<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>James F. Gould</td>
<td>05/30/2013</td>
<td>1.0.0</td>
<td>Initial Revision</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>08/14/2013</td>
<td>1.1.0</td>
<td>Updated “Figure 24 – Domain Create in Sunrise Create Form Example” to the EPPEndodedSignedMark class and added description of changes in version 1.1.0.</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>09/04/2013</td>
<td>1.1.1</td>
<td>Updated the changes for version 1.1.0, and Related Domain Extension section, to support the Domain Info Form and the Related Info Form of the Related Domain Extension. Also added support for the multiple related domain transform commands to the Related Domain Extension.</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>05/01/2014</td>
<td>1.2.0</td>
<td>Updated the Changes from Previous Version section to include the information for the 1.2.0, 1.3.0, and 1.4.0 releases.</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>04/07/2015</td>
<td>1.3.0</td>
<td>Added the new extensions including: Change Poll Mapping (changepoll), Registry Fee Extension (fee), Allocation Token Extension (allocationtoken), IDN Map Extension (idnmap), and the IDN Table Mapping (idntable). The Launch Phase Extension (launch) was updated to be compliant with draft-ietf-eppext-launchphase-05. Also added the subID attribute to the Suggestion Mapping (suggestion).</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>11/2/2015</td>
<td>1.4.0</td>
<td>Added the new extensions including: Verification Code Extension (verificationcode) and China Name Verification Mapping (vsp).</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>11/10/2015</td>
<td>1.4.1</td>
<td>Bumped up the reference of the Verification Code Extension to version 02.</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>12/3/2015</td>
<td>1.5.0</td>
<td>Added information for the 1.7.0 release of the EPP SDK</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>2/18/2016</td>
<td>1.8.0</td>
<td>Added information for the 1.8.0 release of the EPP SDK.</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>3/25/2016</td>
<td>1.9.0</td>
<td>Updated some of the SSLProtocol configuration description.</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>4/20/2016</td>
<td>1.10.0</td>
<td>Added support for draft-brown-epp-fees-07.</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>10/7/2016</td>
<td>1.11.0</td>
<td>Added support for session pool client transaction identifier generator configuration.</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>12/16/2016</td>
<td>1.12.0</td>
<td>Add description of supporting relaxed contact validation and the use of the new optional EPP.Contact.RelaxedValidation epp.config property.</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>2/16/2017</td>
<td>1.12.1</td>
<td>Add description of supporting enabling or disabling the use of an Entity Resolver with the use of the new optional EPP.UseEntityResolver epp.config property.</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>8/1/2017</td>
<td>1.12.2</td>
<td>Added description of the optional EPP.SendReceiveLogger property for setting the logger used for sent and received packets.</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Date</td>
<td>Version</td>
<td>Changes</td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>4/16/2018</td>
<td>1.13.0</td>
<td>Added support and configuration settings associated with the Login Security EPP Extension.</td>
</tr>
<tr>
<td></td>
<td>5/1/2018</td>
<td>1.13.1</td>
<td>Replaced references of reseller and resellerext to org and orgext.</td>
</tr>
<tr>
<td></td>
<td>7/2/2018</td>
<td>1.13.2</td>
<td>Added definition of the optional EPP.PollMessageFilter property.</td>
</tr>
<tr>
<td></td>
<td>8/6/2018</td>
<td>1.13.3</td>
<td>Added the definition of the EPP.RegistryPolicyAdapters property.</td>
</tr>
<tr>
<td></td>
<td>9/4/2018</td>
<td>1.13.4</td>
<td>Removed WhoWas and Suggestion for the SDK.</td>
</tr>
<tr>
<td></td>
<td>9/7/2018</td>
<td>1.13.5</td>
<td>Updated to support draft-ietf-regext-epp-fees-13.</td>
</tr>
<tr>
<td></td>
<td>9/24/2018</td>
<td>1.13.6</td>
<td>Added support for draft-gould-regext-login-security-policy-00 and updated some of the referenced draft versions.</td>
</tr>
<tr>
<td></td>
<td>10/15/2018</td>
<td>1.13.7</td>
<td>Added support for draft-ietf-regext-validate-04.</td>
</tr>
<tr>
<td></td>
<td>1/17/2019</td>
<td>1.13.9</td>
<td>Updated some of the referenced drafts and RFCs. Also updated the change log for the 1.11.0 release.</td>
</tr>
<tr>
<td></td>
<td>2/20/2019</td>
<td>1.13.10</td>
<td>Updated the 1.11.0 change log to include support for Java 11 and to reflect the change from base support of Java 7 to Java 8.</td>
</tr>
<tr>
<td></td>
<td>5/13/2019</td>
<td>1.13.11</td>
<td>Revised the version of the supported Internet Drafts and RFC’s including the registry fee extension, the change poll extension, the organization mapping, and the organization extension.</td>
</tr>
</tbody>
</table>
## References

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Title</th>
<th>Revision</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scott Hollenbeck</td>
<td>Extensible Provisioning Protocol</td>
<td></td>
<td>8/2009</td>
</tr>
<tr>
<td>Scott Hollenbeck</td>
<td>Extensible Provisioning Protocol (EPP) Transport Over TCP</td>
<td></td>
<td>8/2009</td>
</tr>
<tr>
<td>Scott Hollenbeck</td>
<td>Extensible Provisioning Protocol (EPP) Domain Name Mapping</td>
<td></td>
<td>8/2009</td>
</tr>
<tr>
<td>Scott Hollenbeck</td>
<td>Extensible Provisioning Protocol (EPP) Host Mapping</td>
<td></td>
<td>8/2009</td>
</tr>
<tr>
<td>Scott Hollenbeck</td>
<td>Extensible Provisioning Protocol (EPP) Contact Mapping</td>
<td></td>
<td>8/2009</td>
</tr>
<tr>
<td>Scott Hollenbeck</td>
<td>ConsoliDate Mapping for the Extensible Provisioning Protocol</td>
<td>01</td>
<td>3/22/2005</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>EPP RGP Poll Mapping</td>
<td>00</td>
<td>12/10/2013</td>
</tr>
<tr>
<td>James F. Gould and Mahendra Jain</td>
<td>EPP Low Balance Mapping</td>
<td>00</td>
<td>12/10/2013</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>IDN Language Tag</td>
<td>00</td>
<td>12/10/2013</td>
</tr>
<tr>
<td>James F. Gould and Srikanth Veeramachaneni</td>
<td>Whois Info Extension</td>
<td>00</td>
<td>1/10/2014</td>
</tr>
<tr>
<td>Rupert Fernando</td>
<td>Jobs Contact Extension Mapping</td>
<td>1.1</td>
<td>09/07/2007</td>
</tr>
<tr>
<td>James F. Gould and Mahendra Jain</td>
<td>Premium Domain Extension Mapping</td>
<td>00</td>
<td>1/10/2014</td>
</tr>
<tr>
<td>James F. Gould and</td>
<td>Client Object Attribute Extension Mapping</td>
<td>00</td>
<td>1/2/2014</td>
</tr>
<tr>
<td>Author(s)</td>
<td>Mapping/Extension Description</td>
<td>Version</td>
<td>Date</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>Jeff Faust</td>
<td>Balance Mapping</td>
<td>00</td>
<td>12/16/2013</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Defensive Registration Mapping</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Email Forwarding Mapping</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>James F. Gould</td>
<td>RFC 8334 - Launch Phase Extension</td>
<td></td>
<td>3/2018</td>
</tr>
<tr>
<td>Wil Tan</td>
<td>Nagesh Chigurupali</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gavin Brown</td>
<td>Srikanth Veeramachaneni</td>
<td></td>
<td></td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Personal Registration Extension Mapping</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Verisign Registry Mapping</td>
<td>00</td>
<td>1/10/2014</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Related Domain Extension Mapping</td>
<td>00</td>
<td>1/2/2014</td>
</tr>
<tr>
<td>Nagesh Chigurupali</td>
<td>Namestore Extension Mapping</td>
<td>00</td>
<td>12/10/2013</td>
</tr>
<tr>
<td>Srikanth Veeramachaneni</td>
<td>RFC 8495 - Allocation Token Extension</td>
<td></td>
<td>11/2018</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>RFC 8590 - Change Poll Extension</td>
<td></td>
<td>5/2019</td>
</tr>
<tr>
<td>Trung Tran</td>
<td>Namestore Extension Mapping</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>(Neustar)</td>
<td>RFC 8590 - Change Poll Extension</td>
<td></td>
<td></td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Internationalized Domain Name (IDN) Table Mapping</td>
<td>07</td>
<td>10/15/2018</td>
</tr>
<tr>
<td>Francisco Obispo</td>
<td>Internationalized Domain Name Mapping Extension</td>
<td>02</td>
<td>1/5/2015</td>
</tr>
<tr>
<td>Luis Enrique Munoz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Uniregistry)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Francisco Obispo</td>
<td>Internationalized Domain Name Mapping Extension</td>
<td>02</td>
<td>1/5/2015</td>
</tr>
<tr>
<td>Luis Enrique Munoz</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Uniregistry)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Roger Carney</td>
<td>Registry Fee Extension</td>
<td>03</td>
<td>12/29/2014</td>
</tr>
<tr>
<td>(GoDaddy)</td>
<td>Registry Fee Extension</td>
<td>04</td>
<td>2/17/2015</td>
</tr>
<tr>
<td>Gavin Brown</td>
<td>Registry Fee Extension</td>
<td>05</td>
<td>08/12/2015</td>
</tr>
<tr>
<td>(CentralNic)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feature</td>
<td>Page</td>
<td>Date</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------------------------</td>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Registry Fee Extension</td>
<td>06</td>
<td>11/4/2015</td>
</tr>
<tr>
<td></td>
<td></td>
<td>07</td>
<td>04/8/2016</td>
</tr>
<tr>
<td></td>
<td></td>
<td>08</td>
<td>10/1/2017</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>05/1/2019</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Verification Code Extension</td>
<td>06</td>
<td>1/10/2019</td>
</tr>
<tr>
<td>Xie Jiagui</td>
<td>China Name Verification Mapping</td>
<td>00</td>
<td>10/26/2016</td>
</tr>
<tr>
<td>Liu Hongyan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Teleinfo)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>James F. Gould</td>
<td>China Name Verification Mapping</td>
<td>00</td>
<td>10/26/2016</td>
</tr>
<tr>
<td>L. Zhou</td>
<td>Organization Mapping</td>
<td>00</td>
<td>10/26/2016</td>
</tr>
<tr>
<td>N. Kong</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X. Lee</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CNNIC)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. Qi</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Teleinfo)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Organization Mapping</td>
<td>00</td>
<td>10/26/2016</td>
</tr>
<tr>
<td>L. Zhou</td>
<td>Organization Extension</td>
<td>01</td>
<td>4/9/2019</td>
</tr>
<tr>
<td>N. Kong</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X. Lee</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(CNNIC)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C. Qi</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Teleinfo)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Organization Extension</td>
<td>01</td>
<td>4/9/2019</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Login Security Extension</td>
<td>01</td>
<td>4/9/2019</td>
</tr>
<tr>
<td>Lin Jia</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Verisign)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Roger Carney</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jody Kolker</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(GoDaddy)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Registry Mapping</td>
<td>03</td>
<td>8/17/2018</td>
</tr>
<tr>
<td></td>
<td></td>
<td>04</td>
<td>10/22/2018</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Launch Phase Policy Extension</td>
<td>01</td>
<td>8/17/2018</td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Login Security Policy Extension</td>
<td>03</td>
<td>1/16/2019</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------</td>
<td>----</td>
<td>----------</td>
</tr>
<tr>
<td>Roger Carney</td>
<td>Validate Mapping</td>
<td>04</td>
<td>10/11/2018</td>
</tr>
<tr>
<td>Joseph Snitker</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>James F. Gould</td>
<td>Unhandled Namespaces</td>
<td>01</td>
<td>3/26/2019</td>
</tr>
<tr>
<td>Martin Casanova(SWITCH)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTLD</td>
<td>Consolidated Top-Level Domain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS</td>
<td>Delegation Signer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPP</td>
<td>Extensible Provisioning Protocol</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDN</td>
<td>Internationalized Domain Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFC</td>
<td>Request for Comments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RGP</td>
<td>Registry Grace Period</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML</td>
<td><a href="http://www.w3c.org/XML/">http://www.w3c.org/XML/</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Contents

REFERENCES ........................................................................................................ V
DEFINITIONS, ACRONYMS, AND ABBREVIATIONS .......................................... IX

1. INTRODUCTION .................................................................................................. 1

2. CHANGES FROM PREVIOUS VERSION ................................................................ 3
   2.1 VERSION 1.11.0 .................................................................................................. 3
   2.2 VERSION 1.10.0 .................................................................................................. 6
   2.3 VERSION 1.9.0 .................................................................................................. 6
   2.4 VERSION 1.8.0 .................................................................................................. 7
   2.5 VERSION 1.7.0 .................................................................................................. 8
   2.6 VERSION 1.6.0 .................................................................................................. 8
   2.7 VERSION 1.5.0 .................................................................................................. 8
   2.8 VERSION 1.4.0 .................................................................................................. 9
   2.9 VERSION 1.3.0 .................................................................................................. 9
   2.10 VERSION 1.2.0 ................................................................................................ 9
   2.11 VERSION 1.1.0 ................................................................................................ 9
   2.12 VERSION 1.0.0 ................................................................................................10

3. SUPPORTED TRANSPORTS .................................................................................. 11

4. QUICK START INSTRUCTIONS .......................................................................... 12
   4.1.1 RUNNING SDK TESTS VIA STUB SERVER ............................................. 13
   4.13 CHANGES REQUIRED TO INTERFACE WITH VERISIGN SERVERS ...... 13
   4.14 SET THE CLASSPATH OF THE APPLICATION ....................................... 15
   4.15 EXAMPLE CODE TO INITIALIZE THE VERISIGN BUNDLE EPP SDK AND START SESSION ..... 16
   4.16 EXAMPLE CODE TO INITIALIZE THE VERISIGN BUNDLE EPP SDK AND USING SESSION POOL ... 16
   4.17 EXAMPLE / TEST PROGRAMS .................................................................... 18

5. SDK OVERVIEW .................................................................................................. 23
   5.1 SDK DIRECTORIES ....................................................................................... 24
   5.2 SDK PACKAGES ............................................................................................ 24

6. SDK CONFIGURATION ....................................................................................... 26
   6.1 CONFIGURATION FILE .................................................................................. 26
   6.18 LIBRARIES .................................................................................................. 37
   6.19 DIAGNOSTIC AND ERROR LOGGING ...................................................... 38
   6.19.1 Basic Configuration Mode (EPP.LogLevel and EPP.LogFile) .................. 38
   6.19.2 Log4J Configuration File Mode (EPP.LogCfgFile and EPP.LogCfgFileWatch) ........ 38
   6.19.3 Custom Mode ......................................................................................... 40
   6.19.4 SDK Log Categories .............................................................................. 40
   6.20 ADDING AN EPP COMMAND MAPPING TO THE SDK ......................... 41

7. GENERIC EPP CLIENT INTERFACES .................................................................. 43
   7.1 EPPAPPLICATION ....................................................................................... 43
<table>
<thead>
<tr>
<th>Page</th>
<th>Extension</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.27.18</td>
<td>Launch Extension</td>
<td>188</td>
</tr>
<tr>
<td>13.27.19</td>
<td>PersReg Extension</td>
<td>199</td>
</tr>
<tr>
<td>13.27.20</td>
<td>Related Domain Extension</td>
<td>201</td>
</tr>
<tr>
<td>13.27.21</td>
<td>Change Poll Extension</td>
<td>208</td>
</tr>
<tr>
<td>13.27.22</td>
<td>Registry Fee Extension</td>
<td>209</td>
</tr>
<tr>
<td>13.27.23</td>
<td>Allocation Token Extension</td>
<td>212</td>
</tr>
<tr>
<td>13.27.24</td>
<td>IDN Map Extension</td>
<td>213</td>
</tr>
<tr>
<td>13.27.25</td>
<td>Verification Code Extension</td>
<td>214</td>
</tr>
<tr>
<td>13.27.26</td>
<td>Organization Extension</td>
<td>215</td>
</tr>
<tr>
<td>13.27.27</td>
<td>Login Security Extension</td>
<td>216</td>
</tr>
<tr>
<td>13.27.28</td>
<td>Launch Policy Extension</td>
<td>218</td>
</tr>
<tr>
<td>13.27.29</td>
<td>Login Security Policy Extension</td>
<td>220</td>
</tr>
<tr>
<td>13.27.30</td>
<td>Validate Extension</td>
<td>222</td>
</tr>
</tbody>
</table>
1. Introduction
This document provides instructions on how to use the Verisign Bundle Extensible Provisioning Protocol (EPP) Software Development Kit (SDK) includes all of the API’s required to interface with the Verisign EPP servers that include:

1. COM/NET Shared Registry System (SRS), hereon referred to as SRS, supports the .COM and .NET TLDs. The SRS commands require the NameStore Extension for specifying the target registry. The Registry Mapping can be used to provide the list of available TLDs as well as the feature and policy information for the TLDs.
2. Namestore Platform hereon, referred to as Namestore, supports the following products:
   a. Consolidated Top-Level Domain (CTLD) - Is a platform of domain name registries like .JOBS, .TV, .CC, and a new set of Top Level Domains (TLDs). CTLD commands require the Namestore Extension for specifying the target registry. The Registry Mapping can be used to provide the list of available TLDs as well as the feature and policy information for the TLDs.

The SDK includes a full implementation of the EPP specifications independent of the services supported by the Registry services (i.e. Namestore and SRS). The com.verisign.epp.interfaces.EPPDomain and com.verisign.epp.interfaces.EPPHost fully support the IETF Domain and Host mappings, while com.verisign.namestore.interfaces.NSDomain and com.verisign.namestore.interfaces.NSHost provide convinence subclasses for easily passing the Namestore Extension and for supporting extension API’s like Sync and RGP Restore Request/Report. The SDK also provides a Stub Server that can run over TCP or SSL, and a set of test client code (*Tst.java) that validates the SDK API’s and can be used as samples. For example, com.verisign.epp.namestore.interfaces.NSPollTst includes sample code for processing each of the supported poll messages produced by Namestore and the SRS.

The instructions provided include an overview of the Verisign Bundle EPP SDK, how to configure it, how to use it, and how to extend it. Please see http://www.verisigninc.com/en_US/products-and-services/domain-name-services/registry-products/epp-sdk for updates to the Programmer’s Guide between Verisign Bundle EPP SDK releases.

The Verisign Bundle EPP SDK provide detailed interface information in HTML Javadoc. This document does not duplicate the detailed interface information contained in the HTML Javadoc. Descriptions are provided of the main interface elements, the pre-conditions, the post-conditions, and example code.
It is assumed that the reader has reviewed the associated EPP specifications and has a general understanding of the EPP concepts. Much of the EPP details are encapsulated in the SDK, but having a solid understanding of the EPP concepts will help in effectively using the SDK.
2. Changes from Previous Version

This section describes changes between major and minor versions of the SDK. The version numbers reflect SDK version numbers and not version numbers of the Programmer’s Guide.

2.1 Version 1.11.0

1. Updated draft draft-ietf-regext-epp-fees to draft-ietf-regext-epp-fees-16.
2. Updated change poll extension to RFC 8590.
3. Updated the EPP extension RFCs to include both the TXT and HTML documents and to follow the naming convention epp-<extension>-rfc<rfc-number>.<ext>.
4. Updated organization mapping and organization extension to RFC 8543 and RFC 8544, respectively.
5. Updated from XercesJ 2.11.0 to 2.12.0.
6. Added support for draftietf-regext-login-security-01.
7. Updated compilation to target Java 8 instead of Java 7.
8. Updated to compile and run with Java 11. Added jaxb-api-2.3.1.jar and fixed many warnings due to deprecated classes and methods.
10. Added support for draft-gould-casanova-regext-unhandled-namespaces-00.
13. Added support for draft-gould-regext-login-security-policy-03.
15. Added support for draft-gould-regext-launch-policy-01.
17. Updated draft-ietf-regext-verificationcode to draft-ietf-regext-verificationcode-06.
22. Removed who was and suggestion from the SDK.
24. Added support for secure packet logging by creating the com.verisign.epp.util.EPPSendReceiveLogger interface along with the default com.verisign.epp.util.EPPRawSendReceiveLogger class and the com.verisign.epp.namestore.util.EPPSecureSendReceiveLogger class. The default is to use the EPPRawSendReceiveLogger with the option of overriding the default using the EPPSendReceiveLogger epp.config property. Set the property to com.verisign.epp.namestore.util.EPPSecureSendReceiveLogger to get the password and auth info attributes logged with the "MASKED" value.
25. Fixed a bug in com.verisign.epp.codec.emailFwd.EPPEmailFwdCheckResult that set the check reason to the email forwarding name on decode. Updated the
EPPEmailFwdCheckResult.equals methods to ensure that all attributes are checked. Made related cleanup changes to
com.verisign.epp.codec.defReg.EPPDefRegCheckResult.

26. Added an additional check in
com.verisign.epp.util.EPPSchemaCachingEntityResolver to ensure that XML schemas in the only XML schemas that can be dynamically
loaded.

27. Deprecated com.verisign.epp.namestore.interfaces.NSSubProduct. Removed BZ and revised COM, NET, ORG, and NAME to use the a-label TLD in
com.verisign.epp.namestore.interfaces.NSSubProduct.

28. Changed the target JDK to be 1.7 instead of 1.6 due to changing to use TLSv1.2 as the default EPP.SSLProtocol setting.

29. hanged to use TLSv1.2 as the default EPP.SSLProtocol setting across each of the
epp.config files.

30. Added support for draft-ietf-regext-epp-fees-08 that adds the
urn:ietf:params:xml:ns:fee-0.23 XML namespace with it's associated fee-0.23.xsd XML schema, the com.verisign.epp.codec.fee.v23 package, and related classes.

31. Added support for draft-ietf-regext-epp-fees-13 that adds the
urn:ietf:params:xml:ns:fee-1.0 XML namespace with it's associated fee-
1.0.xsd XML schema, the com.verisign.epp.codec.fee.v1_0 package, and related
classes.

32. Added the com.verisign.epp.interfaces.EPPLoginAdapter class, the optional
"EPP.LoginAdapter" epp.config property, and added invoking the set
LoginAdapter within com.verisign.epp.interfaces.EPPSession to support adding
an EPP login extension.

33. Added hasService(String) : boolean and hasExtensionService(String) : boolean to
comeverisign.epp.codec.gen.EPPServiceMenu and
comeverisign.epp.codec.gen.EPPGreeting to be able to easily identify whether the
server supports a specific object or extension service.

34. Added EPP.UseUserAgent (EPPEnv.isUseUserAgent() : boolean) and
EPP.SDKVersion (EPPEnv.getSDKVersion() : boolean) epp.config properties to
support providing the user agent information in the login security extension.

35. Added support for the Login Security Extension.

36. Added the getNamespace() : String method to the
comeverisign.epp.codec.gen.EPPCodecComponent interface and all of the
implemented classes to make all components self describing for the XML
namespace.

37. Added the comeverisign.epp.codec.gen.EPPPollMessageFilter utility class to filter
EPP poll message responses based on the login services (object and command /
response) supported by the client.

38. Modified the com.verisign.epp.serverstub.GenHandler.doPoll(EPPEvent, Object)
method to call comeverisign.epp.codec.gen.EPPPollMessageFilter in no-op and
standard mode to demonstrate filtering the poll messages based on the client login
services.

39. Added the getNamespace() : String method to some of the responses
(EPPRgpPollResponse, EPPRelatedDomainExtDeleteResp,
EPPBalanceInfoResp, EPPLowBalancePollResponse, EPPNameVerificationPendActionMsg) to ensure that they return the correct XML namespace.

40. Removed old fee extension versions including v06, v07, and v08.
41. Replaced references of reseller and resellerext to org and orgext.
42. Updated draft draft-ietf-regext-allocation-token to draft-ietf-regext-allocation-token-08.
43. Updated draft draft-ietf-regext-change-poll to draft-ietf-regext-change-poll-08.
44. Updated draft draft-ietf-regext-epp-fees to draft-ietf-regext-epp-fees-11.
46. Updated draft-ietf-regext-org to draft-ietf-regext-org-10.
47. Updated draft-ietf-regext-org-ext draft to draft-ietf-regext-org-ext-09.
49. Added support for draft-gould-carney-regext-registry-03.
50. Replaced the expired test SMDs, SMD CRL, and revoked SMD list based on a posting by ICANN.
51. Added support for draft-gould-regext-login-security-01.
52. Fixed some of the XML namespace prefix and URI handling from within com.verisign.epp.codec.gen.EPPValue to support the different poll message filters.
53. Renamed com.verisign.epp.codec.gen.EPPPollMessageFilter to com.verisign.epp.codec.gen.EPPXmlMsgPollMessageFilter, added com.verisign.epp.codec.gen.EPPExtValuePollMessageFilter, and added com.verisign.epp.codec.gen.EPPFullExtValuePollMessageFilter to support three different approaches to filtering poll message data that is unsupported by the client based on the client's login services. The com.verisign.epp.serverstub.GenHandler.doPoll(EPPEvent, Object) is predefined to filter via EPPFullExtValuePollMessageFilter, but the filtering can be replaced to use one of the other filters.
54. Removed com.verisign.epp.util.TestErrorHandler and replaced it's used with com.verisign.epp.util.EPPXMLErrorHandler in the test classes.
55. Added the maskString(String) : String method to the EPPSendReceiverLogger class and it's derived classes for use in the EPPXMLErrorHandler class for masking sensitive XML parser errors like the 'pwType'.

VeriSign Inc. Proprietary Information
Verisign Bundle EPP SDK Programmer's Guide • page 5
2.2 **Version 1.10.0**

1. Made setting of the XML entity resolver, based on the com.verisign.epp.util.EPPSchemaCachingEntityResolver, a configurable setting using the optional EPP.UseEntityResolver config property, with a default value of true.

2. Added relaxed contact validation support used to transition a thin registry to thick. Relaxed validation leverages a new contact XSD (contact-1.0-relaxed.xsd) that makes the <contact:name>, <contact:addr>, <contact:city>, <contact:street>, <contact:cc>, and <contact:email> elements optional. The new optional EPP.Contact.RelaxedValidation epp.config property must be set to true for the client and server stub to use and validate to the relaxed schema.

3. Removed the setting for the EntityResolver in com.verisign.epp.util.EPPSchemaCachingParser and deprecated com.verisign.epp.util.EPPSchemaCachingEntityResolver, since the XML schemas are pre-cached in com.verisign.epp.util.EPPSchemaCachingParser, so there is no need to dynamically load them anymore.


5. Added logging to EPPGenHandler.handleEvent(EPPEvent, Object).

6. Added getFirstPostalInfo() : EPPContactPostalDefinition to EPPContactInfoResp and EPPContact.

7. Added EPPDomainInfoResp.getContactByType(String) : EPPDomainContact, EPPDomainInfoResp.getAdminContact() : EPPDomainContact, EPPDomainInfoResp.getTechContact() : EPPDomainContact, and EPPDomainInfoResp.getBillingContact() : EPPDomainContact.

8. Added com.verisign.epp.pool.EPPClientTransIdGenerator interface, concrete com.verisign.epp.pool.ClientTransIdGenerator class, and the optional clientTransIdGenerator pool property to define the concrete EPPClientTransIdGenerator to use for generating client transaction identifiers for the login and logout commands for the pool.

9. Added EPPSession.sendMessage(EPPMessage) to help in testing a negative test case, but also as a potentially useful method for sending any EPPMessage to the session.

10. Added more debug logging to EPPSSLImpl.

11. Added validation of the EPPMessage type in the EPPDispatcher and the EPPByteArrayDispatcher to ensure that receiving an EPPResponse or EPPGreeting in the server will return an appropriate error message.

2.3 **Version 1.9.0**

1. Added additional SSL Protocol options in the comments of the EPP.SSLProtocol property and set the default protocol to TLSv1 in the epp.config.
2. Updated the EPPUtil.decodeBooleanAttr(Element, String) method to properly identify a non-existent attribute and to enhance the format of the EPPDecodeException value to easier support.

3. Made the refundable and grace-period attributes of com.verisign.epp.codec.fee.v09.EPPFeeValue truly optional with no default value, while the applied attribute remained with the "immediate" default value based on the default value in the XSD. Added "has" methods for these attributes, changed the refundable to a Boolean to support a null value, and changed the encode and decode methods to handle non-existent attributes.

4. Added convenient constructor that takes both a EPPFeeValue and a currency (String) parameter for the com.verisign.epp.codec.fee.v09 classes of EPPFeeTransform, EPPFeeCreate, EPPFeeRenew, EPPFeeTransfer, and EPPFeeUpdate.

5. Added methods for passing and returning the Base64 encoded signed code into com.verisign.epp.codec.verificationcode.EPPEncodedSignedCodeValue. The new EPPEncodedSignedCodeValue methods: decodeValue(byte[]), decodeValue(String), encodeValue(boolean) : String, encodeValueByteArray() : byte[], and encodeValueByteArray(boolean) : byte[] where added.

6. Added support for running the com.verisign.epp.verificationcode.china.ChinaVerificationCodeTst test without the VSP pool and by loading the DNVC from the dnvc.b64 file and the RNVC from the rnvc.b64 file.

2.4 Version 1.8.0

1. Added hasTrustAnchor(): boolean and getTrustAnchor() : TrustAnchor to EPPSignedCode for getting the matching trust anchor from the PKIXParameters upon successful validation.


3. Added the com.verisign.epp.verificationcode.china.ChinaVerificationCodeTst along with associated changes to the build and stub server behavior to support a test of the 7 predefined verification flows that will work against the Stub Server and the OT&E servers.

4. Added com.verisign.epp.pool.EPPSessionPool.hasSystemSessionPool(String) : boolean method to determine if a specific system session pool existed.

5. Fixed com.verisign.epp.codec.verificationcode.EPPVerificationCode.getVspId() : int to return UNDEFINED instead of throwing NumberFormatException when the vspId is not an integer.

6. Added validation of the verification code (vsp-id and code types) using formatted trust anchor alias names in the trust store.
2.5 Version 1.7.0

1. Turned on error handling with EPPXMLErrorHandler by default within EPPSchemaCachingParser. It can be disabled by calling EPPSchemaCachingParser.setErrorHandler(null).

2. Added support for an XMLSignature Parser Pool to increase performance in parsing and validating the XML Signature for the Verification Code extension and the Launch extension.

3. Added support for draft-gould-eppext-verificationcode-03, which added regular expression pattern for the format of the verification code token value in the XML schema.

4. Added support for draft-brown-epp-fees-05 and draft-brown-epp-fees-06.

2.6 Version 1.6.0

5. Added the Verification Code Extension (verificationcode), that complies with draft-gould-eppext-verificationcode-01, to the Verisign Bundle.

6. Added the China Name Verification Mapping (vsp), that complies with draft-xie-eppext-nv-mapping-01, to the Verisign Bundle.

7. Bumped down the info level logs to debug level logs in com.verisign.epp.util.EPPSchemaCachingParser.

2.7 Version 1.5.0

8. Added the EPPSession.sendPacket(byte []) method and EPPXMLStream.writePacket(byte[], OutputStream) to enable sending a packet through the session within having to go through the Codec.

9. Added the Change Poll Mapping (changepoll), that complies with draft-gould-change-poll-00.

10. Added the Registry Fee Extension (fee), that complies with draft-brown-epp-fees-03 and draft-brown-epp-fees-04.

11. Added the Allocation Token Extension (allocationtoken), that complies with draft-gould-allocation-token-00.

12. Updated Launch Phase Extension (launch) to comply with draft-ietf-eppext-launchphase-03 and subsequently draft-ietf-eppext-launchphase-05, by adding support for the Trademark Check Form.

13. Added the IDN Map Extension (idnmap), that complies with draft-ietf-eppext-idnmap-01.

14. Added the subID attribute in the info element of NameSuggestion.

15. Added the IDN Table Mapping (idntable), that complies with draft-gould-idntable-02.
2.8 Version 1.4.0

16. Switched to be dependent on Java 6 instead of Java 5. The launch extension already was dependent on Java 6 and with the new dependency on the use of JAXB, the entire SDK is being moved to be dependent on Java 6.

17. Fixed a bug with encoding the XSD dateTime type in EPPUtil by using 4 digit precision for the seconds. The fix was to use a default of 3 digit precision (milliseconds) on the encode and to use JAXB DatatypeConverter.parseDateTime for parsing the dateTime value. Access methods were added (getTimeInstantFormat(): String and setTimeInstantFormat(String)) to enable changing the encoding format if needed.

2.9 Version 1.3.0

18. Updated to the 25nov13 test SMD's (SMD's without whitespace) and test SMD Revocation List. The SMD Revocation List was converted from UTF-8 to ASCII.

19. Updated to support draft-tan-epp-launchphase-12 by updating the launch-1.0.xsd, updating the draft-tan-epp-launchphase.txt in the doc directory, and updating the code and tests to support the new validatorID optional attribute.

2.10 Version 1.2.0

20. Fixed the XML namespace handling in the com.verisign.epp.codec.registry.EPPRegistryRegEx class and some of the client classes to com.verisign.epp.codec.registry.EPPRegistryRegEx.

21. Changed com.verisign.epp.codec.signedMark.EPPSignedMark to handle XML parsing exception. This is specially required while parsing the XML obtained from the encoded SMD.

2.11 Version 1.1.0


23. Added Domain Info Form and the Related Info Form to the info command of the Related Domain Extension, version 1.2. Added support for the two forms to com.verisign.epp.codec.relateddomainext.EPPRelatedDomainExtInfo as well as com.verisign.epp.interfaces.EPPRelatedDomain.

24. Addressed issue with using the sample SMD's with the signedMark and the default XML parser setting of normalizing the XML.

25. Added the EPPSignedMark(EPPEncodedSignedMark) constructor to convert an encodedSignedMark to a signedMark.

27. Split com.verisign.epp.codec.signedMark.EPPSignedMark into com.verisign.epp.codec.signedMark.EPPSignedMark and com.verisign.epp.codec.signedMark.EPPEncodedSignedMark to isolate the base64 encoding and to address issues with retaining any extra whitespace contained in the signed mark to address the validation issue with the ICANN sample SMD's.

28. Updated the com.verisign.epp.codec.launch.EPPLaunchTst tests to include validation of all of the ICANN sample Signed Mark Data (SMD) files. Added the ICANN sample SMD files to the distribution to support the tests.

29. The certs.crl file was split into the eppsdk.crl and the tmch-pilot.crl. The eppsdk.crl includes the CRL for the signed marks signed by the EPP SDK and the tmch-pilot.crl includes the CRL provided by ICANN for testing.

30. Added the test ICANN CA certificate to signedMarkTrust.jks to support validating the ICANN sample Signed Mark Data (SMD).

31. Added SMD revocation list validation to the LaunchDomainHandler class.

32. Added the ICANN test SMD revocation list (smd-test-revocation.csv) to the distribution to support the testing of the ICANN sample Signed Mark Data (SMD).

33. Created the com.verisign.epp.codec.signedMark.SMDRevocationList and com.verisign.epp.codec.signedMark.RevokedSMD classes to decode the SMD revocation list and to include the SMDRevocationList.isRevoked(EPPSignedMark) : boolean method to determine if a signed mark is revoked.

2.12 Version 1.0.0

1. Merged the NameStore / SRS Bundle SDK 3.15.0.5, the Dotname SDK 1.5.0.1, the Launch SDK 2.1.0.0 into a single Verisign Bundle EPP SDK distribution.

2. Added the Registry Mapping

3. Added the Related Domain Extension
3. **Supported Transports**

The SDK supports TCP and SSL as transports. The default configuration of the SDK is to use the TCP transport for the ease of setup. The TCP and the SSL transports follow RFC 5734 “Extensible Provisioning Protocol (EPP) Transport Over TCP”.
4. Quick Start Instructions

The Verisign Bundle EPP SDK is distributed in two forms, a source code distribution and a binary distribution. Both distributions are preconfigured to run a TCP/IP Stub Server and include a suite of tests that run against the TCP/IP Stub Server. The Stub Server is described in section 10. The following steps are common to either transport:

1. Uncompress the Verisign Bundle EPP SDK. With the source distribution, the Unix filename is `epp-launch-bundle-${BUILD_VER}-src.tar.gz` and the Windows filename is `epp-launch-bundle-${BUILD_VER}-src.zip`. With the binary distribution, the Unix filename is `epp-launch-bundle-${BUILD_VER}-bin.tar.gz` and the Windows filename is `epp-launch-bundle-${BUILD_VER}-bin.zip`. `${BUILD_VER}` is the version number for the release (e.g. 1.0.0).

2. Change to the Verisign Bundle EPP SDK directory: `epp-verisign-${BUILD_VER}/bundles/verisign`

3. Edit any configuration changes in `epp.config`. Build related changes could be made in `build.properties`.

4. Execute one of the Ant build.xml targets defined in Table 1 – Verisign Bundle build.xml Targets.

<table>
<thead>
<tr>
<th>Target</th>
<th>Distribution (src, bin, or both)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean</td>
<td>both</td>
<td>Cleans the built files and directories</td>
</tr>
<tr>
<td>compile</td>
<td>src</td>
<td>Compiles the source files</td>
</tr>
<tr>
<td>dist</td>
<td>src</td>
<td>Creates the distributions (-Dbuild.version required)</td>
</tr>
<tr>
<td>dist-bin</td>
<td>src</td>
<td>Creates the binary distribution (-Dbuild.version required)</td>
</tr>
<tr>
<td>dist-src</td>
<td>src</td>
<td>Creates the source distribution (-Dbuild.version required)</td>
</tr>
<tr>
<td>doc</td>
<td>src</td>
<td>Creates the HTML API documentation</td>
</tr>
<tr>
<td>format</td>
<td>src</td>
<td>Formats the source code</td>
</tr>
<tr>
<td>init</td>
<td>both</td>
<td>Initializes the build for rest of targets</td>
</tr>
</tbody>
</table>
| jar      | src                             | Creates the jar file (epp-verisign-bundle-{$BUILD_VER}.jar)  
|          |                                 | Default for src distribution                         |
1.1 Running SDK Tests via Stub Server

The SDK works with JDK 1.5 and higher, but JDK 1.6 is needed when using the launch extension due to the dependency on XML-DSig. Follow the directions below to run the suite of tests against the TCP/IP Stub Server. Use build.bat on Windows and build.sh on Unix to execute the Ant targets. The directions only reference build.sh, so replace build.sh with build.bat when running in Windows.

- build.sh test-client-server

When running the test-client-server target, the following is a sample result of a successful execution.

BUILD SUCCESSFUL
Total time: 1 minute 11 seconds

4.13 Changes Required to Interface with Verisign Servers

The SDK configuration has to be changed to communicate with the real Verisign Servers, since the transport is SSL for the Production servers. The client mappings/extensions that are used might have to be changed. Set the properties in “Table 2 - Changes Required to Interface with Verisign Servers via SSL” in epp.config.

<table>
<thead>
<tr>
<th>Property</th>
<th>Update To</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPP.MapFactories</td>
<td>MapFactories for products that will be provisioned. For example, if .tv and .cc domains are only provisioned, set EPP.MapFactories to:</td>
<td>All supported map factories.</td>
</tr>
<tr>
<td></td>
<td>• com.verisign.epp.codec.do</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>main.EPPDomainMapFactory</td>
<td></td>
<td>com.verisign.epp.codec.host.EPPHostMapFactory</td>
</tr>
<tr>
<td>EPP.CmdRspExtensions</td>
<td>Dependent command/response extensions needed by the products that will be provisioned. For example, if .cc domains are only provisioned, set EPP.CmdRspExtensions to:</td>
<td>All supported extension factories</td>
</tr>
<tr>
<td></td>
<td>• com.verisign.epp.codec.namestoreext.EPPNamestoreExtEPPNamestoreExtFactory</td>
<td></td>
</tr>
<tr>
<td>EPP.CmdRspExtensions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPP.SSLKeyFileName</td>
<td>JSSE keystore file name that contains the CA issued certificate chain and the associated private key.</td>
<td>../../lib/keystore/testkey</td>
</tr>
<tr>
<td>EPP.SSLPassPhrase</td>
<td>Password needed to access EPP.SSLKeyFileName file.</td>
<td>passphrase</td>
</tr>
<tr>
<td>EPP.SSLKeyPassPhrase</td>
<td>Password needed to access the private key defined in the EPP.SSLKeyFileName file. If this property is not defined, EPP.SSLPassPhrase will be used for accessing both the keystore and the private key.</td>
<td>Not Defined</td>
</tr>
<tr>
<td>EPP.SSLProtocol</td>
<td>SSL protocol of the configured provider. Specify supported protocol of the JRE or of the selected JSSE provider. Example values include: SSL, SSLv2, SSLv3, TLS, TLSv1, TLSv1.1, TLSv1.2.</td>
<td></td>
</tr>
<tr>
<td>EPP.SSLKeyStore</td>
<td>SSL keystore file type.</td>
<td>JKS</td>
</tr>
<tr>
<td>EPP.SSLTrustStoreFileName</td>
<td>The trust store is a file that contains the certificate or chain of certificates that this client trusts. If this property is not defined, than the default JRE truststore ($JAVA_HOME/lib/security/cacerts) will be used.</td>
<td>../../lib/keystore/testkey</td>
</tr>
<tr>
<td></td>
<td><strong>Recommend commenting out this property.</strong></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>EPP.SSLTrustStorePassPhrase</td>
<td>Password for accessing the trust store defined by the EPP.SSLTrustStoreFileName property. This property is required if EPP.SSLTrustStoreFileName is defined.</td>
<td>passphrase</td>
</tr>
<tr>
<td></td>
<td><strong>Recommend commenting out this property.</strong></td>
<td></td>
</tr>
<tr>
<td>EPP.ClientSocketName</td>
<td>Class used to make client connections.</td>
<td>com.verisign.epp.transport.client.EPPPlainClientSocket</td>
</tr>
<tr>
<td><strong>Set this to</strong></td>
<td><strong>com.verisign.epp.transport.client.EPPSSLClientSocket for SSL.</strong></td>
<td></td>
</tr>
<tr>
<td>EPP.ServerName</td>
<td>Set to the Verisign server name or IP address. The following are possible values:</td>
<td>localhost</td>
</tr>
</tbody>
</table>
|                                 | 1. NameStore OTE - otessl.verisign-grs.com  
2. NameStore Production – namestoressl.verisign-grs.com  
3. SRS OTE – epp-ote.verisign-grs.com  
4. SRS Production - epp.verisign-grs.net  
5. NAME OTE - nameeppote.verisign-grs.com  
6. NAME Production - nameepp.verisign-grs.net |                |
| **Recommend configuring and using** | **the session pooling feature defined in section 1.1 to interface with multiple Verisign servers with a single client.**                                                                                       |                |
| EPP.ServerPort                   | Set to the Verisign server port number, which should be 700.                                                                                                                                             | 1700           |

**The SDK uses JSSE in the JDK.** Use the EPP.SSL properties of the SDK to configure JSSE. Please consult the JSSE documentation for details of how to construct java keystores and truststores.

### 4.14 Set the classpath of the application

When including the SDK in a client program, the following dependent .jar files must be included in the CLASSPATH:

- `epp-verisign-${BUILD_VER}/lib/epp/epp-verisign-bundle-${BUILD_VER}.jar`  
  This .jar file includes all of the Verisign EPP mappings and extensions supported by the Verisign servers.
- `epp-verisign-${BUILD_VER}/lib/*.jar`  
  These are dependent third party .jar files including JUnit, Log4j, PoolMan, and XercesJ
4.15 Example Code to initialize the Verisign Bundle EPP SDK and Start Session

The following code can be used to initialize a session with an EPP Server.

```java
public static void main(String[] args) {
    try {
        EPPApplicationSingle.getInstance().initialize("epp.config");
    } catch (EPPCommandException e) {
        e.printStackTrace();
    }

    try {
        EPPSession session = new EPPSession();
        session.setTransId("ABC-12345");
        session.setVersion("1.0");
        session.setLang("en");
        session.setClientId("myname");
        session.setPassphrase("mypass");
        session.initSession();
        // Invoke commands on Interface classes here...
    } // try
    catch (EPPCommandException e) {
        EPPResponse response = session.getResponse();
        // Is a server specified error?
        if ((response != null) && (!response.isSuccess())) {
            System.out.println("Server Error : " + response);
        }
        else {
            e.printStackTrace();
            System.out.println("initSession Error : " + e);
        }
    }
}
```

4.16 Example Code to initialize the Verisign Bundle EPP SDK and using Session Pool

The example provided in section 4.15 shows a simple method of creating a new EPP session. The recommended approach is to use a session pool to manage sessions and to borrow, return, and invalidate sessions in the pool as needed. The session pool manages idle timeouts, manages absolute timeouts, maintains the configured number of sessions,
and provides for a configurable session create retry. There can be more than one session pool configured, each with a pool name, so that the client can manage pools with different settings (server info, protocol, transport, login name, login password, SSL settings, number of sessions, etc.) from a single client. Refer to section 1.1 for more information on the session pools. The example below shows initializing the SDK with initializing the session pools and borrowing / returning / invalidating a session in the pool using the “test” session pool.

```java
try {
    EPPApplicationSingle.getInstance().initialize("epp.config");
    EPPSessionPool.getInstance().init();
}
catch (Exception e){
  e.printStackTrace();
  System.exit(1);
}

EPPSession theSession = null;
try {
    theSession = EPPSessionPool.getInstance().borrowObject("test");
    NSDomain theDomain = new NSDomain(theSession);
    theDomain.addDomainName("example.com");
    theDomain.setSubProductID(NSSubProduct.COM);
    EPPDomainCheckResp theResponse = theDomain.sendCheck();
    ...
}
catch (EPPCommandException ex) {
  if (ex.hasResponse()) {
    if (ex.getResponse().getResult().shouldCloseSession()) {
        EPPSessionPool.getInstance().invalidateObject("test", theSession);
        theSession = null;
    }
  }
  else {
    EPPSessionPool.getInstance().invalidateObject("test", theSession);
    theSession = null;
  }
}
finally {
  if (theSession != null)
```
EPPSessionPool.getInstance().returnObject(theSession);

// Cleanly close the session pools at the end of the program
EPPSessionPool.getInstance().close();

4.17 Example / Test Programs
The best examples are running programs. The Verisign Bundle EPP SDK includes a
suite of client tests that are fully run against a Stub Server and that can be used as
samples of using the SDK. Download the source distribution of the Verisign Bundle EPP
SDK to review the source of the tests included in Table 3 - SDK Interface Test Classes.

Table 3 - SDK Interface Test Classes

<table>
<thead>
<tr>
<th>Test Classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.interfaces.EPPBalanceTst</td>
<td>Test of sending a Balance Info Command.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPCoaDomainTst</td>
<td>Test of the Client Object Attribute (COA) extension. Tests include a</td>
</tr>
<tr>
<td></td>
<td>domain create with COA, and adding, changing and removing COAs using domain</td>
</tr>
<tr>
<td></td>
<td>updates. Also tests using the domain info command to retrieve a COA from</td>
</tr>
<tr>
<td></td>
<td>an existing owned domain.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPContactRelaxedValidationTst</td>
<td>Test of using the EPPContact interface to test contact create command and</td>
</tr>
<tr>
<td></td>
<td>contact info response with relaxed contact validation. Relaxed contact</td>
</tr>
<tr>
<td></td>
<td>validation loads a relaxed contact XSD (contact-1.0-relaxed.xsd) that makes</td>
</tr>
<tr>
<td></td>
<td>many of the RFC 5733 required elements optional. This test does not do</td>
</tr>
<tr>
<td></td>
<td>anything unless the EPP.Contact.RelaxedValidation epp.config property is</td>
</tr>
<tr>
<td></td>
<td>set to true.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPContactTst</td>
<td>Test of using the EPPContact interface for all of the RFC 5733 contact</td>
</tr>
<tr>
<td></td>
<td>commands. This test creates an individual EPP session</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPDomainTst</td>
<td>Test of using the EPPDomain interface for all of the RFC 5731 domain commands. This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPHostTst</td>
<td>Test of using the EPPHost interface for all of the RFC 5732 host commands. This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPIdnDomainTst</td>
<td>Tests send a domain create command with the IDN language extension (EPPIdnLangTag). This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPJobsContactTst</td>
<td>Test of using the RFC 5733 contact commands along with the Jobs Contact Extension. This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPLaunchTst</td>
<td>Test of sending launch phase commands (check, create, info, update, and delete).</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPLowBalanceDomainTst</td>
<td>Test of processing the Low Balance Poll Message by sending a domain create of “test.com” against the Stub Server that will then insert the Low Balance Poll Message. This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPNamestoreExtDomainTst</td>
<td>Test of using the RFC 5731 domain commands along with the NameStore Extension (EPPNamestoreExtNamestoreExt). This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPNamestoreExtHostTst</td>
<td>Test of using the RFC 5732 host commands along with the</td>
</tr>
<tr>
<td>Class</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NameStore Extension (EPPNamestoreExtNamestoreExt)</td>
<td>This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPPremiumDomainTst</td>
<td>Test of using the Premium Domain Extension to domain check command, domain check response and domain update command. This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPRegistryTst</td>
<td>Test of sending Registry Mapping commands.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPRelatedDomainTst</td>
<td>Test of sending Related Domain Extension command (domain info with extension).</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPRgpDomainTst</td>
<td>Test of using the Domain Registry Grace Period (RGP) extension defined in RFC 3915 to restore a domain and to retrieve RGP statuses from the Stub Server. This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPSecDNSDomainTst</td>
<td>Test of using the DNSSEC Extension RFC 4310 (secDNS-1.0) and RFC 5910 (secDNS-1.1) to create, info, and update a domain with DS data. This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPSessionTst</td>
<td>Test of doing the RFC 5730 commands (login, hello, poll, and logout).</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPSuggestionTst</td>
<td>Test of sending a set of randomized Name Suggestion commands. This test creates an individual EPP session without the SDK session pool.</td>
</tr>
</tbody>
</table>
| com.verisign.epp.interfaces.EPPSyncDomainTst                        | Test of using ConsolDate extension (EPPSyncExtUpdate) to
<table>
<thead>
<tr>
<th>Class Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.interfaces.EPPWhoisDomainTst</td>
<td>Test using the EPP Whois Info Extension with the RFC 5731 domain info command to retrieve the additional information sent by the Stub Server. This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPWhoWasTst</td>
<td>Test of sending a set of WhoWas commands. This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.namestore.interfaces.NSContactTst</td>
<td>Test of using NSContact for all of the contact commands. This test also utilizes the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.namestore.interfaces.NSDomainTst</td>
<td>Test of using NSDomain for all of the domain commands. This test also utilizes the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.namestore.interfaces.NSHostTst</td>
<td>Test of using NSHost for all of the host commands. This test also utilizes the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.namestore.interfaces.NSPollTst</td>
<td>Test of processing each of the poll messages produced by NameStore and the SRS by sending a domain create command for “NSPollTst.com” to the Stub Server, which will then insert all possible poll message types for consumption by NSPollTst. This test also utilizes the SDK session pool.</td>
</tr>
<tr>
<td>com.verisign.epp.pool.EPPSessionPoolTst</td>
<td>Test of using the EPPSessionPool for an individual session pool. It utilizes the session pool to send a hello and poll command. It also tests the absolute and idle timeout features of the pool.</td>
</tr>
<tr>
<td>com.verisign.epp.pool.EPPSystemSessionPoolTst</td>
<td>Test using two session pools</td>
</tr>
</tbody>
</table>
(default and “test”).
5. SDK Overview

The Verisign Bundle EPP SDK provide an interface for users to easily implement client applications that use EPP as the underlying protocol. The primary goal of the SDK is the same as EPP itself, which is to be extensible. New EPP Command Mappings can be created and added to the SDK. The SDK provides the following features:

- Extensible EPP Client Interface
- EPP Session Management
- Pluggable Transport Package with TCP/IP and SSL/TLS transport implementations
- Encapsulation of XML Encoding and Decoding
- Diagnostic logging using a powerful, open logging facility
- Extensible EPP Stub Server
- Use of an XML Parser Pool with XML schema caching
- Session Pooling with support for a separate SSL configuration per Session Pool.
- Support for pipelining when using `com.verisign.epp.interfaces.EPPSession`. Pipelining is sending multiple commands and processing the responses asynchronously.

The EPP Session Management includes the handling of the EPP Greeting, the EPP Login, the EPP Logout, the EPP Hello, and the EPP Poll commands. The default behavior is to derive the EPP Login services from the classes defined in the `EPP.MapFactories` and the optional `EPP.ProtocolExtensions` and `EPP.CmdRspExtensions` configuration parameters. These configuration parameters can be overridden by calling `EPPSession.setServices()` and `EPPSession.addExtensions()` in the `EPPSession` Interface. As described in Section 1.1 - Transport, the transport layer can be easily replaced. The XML encoding and decoding is completely encapsulated in the SDK, although the XML messages can be logged as described in Section 6.19 - Diagnostic and Error Logging.
1.1 SDK Directories

Once unpacked, the Verisign Bundle EPP SDK will have the following directories:

Table 4 - Verisign Bundle EPP SDK Directories

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>epp-verisign-${BUILD_VER}/bundles/verisign</td>
<td>Main directory for the Verisign Bundle EPP SDK. A bundle is a packaging of multiple EPP SDK mappings and extensions. This directory contains the configuration files and a build script that includes the Ant targets defined in Table 1 – Verisign Bundle build.xml Targets.</td>
</tr>
<tr>
<td>epp-verisign-${BUILD_VER}/bundles/verisign/doc</td>
<td>This directory contains bundled documentation for the Verisign Bundle EPP SDK. There are EPP Mapping documents in PDF format (.pdf) or ASCII format (.txt). These documents describe the XML schema definitions for the bundled products.</td>
</tr>
<tr>
<td>epp-verisign-${BUILD_VER}/bundles/verisign/doc/html</td>
<td>This directory contains the bundled interface specification in Javadoc format.</td>
</tr>
<tr>
<td>epp-verisign-${BUILD_VER}/lib</td>
<td>This directory contains dependent JAR files including JUnit, Log4j, PoolMan, and XercesJ.</td>
</tr>
<tr>
<td>epp-verisign-${BUILD_VER}/lib/epp</td>
<td>This directory contains the bundled JAR file for the Verisign Bundle EPP SDK (epp-verisign-bundle-${BUILD_VER}.jar).</td>
</tr>
</tbody>
</table>

1.2 SDK Packages

The SDK consists of sub-packages under com.verisign.epp. Some packages are extended by sub-packages (i.e. com.verisign.epp.codec), and some packages are extended with new classes in the existing packages (i.e. com.verisign.epp.interfaces). Table 5 - Verisign Bundle EPP SDK High-Level Packages provides an overview of the high-level SDK packages.

Table 5 - Verisign Bundle EPP SDK High-Level Packages

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec</td>
<td>EPP Encoder/Decoder package. There is one sub-package per implemented EPP specification (i.e. gen for the EPP General Specification and domain for the EPP Domain Command Mapping Specification).</td>
</tr>
<tr>
<td>com.verisign.epp.exception</td>
<td>General EPP SDK exception classes</td>
</tr>
<tr>
<td>Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Client interface classes including EPPApplication and EPPSession</td>
</tr>
<tr>
<td>com.verisign.epp.framework</td>
<td>EPP Server Framework classes used by the Stub Server</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Stub Server classes including handlers for each of the supported EPP Command Mappings.</td>
</tr>
<tr>
<td>com.verisign.epp.util</td>
<td>Set of SDK utility classes including EPPEnv.</td>
</tr>
<tr>
<td>com.verisign.epp.pool</td>
<td>Session Pool classes</td>
</tr>
</tbody>
</table>
6. SDK Configuration

1.1 Configuration File

The Verisign Bundle EPP SDK configuration file is a Java properties file that is passed to
EPPApplication.initialize(String) to initialize the SDK. It contains configuration
parameters that initialize the logging facility, that specify the EPP Command Mapping
classes, that initialize the XML Parser Pools, and that initialize the transport layer. Each
of the parameters has accessor methods in EPPEnv. The property file is searched in the
following ways:

1. On the file system (i.e. new File(String))
2. In the system ClassLoader (i.e.
   ClassLoader.getClassLoader().getResourceAsStream(String))
3. In the ClassLoader of the Environment class (i.e.
   Environment.class.getClassLoader().getResourceAsStream(String)).

The parameters that include “Client” and “Server” indicate two separate parameters for
the Client and the Server, respectively. For example, the parameter
“PoolMan[Client/Server]” indicates that there are two parameters PoolManClient and
PoolManServer, where the Client uses PoolManClient and the Server uses
PoolManServer. The Process column with a value “Client/Server” indicates that the
Client is the process for the Client form of the parameter (i.e. PoolManClient) and that
the Server is the process for the Server form of the parameter (i.e. PoolManServer). The
required column and the description apply to both forms of the parameter. You may
optionally setup separate epp.config files for client and server, when running the stub
server. The parameters containing “Server” are required in the server’s epp.config;
parameters containing “Client” are required in the client’s epp.config.

Table 6 - SDK Configuration File Parameters shows the configuration parameters in
the configuration file. The table includes each configuration parameter, a description, the
process that uses the parameter (Client, Server, or Both), and whether the parameter is
required. The JSSE parameters are only required if EPP.ClientSocketName or
EPP.ServerSocketName use a JSSE class. Bolded parameters are new to this release of
the SDK.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Process</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPP.LogMode</td>
<td>Both</td>
<td>Yes</td>
<td>Log Configuration Mode. The mode controls the way by which the logging facility (Log4J) is initialized. There are three different modes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• BASIC - Initialize logging using EPP.LogLevel and EPP.LogFile</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CFGFILE - Initialize logging using EPP.LogCfgFile and optionally EPP.LogCfgFileWatch</td>
</tr>
</tbody>
</table>
CUSTOM - SDK will not initialize the logging facility and it is left up to the client application.

The Stub Server does not consult EPP.LogMode, and will initialize its logging facility based on the following:

If EPP.LogCfgFile is defined
Use EPP.LogCfgFile and Use EPP.LogCfgFileWatch Else if EPP.LogFile and EPP.LogLevel is defined
Use EPP.LogFile and use EPP.LogLevel Else
Print error and stop program.

<table>
<thead>
<tr>
<th>EPP.LogLevel</th>
<th>Both</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log4J Log Level. The root category will be set to the specified priority. The possible values in order of severity from lowest to highest include: DEBUG – Recommended for debugging only INFO – Recommended for production mode WARN ERROR FATAL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EPP.LogFile</th>
<th>Both</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log4J Log File Name. Logs will be appended to the log file. The default file is “epp.log”.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EPP.LogCfgFile</th>
<th>Both</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log4J XML Configuration File Name used to define the log levels and the appenders (file, syslog, etc.). The default file is “logconfig.xml”.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EPP.LogCfgFileWatch</th>
<th>Both</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interval in milliseconds to monitor for changes to EPP.LogCfgFile. If EPP.LogCfgFile is updated, the log settings will be re-loaded. The default setting is 5000 milliseconds (5 seconds).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EPP.ConTimeOut</th>
<th>Both</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection and read timeout in milliseconds. A setting of 0 indicates no timeout. The default setting is 500000 milliseconds (500 seconds).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EPP.ClientSocketName</th>
<th>Client</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concrete client socket class. The class must implement the EPPClientCon interface and is only required for TCP or SSL. The concrete EPPClientCon class is instantiated when an EPPSession is instantiated and is closed when EPPSession.endSession() is called. The classes provided in the SDK include:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>com.verisign.epp.transport.client.EPPPlainClientSocket - Plain TCP/IP socket connection(s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>com.verisign.epp.transport.client.EPPSSLClientSocket - SSL TCP/IP socket connection(s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>com.verisign.epp.transport.client.EPPPlainProxyClientSocket - Plain TCP/IP socket connection(s) that connects through an Apache Proxy Server (mod_proxy). The</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EPP.ProxyServersLocator property must be set and the EPP.ProxyServers and EPP.ProxyServersRandomize should be set.

com.verisign.epp.transport.client.EPPSSLProxyClientSocket - SSL TCP/IP socket connection(s) that connects through an Apache Proxy Server. The EPP.ProxyServersLocator property must be set and the EPP.ProxyServers and EPP.ProxyServersRandomize should be set.

<table>
<thead>
<tr>
<th>Property</th>
<th>Client</th>
<th>Server</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPP.ProxyServersLocator</td>
<td>Client</td>
<td>No</td>
<td>Defines the concrete class of the com.verisign.epp.transport.client.EPPProxyServersLocator interface that returns the list of Apache Proxy Servers to connect through. This property is required if the EPP.ClientSocketName property is set to either com.verisign.epp.transport.client.EPPPlainProxyClientSocket or com.verisign.epp.transport.client.EPPSSLProxyClientSocket. The default value is com.verisign.epp.transport.client.EPPConfigProxyServersLocator to load the proxy servers from the EPP.ProxyServers property.</td>
</tr>
<tr>
<td>EPP.ProxyServers</td>
<td>Client</td>
<td>No</td>
<td>Defines the list of Apache Proxy Servers to connect through when the EPP.ProxyServersLocator property is set to com.verisign.epp.transport.client.EPPConfigProxyServersLocator. The format required for the property value is: (&lt;proxy server&gt;:&lt;port number&gt;)(,&lt;proxy server&gt;:&lt;port number&gt;)* Where &lt;proxy server&gt; ::= [&quot;?&lt;ip address&gt;</td>
</tr>
<tr>
<td>EPP.ProxyServersRandomize</td>
<td>Client</td>
<td>No</td>
<td>Defines whether or not the Apache Servers defined by the EPP.ProxyServers property or what the EPP.ProxyServersLocator class returns randomized per connection or attempted in order.</td>
</tr>
<tr>
<td>EPP.ClientHost</td>
<td>Client</td>
<td>No</td>
<td>Host name or IP Address that the client will connect from. If not defined the client host will default to the loopback address.</td>
</tr>
<tr>
<td>EPP.ServerName</td>
<td>Both</td>
<td>Yes</td>
<td>Host name or IP Address that the server will listen on and that the client will connect to. The default setting is &quot;localhost&quot;.</td>
</tr>
<tr>
<td>EPP.ServerPort</td>
<td>Both</td>
<td>Yes</td>
<td>Port that the server will listen on and that the client will connect to.</td>
</tr>
<tr>
<td>EPP.ServerSocketName</td>
<td>Server</td>
<td>Yes</td>
<td>Concrete server socket class used by the Stub Server. The classes provided in the SDK include: com.verisign.epp.transport.server.EPPPlainServer - Plain TCP/IP socket connection(s)</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Mandatory</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EPP.MapFactories</td>
<td>Client</td>
<td>Yes</td>
<td>Space separated list of fully qualified EPP Command Mapping factory class names. There is one EPP Mapping Factory per EPP Command Mapping. See the Programmer Guide for the desired EPP Command Mapping for more details.</td>
</tr>
<tr>
<td>EPP.ServerAssembler</td>
<td>Server</td>
<td>No</td>
<td>Fully qualified class name of the class the Stub Server will use to assemble EPP packets. This class must implement the com.verisign.epp.framework.EPPAssembler interface. If nothing is specified then com.verisign.epp.framework.EPPXMLAssembler is used.</td>
</tr>
<tr>
<td>EPP.CmdRspExtensions</td>
<td>Client</td>
<td>No</td>
<td>Space separated list of fully qualified EPP Command Extension factory class names. There is one EPP Command Response Extension Factory mapping per EPP Command Response Extensions. See the Programmer Guide for the desired EPP Command Mapping for more details.</td>
</tr>
<tr>
<td>EPP.ServerEventHandler</td>
<td>Server</td>
<td>Yes</td>
<td>Space separated list of fully qualified EPP Event Handler class names loaded in the Stub Server. There is one EPP Event Handler per EPP Command Mapping. There is one handler required for EPP general handling, which is com.verisign.epp.serverstub.GenHandler. See the Programmer Guide for the desired EPP Command Mapping for more details.</td>
</tr>
<tr>
<td>EPP.PollHandlers</td>
<td>Server</td>
<td>No</td>
<td>Space separated list of fully qualified EPP Poll Handler class names loaded in the Stub Server. Each EPP Command Mapping that supports EPP Poll will include an EPP Poll Handler. See the Programmer Guide for the desired EPP Command Mapping for more details.</td>
</tr>
</tbody>
</table>
| EPP.SSLProtocol       | Both  | Yes       | JSSE Protocol used. The possible values include:  
- TLS - Supports some version of TLS  
- SSL - Supports some version of SSL  
- SSLv2 - Supports SSL version 2 or higher  
- SSLv3 - Supports SSL version 3  
- TLSv1 - Supports TLS version 1.0  
- TLSv1.1 – Supports TLS version 1.1 (Java 7 or higher)  
- TLSv1.2 – Supports TLS version 1.2 (Java 7 or higher) |
<table>
<thead>
<tr>
<th>Property</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPP.SSLKeyStore</td>
<td>Both</td>
<td>JSSE KeyStore format. The default setting is “JKS”.</td>
</tr>
<tr>
<td>EPP.SSLKeyFileName</td>
<td>Both</td>
<td>JSSE KeyStore file used for authentication. The SDK includes a self-signed certificate in the KeyStore “testkeys”.</td>
</tr>
<tr>
<td>EPP.SSLPassPhrase</td>
<td>Both</td>
<td>JSSE KeyStore pass-phrase. The SDK provided KeyStore has a pass-phrase of “passphrase”.</td>
</tr>
<tr>
<td>EPP.SSLKeyPassPhrase</td>
<td>Both</td>
<td>JSSE private key pass-phrase. If not set, EPP.SSLPassPhrase is used. The SDK does not use a different pass-phrase for the private key.</td>
</tr>
<tr>
<td>EPP.SSLEnabledProtocols</td>
<td>Both</td>
<td>Enabled Protocols. If not defined, the default for the provider will be used. If defined, the list of enabled protocols should be provided using spaces as delimiters. Examples include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SSL - Supports some version of SSL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SSLv2 - Supports SSL version 2 or higher</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SSLv3 - Supports SSL version 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TLS - Supports some version of TLS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TLSv1 - Supports TLS version 1</td>
</tr>
<tr>
<td>EPP.SSLEnabledCipherSuites</td>
<td>Both</td>
<td>Enabled Cipher Suites. Space delimited list of cipher suites. Examples include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SSL_RSA_WITH_RC4_128_MD5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SSL_RSA_WITH_RC4_128_SHA</td>
</tr>
<tr>
<td>EPP.SSLTrustStoreFileName</td>
<td>Both</td>
<td>Set this to the keystore file that contains the list of Certificate Authorities that should be trusted. If not set then it defaults to the keystore that comes with the JDK which is: $JAVA_HOME/jre/lib/security/cacerts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>It is recommended to comment out or not define this property when connecting to the NameStore or SRS Servers.</td>
</tr>
<tr>
<td>EPP.SSLTrustStorePassPhrase</td>
<td>Both</td>
<td>JSSE TrustStore pass-phrase. This property is required if EPP.SSLTrustStoreFileName is defined.</td>
</tr>
<tr>
<td>javax.net.debug</td>
<td>Both</td>
<td>JSSE debug options. This is very useful for debugging SSL handshaking issues. The possible values include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- none – No debug</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- all – All debug</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This property will set the javax.net.debug System property.</td>
</tr>
<tr>
<td>PoolMan.[Client/Server/XMLSignature].logFile</td>
<td>Client/Server</td>
<td>Log file to write debug messages, if PoolMan.[Client/Server/XMLSignature].debugging is true. Default value is PoolMan.[Client/Server/XMLSignature].log.</td>
</tr>
<tr>
<td>PoolMan.[Client/Server/</td>
<td>Client/Server</td>
<td>Initial number of objects to create in the pool. Default value</td>
</tr>
<tr>
<td>XMLSignature].logFile</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VeriSign Inc. Proprietary Information
Verisign Bundle EPP SDK Programmer's Guide ● page 30
<table>
<thead>
<tr>
<th>XMLSignature</th>
<th>initialObject</th>
<th>er</th>
<th>is 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PoolMan.[Client/Server/ XMLSignature].minSize</td>
<td>Client/Server</td>
<td>No</td>
<td>Minimum number of objects that can be in the pool. Default value is 0.</td>
</tr>
<tr>
<td>PoolMan.[Client/Server/ XMLSignature].maxSize</td>
<td>Client/Server</td>
<td>No</td>
<td>Maximum number of objects that can be in the pool. Default value is Integer.MAX_VALUE</td>
</tr>
<tr>
<td>PoolMan.[Client/Server/ XMLSignature].maxSoft</td>
<td>Client/Server</td>
<td>No</td>
<td>If the maximum size of a pool is reached but requests are still waiting on objects, PoolMan will create new emergency objects if this value is set to true. This will temporarily increase the size of the pool, but the pool will shrink back down to acceptable size automatically when the skimmer activates. If this value is set to false, the requests will sit and wait until an object is available. Default value is true.</td>
</tr>
<tr>
<td>PoolMan.[Client/Server/ XMLSignature].skimmer Frequency</td>
<td>Client/Server</td>
<td>No</td>
<td>The length of time the pool skimmer waits between reap cycles. Each reap cycle involves evaluating all objects (both checked in and checked out) to determine whether to automatically return them to the pool and whether to destroy them if they have timed out. Default is 420 seconds (7 minutes)</td>
</tr>
<tr>
<td>PoolMan.[Client/Server/ XMLSignature].shrinkBy</td>
<td>Client/Server</td>
<td>No</td>
<td>Each time the pool is sized down by the skimmer, this value determines the maximum number of objects that can be removed from it in any one reap cycle. It prevents backing off the pool too quickly at peak times. Default is 5.</td>
</tr>
<tr>
<td>PoolMan.[Client/Server/ XMLSignature].debugging</td>
<td>Client/Server</td>
<td>No</td>
<td>Write debug messages? Debug messages are written to the file specified by the PoolMan.[Client/Server/XMLSignature].logFile parameter. Default is false.</td>
</tr>
<tr>
<td>PoolMan.[Client/Server/ XMLSignature].objectTimeout</td>
<td>Client/Server</td>
<td>No</td>
<td>The length of time, in seconds, that each object has to live before being destroyed and removed from the pool. Default value is 1200 seconds (20 minutes)</td>
</tr>
<tr>
<td>PoolMan.[Client/Server/ XMLSignature].userTimeout</td>
<td>Client/Server</td>
<td>No</td>
<td>The length of time in seconds that user has to keep an object before it is automatically returned to the pool. Default value is 1200 seconds (20 minutes)</td>
</tr>
<tr>
<td>EPP.Validating</td>
<td>Client/Server</td>
<td>No</td>
<td>Turns on/off XML schema validation. The default is false for clients for improved performance, but can be turned to true if response validation is important. Set to true to test against the Stub Server with XML schema validation. Default is true.</td>
</tr>
<tr>
<td>EPP.FullSchemaChecking</td>
<td>Client/Server</td>
<td>No</td>
<td>Turns on/off strict XML schema validation. Set to true to test against the Stub Server with full XML schema validation. EPP.Validating must be set to true for the EPP.FullSchemaChecking setting to have any impact. Default is true.</td>
</tr>
<tr>
<td>Property</td>
<td>Client/Server</td>
<td>Required</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>EPP.MaxPacketSize</strong></td>
<td>Client/Serv</td>
<td>No</td>
<td>Maximum packet size of bytes accepted to ensure that the client is not overrun with an invalid packet or a packet that exceeds the maximum size. The default is 10000 if property is not defined.</td>
</tr>
<tr>
<td><strong>EPP.SessionPool.poolableClassName</strong></td>
<td>Client</td>
<td>No</td>
<td>Fully qualified class name of object pooled by the EPPSessionPool. Set to com.verisign.epp.pool.EPPSessionPoolableFactory to TCP session pooling. Default is com.verisign.epp.pool.EPPSessionPoolableFactory</td>
</tr>
<tr>
<td><strong>EPP.SessionPool.clientTransIdGenerator</strong></td>
<td>Client</td>
<td>No</td>
<td>Fully qualified class name of client transaction identifier generator that implements the com.verisign.epp.pool.EPPClientTransIdGenerator interface. Default is null (not set)</td>
</tr>
<tr>
<td><strong>EPP.SessionPool.clientId</strong></td>
<td>Client</td>
<td>No</td>
<td>Client id/name used to authenticate session in the session pool. Required if using the EPPSessionPool.</td>
</tr>
<tr>
<td><strong>EPP.SessionPool.password</strong></td>
<td>Client</td>
<td>No</td>
<td>Password used to authenticate session in the session pool. Required if using the EPPSessionPool.</td>
</tr>
<tr>
<td><strong>EPP.SessionPool.absoluteTimeout</strong></td>
<td>Client</td>
<td>No</td>
<td>Absolute timeout of session in milliseconds of a session in the session pool. Sessions past the absolute timeout will be refreshed in the pool. Default is 82800000 (23 hours)</td>
</tr>
<tr>
<td><strong>EPP.SessionPool.idleTimeout</strong></td>
<td>Client</td>
<td>No</td>
<td>Idle timeout of session in milliseconds of a session in the session pool. Sessions past the idle timeout will send an EPP hello command to keep the session alive. Required if using the EPPSessionPool. Default is 480000 (8 minutes)</td>
</tr>
<tr>
<td><strong>EPP.SessionPool.minIdle</strong></td>
<td>Client</td>
<td>No</td>
<td>Minimum number of idle sessions in the session pool. Default is 5</td>
</tr>
<tr>
<td><strong>EPP.SessionPool.maxIdle</strong></td>
<td>Client</td>
<td>No</td>
<td>Maximum number of idle sessions in the session pool. Default is 10</td>
</tr>
<tr>
<td><strong>EPP.SessionPool.maxActive</strong></td>
<td>Client</td>
<td>No</td>
<td>Maximum number of active sessions borrowed from the session pool. Default is 10</td>
</tr>
<tr>
<td><strong>EPP.SessionPool.initMaxActive</strong></td>
<td>Client</td>
<td>No</td>
<td>Boolean value that will pre-initialize maxActive sessions in the pool. Default is false</td>
</tr>
<tr>
<td><strong>EPP.SessionPool.borrowRetries</strong></td>
<td>Client</td>
<td>No</td>
<td>Number of retries when there is a failure in borrowing a new session from the pool. This eliminates the client having to implement its own retry loop on a call to EPPSessionPool.borrowObject() and also applies to pre-initializing the sessions when the pool is initialized.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Required</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EPP.SessionPool.initMaxActive</td>
<td></td>
<td></td>
<td>Default is 0. The maximum number of milliseconds a client will block waiting for a session from the session pool. Default is 60000 (1 minute).</td>
</tr>
<tr>
<td>EPP.SessionPool.timeBetweenEvictionRunsMillis</td>
<td></td>
<td></td>
<td>Frequency in milliseconds to scan idle sessions in the session pool for timeouts. Default is 500 (1/2 second).</td>
</tr>
<tr>
<td>EPP.SessionPool.systemPools</td>
<td></td>
<td></td>
<td>Default the set of system session pools in a comma separated list of names (i.e. srs,namestore). The system name “default” initializes a default pool that uses the EPP.SessionPool.&lt;param&gt; property along with other properties like EPP.ServerName and EPP.ServerPort. The default pool uses the EPPSessionPool methods that don’t take a &lt;system&gt; parameter.</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.poolableClassName</td>
<td></td>
<td></td>
<td>System specific setting of EPP.SessionPool.poolableClassName, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore).</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.clientTransIdGenerator</td>
<td></td>
<td></td>
<td>System specific setting of EPP.SessionPool.clientTransIdGenerator, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore).</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.serverName</td>
<td></td>
<td></td>
<td>System specific setting of EPP.ServerName, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore).</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.serverPort</td>
<td></td>
<td></td>
<td>System specific setting of EPP.ServerPort, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore).</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.clientHost</td>
<td></td>
<td></td>
<td>TCP client host name. If not defined, the loopback will be used.</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.clientId</td>
<td></td>
<td></td>
<td>System specific setting of EPP.SessionPool.clientId, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore).</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.password</td>
<td></td>
<td></td>
<td>System specific setting of EPP.SessionPool.password, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore).</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.absoluteTimeout</td>
<td></td>
<td></td>
<td>System specific setting of EPP.SessionPool.absoluteTimeout, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore).</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.idleTimeout</td>
<td></td>
<td></td>
<td>System specific setting of EPP.SessionPool.idleTimeout, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore).</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.minIdle</td>
<td></td>
<td></td>
<td>System specific setting of EPP.SessionPool.minIdle, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore).</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.maxIdle</td>
<td></td>
<td></td>
<td>System specific setting of EPP.SessionPool.maxIdle, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore).</td>
</tr>
<tr>
<td>Variable</td>
<td>Setting</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------</td>
<td>------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>.maxIdle</td>
<td>&lt;system&gt;</td>
<td>is replaced with the system name (i.e. srs, namestore)</td>
<td></td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.maxActive</td>
<td>Client</td>
<td>System specific setting of EPP.SessionPool.maxActive, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</td>
<td></td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.initMaxActive</td>
<td>Client</td>
<td>System specific setting of EPP.SessionPool.initMaxActive, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</td>
<td></td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.borrowRetries</td>
<td>Client</td>
<td>System specific setting of EPP.SessionPool.borrowRetries, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</td>
<td></td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.maxWait</td>
<td>Client</td>
<td>System specific setting of EPP.SessionPool.maxWait, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</td>
<td></td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.timeBetweenEvictionRunsMillis</td>
<td>Client</td>
<td>System specific setting of EPP.SessionPool.timeBetweenEvictionRunsMillis, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</td>
<td></td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLProtocol</td>
<td>Client</td>
<td>Protocol to use for pool, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore). If defined the pool will have its own SSL configuration. The required pool SSL properties include SSLKeyStore, SSLKeyFileName, and SSLKeyPassPhrase. Default is TLSv1</td>
<td></td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLKeyStore</td>
<td>Client</td>
<td>Type of identity KeyStore, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore). Required if SSLProtocol is defined for pool. Default is JKS</td>
<td></td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLKeyFileName</td>
<td>Client</td>
<td>Name of the identity KeyStore file, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore). Required if SSLProtocol is defined for pool. Default is ../../lib/keystore/testkeys</td>
<td></td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLPassPhrase</td>
<td>Client</td>
<td>Passphrase/password to access the identity KeyStore file defined by SSLKeyFileName pool property, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore). Required if SSLProtocol is defined for pool. Default is passphrase</td>
<td></td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLEnabledProtocols</td>
<td>Client</td>
<td>Optional space delimited list of enabled SSL protocols, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</td>
<td></td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLEnabledCipherSuites</td>
<td>Client</td>
<td>Optional space delimited list of SSL cipher suites, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore). Examples include:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SSL_RSA_WITH_RC4_128_MD5</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Required</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLKeyPassPhrase</td>
<td>Client</td>
<td>No</td>
<td>Optional passphrase/password for the private key stored in the identity KeyStore, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore). If undefined the pool SSLPassPhrase will be used.</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLTrustStore</td>
<td>Client</td>
<td>No</td>
<td>Optional Type of the Trust Store, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore). If not defined and the pool SSLTrustStoreFileName is defined, the pool SSLKeyStore will be used for the Trust Store. Default is JKS</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLTrustStoreFileName</td>
<td>Client</td>
<td>No</td>
<td>Pool trust store file that contains the list of Certificate Authorities that should be trusted. If not set than the pool defaults to the keystore that comes with the JDK which is: $JAVA_HOME/jre/lib/security/cacerts. It is recommended to comment out or not define this property when connecting to the NameStore or SRS Servers.</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLTrustStorePassPhrase</td>
<td>Client</td>
<td>No</td>
<td>Pool passphrase/password to access the Trust Store file defined by the pool SSLTrustStoreFileName property.</td>
</tr>
</tbody>
</table>
| EPP.SessionPool.<system>.SSLDebug | Client | No | Defines the SSL debug Java system property javax.net.debug value. The possible values include:  
  - none – No debug  
  - all – All debug  
This property only needs to be defined once for all pools, since each pool property will result in resetting the javax.net.debug system property. |
| EPP.Test.clientId | Client | No | Optional setting for configuring the login clientId used by the tests. This will allow the tests to target servers other than the Stub Server like OT&E. Not all tests might have been updated to utilize this property. Default is “ClientX” |
| EPP.Test.password | Client | No | Optional setting for configuring the login password used by the tests. This will allow the tests to target servers other than the Stub Server like OT&E. Not all tests might have been updated to utilize this property. Default is “password” |
| EPP.Test.stubServer | Client | No | Optional Boolean setting to specify to the tests that the target server is the Stub Server. This allows the tests to be customized to run against the Stub Server or against a real server like OT&E. Default is true |
| EPP.Contact.RelaxedValidation | Client/Server | No | Optional Boolean setting that loads and validates to a relaxed form of the contact XSD (contact-1.0-relaxed.xsd) |

- SSL_RSA_WITH_RC4_128_SHA
<table>
<thead>
<tr>
<th>Property</th>
<th>Category</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPP.UseEntityResolver</td>
<td>Client/Server</td>
<td>No</td>
<td>Use an Entity Resolver, that supports the dynamic loading of XML schemas based on the client or server including the xsi:schemaLocation attribute for an XML namespace that has not already been loaded into the XML parser. XML schemas will be pre-loaded based on the registered factories, so the recommendation is to set this property to false. The default setting is true for backward compatibility. Default is true</td>
</tr>
<tr>
<td>EPP.SendReceiveLogger</td>
<td>Client/Server</td>
<td>No</td>
<td>Concrete com.verisign.epp.util.EPPSendReceiveLogger to use for logging sent and received packets. The default value is to use com.verisign.epp.util.EPPRawSendReceiveLogger if the EPP.SendReceiveLogger is not defined. An alternative logger is to use com.verisign.epp.namestore.util.EPPSecureSendReceiveLogger that will mask sensitive attributes like the login password and the auth info values with the word &quot;MASKED&quot;. Default is false</td>
</tr>
<tr>
<td>EPP.LoginAdapter</td>
<td>Client</td>
<td>No</td>
<td>Concrete optional com.verisign.epp.interfaces.EPPLoginAdapter that can customize (adapt) the makeup of the EPP Login Command sent to the server. The LoginAdapter is called prior to sending the login command to the server in com.verisign.epp.interfaces.EPPSession. Default is com.verisign.epp.interfaces.EPPLoginSecLoginAdapter to support automatically adding the Login Security Extension to the login command when the password or new password is longer than 16 characters or the EPP.UseUserAgent property is not set to false.</td>
</tr>
<tr>
<td>EPP.UseUserAgent</td>
<td>Client</td>
<td>No</td>
<td>Include the userAgent in the login security extension? The default is true, but it can be disabled by setting the EPP.UseUserAgent property to false. Default is true</td>
</tr>
<tr>
<td>EPP.SDKVersion</td>
<td>Client</td>
<td>No</td>
<td>Defines the version of the Verisign EPP SDK distribution for inclusion in the Login Security Extension user agent. Default is automatically set by the Verisign EPP SDK distribution.</td>
</tr>
</tbody>
</table>
Set one of the
com.verisign.epp.codec.gen.EPPPollMessageFilter classes
that is called from within
com.verisign.epp.serverstub.GenHandler.doPoll(EPPEvent,
Object) to filter the poll message responses based on the
client login services. The server must not return extensions
(object or command / response) that the client does not
support based on the login services that are passed in the
Login Command. The EPP.PollMessageFilter property
is optional and if set should be set only once.

Default is null (undefined)

Table 6 - SDK Configuration File Parameters

6.18 Libraries
The Verisign Bundle EPP SDK library, epp-verisign-bundle-{$BUILD_VER}.jar, is a
bundled distribution that includes all required classes from the core EPP SDK, packages
and classes for each Namestore EPP Command Mapping, and a set of dependent
libraries. The core EPP classes include all of the SDK base frameworks and an
implementation of the general EPP specification. This includes session management and
the building block classes for EPP. The dependent libraries included in the SDK are:

- Ant 1.9.11 – .jar files in epp-verisign-$\{BUILD_VER\}/lib/ant
- junit-4.12.jar – Library used for SDK tests
- log4j-1.2.8.jar – Library used for SDK diagnostic and error logging
- XercesJ 2.12.0 – Library used for XML parsing. This includes xercesImpl-
  2.12.0.jar and xml-apis-2.12.0.jar
- poolman-2.1-b1.jar – Library used for pooling the XML Parsers and Transformers
- Jalopy 1.0b10 – Library used for code formatting. .jar files in epp-verisign-
  $\{BUILD_VER\}/lib/jalopy
- Apache Commons Pool 1.1 – Library used for implementing the EPP session
  pool.
- Apache Commons Codec 1.6 – Library used for Base64 encoding and decoding
  within the launch extension.
- Apache Commons Collections 3.2.1 – Library used for the EPP session pool.
- dnsjava 2.1.5 – Library used in the secdns (DNSSEC) extension.
Each set of product specific EPP Command Mappings are included in the `epp-verisign-bundle-{$BUILD_VER}.jar` library. The library includes both client packages and classes and Stub Server classes, so that a single library contains all of the SDK classes required for all Namestore EPP Command Mappings.

6.19 Diagnostic and Error Logging
The Verisign Bundle EPP SDK uses Log4J([http://jakarta.apache.org/ant/index.html](http://jakarta.apache.org/ant/index.html)) for diagnostic and error logging. The `EPP.LogMode` configuration parameter has three possible modes:

1. **BASIC** - Sets the root log level with `EPP.LogLevel`, and sets the log file appender with `EPP.LogFile`
2. **CFGFILE** - Sets the Log4J configuration file with `EPP.LogCfgFile`, and optionally sets the Log4J configuration file monitor with `EPP.LogCfgFileWatch`.
3. **CUSTOM** – The SDK will not initialize Log4J and leaves the initialization up to the client application.

The EPP Stub Server included in the SDK does not use the `EPP.LogMode` configuration parameter, and first looks to the `EPP.LogCfgFile` parameter, and second looks to the `EPP.LogLevel` and the `EPP.LogFile` parameters. The EPP Stub Server uses `EPP.LogCfgFileWatch` only if `EPP.LogCfgFile` is defined.

6.19.1 Basic Configuration Mode (EPP.LogLevel and EPP.LogFile)
The basic configuration mode is meant to set the logging configuration without requiring the use of a Log4J XML configuration file. The `EPP.LogLevel` parameter will set the priority level of the root category. By default, it is set to DEBUG, so that all of the SDK messages will be logged. The `EPP.LogFile` parameter will set the log file name where the logs will be appended. The format of the messages is the Log4J format `"=%d{yyyyMMdd HHmmss}  %c  %-5p  %m\n",` which will log the date in the format “`yyyyMMdd HHmmss`”, the category as the fully qualified class name, the priority (DEBUG, INFO, WARN, ERROR, or FATAL), and the log message.

6.19.2 Log4J Configuration File Mode (EPP.LogCfgFile and EPP.LogCfgFileWatch)
The Log4J Configuration File Mode allows for the use of a Log4J configuration file to configure the SDK logging. The `EPP.LogCfgFile` parameter gives the name of the Log4J configuration file. By using the Log4J configuration file, logs can be routed to different destinations (i.e. syslog, date rolling files), priority levels can be set by category, and the configuration file changes can be applied without restarting the application if `EPP.LogCfgFileWatch` is defined. A daemon thread will check for configuration file changes every `EPP.LogCfgFileWatch` milliseconds.
Figure 1 - Default SDK Log4J Configuration File shows the configuration supplied in the SDK, which includes two appenders, DATEFILE and ERROR, and a setting for the root category. All logs will be sent to the DATEFILE appender, which will result in the log files epp.log[yyyyMMdd]. “epp.log” is the current day log file, and yyyyMMdd will be appended for previous days. All logs with the priority of WARN, ERROR, or FATAL will be sent to the ERROR appender, which will result in the log files epp.err[yyyyMMdd]. “epp.err” is the current day log file, and yyyyMMdd will be appended for previous days. The default format of the logs includes the date in the format “yyyyMMdd HHmmss”, the category as the fully qualified class name, the priority (DEBUG, INFO, WARN, ERROR, or FATAL), and the log message. The root category is set with a priority of “debug”, so that all SDK messages will be logged.

```xml
<log4j:configuration>

<!--
Direct diagnostic logging to a rolling log file prefixed with epp.log.
-->
<appender name="DATEFILE"
    class="org.apache.log4j.DailyRollingFileAppender">
    <param name="File" value="epp.log" />
    <param name="DatePattern" value="yyyyMMdd" />
    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%d{yyyyMMdd HHmmss}  %c -5p %m\n"/>
    </layout>
</appender>

<!--
Direct warning and errors to a rolling error log prefixed with epp.err.
-->
<appender name="ERROR"
    class="org.apache.log4j.DailyRollingFileAppender">
    <param name="File" value="epp.err" />
    <param name="DatePattern" value="yyyyMMdd" />
    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%d{yyyyMMdd HHmmss}  %c -5p %m\n"/>
    </layout>
    <filter class="org.apache.log4j.varia.PriorityRangeFilter">
        <param name="PriorityMin" value="WARN"/>
        <param name="PriorityMax" value="FATAL"/>
        <param name="AcceptOnMatch" value="true"/>
    </filter>
</appender>

<!--
Default level (info) and appender:
  o debug - Is the default level
  o All logs will go to the dated log file (DATEFILE)
  o All warnings and errors will go to the dated log file (ERROR)
</log4j:configuration>
```
-->
<root>
  <priority value="debug" />
  <appender-ref ref="DATEFILE" />
  <appender-ref ref="ERROR" />
</root>
</log4j:configuration>

Figure 1 - Default SDK Log4J Configuration File

6.19.3 Custom Mode
To integrate the SDK into a Log4J application, setting the EPP.LogMode parameter to CUSTOM can turn off the initialization of Log4J by the SDK. See Section 4.3.4 - SDK Log Categories for more information on the SDK categories that can be set in the custom Log4J configuration.

6.19.4 SDK Log Categories
The SDK Log Categories are based on the SDK fully qualified class names. All of the SDK classes are contained in the package com.verisign.epp. In general, any SDK errors are logged at the ERROR priority level. Table 7 - SDK Log Categories lists the primary categories defined in the SDK.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.util.EPPXMLStream</td>
<td>Class that handles the reading and writing of the EPP XML messages. When the “debug” priority level is enabled, the packets that are read and written will be logged.</td>
</tr>
<tr>
<td>com.verisign.epp.util.EPPXmlEntityResolver</td>
<td>Class that handles the loading and caching of XML schemas. The loading is done from the CLASSPATH, where the schemas need to reside in the directory “schemas”.</td>
</tr>
<tr>
<td>com.verisign.epp.transport</td>
<td>Package that contains the SDK transport classes. When the “debug” priority level is enabled, the trace of the primary classes will be logged.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Package that contains the client interface classes. Currently, these classes don’t log messages, but it is likely that logs will be added in future releases.</td>
</tr>
<tr>
<td>com.verisign.epp.codec</td>
<td>Package that contains the EPP encoding/decoding classes. Currently, these classes don’t log messages, but it is likely that logs will be added in future releases.</td>
</tr>
</tbody>
</table>
### Table 7 - SDK Log Categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.pool</td>
<td>Package that contains the EPP session pool classes. Logs include the number of active sessions, idle sessions, and identifies the session being borrowed or returned from/to the session pool.</td>
</tr>
</tbody>
</table>

### 6.20 Adding an EPP Command Mapping to the SDK

The SDK can be easily extended to support new EPP Command Mappings. As described in section 6.18, an EPP Command Mapping is associated with a single library. This SDK distribution bundles EPP Command Mappings, so manually adding mappings might not be required. Do the following to add an EPP Command Mapping to the SDK:

1. Add the EPP Command Mapping library (i.e. `epp-domain.jar`) to the application and Stub Server CLASSPATH. When using the standard SDK build scripts, the .jar files can be added to `epp-verisign-${BUILD_VER}/lib/epp` for automatic inclusion in the CLASSPATH.

2. Add EPP Command Mapping factory to the `EPP.MapFactories` configuration parameter. See the EPP Command Mapping Programmer Guide for the name of the class.

3. Add EPP Command Mapping factory to the `EPP.ProtocolExtensions` configuration parameter. See the EPP Command Mapping Programmer Guide for the name of the class.

4. Add EPP Command Mapping factory to the `EPP.CmdRspExtensions` configuration parameter. See the EPP Command Mapping Programmer Guide for the name of the class.

5. Add EPP Command Mapping handler to the `EPP.ServerEventHandler` configuration parameter. See the EPP Command Mapping Programmer Guide for the name of the class.

6. Add EPP Command Mapping poll handler to the `EPP.PollHandlers` configuration parameter. See the EPP Command Mapping Programmer Guide for the name of the class. Only EPP Command Mappings that support the EPP Poll command will include a poll handler.

By default, the services contained in the EPP `<login>` and the EPP `<greeting>` are controlled by the classes listed in `EPP.MapFactories`, `EPP.ProtocolExtensions`, `EPP.CmdRspExtensions` and `EPP.ServerEventHandler`. The client EPP `<login>` services by default will be determined by finding the intersection of the services included in the EPP `<greeting>` and the services defined by factories referenced by the `EPP.MapFactories`, `EPP.ProtocolExtensions`, `EPP.CmdRspExtensions` properties. The
client can override the services with a call to `EPPSession.setServices(String [])` or the client can override the extensions with a call to `EPPSession.addExtensions(Vector ext, Vector ext)` before calling `EPPSession.initSession()`.
7. Generic EPP Client Interfaces
The generic EPP client interface classes are contained in the `com.verisign.epp.interfaces` package and are meant to be the primary classes that a client application will use.

1.1 EPPApplication

**Figure 2 - EPPApplication Class Diagram** shows the `EPPApplication` classes used to initialize the SDK subsystems. `EPPApplicationSingle` is a Singleton version of `EPPApplication`. The subsystems initialized by `EPPApplication`, include:

- The Environment Settings based on the contents of the configuration file. `EPPEnv` provides an interface to the configuration information.
- The Logging Facility based on the `EPP.Log` configuration parameters. If `EPP.LogMode` is set to `CUSTOM`, than the Logging Facility will not be initialized by `EPPApplication`.
- The EPP Encoder/Decoder (CODEC) based on the `EPP.MapFactories` configuration parameter

`EPPApplication.initialize()` must be the first SDK method called, and it reads the configuration file passed in as an argument.

```java
try {
    EPPApplicationSingle.getInstance().initialize("epp.config");
} catch (EPPCommandException e) {
    System.err.println("Error initializing the EPP Application: " + e);
}
```

**Figure 3 - EPPApplication Initialization Sample Code** shows the code required to initialize `EPPApplication` with the epp.config configuration file. After `EPPApplication` is properly initialized, sessions can be created with the EPP Server as described in section 0.
// Create one or more EPP Server sessions.
EPPSession session = new EPPSession();

Figure 3 - EPPApplication Initialization Sample Code

7.21 EPPSession

7.21.1 Overview

Figure 4 - EPPSession Class Diagram shows the class that is responsible for managing a session with an EPP Server. An EPPSession represents an authenticated connection with the EPP Server, and is passed in the constructor of the EPP Command Mapping Interface classes. For example, EPPDomain is created with an instance of EPPSession. Each EPPSession is associated with one EPPClientCon, which represents one EPP Server connection.
The following is a description of the sequence in creation, usage, and closing of an `EPPSession`:

1. The client will create an instance of an `EPPSession`.
2. `EPPSession` will create a connection to the EPPServer, using the concrete `EPPClientCon` defined by the `EPP.ClientSocketName` configuration parameter.

---

**Figure 4 - EPPSession Class Diagram**

The following is a description of the sequence in creation, usage, and closing of an `EPPSession`:

1. The client will create an instance of an `EPPSession`.
2. `EPPSession` will create a connection to the EPPServer, using the concrete `EPPClientCon` defined by the `EPP.ClientSocketName` configuration parameter.
3. The client will set the authentication information (Client ID, Password) and authenticate with the EPP Server by calling `EPPSession.initSession()`.
   The `EPP.initSession()` will automatically intersect the configured services using the `EPP.MapFactories`, `EPP.ProtocolExtensions`, `EPP.CmdRspExtensions` configuration properties with the services included in the EPP `<greeting>` with setting the EPP `<login>` services. Optionally, the client can call `EPPSession.setServices()` with the list of client services, and also the client can optionally call `EPPSession.addExtensions()` with a vector of `EPPProtocolExtensions` or vector of `CommandResponseExtensions` or a combination of both, before calling `EPPSession.initSession()`.

4. The `EPPSession` will read the EPP `<greeting>` from the EPP Server, will create an EPP `<login>`, and will send the EPP `<login>` to the EPP Server.

5. The client will creates a Mapping Interface object (i.e. `EPPDomain`) with the `EPPSession` instance.

6. The client will execute zero or more commands through the Mapping Interface object.

7. The client will end the session by calling `EPPSession.endSession()`.

8. The `EPPSession` will send an EPP `<logout>` to the EPP Server and will call `EPPClientCon.close()` to close the connection with the EPP Server.

### 7.21.2 Sample Code

**Figure 5 - EPPSession Life Cycle Sample Code** shows the code associated with the `EPPSession` life cycle. One optional step is shown, which is setting the services to `EPPDomainMapFactory.NS` *(urn:iana:xml:ns:domain-1.0)* before initializing the session. The exception handlers look for an `EPPResponse` to print out the error information sent by the EPP Server. If no `EPPResponse` exists and `isSuccess()` returns `true`, than the exception is not associated with an EPP Server error.

```java
// Create session and set session attributes.
EPPSession session = new EPPSession();

session.setTransId("ABC-12345-XYZ");
session.setVersion("1.0");
session.setLang("en-US");
session.setClientID("ClientX");
session.setPassword("foo-BAR2");
session.setNewPassword("bar-FOO2");

// Optional step: Override the services sent with an EPP Login, which
// by default are derived from the classes defined by
// the EPP.MapFactories configuration parameter.
// Optional step: Override the extensions sent with an EPP Login, which
// by default are derived from the classes defined by
// the EPP.ProtocolExtensions and EPP.CmdRspExtensions configuration
// Parameters.
Vector ProtocolExtensions=new Vector();
```
ProtocolExtensions.addElement(new String("protocolExtURI"));
Vector CommandResponseExtensions=new Vector();
CommandResponseExtensions.addElement(new String("commandExtURI"));

try {
    session.setServices(new String[]{EPPDomainMapFactory.NS});
    session.addExtensions(protocolExtensions,CommandResponseExtensions);
} catch (EPPCommandException ex) {
    System.err.println("Service " + EPPDomainMapFactory.NS + " not valid");
}

// Initialize the session
try {
    session.initSession();
} catch (EPPCommandException e) {
    EPPResponse response = session.getResponse();
    // Is an EPP Server specified error?
    if ((response != null) && (!response.isSuccess())) {
        System.err.println("initSession Server Error : " + response);
    } else { // Internal SDK error
        System.err.println("initSession Internal Error : " + e);
    }
}

// Create EPP Command Mapping Interface object
// See appropriate EPP Command Mapping Programmer Guide

// End the session
try {
    session.endSession();
} catch (EPPCommandException e) {
    EPPResponse response = session.getResponse();
    // Is an EPP Server specified error?
    if ((response != null) && (!response.isSuccess())) {
        System.err.println("endSession Server Error : " + response);
    } else { // Internal SDK error
        System.err.println("endSession Internal Error : " + e);
    }
}
7.21.3 initSession() Method
Initializes a session with the EPP Server.

Pre-Conditions
The following methods must be previously called:

- `setTransId(String)` – Sets the client transaction identifier
- `setClientID(String)` – Sets the client login identifier
- `setPassword(String)` – Sets the client password

The following methods can be previously called:

- `setVersion(String)` – Sets the EPP protocol version. Default is “1.0”.
- `setNewPassword(String)` – Requests a change in password. The EPP Server might not support this. See the SDK release notes for more details.
- `setLang(String)` – Sets the desired response message language. Default is “en-US”. The EPP Server might not support any language other than “en-US”. See the SDK release notes for more details.
- `setServices(String [])` – Set the services to use with the session. The default services are derived from the classes defined by the `EPP.MapFactories` configuration parameter intersected with the EPP <greeting> services. This is useful if the client wants to explicitly define the services.
- `addExtensions(Vector,Vector)` – Set the Extensions to use with the session. The default extensions are derived from the classes defined by the `EPP.ProtocolExtensions` and the `EPP.CmdRspExtensions` configuration properties intersected with the EPP <greeting> extension services. This is useful if the client wants to explicitly define the extension services.
- `setMode(int)` – Sets the command/response processing mode to either `EPPSession.MODE_SYNC` (default) or `EPPSession.MODE_ASYNC`. `EPPSession.MODE_ASYNC` is used for pipelining where a call to a `send` method will immediately return after sending the command and the client is responsible for calling `EPPSession.readResponse() : EPPResponse` to get the response asynchronously.

Post-Conditions
The session with the EPP Server has been authenticated for the services set by `setServices` or derived from the classes defined in the `EPP.MapFactories` configuration parameter. The successful EPPResponse can be retrieved by calling `getResponse()`.

Exceptions
`EPPCommandException` that contains the `EPPResponse` returned from the EPP Server. The `getResponse()` method returns the associated `EPPResponse`. If the exception is thrown before reading the EPP Server response, than `getResponse()` will return `null`
EPP Status Codes
The following are expected EPP Status Codes when a response has been received from the EPP Server:

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Constant Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPResult.SUCCESS</td>
<td>1000</td>
<td>Session has successfully been initialized. initSession will not throw an exception, and the successful response can be retrieved by calling getResponse().</td>
</tr>
<tr>
<td>EPPResult.COMMAND_SYNTAX_ERROR</td>
<td>2001</td>
<td>Malformed EPP message.</td>
</tr>
<tr>
<td>EPPResult.COMMAND_USE_ERROR</td>
<td>2002</td>
<td>Session has already been established, which could be due to initSession being called more than once.</td>
</tr>
<tr>
<td>EPPResult.PARAM_OUT_OF_RANGE</td>
<td>2004</td>
<td>Input attribute (i.e. client identifier, password) not valid</td>
</tr>
<tr>
<td>EPPResult.COMMAND_FAILED</td>
<td>2500</td>
<td>Internal EPP Server error</td>
</tr>
<tr>
<td>EPPResult.AUTHENTICATION_ERROR</td>
<td>2200</td>
<td>Not a valid user</td>
</tr>
<tr>
<td>EPPResult.AUTHORIZATION_ERROR</td>
<td>2201</td>
<td>User is not authorized to login.</td>
</tr>
<tr>
<td>EPPResult.TIMEOUT_END</td>
<td>2501</td>
<td>Command timeout has occurred. The EPP Server closes the connection.</td>
</tr>
</tbody>
</table>

Table 8 - EPPSession.initSession EPP Status Code Matrix

7.21.4 endSession() Method
Ends a session initialized by initSession().

Pre-Conditions
A session has been successfully initialized by initSession().

The following methods must be previously called:

- setTransId(String) – Sets the client transaction identifier

Post-Conditions
The connection is closed with the EPP Server. The successful EPPResponse can be retrieved by calling getResponse().

**Exceptions**

EPPCommandException that contains the EPPResponse returned from the EPP Server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the EPP Server response, than getResponse() will return null.

**EPP Status Codes**
The following are expected EPP Status Codes when a response has been received from the EPP Server:

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Constant Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPResult.SUCCESS_END_SESSION</td>
<td>1500</td>
<td>Session has successfully been closed. endSession will not throw an exception, and the successful response can be retrieved by calling getResponse().</td>
</tr>
<tr>
<td>EPPResult.COMMAND_SYNTAX_ERROR</td>
<td>2001</td>
<td>Malformed EPP message.</td>
</tr>
<tr>
<td>EPPResult.COMMAND_USE_ERROR</td>
<td>2002</td>
<td>Session has not been established.</td>
</tr>
<tr>
<td>EPPResult.PARAM_OUT_OF_RANGE</td>
<td>2004</td>
<td>Input attribute not valid</td>
</tr>
<tr>
<td>EPPResult.COMMAND_FAILED</td>
<td>2500</td>
<td>Internal EPP Server error</td>
</tr>
<tr>
<td>EPPResult.TIMEOUT_END</td>
<td>2501</td>
<td>Command or session timeout has occurred. The EPP Server closes the connection.</td>
</tr>
</tbody>
</table>

---

**Table 9 - EPPSession.initSession EPP Status Code Matrix**

**7.21.5 hello() Method**

Sends an EPP Hello message to get the EPP Greeting from the server. This can be done with an unauthenticated and with an authenticated session.

**Pre-Conditions**

None.

**Post-Conditions**

An EPPGreeting is returned.

**Exceptions**

EPPCommandException indicates that the EPPGreeting was not successfully received.
EPP Status Codes
None.

7.21.6 sendPoll() Method
Sends a poll command to either request a poll message or to send a poll message acknowledgement.

Pre-Conditions
A session has been successfully initialized by initSession().

The following methods must be previously called:

- `setTransId(String)` – Sets the client transaction identifier
- `setPollOp(String)` – Sets the poll operation to either `EPPSession.OP_REQ` for requesting a poll message and `EPPSession.OP_ACK` to acknowledge a poll message.

The following methods can be previously called:

- `setMsgID(String)` – Sets the message identifier associated with a poll command where the poll operation is set to `EPPSession.OP_ACK`.

Post-Conditions
A poll message is contained in the `EPPResponse` when the poll operation is `EPPSession.OP_REQ` and the poll message is removed from the poll queue when the poll operation is `EPPSession.OP_ACK`.

Exceptions
`EPPCommandException` that contains the `EPPResponse` returned from the EPP Server. The `getResponse()` method returns the associated `EPPResponse`. If the exception is thrown before reading the EPP Server response, than `getResponse()` will return `null`.

EPP Status Codes
The following are expected EPP Status Codes when a response has been received from the EPP Server:

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Constant Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPResult. SUCCESS</td>
<td>1000</td>
<td>Poll acknowledgement command was successful</td>
</tr>
<tr>
<td>EPPResult.SUCCESS_POLL_NO_MSGS</td>
<td>1300</td>
<td>Successful Poll request command. There are no</td>
</tr>
</tbody>
</table>
Table 10 - EPPSession.sendPoll EPP Status Code Matrix

<table>
<thead>
<tr>
<th>EPPResult</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESS_POLL_MSG</td>
<td>1301</td>
<td>Successful Poll request command. A poll and a queue size is returned.</td>
</tr>
<tr>
<td>COMMAND_SYNTAX_ERROR</td>
<td>2001</td>
<td>Malformed EPP message.</td>
</tr>
<tr>
<td>COMMAND_USE_ERROR</td>
<td>2002</td>
<td>Session has not been established.</td>
</tr>
<tr>
<td>PARAM_OUT_OF_RANGE</td>
<td>2004</td>
<td>Input attribute not valid</td>
</tr>
<tr>
<td>COMMAND_FAILED</td>
<td>2500</td>
<td>Internal EPP Server error</td>
</tr>
<tr>
<td>TIMEOUT_END</td>
<td>2501</td>
<td>Command or session timeout has occurred. The EPP Server closes the connection.</td>
</tr>
</tbody>
</table>

7.22 EPPEnv

EPPEnv is a utility class initialized by EPPApplication that provides an interface for the SDK configuration parameters. EPPEnvSingle is the Singleton class for EPPEnv. Figure 6 - EPPEnv Class Diagram shows the class diagram for EPPEnv and EPPEnvSingle.
Each configuration parameter has an associated EPPEnv accessor method. Table 11 - EPPEnv Method Mappings shows the mapping of the EPPEnv assessor methods to the configuration parameters.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Configuration Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>get[Client/Server]ParserDebug()</td>
<td>EPP.PoolMan.[Client/Server].debugging</td>
</tr>
<tr>
<td>get[Client/Server]ParserInitObjs()</td>
<td>EPP.PoolMan.[Client/Server].initialObjects</td>
</tr>
<tr>
<td>get[Client/Server]ParserLogFile()</td>
<td>EPP.PoolMan.[Client/Server].logFile</td>
</tr>
<tr>
<td>get[Client/Server]ParserMaxSize()</td>
<td>EPP.PoolMan.[Client/Server].maximumSize</td>
</tr>
<tr>
<td>get[Client/Server]ParserMaxSoft()</td>
<td>EPP.PoolMan.[Client/Server].maximumSoft</td>
</tr>
<tr>
<td>get[Client/Server]ParserMinSize()</td>
<td>EPP.PoolMan.[Client/Server].minimumSize</td>
</tr>
<tr>
<td>get[Client/Server]ParserObjTimeout()</td>
<td>EPP.PoolMan.[Client/Server].objectTimeout</td>
</tr>
<tr>
<td>Method Name</td>
<td>Class Name</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>get[Client/Server]ParserShrinkBy</td>
<td>EPP.PoolMan.[Client/Server].shrinkBy</td>
</tr>
<tr>
<td>get[Client/Server]ParserSkimmerFreq()</td>
<td>EPP.PoolMan.[Client/Server].skimmerFrequency</td>
</tr>
<tr>
<td>get[Client/Server]ParserUserTimeout()</td>
<td>EPP.PoolMan.[Client/Server].userTimeout</td>
</tr>
<tr>
<td>getClientSocketName()</td>
<td>EPP.ClientSocketName</td>
</tr>
<tr>
<td>getCmdResponseExtensions()</td>
<td>EPP.CmdRspExtensions</td>
</tr>
<tr>
<td>getConTimeOut</td>
<td>EPP.ConTimeOut</td>
</tr>
<tr>
<td>getFullSchemaChecking()</td>
<td>EPP.FullSchemaChecking</td>
</tr>
<tr>
<td>getGreetingServerName()</td>
<td>EPP.GreetingServerName</td>
</tr>
<tr>
<td>getKeyStore()</td>
<td>EPP.SSLKeyStore</td>
</tr>
<tr>
<td>getLogCfgFile()</td>
<td>EPP.LogCfgFile</td>
</tr>
<tr>
<td>getLogCfgFileWatch()</td>
<td>EPP.LogCfgFileWatch</td>
</tr>
<tr>
<td>getLogFile()</td>
<td>EPP.LogFile</td>
</tr>
<tr>
<td>getLogLevel()</td>
<td>EPP.LogLevel</td>
</tr>
<tr>
<td>getLogMode()</td>
<td>EPP.LogMode</td>
</tr>
<tr>
<td>getMapFactories()</td>
<td>EPP.MapFactories</td>
</tr>
<tr>
<td>getPollHandlers()</td>
<td>EPP.PollHandlers</td>
</tr>
<tr>
<td>getProtocolExtensions()</td>
<td>EPP.ProtocolExtensions</td>
</tr>
<tr>
<td>getProxyServerLocator()</td>
<td>EPP.ProxyServersLocator</td>
</tr>
<tr>
<td>getProxyServers()</td>
<td>EPP.ProxyServers</td>
</tr>
<tr>
<td>getProxyServersRandomize()</td>
<td>EPP.ProxyServersRandomize</td>
</tr>
<tr>
<td>getServerEventHandlers()</td>
<td>EPP.ServerEventHandlers</td>
</tr>
<tr>
<td>getServerName()</td>
<td>EPP.ServerName</td>
</tr>
<tr>
<td>getServerPort()</td>
<td>EPP.ServerPort</td>
</tr>
</tbody>
</table>
Table 11 - EPPEnv Method Mappings

<table>
<thead>
<tr>
<th>getServerSocketName()</th>
<th>EPP.ServerSocketName</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSSLEnabledCipherSuites()</td>
<td>EPP.SSLEnabledCipherSuites</td>
</tr>
<tr>
<td>getSSLEnabledProtocol()</td>
<td>EPP.SSLEnabledProtocols</td>
</tr>
<tr>
<td>getSSLKeyFileName()</td>
<td>EPP.SSLKeyFileName</td>
</tr>
<tr>
<td>getSSLKeyManager()</td>
<td>EPP.SSLKeyManager</td>
</tr>
<tr>
<td>getSSLKeyPassPhrase()</td>
<td>EPP.SSLKeyPassPhrase</td>
</tr>
<tr>
<td>getSSLPassPhrase()</td>
<td>EPP.SSLPassPhrase</td>
</tr>
<tr>
<td>getSSLProtocol()</td>
<td>EPP.SSLProtocol</td>
</tr>
<tr>
<td>getSSLTrustStoreFileName()</td>
<td>EPP.SSLTrustStoreFileName</td>
</tr>
<tr>
<td>getSSLTrustStorePassPhrase()</td>
<td>EPP.SSLTrustStorePassPhrase</td>
</tr>
<tr>
<td>getValidating()</td>
<td>EPP.Validating</td>
</tr>
</tbody>
</table>

8. XML Parser Pool
The XML Parser included in the SDK is XercesJ (http://xml.apache.org/xerces-j/index.html), which is not thread-safe. There are three approaches to overcoming this limitation, which include:

1. A parser per thread
2. A parser per operation
3. A parser pool

Having a parser per thread requires the SDK to have knowledge/control of the thread, which is not flexible. Instantiating a parser per operation represents a scalability issue because of the expense of instantiating and garbage collecting a parser for every operation. Utilizing a parser pool provides a more scalable solution and does not require the SDK to have knowledge/control of the thread.

PoolMan 2.1-b1 (http://sourceforge.net/projects/poolman/) was used to implement the XML Parser Pool. There is an XML Parser Pool in the SDK Client and the SDK Stub Server. The configuration parameters EPP.Poolman.[Client/Server] can be used to configure the client and Stub Server, respectively. Table 6 - SDK Configuration File
Parameters describes the XML Parser Pool configuration parameters. The $EPPEnv$ class includes accessor methods for the configuration parameters as described in Table 11 - $EPPEnv$ Method Mappings.
9. Extending the SDK

1.1 Transport

The SDK allows for the replacement of the transport layer with a combination of the EPP.ServerName and EPP.ClientSocketName configuration parameters. The pre-built transports in the SDK include:

- com.verisign.epp.transport.client.EPPPlainClientSocket – For TCP/IP connections
- com.verisign.epp.transport.client.EPPSSLClientSocket – For SSL/TLS connections

A transport class must implement com.verisign.epp.transport.EPPClientCon and have a default constructor. Figure 7 - EPPClientCon Class Diagram shows the class diagram for EPPClientCon.

![Figure 7 - EPPClientCon Class Diagram](image)

When a new EPPSession is created, the EPPSession will create an instance of the class defined by the EPP.ClientSocketName configuration parameter, call the EPPClientCon.initialize() method, and retrieve the input/output stream by calling EPPClientCon.getInputStream() and EPPClientCon.getOutputStream(). When the EPPSession is ended by a call to EPPSession.endSession(), the EPPSession will call EPPClientCon.close().

Figure 8 - SDK TCP/IP Transport Sample Code shows the code required to implement a TCP/IP transport using the host name, port number, and connection timeout SDK configuration settings. A custom transport can be used by setting EPP.ClientSocketName with the fully qualified name of the EPPClientCon class and by using EPPSession.

```
import java.net.*;
import java.io.*;
import com.verisign.epp.transport.*;
import com.verisign.epp.util.*;

public class EPPClientConSample implements EPPClientCon {
```
public EPPClientConSample() throws EPPConException {
    try {
        _hostName = EPPEnv.getServerName();
        _portNum = EPPEnv.getServerPort();
        _conTimeout = EPPEnv.getConTimeOut();
    }
    catch (EPPEnvException ex) {
        throw new EPPConException("Error initializing attributes: " + ex);
    }

    _socket = null;
    _input = null;
    _output = null;
}

public void initialize() throws EPPConException {
    try {
        _socket = new Socket(_hostName, _portNum);
        _socket.setSoTimeout(_conTimeout);
    }
    catch (UnknownHostException ex) {
        throw new EPPConException("Creating socket: " + ex);
    }
    catch (IOException ex) {
        throw new EPPConException("Creating socket: " + ex);
    }
    catch (SecurityException ex) {
        throw new EPPConException("Creating socket: " + ex);
    }

    try {
        _input = _socket.getInputStream();
        _output = _socket.getOutputStream();
    }
    catch (IOException ex) {
        throw new EPPConException("Getting streams: " + ex);
    }
}

public InputStream getInputStream() throws EPPConException {
    if (_input == null) {
        throw new EPPConException("The input stream is null");
    }

    return _input;
}

public OutputStream getOutputStream() throws EPPConException {
    if (_output == null) {
        throw new EPPConException("The output stream is null");
    }

    return _output;
}

public void close() throws EPPConException {

if (_socket != null) {
    try {
        _socket.close();
    } catch (IOException ex) {
        throw new EPPConException("Closing socket: "+ ex);
    }

    _socket = null;
    _input = null;
    _output = null;
}

private InputStream _input;
private OutputStream _output;
private Socket _socket;
private String _hostName;
private int _portNum;
private int _conTimeout;
}

Figure 8 - SDK TCP/IP Transport Sample Code
10. **Stub Server**
The Verisign Bundle EPP SDK includes an extensible Stub Server that implements all of the installed EPP commands and returns back complete hard-coded successful EPP responses.

1.1 **Event Handlers**
The Stub Server uses event handlers to process and respond to the EPP commands it receives. The `EPP.ServerEventHandlers`, located in the `epp.config` file, controls what EPP commands will be supported by the Stub Server. The general EPP commands including EPP Login, EPP Logout, EPP Hello, EPP Poll, and the creation of the EPP Greeting is handled by `com.verisign.epp.serverstub.GenHandler`.

Each EPP Command Mapping will include a handler in the `com.verisign.epp.serverstub` package. For example, the EPP Domain Command Mapping has the handler `com.verisign.epp.serverstub.DomainHandler`. The handlers set in `EPP.ServerEventHandlers` will create the list of services in the EPP Greeting. For example, by adding `com.verisign.epp.serverstub.DomainHandler` to `EPP.ServerEventHandlers`, `urn:ietf:params:xml:ns:domain-1.0` will be added to the EPP Greeting service menu.

10.23 **Poll Handlers**
Each EPP Command Mapping that supports EPP Poll will include a handler in the `com.verisign.epp.serverstub` package. For example, the EPP Domain Command Mapping has the handler `com.verisign.epp.serverstub.DomainPollHandler`. The handlers loaded in the server are included with the `EPP.PollHandlers` configuration parameter. The Stub Server implements an in-memory poll queue, which can be used by server handlers to insert messages into the queue. See the product sections of this programmer’s guide for more details about polling requirements and adding the product specific EPP Command Mappings. There is only one in-memory queue for the server, so different connections will pull messages from the same queue. See the `EPPSession.sendPoll()` for more information on sending an EPP Poll command.
11. Client Implementation Notes

1.1 Pooling

Connection pooling is a common pattern for scalable systems. *EPPSession* is associated with a single connection for its entire lifecycle. Since an *EPPSession* represents an authenticated connection with the EPP Server, it is recommended to pool *EPPSession* instances. The EPP Server does have a session timeout, so the *EPPSession* instances will have to be periodically refreshed. The *EPPSessionPool* can be used to manage a pool of sessions. Refer to the EPP.SessionPool parameters in Table 6 - SDK Configuration File Parameters for details on configuring the session pool. *EPPSessionPool.init()* will initialize the session pool assuming the SDK has already been initialized by calling *EPPApplicationSingle.initialize*(config_file : String). *EPPSessionPool.close* will cleanly close the underlying *EPPSession* instances that should be called at the ending of the client program. Below is a sample of the block of code using the EPPSessionPool for sending a domain check:

```java
EPPSession theSession = null;
try {
    theSession = EPPSessionPool.getInstance().borrowObject();
    NSDomain theDomain = new NSDomain(theSession);
    theDomain.addDomainName("example.com");
    theDomain.setSubProductID(NSSubProduct.COM);
    EPPDomainCheckResp theResponse = theDomain.sendCheck();
    ...
} catch (EPPCommandException ex) {
    if (ex.hasResponse()) {
        if (ex.getResponse().getResult().shouldCloseSession()) {
            EPPSessionPool.getInstance().invalidateObject(theSession);
            theSession = null;
        }
    }
    else if (theSession != null) {
        EPPSessionPool.getInstance().invalidateObject(theSession);
        theSession = null;
    }
} finally {
```
if (theSession != null)
    EPPSessionPool.getInstance().returnObject(theSession);

11.23.1 Multiple Session Pools

Multiple session pools can be configured and used in the SDK by defining a list of system names in the EPP.SessionPool.systemPools property. The system name “default” is for backward compatible with the EPP.SessionPool.<prop>, EPP.ServerName, and EPP.ServerPort properties. The EPPSessionPool.init() method will attempt to initialize a pool for each system name specified in the comma separated list of system names. The EPPSessionPool.close() method will cleanly close the underlying EPPSession instances contained in the pools and should be called at the end of the client program. The properties for a system session pool follow the naming convention, EPP.SessionPool.<system>.<prop>, where <system> is the system name. The following epp.config properties define the use of the “default” system pool along with a new system pool called “test”.

```java
EPPSessionPool.systemPools=default,test
EPPSessionPool.test.poolableClassName=com.verisign.epp.pool.EPPSessionPoolableFactory
EPPSessionPool.test.clientTransIdGenerator=com.verisign.epp.pool.ClientTransIdGenerator
EPPSessionPool.test.serverName=localhost
EPPSessionPool.test.serverPort=1700
EPPSessionPool.test.clientId=username
EPPSessionPool.test.password=password
EPPSessionPool.test.absouteTimeout=82800000
EPPSessionPool.test.idleTimeout=480000
EPPSessionPool.test.minIdle=0
EPPSessionPool.test.maxIdle=-1
EPPSessionPool.test.maxActive=10
EPPSessionPool.test.initMaxActive=true
EPPSessionPool.test.borrowRetries=3
EPPSessionPool.test.maxWait=60000
EPPSessionPool.test.timeBetweenEvictionRunsMillis=500
```

The example below illustrates the same session pool sample using the “test” system instead of the default system.

```java
EPPSession theSession = null;
try {
    theSession = EPPSessionPool.getInstance().borrowObject("test");
    NSDomain theDomain = new NSDomain(theSession);
    theDomain.addDomainName("example.com");
    theDomain.setSubProductID(NSSubProduct.COM);
    EPPDomainCheckResp theResponse = theDomain.sendCheck();
```
The example below illustrates the same session pool sample using the “default” system instead of using the equivalent default system methods of EPPSessionPool.

```java
EPPSession theSession = null;
try {
    theSession = EPPSessionPool.getInstance().borrowObject(EPPSessionPool.DEFAULT);
    NSDomain theDomain = new NSDomain(theSession);
    theDomain.addDomainName("example.com");
    theDomain.setSubProductID(NSSubProduct.COM);
    EPPDomainCheckResp theResponse = theDomain.sendCheck();
    ...
}
```

```java
catch (EPPCommandException ex) {
    if (ex.hasResponse()) {
        if (ex.getResponse().getResult().shouldCloseSession()) {
            EPPSessionPool.getInstance().invalidateObject("test", theSession);
            theSession = null;
        }
    } else if (theSession != null) {
        EPPSessionPool.getInstance().invalidateObject("test", theSession);
        theSession = null;
    }
}
```

```java
finally {
    if (theSession != null)
        EPPSessionPool.getInstance().returnObject("test", theSession);
}
```
EPPSessionPool.getInstance().invalidateObject(EPPSessionPool.DEFAULT, theSession);
    theSession = null;
}
else if (theSession != null) {
    EPPSessionPool.getInstance().invalidateObject(EPPSessionPool.DEFAULT, theSession);
    theSession = null;
}
}
finally {
    if (theSession != null)
    EPPSessionPool.getInstance().returnObject(EPPSessionPool.DEFAULT, theSession);
}

11.23.2 Separate SSL Configuration Per Session Pool

Section 11.23.1 defines how to use multiple session pools in the SDK. The SDK provides a set of SSL configuration properties that by default will be used across all session pools. Each session pool can define its own SSL configuration properties. All of the SSL configuration properties are available as session pool properties. Please refer to “Table 6 - SDK Configuration File Parameters” for more detail of the session pool SSL properties. The SDK includes the following files to test and demonstrate the use of separate SSL configurations per session pool:

1. bundles/verisign/epp-client.config – SDK configuration file containing a default SSL configuration that references “./lib/keystore/client1-identity.jks” for the identity keystore and “./lib/keystore/client-truststore.jks” for the truststore, and the “test” system session pool SSL configuration that references “./lib/keystore/client2-identity.jks” for the identity keystore and “./lib/keystore/client-truststore.jks” for the truststore. The “./lib/keystore/client-truststore.jks” truststore contains the self-signed server certificate contained in “../../lib/keystore/testkeys”.

2. bundles/verisign/epp-server.config – SDK configuration to run the Stub Server using “././lib/keystore/testkeys” as the identity keystore and “././lib/keystore/server-truststore.jks” as the truststore. “././lib/keystore/server-truststore.jks” contains the certificates for “././lib/keystore/client1-identity.jks” and
“../lib/keystore/client2-identity.jks” so that both client session pools can establish a two-way SSL connection.

3. lib/keystore/client1-identity.jks – Identity keystore used by the first session pool (“default”).

4. lib/keystore/client2-identity.jks – Identity keystore used by the second session pool (“test”).

5. lib/keystore/client-truststore.jks – Truststore used by clients containing the server certificate from “lib/keystore/testkeys”.

6. lib/keystore/server-truststore.jks – Truststore used by server containing the certificates from “lib/keystore/client1-identity.jks” and “lib/keystore/client2-identity.jks”.

To run a test of using two session pools with separate SSL configurations, follow the steps below (using UNIX conventions):

1. cd epp-verisign-{$BUILD_VER}/bundles/verisign
2. build.sh -DEPP.ConfigFile=epp-server.config start-server
3. In a separate Window:
   a. cd epp-verisign-{$BUILD_VER}/bundles/verisign
   b. build.sh -DEPP.ConfigFile=epp-client.config test-client

11.24 Threading

EPPSession and the SDK classes that use EPPSession (i.e. EPPDomain) are not thread safe. It is recommended that each thread use its own EPPSession or use a session pool.

11.25 Pipelining

The EPPSession class supports pipelining by changing the mode from the default of EPPSession.MODE_SYNC to EPPSession.MODE_ASYNC. When the mode is set to EPPSession.MODE_ASYNC calling any of the send methods (i.e. EPPDomain.sendCreate() : EPPDomainCreateResp) will return null and the EPPSession.readResponse() : EPPResponse method needs to be called to asynchronously read the responses.

If Session Pooling, as described in section 1.1, is being used along with Pipelining, the mode must be set to MODE_SYNC when returning sessions back to the pool. Commands like login, logout, and hello are synchronous, so sessions in the pool must be set to MODE_SYNC. The following code shows using a session pool and sending pipelining domain check commands:

```java
EPPSession theSession = null;
try {
```
theSession = EPPSessionPool.getInstance().borrowObject();

if (!theSession.isModeSupported(EPPSession.MODE_ASYNC) {
    throw new Exception(“EPPSession does NOT support MODE_ASYNC”);
}

theSession.setMode(EPPSession.MODE_ASYNC);
NSDomain theDomain = new NSDomain(theSession);
// Pipeline 10 domain check commands
for (int i = 0; i < 10; i++) {
    theDomain.addDomainName(“example” + i + “.com”);
    // It’s good to set the client trans id for mapping responses.
    theDomain.setTransId(“ASYNC-DOMAIN-CHECK-” + i);
    theDomain.setSubProductID(NSSubProduct.COM);
    // The response is null with MODE_ASYNC
    theDomain.sendCheck();
}
// Get the domain check responses asynchronously
EPPDomainCheckResp theResponse;
for (int i = 0; i < 10; i++) {
    theResponse = (EPPDomainCheckResp) theSession.readResponse();
    System.out.println(“Received async domain check “ + i + “: “ + theResponse);
}
}

catch (EPPCommandException ex) {
    if (ex.hasResponse()) {
        if (ex.getResponse().getResult().shouldCloseSession()) {
            theSession.setMode(EPPSession.MODE_SYNC);
            EPPSessionPool.getInstance().invalidateObject(theSession);
            theSession = null;
        }
    }
}
else if (theSession != null) {
    theSession.setMode(EPPSession.MODE_SYNC);
    EPPSessionPool.getInstance().invalidateObject(theSession);
}
12. Poll Messages

The servers can send different kinds of EPP poll messages. “Table 12 - Poll Message System Mapping” includes all of the possible EPP poll messages with a mapping of the systems that support them.

Table 12 - Poll Message System Mapping

<table>
<thead>
<tr>
<th>Poll Message</th>
<th>Names Store</th>
<th>N A M E</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.domain.EPPDomainTransferResp</td>
<td>X</td>
<td>X</td>
<td>Used for transfer notifications for the domain transfer actions: request, cancelled, approved, rejected, and auto approved.</td>
</tr>
<tr>
<td>com.verisign.epp.codec.domain.EPPDomainPendActionMsg</td>
<td>X</td>
<td></td>
<td>Used for domain registration pending action notifications.</td>
</tr>
<tr>
<td>com.verisign.epp.codec.domain.EPPDomainPendActionMsg with com.verisign.epp.codec.launch.EPPLaunchInfData extension</td>
<td>X</td>
<td></td>
<td>Used to notify the result of a launch application.</td>
</tr>
<tr>
<td>com.verisign.epp.codec.contact.EPPContactTransferResp</td>
<td>X</td>
<td>X</td>
<td>Used for transfer notifications for the contact transfer actions: request, cancelled, approved, rejected, and auto approved.</td>
</tr>
<tr>
<td>Package</td>
<td>X</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>---</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>com.verisign.epp.codec.emailFwd.EPPEmailFwdTransferResp</td>
<td>X</td>
<td>Used for transfer notifications for the email forwarding transfer actions: request, cancelled, approved, rejected, and auto approved.</td>
<td></td>
</tr>
<tr>
<td>com.verisign.epp.codec.defReg.EPPDefRegTransferResp</td>
<td>X</td>
<td>Used for transfer notifications for the defensive registration transfer actions: request, cancelled, approved, rejected, and auto approved.</td>
<td></td>
</tr>
<tr>
<td>com.verisign.epp.codec.nameWatch.EPPNameWatchTransferResp</td>
<td>X</td>
<td>Used for transfer notifications for the namewatch transfer actions: request, cancelled, approved, rejected, and auto approved.</td>
<td></td>
</tr>
<tr>
<td>com.verisign.epp.codec.rgp.poll.EPPRgpPollResponse</td>
<td>X</td>
<td>Used for Registry Grace Period (RGP) pending restore notifications.</td>
<td></td>
</tr>
</tbody>
</table>

The test `com.verisign.epp.namestore.interface.NSPollTst` provides a sample of processing some of the poll messages. The following is a portion of the `com.verisign.epp.namestore.interfaces.NSPollTst`. 
// Transfer notification
if (theResponse instanceof EPPDomainTransferResp) {
    System.out.println("testPoll: Got transfer notification");

    EPPDomainTransferResp theMsg = (EPPDomainTransferResp) theResponse;
    String theStatus = theMsg.getTransferStatus();

    // Transfer request?
    if (theStatus.equals(EPPDomainTransferResp.TRANSFER_PENDING)) {
        System.out.println("testPoll: Got transfer request notification");
    } // Transfer approved?
    else if (theStatus.equals(EPPDomainTransferResp.TRANSFER_CLIENT_APPROVED)) {
        System.out.println("testPoll: Got transfer approve notification");
    } // Transfer cancelled?
    else if (theStatus.equals(EPPDomainTransferResp.TRANSFER_CLIENT_CANCELLED)) {
        System.out.println("testPoll: Got transfer cancelled notification");
    } // Transfer rejected?
    else if (theStatus.equals(EPPDomainTransferResp.TRANSFER_CLIENT_REJECTED)) {
        System.out.println("testPoll: Got transfer rejected notification");
    } // Transfer auto approved?
    else if (theStatus.equals(EPPDomainTransferResp.TRANSFER_SERVER_APPROVED)) {
        System.out.println("testPoll: Got transfer auto approve notification");
    } // Transfer auto cancelled?
    else if (theStatus.equals(EPPDomainTransferResp.TRANSFER_SERVER_CANCELLED)) {
        System.out.println("testPoll: Got transfer auto cancelled notification");
    } else {
        System.out.println("testPoll: Unknown transfer status [" + theStatus + "]");
    }
} // low balance notification
else if (theResponse instanceof EPPLowBalancePollResponse) {
    System.out.println("testPoll: Got low balance notification");
} // RGP notification
else if (theResponse instanceof EPPRgpPollResponse) {
    System.out.println("testPoll: Got RGP notification");
} // Pending action notification
else if (theResponse instanceof com.verisign.epp.codec.domain.EPPDomainPendActionMsg) {
    System.out.println("testPoll: Got domain pending action notification");
} // Unknown general message
else {
    System.out.println("testPoll: Got general notification");
13. **Mappings and Extensions**

This section provides a description of each of the Extensible Provisioning Protocol (EPP) Mappings and Extensions supported by the Verisign Bundle EPP SDK that includes:

1. Definition of the files (i.e. library, schema)
2. Description of the interface classes, including the pre-conditions, the post-conditions, the exceptions, the EPP status codes, and sample code of each of the action methods.

Each of the Verisign systems support a different set of mappings and extensions included in the Verisign Bundle EPP SDK. “Table 13 - Mapping and Extension System Support” shows the mappings and extensions supported by the two Verisign EPP systems (Common Top Level Domain – CTLD and COM/NET Shared Registry System - SRS) along with a short description. The name of the Mapping and Extension corresponds to the directory name included in the source distribution.

<table>
<thead>
<tr>
<th>Mapping / Extension</th>
<th>CTLD</th>
<th>SRS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocationtoken</td>
<td></td>
<td></td>
<td>draft-ietf-regext-allocation-token IETF draft</td>
</tr>
<tr>
<td>coa</td>
<td></td>
<td>X</td>
<td>Client Object Attribute Extension to allow the creation and maintenance of key/value pairs associated with Objects.</td>
</tr>
<tr>
<td>changepoll</td>
<td>X</td>
<td>X</td>
<td>draft-ietf-regext-change-poll IETF draft</td>
</tr>
<tr>
<td>contact</td>
<td>X</td>
<td>X</td>
<td>Standard IETF Contact Mapping</td>
</tr>
<tr>
<td>defreg</td>
<td></td>
<td>X</td>
<td>Defensive Registration Mapping</td>
</tr>
<tr>
<td>domain</td>
<td></td>
<td></td>
<td>Standard IETF Domain Mapping</td>
</tr>
<tr>
<td>emailfwd</td>
<td></td>
<td>X</td>
<td>Email Forwarding Mapping</td>
</tr>
<tr>
<td>fee</td>
<td>draft-brown-epp-fees-06</td>
<td>X</td>
<td>draft-ietf-regext-epp-fees IETF draft</td>
</tr>
<tr>
<td>Extension</td>
<td>X</td>
<td>X</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------</td>
<td>---</td>
<td>---</td>
<td>-------</td>
</tr>
<tr>
<td>host</td>
<td></td>
<td></td>
<td>Standard IETF Host Mapping</td>
</tr>
<tr>
<td>idn</td>
<td>X</td>
<td>X</td>
<td>Extension for the International Domain Name Tag required for IDN domain creates.</td>
</tr>
<tr>
<td>jobsContact</td>
<td></td>
<td></td>
<td>Extension for .JOBS specific contact attributes</td>
</tr>
<tr>
<td>launch</td>
<td></td>
<td></td>
<td>Standard IETF Launch Phase EPP Extension</td>
</tr>
<tr>
<td>launchpolicy</td>
<td></td>
<td></td>
<td>draft-gould-regext-launch-policy IETF draft.</td>
</tr>
<tr>
<td>loginsec</td>
<td></td>
<td></td>
<td>draft-gould-regext-login-security IETF draft.</td>
</tr>
<tr>
<td>namestoreext</td>
<td>X</td>
<td>X</td>
<td>Namestore Extension needed to specify the target sub-product for a command.</td>
</tr>
<tr>
<td>namewatch</td>
<td></td>
<td></td>
<td>NameWatch Mapping</td>
</tr>
<tr>
<td>nsfinance</td>
<td>X</td>
<td>X</td>
<td>Low Balance Poll Mapping and Balance Mapping</td>
</tr>
<tr>
<td>org</td>
<td></td>
<td></td>
<td>Organization Mapping &amp; Extension</td>
</tr>
<tr>
<td>persreg</td>
<td></td>
<td></td>
<td>Personal Registration Extension</td>
</tr>
<tr>
<td>premiumdomain</td>
<td>X</td>
<td></td>
<td>Extensions to the domain check command, domain check response and domain</td>
</tr>
<tr>
<td>Related Domain Extension</td>
<td>update command to support premium features.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>relateddomain</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rgp</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>secdns</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sync</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>verificationcode</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>whois</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* = future

“Table 14 – Mapping and Extension Directory Standard Files” defines the standard set of files that reside in the mapping and extension directories defined in “Table 13 - Mapping and Extension System Support”. The mappings and extensions can be worked on in isolation by using the files in the directories, but most of the time the Verisign bundle directory should be used, which is the bundles/verisign directory.
Table 14 – Mapping and Extension Directory Standard Files

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>build.bat</td>
<td>This is a Windows batch file that executes Ant to compile and execute the test programs.</td>
</tr>
<tr>
<td>build.sh</td>
<td>This is a Bash shell script that executes Ant to compile and execute the test programs.</td>
</tr>
<tr>
<td>build.xml</td>
<td>This is the Ant XML configuration file.</td>
</tr>
<tr>
<td>epp.config</td>
<td>This is a sample EPP configuration file. See section 1.1 for information on configuring the SDK.</td>
</tr>
<tr>
<td>common-targets.xml</td>
<td>Common Ant targets used bye both the source and binary distributions.</td>
</tr>
<tr>
<td>logconfig.xml</td>
<td>Log4J XML configuration file.</td>
</tr>
</tbody>
</table>

“Table 15 - Mapping and Extension CODEC Packages” defines the CODEC Java packages associated with each of the mappings and extensions along with a description of how they CODEC packages are configured in the SDK.

Table 15 - Mapping and Extension CODEC Packages

<table>
<thead>
<tr>
<th>Mapping / Extension</th>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coa</td>
<td>com.verisign.epp.codec.coae xt</td>
<td>The COA Extension Encoder/Decoder package. All of the detail of encoding and decoding the COA Extension is in this package. The EPPCoaExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>contact</td>
<td>com.verisign.epp.codec.contact</td>
<td>The standard EPP Contact Encoder/Decoder package. All of the detail of encoding and decoding the EPP Contact messages are in this package. The EPPContactMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
</tbody>
</table>
| defreg              | com.verisign.epp.codec.defre  | The EPP Defensive Registration Encoder/Decoder package. All the detail of encoding and decoding the EPP
<table>
<thead>
<tr>
<th>Package</th>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defensive_reg</td>
<td>defensive_reg</td>
<td>The Defensive Registration message are in this package. The EPPDefRegMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>domain</td>
<td>com.verisign.epp.codec.domain</td>
<td>The standard EPP Domain Encoder/Decoder package. All of the detail of encoding and decoding the EPP Domain messages are in this package. The EPPDomainMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>emailfwd</td>
<td>com.verisign.epp.codec.emailfwd</td>
<td>The EPP Email Forwarding Encoder/Decoder package. All the detail of encoding and decoding the EPP Email Forwarding message are in this package. The EPPEmailFwdMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>fee</td>
<td>com.verisign.epp.codec.fee.v09</td>
<td>The Registry Fee Extension Encoder/Decoder packages. All the detail of encoding and decoding the EPP Registry Fee versioned messages are in these packages. The desired set of EPPFeeExtFactory classes must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td></td>
<td>com.verisign.epp.codec.fee.v11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>com.verisign.epp.codec.fee.v23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>com.verisign.epp.codec.fee.v1_0</td>
<td></td>
</tr>
<tr>
<td>host</td>
<td>com.verisign.epp.codec.host</td>
<td>The standard EPP Host Encoder/Decoder package. All of the detail of encoding and decoding the EPP Host messages are in this package. The EPPHostMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>idn</td>
<td>com.verisign.epp.codec.idnext</td>
<td>The IDN Tag Extension Encoder/Decoder package. All of the detail of encoding and decoding the IDN Tag Extension is in this package. The EPPIdnExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>jobsContact</td>
<td>com.verisign.epp.codec.jobsccontact</td>
<td>The .JOBS Contact Extension Encoder/Decoder package. All of the detail of encoding and decoding the .JOBS Extension is in this package. The EPPJobsContactExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>Package</td>
<td>Class Path</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>launch</td>
<td>com.verisign.epp.codec.launch</td>
<td>The standard Launch Phase Extension Encoder/Decoder package. All of the detail of encoding and decoding the Launch Phase Extension is in this package. The EPPLaunchExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>launchpolicy</td>
<td>com.verisign.epp.codec.launchpolicy.v01</td>
<td>The Launch Policy Extension Encoder/Decoder package. All of the detail of encoding and decoding the Login Security Extension is in this package. The EPPLaunchPolicyExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name. The EPPLaunchPolicyAdapter should be added to the EPP.RegistryPolicyAdapter configuration parameter using the full package and class name when running the Stub Server.</td>
</tr>
<tr>
<td>loginsec</td>
<td>com.verisign.epp.codec.loginsec.v01</td>
<td>The Login Security Extension Encoder/Decoder package. All of the detail of encoding and decoding the Login Security Extension is in this package. The EPPLLoginSecExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>loginsec</td>
<td>com.verisign.epp.codec.loginsec.v02</td>
<td>The Login Security Extension Encoder/Decoder package. All of the detail of encoding and decoding the Login Security Extension is in this package. The EPPLLoginSecExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>namestoreext</td>
<td>com.verisign.epp.codec.namestoreext</td>
<td>NamestoreExt Extension Encoder/Decoder package. All of the detail of encoding and decoding the EPP NamestoreExt messages are in this package. The EPPNamestoreExtExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>namewatch</td>
<td>com.verisign.epp.codec.namewatch</td>
<td>The EPP NameWatch Encoder/Decoder package. All the detail of encoding and decoding the EPP NameWatch message are in this package. The EPPNameWatchMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>Package Name</td>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>nsfinance</td>
<td>com.verisign.epp.codec.balance</td>
<td>The EPP Balance Encoder/Decoder package. All the detail of encoding and decoding the EPP Balance message are in this package. The EPPBalanceMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>nsfinance</td>
<td>com.verisign.epp.codec.lowbalancepoll</td>
<td>The EPP Low Balance Poll Encoder/Decoder package. All the detail of encoding and decoding the EPP Low Balance Poll message are in this package. The EPPLowBalancePollMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>org</td>
<td>com.verisign.epp.codec.org</td>
<td>The Organization Mapping Encoder/Decoder package. All of the detail of encoding and decoding the Organization Mapping is in this package. The EPPOrgMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>orgext</td>
<td>com.verisign.epp.codec.orgext</td>
<td>The Organization Extension Encoder/Decoder package. All of the detail of encoding and decoding the Organization Extension is in this package. The EPPOrgExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>persreg</td>
<td>com.verisign.epp.codec.persreg</td>
<td>The Personal Registration Encoder/Decoder package. All of the detail of encoding and decoding the Personal Registration Extension is in this package. The EPPPersRegExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>premiumdomain</td>
<td>com.verisign.epp.codec.premiumdomain</td>
<td>The Premium Domain Extension Encoder/Decoder package. All of the detail of encoding and decoding the Premium Domain Extension is in this package. The EPPPremiumDomainExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>package</td>
<td>class path</td>
<td>description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>registry</td>
<td>com.verisign.epp.codec.registry</td>
<td>The EPP Registry Encoder/Decoder package. All the detail of encoding and decoding the EPP Registry messages are in this package. The EPPRegistryMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>relateddomain</td>
<td>com.verisign.epp.codec.relateddomainext</td>
<td>The EPP Related Domain Encoder/Decoder package. All the detail of encoding and decoding the EPP Related Domain Extension are in this package. The EPPRelatedDomainExtFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>rgp</td>
<td>com.verisign.epp.codec.rgpext</td>
<td>The RGP Extension Encoder/Decoder package. All of the detail of encoding and decoding the RGP extension is in this package. The EPPRgpExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>rgp</td>
<td>com.verisign.epp.codec.rgppoll</td>
<td>The RGP Poll Encoder/Decoder package. All of the detail of encoding and decoding the RGP poll message is in this package. The EPPRgpPollMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>secdns</td>
<td>com.verisign.epp.codec.secdns</td>
<td>The SecDNS Extension Encoder/Decoder package. All of the detail of encoding and decoding the SecDNS Extension is in this package. The EPPSecDNSExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>suggestion</td>
<td>com.verisign.epp.codec.suggestion</td>
<td>The EPP Name Suggestion Encoder/Decoder package. All the detail of encoding and decoding the EPP Name Suggestion message are in this package. The EPPSuggestionMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
</tbody>
</table>
The Sync Extension Encoder/Decoder package. All of the detail of encoding and decoding the Sync extension is in this package.

The EPPSyncExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.

The Whois Extension Encoder/Decoder package. All of the detail of encoding and decoding the Whois Extension is in this package.

The EPPWhoisExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.

The EPP WhoWas Encoder/Decoder package. All the detail of encoding and decoding the EPP WhoWas message are in this package.

The EPPWhoWasMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.

The EPP Mappings and Extensions are defined using XML schema files. These files are located in the schemas directory of epp-verisign-bundle-{$BUILD_VER}.jar. Extract the epp-verisign-bundle-{$BUILD_VER}.jar in the binary distribution to view the schema files or look to the location column defined in “Table 16 – Mapping and Extension XML Schema Files” to view the schema files in the source distribution. The EPPSchemaParsingEntityResolver look for the schemas in the schemas folder of the classpath.

### Table 16 – Mapping and Extension XML Schema Files

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coa-1.0.xsd</td>
<td>coa/schemas</td>
<td>Client Object Attribute (COA) XML schema</td>
</tr>
<tr>
<td>contact-1.0.xsd</td>
<td>contact/schemas</td>
<td>Standard EPP Contact XML Schema.</td>
</tr>
<tr>
<td>defreg-1.0.xsd</td>
<td>defreg/schemas</td>
<td>Defensive Registration XML Schema.</td>
</tr>
<tr>
<td>domain-1.0.xsd</td>
<td>domain/schemas</td>
<td>Standard EPP Domain XML Schema.</td>
</tr>
<tr>
<td>filename</td>
<td>package</td>
<td>description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>emailfwd-1.0.xsd</td>
<td>emailfwd/schemas</td>
<td>Email Forwarding XML Schema.</td>
</tr>
<tr>
<td>fee-0.9.xsd</td>
<td>fee/schemas</td>
<td>Registry Fee Extension versioned XML Schema.</td>
</tr>
<tr>
<td>fee-0.11.xsd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fee-0.23.xsd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fee-1.0.xsd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>host-1.0.xsd</td>
<td>host/schemas</td>
<td>Standard EPP Host XML Schema.</td>
</tr>
<tr>
<td>idnLang-1.0.xsd</td>
<td>idn/schemas</td>
<td>IDN Language Tag Extension XML Schema.</td>
</tr>
<tr>
<td>jobsContacts-1.0.xsd</td>
<td>jobsContact/schemas</td>
<td>.JOBS Contact Extension XML Schema.</td>
</tr>
<tr>
<td>loginSec-0.1.xsd</td>
<td>loginsec/schemas</td>
<td>Login Security Extension XML Schema.</td>
</tr>
<tr>
<td>loginSec-0.2.xsd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>launchPolicy-0.1.xsd</td>
<td>launchpolicy/schemas</td>
<td>Launch Policy Extension XML Schema.</td>
</tr>
<tr>
<td>namestoreExt-1.0.xsd</td>
<td>namestoreext/schemas</td>
<td>NamestoreExt XML Schema.</td>
</tr>
<tr>
<td>launch-1.0.xsd</td>
<td>launch/schemas</td>
<td>Launch Phase EPP Extension XML schemas.</td>
</tr>
<tr>
<td>mark-1.0.xsd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>signedMark-1.0.xsd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xmldsig-core-schema.xsd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>namestoreExt-1.0.xsd</td>
<td>namestoreext/schemas</td>
<td>Namestore Extension XML Schema.</td>
</tr>
<tr>
<td>namewatch-1.0.xsd</td>
<td>namewatch/schemas</td>
<td>NameWatch XML Schema.</td>
</tr>
<tr>
<td>balance-1.0.xsd</td>
<td>nsfinance/schemas</td>
<td>Namestore Finance (Balance and Low Balance Poll) XML schemas.</td>
</tr>
<tr>
<td>lowbalance-poll-1.0.xsd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>org-1.0.xsd</td>
<td>org/schemas</td>
<td>Organization Mapping and Extension XML Schema.</td>
</tr>
<tr>
<td>orgext-1.0.xsd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>persReg-1.0.xsd</td>
<td>persreg/schemas</td>
<td>Personal Registration Extension XML Schema.</td>
</tr>
<tr>
<td>registry-0.1.xsd (IETF draft)</td>
<td>registry/schemas</td>
<td>Registry Mapping XML Schema.</td>
</tr>
</tbody>
</table>
### 13.26 Namestore Client Interfaces

The Client Interfaces package `com.verisign.epp.namestore.interfaces` consists of a set of high-level client interface classes for setting the properties of commands and sending those commands to an EPP server using an established EPP Session. Generally, there is one Client Interface class per supported EPP mapping and a set of support classes for attaching extensions to the commands. A set of higher-level utility Client Interface classes and support classes have been defined to make setting of the extensions easier, which include:

- `com.verisign.epp.namestore.interfaces.NSDomain` – Extension of the `com.verisign.epp.namestore.interfaces.EPPDomain` Client Interface that adds methods for RGP (restore request and restore report), sync, DNSSEC, Namestore Extension, and the Whois Info Extension. This interface is defined in section 13.27.1.

- `com.verisign.epp.namestore.interfaces.NSHost` – Extension of the `com.verisign.epp.namestore.interfaces.EPPHost` Client Interface that adds a method for the Namestore Extension. This interface is defined in section 13.27.2.

- `com.verisign.epp.namestore.interfaces.NSContact` - Extension of the `com.verisign.epp.namestore.interfaces.EPPContact` Client Interface that adds a method for the Namestore Extension. This interface is defined in section 13.27.3.
### 13.27 Mappings

#### 13.27.1 Domain Mapping (NSDomain Interface)

The Domain Mapping is first handled by the `com.verisign.epp.interfaces.EPPDomain` interface class, which is extended by the `com.verisign.epp.namestore.interfaces.NSDomain` interface to add support for common extensions. Convenience methods are provided in `NSDomain` to make managing domains easier. For example, the method `setSubProductID` is provided instead of having to manually add the `EPPNamestoreExtNamestoreExt` with each action.

The `NSDomain` interface has the following relevant methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NSDomain(EPPSession aNewSession)</code></td>
<td>This is the constructor method and it requires an EPP session object to be passed that has been authenticated (e.g. logged in).</td>
</tr>
<tr>
<td><code>void addDomainName(java.lang.String)</code></td>
<td>This method adds a domain name to the object for use with the action methods.</td>
</tr>
<tr>
<td><code>void addExtension(EPPCodecComponent)</code></td>
<td>Adds a extension to be sent with the command.</td>
</tr>
<tr>
<td><code>void addHostName(java.lang.String)</code></td>
<td>Adds a host name to be associated with the domain.</td>
</tr>
<tr>
<td><code>void addContact(String, String)</code></td>
<td>Adds contact for call to sendCreate() or sendUpdate().</td>
</tr>
<tr>
<td><code>EPPResponse getResponse()</code></td>
<td>This method returns the EPP Response for the last executed command on the interface.</td>
</tr>
<tr>
<td><code>Date getExpirationDate()</code></td>
<td>Returns the expiration date of the domain.</td>
</tr>
<tr>
<td><code>Vector getExtensions()</code></td>
<td>Gets all set command extensions.</td>
</tr>
<tr>
<td><code>Boolean hasExtension(Class)</code></td>
<td>Does a command extension exist with the specified Class.</td>
</tr>
<tr>
<td><code>EPPResponse sendCheck()</code></td>
<td>This method sends the domain check command to the server.</td>
</tr>
<tr>
<td><code>EPPResponse sendCreate()</code></td>
<td>This method sends the domain create command to the server.</td>
</tr>
<tr>
<td><code>EPPResponse sendDelete()</code></td>
<td>This method sends the domain delete command to the server.</td>
</tr>
<tr>
<td><code>EPPResponse sendInfo()</code></td>
<td>This method sends the domain info command to the server.</td>
</tr>
<tr>
<td><code>EPPResponse sendUpdate()</code></td>
<td>This method sends the domain update command to the server.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>sendRenew()</code></td>
<td>This method sends the domain renewal command to the server.</td>
</tr>
<tr>
<td><code>sendTransfer()</code></td>
<td>This method sends the domain transfer command to the server.</td>
</tr>
<tr>
<td><code>sendRestoreRequest()</code></td>
<td>This method sends the Registry Grace Period (RGP) restore request command.</td>
</tr>
<tr>
<td><code>sendRestoreReport()</code></td>
<td>This method sends the Registry Grace Period (RGP) restore report command.</td>
</tr>
<tr>
<td><code>sendSync()</code></td>
<td>This method sends the ConsoliDate sync command.</td>
</tr>
<tr>
<td><code>setTransferOpCode(String)</code></td>
<td>Sets the operation type for future transfer commands. Valid values are “TRANSFER_REQUEST”, “TRANSFER_APPROVE” or “TRANSFER_REJECT”</td>
</tr>
<tr>
<td><code>setTransId(String)</code></td>
<td>This method sets the client transaction identifier.</td>
</tr>
<tr>
<td><code>setDay(int)</code></td>
<td>Sets the target day for a call to <code>sendSync()</code>.</td>
</tr>
<tr>
<td><code>setMonth(int)</code></td>
<td>Sets the target month using the Calendar month constants (Calendar.JANUARY to Calendar.DECEMBER) for a call to <code>sendSync()</code>.</td>
</tr>
<tr>
<td><code>setIDNLangTag(String)</code></td>
<td>Sets the IDN language tag for a call to <code>sendCreate()</code> of an IDN domain.</td>
</tr>
<tr>
<td><code>setSubProductID(String)</code></td>
<td>Sets the NameStore sub-product id associated with the action method. The NSSubProduct class includes a set of constant that can be used as the <code>setSubProductID</code> argument value.</td>
</tr>
<tr>
<td><code>setPeriodLength(int)</code></td>
<td>Sets the registration period for a create, renew, or transfer command. The default value is 1 year.</td>
</tr>
<tr>
<td><code>setPeriodUnit(String)</code></td>
<td>Sets the unit of the registration period as defined by <code>setPeriodLength(int)</code> according to the EPP specification. The servers currently only support the default value of “y” for years.</td>
</tr>
<tr>
<td><code>setExpirationDate(Date)</code></td>
<td>Sets the current expiration date for a call to <code>sendRenew()</code>.</td>
</tr>
<tr>
<td><code>setUpdateAttrib(…)</code></td>
<td>Sets attribute to update for a call to <code>sendUpdate()</code>.</td>
</tr>
<tr>
<td><code>setAuthString(String)</code></td>
<td>Sets authorization string for a call to <code>sendCreate()</code>, <code>sendTransfer()</code>, or <code>sendInfo()</code></td>
</tr>
<tr>
<td><code>setRegistrant(String)</code></td>
<td>Sets the domain registrant for a call to <code>sendCreate()</code> or <code>sendUpdate()</code>.</td>
</tr>
<tr>
<td><code>setHosts(String)</code></td>
<td></td>
</tr>
</tbody>
</table>
Sets the desired level of host information using one of the HOSTS_* constant values for a call to sendInfo(). The possible values include:

- HOSTS_ALL – Get information on all hosts (delegated and subordinate). This is the default value.
- HOSTS_DELEGATED – Get information on just the delegated hosts.
- HOSTS_SUBORDINATE – Get information on just the subordinate hosts.

```java
void setWhoisInfo(boolean)
Sets the flag that determines if the whois info extension should be included in the response to sendInfo(). The target server needs to support the “Extensible Provisioning Protocol Extension Mapping: Whois Info” to set this flag.
```

```java
void setSecDNSCreate(List<EPPSecDNSExtDsData>)
Sets the list of <EPPSecDNSExtDsData> instances in order to create delegation signer information.
```

```java
void setSecDNSUpdateForAdd(List<EPPSecDNSExtDsData>, boolean)
Sets the list of <EPPSecDNSExtDsData> instances in order to add delegation signer information. It also supports setting of an urgent attribute in the SecDNS update extension which determines the priority of the request.
```

```java
void setSecDNSUpdateForChg(List<EPPSecDNSExtDsData>, boolean)
Sets the list of <EPPSecDNSExtDsData> instances in order to change delegation signer information. It also supports setting of an urgent attribute in the SecDNS update extension which determines the priority of the request.
```

```java
void setSecDNSUpdateForRem(List<Integer>, boolean)
Sets the list of <Integer> instances (i.e keytags of DS records) in order to remove delegation signer information. It also supports setting of an urgent attribute in the SecDNS update extension which determines the priority of the request.
```

```java
void setCoaCreate(List<EPPCoaExtAttr>)
Sets the list of <EPPCoaExtAttr> instances (which in turn each specify a single key/value pair) to be associated with the object being created.
```

```java
void setCoaUpdateForPut(List<EPPCoaExtAttr>)
Sets the list of <EPPCoaExtAttr> instances (which in turn each specify a single key/value pair) to be associated with the object being updated. If the object already has a value associated with the key, this value will be overwritten with the value specified.
```

```java
void setCoaUpdateForRem(List<EPPCoaExtKey>)
Sets the list of <EPPCoaExtKey> instances specifying the key portions of existing key/value pairs to be removed from the object being updated.
```

```java
void addOrgId(EPPOrgExtId)
Adds an org identifier with role for use with sendCreate().
```

```java
void addUpdateOrgId(int, EPPOrgExtId)
```
Action methods are prefixed with `send` and are shown in bold in the previous table. Each action method has a different set of pre-conditions defining what attributes need to be set with the setter methods. Each action method will return a response from the server and will throw an exception if any error occurs. If the exception is associated with an error response from the server, then the response can be retrieved from the exception with a call to `getResponse()`. The following sections describe and provide sample code for the action methods.

### 13.27.1.1 NSDomain() method

The `NSDomain` constructor requires that an authenticated `EPPSession` object be passed upon creation. Once created, the `NSDomain` object can perform multiple functions without re-initializing the `EPPSession` object. For example, you can use the same initialized `NSDomain` object to create and info a domain with the `sendCreate()` and `sendInfo()` commands.

#### 13.27.1.1.1 Pre-Conditions

An authenticated session has been successfully established with an `EPPSession`.

#### 13.27.1.1.2 Post-Conditions

The `NSDomain` instance is ready for the execution of one or more operations.

#### 13.27.1.1.3 Exceptions

None

#### 13.27.1.1.4 Sample Code

The following example shows the steps of initializing an `EPPSession`, then using the `EPPSession` to initialize the `NSDomain` interface.

```java
EPPSession session = new EPPSession();

// optional
session.setTransId("ABC-12345-XYZ");
session.setVersion("1.0");
session.setLang("en");

// required
session.setClientID("ClientXID");
session.setPassword("ClientXPass");

try {
    session.initSession();
} catch (EPPCommandException ex) {
    ex.printStackTrace();
    System.exit(1);
}
```
13.27.1.2 sendCheck() method

The sendCheck() method sends the EPP check domain command to check the allowable flag for one or more domains.

13.27.1.2.1 Pre-Conditions

The following list shows the accessor methods for the required attributes:

- addDomainName(String) – add a domain name to the object in preparation for an action method.
- setSubProductID(String) – sub-product associated with operation. Use one of the constants from com.verisign.epp.namestore.interfaces.NSSubProduct or the TLD value going forward.

The following list shows the accessor methods for the optional attributes:

- addExtension(EPPCodecComponent) - Sets the extension, if any (Ex. EPPPremiumDomainCheck for Premium Domain extension)

13.27.1.2.2 Post-Conditions

On success, an EPPDomainCheckResp is returned, with the following attributes:

- Results – the check results are returned in a vector containing one or more EPPDomainCheckResult objects.

On success, an EPPDomainCheckResp is returned, with the following optional attributes based on authorization level and command attributes set:

- com.verisign.epp.codec.premiumdomain.EPPPremiumDomainCheckResp extension containing premium information. This is available via the getExtension(Class) method.

13.27.1.2.3 Exceptions

An EPPCommandException will be returned that contains the EPPResponse returned from the server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the server response, then getResponse() will return null.

13.27.1.2.4 Sample Code

The following example shows the steps of performing a check on domains through the use of the NSDomain client interface and the sendCheck() method:

```java
NSDomain domain = new NSDomain(session);
```
try {
    // Check single domain name
    domain.setTransId("ABC-12345-XYZ");
    domain.addDomainName("mydomain.tv");
    domain.setSubProductID(NSSubProduct.TV);

    // optionally set the premium extension
    EPPPremiumDomainCheck extension = new EPPPremiumDomainCheck( true );
    domain.addExtension( extension );

    response = domain.sendCheck();

    // Correct number of results?
    Assert.assertEquals(1, response.getCheckResults().size());

    // For each result
    for (int i = 0; i < response.getCheckResults().size(); i++) {
        EPPDomainCheckResult currResult = (EPPDomainCheckResult) response.getCheckResults().elementAt(i);

        if (currResult.isAvailable()) {
            System.out.println("domainCheck: Domain "+ currResult.getName() + " is available");
        } else {
            System.out.println("domainCheck: Domain "+ currResult.getName() + " is not available");
        }
    }

    if (response.hasExtension(EPPPremiumDomainCheckResp.class)) {
        EPPPremiumDomainCheckResp resp = (EPPPremiumDomainCheckResp) response.getExtension(EPPPremiumDomainCheckResp.class);

        // For each result
        for (int i = 0; i < resp.getCheckResults().size(); i++) {
            EPPDomainCheckResult currResult = (EPPDomainCheckResult) resp .getCheckResults().elementAt(i);

            if (currResult.isPremium()) {
                System.out.println("domainCheck: Domain " + currResult.getName() + " is premium");
                if(currResult.getPrice() != null) {
                    System.out.println("domainCheck: Premium price is "+ currResult.getPrice());
                    System.out.println("domainCheck: Premium renewal price is "+ currResult.getRenewalPrice());
                }
            } else {
                System.out.println("domainCheck: Domain " + currResult.getName() + " is not premium");
            }
        }
    }
}
13.27.1.3 sendCreate() method

The sendCreate() method sends the EPP create domain command to the server.

13.27.1.3.1 Pre-Conditions

This method requires that several attributes be set prior to execution. The following list shows the accessor methods for the required attributes:

- addDomainName(String) – add the domain name to the object in preparation for an action method.
- setSubProductID(String) – sub-product associated with operation. Use one of the constants from com.verisign.epp.namestore.interfaces.NSSubProduct or the TLD value going forward.

The following list shows the accessor methods for the optional attributes:

- setPeriodLength(int) – Registration period for the domain create command. The default value is 1 year. Valid values are from 1 to 10.
- setIdNLangTag(String) – Language tag associated with IDN domain create command. This is required if an IDN domain name is specified.
- setSecDNSCreate(List<EPPSecDNSExtDsData>) – Sets the list of <EPPSecDNSExtDsData> instances in order to create delegation signer (DS) information.
- setCoaCreate(List<EPPCoaExtAttr>) – Sets the list of <EPPCoaExtAttr> instances (which in turn each specify a single key/value pair) to be associated with the object being created.

13.27.1.3.2 Post-Conditions

On success, an EPPDomainCreateResp is returned, with the following attributes:

- Name – the name of the domain being created.
- CreationDate – the date that the domain was created.
- ExpirationDate – the date that the domain is due to be renewed.
13.27.1.3.3 Exceptions

An EPPCommandException will be returned that contains the EPPResponse returned from the server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the server response, then getResponse() will return null.

13.27.1.3.4 Sample Code

The following example shows the steps of performing a create of a domain through the use of the NSDomain client interface and the sendCreate() method:
try {

domain.setTransId("ABC-12345-XYZ");

domain.addDomainName("example.tv");
domain.setSubProductID(NSSubProduct.TV);
domain.addHostName1("a.b.com");
domain.addHostName2("c.d.com");
domain.setPeriodUnit(2);

// -- Add Delegation Signer Information
// instantiate a secDNS:KeyData object
EPPSecDNSExtKeyData keyData = new EPPSecDNSExtKeyData();
keyData.setFlags( EPPSecDNSExtKeyData.FLAGS_ZONE_KEY_SEP );
keyData.setProtocol( EPPSecDNSExtKeyData.DEFAULT_PROTOCOL );
keyData.setAlg( EPPSecDNSAlgorithm.RSASHA1 );
keyData.setPubKey( "AQOxXpFbRp7+zPBoTt6zL7Af0aEKzpS4JbVB" +
      "5ofk5E5H5puWU+mHT9hm2kMph6LzDEEL142" +
      "nq0HrgiETFCsN/VM4Zn+meRkELLpCG933Cu/H" +
      "hwvxf2aZenUAA6Vb9fWyXq1EMYRW05K/gH2Ge" +
      "w55k/0o6Ev7DKG2YdJYA17QsaZtFw==" );

// instantiate another secDNS:KeyData object
EPPSecDNSExtKeyData keyData2 = new EPPSecDNSExtKeyData( EPPSecDNSExtKeyData.FLAGS_ZONE_KEY_SEP,
    EPPSecDNSExtKeyData.DEFAULT_PROTOCOL,
    EPPSecDNSAlgorithm.RSASHA1,
    "AQOxXpFbRp7+zPBoTt6zL7Af0aEKzpS4JbVB" +
    "5ofk5E5H5puWU+mHT9hm2kMph6LzDEEL142" +
    "nq0HrgiETFCsN/VM4Zn+meRkELLpCG933Cu/H" +
    "hwvxf2aZenUAA6Vb9fWyXq1EMYRW05K/gH2Ge" +
    "w55k/0o6Ev7DKG2YdJYA17QsaZtFw==" );

// instantiate a secDNS:dsData object
EPPSecDNSExtDsData dsData = new EPPSecDNSExtDsData();
dsData.setKeyTag( 34095 );
dsData.setAlg( EPPSecDNSAlgorithm.RSASHA1 );
dsData.setDigestType( EPPSecDNSExtDsData.SHA1_DIGEST_TYPE );
dsData.setDigest( "BDB4FFPF11566D6E6A5BA44ED0018797564AA289" );
dsData.setMaxSigLife( 604800 );
dsData.setKeyData( keyData );

// instantiate another secDNS:dsData object
EPPSecDNSExtDsData dsData2 = new EPPSecDNSExtDsData( 10563,
    EPPSecDNSAlgorithm.RSASHA1,
    EPPSecDNSExtDsData.SHA1_DIGEST_TYPE,
    "9C20674BFF957211D129B0DFE9410AF753559D4B",
    604800, keyData2 );

// dsData Records
List dsDataRecords = new ArrayList();
dsDataRecords.add( dsData );
dsDataRecords.add( dsData2 );

// Call only if server supports creating delegation signer
// information
theDomain.setSecDNSCreate( dsDataRecords );
response = (EPPDomainCreateResp) domain.sendCreate();

//-- Output response attributes using accessors
System.out.println("domainCreate: name = " +
response.getName());
System.out.println("domainCreate: CreationDate = " +
response.getCreationDate());
System.out.println("domainCreate: ExpirationDate = " +
response.getExpirationDate());

} // end of try block
catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block

13.27.1.4 sendDelete() method

The sendDelete() method sends the EPP delete domain command to the server.

13.27.1.4.1 Pre-Conditions

This method expects that the domain object be populated with the unique identifier of the domain to be deleted. The following method must be called to populate the domain identifier:

- addDomainName(String) – call the add domain name method passing the unique domain name in preparation for an action method.
- setSubProductID(String) – sub-product associated with operation. Use one of the constants from com.verisign.epp.namestore.interfaces.NSSubProduct or the TLD value going forward.

13.27.1.4.2 Post-Conditions

On success, an EPPResponse is returned, with no attributes.

13.27.1.4.3 Exceptions

An EPPCommandException will be returned that contains the EPPResponse returned from the server. The getResponse() method returns the associated EPPResponse. If the
exception is thrown before reading the server response, then `getResponse()` will return `null`.

13.27.1.4.4 Sample Code

The following example shows the steps of performing a delete of an CTLD domain through the use of the `NSDomain` client interface and the `sendDelete()` method:

```java
try {
    domain.setTransId("ABC-12345-XYZ");
    domain.addDomainName("abcdefdg.tv");
    domain.setSubProductID(NSSubProduct.TV);
    response = (EPPResponse) domain.sendDelete();
} // end of try block
catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block
```

13.27.1.5 `sendInfo()` method

The `sendInfo()` method sends the EPP info domain command to the server.

13.27.1.5.1 Pre-Conditions

This method expects that the domain object be populated with the single domain name of the domain to be queried. The following method must be called to populate the domain identifier:

- `addDomainName(String)` – call the add domain name method passing the unique domain name in preparation for an action method.
- `setSubProductID(String)` – sub-product associated with operation. Use one of the constants from `com.verisign.epp.namestore.interfaces.NSSubProduct` or the TLD value going forward.

The following list shows the the accessor methods for the optional attributes:

- `setAuthString(String)` – sets the authorization string for getting full domain information if not sponsoring Registrar.
- `setHosts(String)` – Sets the desired level of host information. Use one of the `HOSTS_` constant values:
HOSTS_ALL – Get information on all hosts (delegated and subordinate). This is the default value.
HOSTS_DELEGATED – Get information on just the delegated hosts.
HOSTS_SUBORDINATE – Get information on just the subordinate hosts.

- setWhoisInfo(boolean) – Sets the flag for the desire for the whois information defined in the
  com.verisign.epp.codec.whois.EPPWhoisInfData class.

13.27.1.5.2 Post-Conditions

On success, an EPPDomainInfoResp is returned, with the following required attributes:

- name – the fully qualified domain name
- roid – the domain roid
- clientId – identifier of sponsoring client

On success, an EPPDomainInfoResp is returned, with the following optional attributes based on authorization level and command attributes set:

- expirationDate - date and time identifying the end of the domain's registration period
- createdBy – identifier of the client that created the domain
- createdDate - date and time of domain creation
- lastUpdatedBy - identifier of the client that last updated the domain
- lastUpdatedDate - date and time of the most recent domain modification
- lastTransferDate - date and time of the most recent successful transfer
- authInfo - authorization information
- hosts - names of host objects
- nses - names of name server objects
- status - one or more current status descriptors
- com.verisign.epp.codec.whois.EPPWhoisInfData extension contains the additional whois information. This is available via the getExtension(Class) method and will only be set if the EPPWhoisInf command extension was included with a flag value of true, which is automatically set using the setWhoisInfo(boolean) method.
- com.verisign.epp.codec.secdnsext.EPPSecDNSExtInfData extension contains the Delegation Signer (DS) information. This is available via the getExtension(Class) method.
**com.verisign.epp.codec.coaext.EPPCoaExtInfData** extension contains the Client Object Attribute (COA) information. This is available via the `getExtension(Class)` method.

13.27.1.5.3 **Exceptions**

An `EPPCommandException` will be returned that contains the `EPPResponse` returned from the server. The `getResponse()` method returns the associated `EPPResponse`. If the exception is thrown before reading the server response, then `getResponse()` will return `null`.

13.27.1.5.4 **Sample Code**

The following example shows the steps of querying an CTLD domain through the use of the `NSDomain` client interface and the `sendInfo()` method:
try {

domain.setTransId("ABC-12345-XYZ");
domain.addDomainName("abcdefg.tv");
domain.setSubProductId(NSSubProduct.TV);
domain.setHosts(EPPDomain.HOSTS_ALL);
domain.setWhoisInfo(true); // Call only if server supports it

response = (EPPDomainInfoResp) domain.sendInfo();

//-- Output required response attributes using accessors
System.out.println("domainInfo: name = " +
                   response.getName());
System.out.println("domainInfo: created by = " +
                   response.getCreatedBy());
System.out.println("domainInfo: expiration date = " +
                   response.getExpirationDate());

//-- Output additional whois information if returned
if (response.hasExtension(EPPWhoisInfData.class)) {
    EPPWhoisInfData whois = (EPPWhoisInfData)
                          response.getExtension(EPPWhoisInfData.class);
    System.out.println("domainInfo: registrar = " +
                       whois.getRegistrar());
    System.out.println("domainInfo: whois server = " +
                       whois.getWhoisServer());
}

//-- Output the secDNS:infData extension if returned
if (response.hasExtension(EPPSecDNSExtInfData.class)) {
    EPPSecDNSExtInfData infData = (EPPSecDNSExtInfData)
                                   response.getExtension(EPPSecDNSExtInfData.class);
    Collection dsDataVec = infData.getDsData();
    EPPSecDNSExtDsData dsData = null;
    if (dsDataVec == null) {
        System.out.println("domainInfo: secDNS:infData
                           dsDataVec = " + dsDataVec);
    } else {
        int i = 0;
        Iterator iter = dsDataVec.iterator();
        while (iter.hasNext()) {
            dsData = (EPPSecDNSExtDsData)iter.next();

            System.out.println("domainInfo:
                       secDNS:infData/dsData[" + i + "]/keyTag = " +
                               dsData.getKeyTag());
            System.out.println("domainInfo:
                       secDNS:infData/dsData[" + i + "]/alg = " +
                               dsData.getAlg());
            System.out.println("domainInfo:
                       secDNS:infData/dsData[" + i + "]/digestType = " +
                               dsData.getDigestType());
            System.out.println("domainInfo:
                       secDNS:infData/dsData[" + i + "]/digest

13.27.1.6 sendUpdate() method

The sendUpdate() method sends the EPP update domain command to the server.
13.27.1.6.1  Pre-Conditions

This method expects that the domain object be populated with the unique identifier of the
domain to be updated and the attributes to change. The following methods must be called
to populate the domain name:

- addDomainName(String) – call the add domain name method passing the
  unique domain name in preparation for an action method.
- setSubProductID(String) – sub-product associated with operation. Use
  one of the constants from com.verisign.epp.namestore.interfaces.NSSubProduct or the TLD value
  going forward.

The following list shows the accessor methods for the optional attributes:

- setSecDNSUpdateForAdd(List<EPPSecDNSExtDsData>, boolean) –
  Sets the list of <EPPSecDNSExtDsData> instances in order to add
delegation signer information. It also supports setting of an urgent
  attribute in the SecDNS update extension which determines the priority of
  the request.
- setSecDNSUpdateForChg(List<EPPSecDNSExtDsData>, boolean) –
  Sets the list of <EPPSecDNSExtDsData> instances in order to change
delegation signer information. It also supports setting of an urgent
  attribute in the SecDNS update extension which determines the priority of
  the request.
- setSecDNSUpdateForRem(List<Integer>, boolean) – Sets the list of
  <Integer> instances (i.e key tags of DS records) in order to remove
delegation signer information. It also supports setting of an urgent
  attribute in the SecDNS update extension which determines the priority of
  the request.
- setCoaUpdateForPut(List<EPPCoaExtAttr>) - Sets the list of
  <EPPCoaExtAttr> instances (which in turn each specify a single
  key/value pair) to be associated with the object being updated. If the
  object already has a value for one or more of the specified keys, the
  existing value will be overwritten by the specified one.
- setCoaUpdateForRem(List<EPPCoaExtKey>) - Sets the list of
  <EPPCoaExtKey> instances specifying the key portions of existing
  key/value pairs to be removed from the object being updated.
- addExtension(EPPCodecComponent) – Sets the extension, if any (Ex.
  EPPPremiumDomainReAssignCmd for Premium Domain (ReAssign)
  extension.)

13.27.1.6.2  Post-Conditions

On success, an EPPResponse is returned, with no attributes.
13.27.1.6.3 Exceptions

An EPPCommandException will be returned that contains the EPPResponse returned from the server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the server response, then getResponse() will return null.

13.27.1.6.4 Sample Code

The following example shows the steps of performing an update of an domain through the use of the NSDomain client interface and the sendUpdate() method:

```java
try {
    domain.setTransId("ABC-12345-XYZ");
    domain.addDomainName("abcdefg.tv");
    domain.setSubProductID(NSSubProduct.TV);

    // Execute update
    domain.addHostName("a.b.com");
    domain.addHostName("a.c.com");

    // instantiate the rem DS Key Tag List
    List remKeyTags = new ArrayList();
    remKeyTags.add(new Integer(34095));
    remKeyTags.add(new Integer(10563));

    // Call only if server supports updating delegation signer
    // information
    domain.setSecDNSUpdateForRmv(remKeyTags);

    // Call only when premium domain (reassign) extension need to be set
    EPPPremiumDomainReAssignCmd extension =
        new EPPPremiumDomainReAssignCmd();
    extension.setShortName("testregistrar");
    domain.addExtension(extension);

    response = domain.sendUpdate();
} // end of try block
catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block
```

13.27.1.7 sendRenew() method

The sendRenew() method sends the EPP renew domain command to the server.
13.27.1.7.1 Pre-Conditions

This method expects that the domain object be populated with the unique identifier of the domain to be updated and the attributes to change. The following methods must be called to populate the domain name:

- addDomainName(String) – call the add domain name method passing the unique domain name in preparation for an action method.
- setSubProductID(String) – sub-product associated with operation. Use one of the constants from com.verisign.epp.namestore.interfaces.NSSubProduct or the TLD value going forward.
- setExpirationDate(Date) – the current expiration date needs to be set to confirm the renewal.
- SetPeriodUnit(String) – the period for which the domain is to be renewed.

13.27.1.7.2 Post-Conditions

On success, an EPPDomainRenewResp is returned, with the following attributes:

- Name – the name of the domain being created.
- ExpirationDate – the new date that the domain is due to be renewed.

13.27.1.7.3 Exceptions

An EPPCommandException will be returned that contains the EPPResponse returned from the server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the server response, then getResponse() will return null.

13.27.1.7.4 Sample Code

The following example shows the steps of performing an renewal of an domain through the use of the NSDomain client interface and the sendRenew() method:
try {
    domain.setTransId("ABC-12345-XYZ");
    domain.addDomainName("abcdefg.tv");
    domain.setSubProductID(NSSubProduct.TV);

    // Execute update
    domain.setExpirationDate(theCurrentExpirationDate);
    domain.setPeriodUnit("1");

    // Set the price value
    vector someExtensions = new vector;
    someExtensions.addElement(new EPPNSDomBillRenew("50.00"));
    domain.getExtension().addExtensions(someExtensions);

    response = (EPPDomainRenewResp) domain.sendRenew();

    //-- Output response attributes using accessors
    System.out.println("domainRenew: name = " + 
    response.getName());
    System.out.println("domainRenew: ExpirationDate = " + 
    response.getExpirationDate());
}
} // end of try block

catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block

13.27.1.8 sendTransfer() method

The sendTransfer() method sends the EPP transfer domain command to the server. There are three different operations that can be performed with a sendTransfer() command – request a transfer, approve a transfer or reject a transfer. A transfer is initiated by the new registrar sending a transfer request. Once a transfer has been requested, the current registrar of record will be notified by an external method. The current registrar is then required to either approve or reject the transfer.

13.27.1.8.1 Pre-Conditions

This method expects that the domain object be populated with the unique identifier of the domain to be updated and the attributes to change. The following methods must be called to populate the domain name:

- addDomainName(String) – call the add domain name method passing the unique domain name in preparation for an action method.
- setSubProductID(String) – sub-product associated with operation. Use one of the constants from
com.verisign.epp.namestore.interfaces.NSSubProduct or the TLD value going forward.

- SetTransferOpCode(String) – sets the operation code for this particular call. Valid values are “TRANSFER_REQUEST”, “TRANSFER_APPROVE” or “TRANSFER_REJECT”

13.27.1.8.2 Post-Conditions

On success, an EPPDomainTransferResp is returned, with attributes depending on the operation code requested.

For “TRANSFER_REQUEST”:

- ActionDate – The date at which the transfer must be approved or rejected by, otherwise it will be accepted by default.

For “TRANSFER_APPROVE” and “TRANSFER_REJECT” no additional attributes are set.

13.27.1.8.3 Exceptions

An EPPCommandException will be returned that contains the EPPResponse returned from the server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the server response, then getResponse() will return null.

13.27.1.8.4 Sample Code

The following example shows the steps of performing a transfer request of a domain through the use of the NSDomain client interface and the sendTransfer() method:
try {
    domain.setTransId("ABC-12345-XYZ");
    domain.addDomainName("abcdefg.cc");
    domain.setSubProductID(NSSubProduct.CC);
    domain.setTransferOpCode("TRANSFER_REQUEST");

    // Execute update
    response = (EPPDomainTransferResp) domain.sendTransfer();

    System.out.println("domainTransfer: Action Date: " +
                        response.getActionDate());
} // end of try block

catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block

The following example shows the steps of performing a transfer approve of a domain through the use of the NSDomain client interface and the sendTransfer() method:

try {
    domain.setTransId("ABC-12345-XYZ");
    domain.addDomainName("abcdefg.cc");
    domain.setTransferOpCode("TRANSFER_APPROVE");

    // Execute update
    response = (EPPDomainTransferResp) domain.sendTransfer();

} // end of try block

catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block

The following example shows the steps of performing a transfer reject of a domain through the use of the NSDomain client interface and the sendTransfer() method:
try {
    domain.setTransId("ABC-12345-XYZ");
    domain.addDomainName("abcdefg.cc");
    domain.setTransferOpCode("TRANSFER_REJECT");

    // Execute update
    response = (EPPDomainTransferResp) domain.sendTransfer();
}
} // end of try block

catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
}
} // end of catch block

13.27.1.9 sendRestoreRequest() method

The sendRestoreRequest() method sends the EPP restore request, which is encoded as an
EPP update command with an rgp:update command/response extension.

13.27.1.9.1 Pre-Conditions

The following list shows the accessor methods for the required attributes:

- addDomainName(String) – add the domain name to the object in
  preparation for an action method.
- setSubProductID(String) – sub-product associated with operation. Use
  one of the constants from
  com.verisign.epp.namestore.interfaces.NSSubProduct or the TLD value
  going forward.

13.27.1.9.2 Post-Conditions

On success, an EPPResponse is returned, with no attributes.

13.27.1.9.3 Exceptions

An EPPCommandException will be returned that contains the EPPResponse returned
from the server. The getResponse() method returns the associated EPPResponse. If the
exception is thrown before reading the server response, then getResponse() will return
null.

13.27.1.9.4 Sample Code

The following example shows the steps of performing an update of an domain through
the use of the NSDomain client interface and the sendRestoreRequest() method:
try {
    domain.setTransId("ABC-12345-XYZ");
    domain.addDomainName("abcdefg.tv");
    domain.setSubProductID(NSSubProduct.TV);

    response = domain.sendRestoreRequest();
} // end of try block

catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block

13.27.1.10  sendRestoreReport() method

The sendRestoreReport() method sends the EPP restore report, which is encoded as an
EPP update command with an rgp:update command/response extension. The EPP restore
report follows a previous call to sendRestoreRequest().

13.27.1.10.1  Pre-Conditions

The following list shows the accessor methods for the required attributes:

- addDomainName(String) – add the domain name to the object in
  preparation for an action method.
- setSubProductID(String) – sub-product associated with operation. Use
  one of the constants from
  com.verisign.epp.namestore.interfaces.NSSubProduct or the TLD value
going forward.

13.27.1.10.2  Post-Conditions

On success, an EPPResponse is returned, with no attributes.

13.27.1.10.3  Exceptions

An EPPCommandException will be returned that contains the EPPResponse returned
from the server. The getResponse() method returns the associated EPPResponse. If the
exception is thrown before reading the server response, then getResponse() will return
null.

13.27.1.10.4  Sample Code

The following example shows the steps of performing an update of an domain through
the use of the NSDomain client interface and the sendRestoreReport() method:
try {
    domain.setTransId("ABC-12345-XYZ");
    domain.addDomainName("abcdefg.tv");
    domain.setSubProductID(NSSubProduct.TV);

    EPPRgpExtReport theReport = new EPPRgpExtReport();
    theReport.setPreWhois("Pre-delete whois data goes here. Both XML and free text are allowed");
    theReport.setPostWhois("Post-delete whois data goes here. Both XML and free text are allowed");
    theReport.setDeleteTime(new Date());
    theReport.setRestoreTime(new Date());
    theReport.setRestoreReason(new EPPRgpExtReportText("Registrant Error"));
    theReport.setStatement1(new EPPRgpExtReportText(
        "This registrar has not" + " restored the Registered Domain in order to " + "assume the rights to use or sell the Registered" + " Name for itself or for any third party");

    theReport.setStatement2(new EPPRgpExtReportText(
        "The information in this report " + " is true to best of this registrar's knowledge, and this" + " registrar acknowledges that intentionally supplying false" + " information in this report shall " + "constitute an incurable material breach of the Registry-Registrar" + " Agreement");

    theReport.setOther("other stuff");

    // Execute restore report
    domain.setReport(theReport);
    theResponse = domain.sendRestoreReport();
} // end of try block
catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block
13.27.1.11 sendSync() method

The sendSync() method sends the EPP restore request, which is encoded as an EPP update command with a sync:update command/response extension.

13.27.1.11.1 Pre-Conditions

The following list shows the accessor methods for the required attributes:

- addDomainName(String) – add the domain name to the object in preparation for an action method.
- setSubProductID(String) – sub-product associated with operation. Use one of the constants from com.verisign.epp.namestore.interfaces.NSSubProduct or the TLD value going forward.
- setMonth(int) – Sets the month using a Calendar constant of the target expiration date.
- setDay(int) – Sets the day of the target expiration date.

13.27.1.11.2 Post-Conditions

On success, an EPPResponse is returned, with no attributes.

13.27.1.11.3 Exceptions

An EPPCommandException will be returned that contains the EPPResponse returned from the server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the server response, then getResponse() will return null.

13.27.1.11.4 Sample Code

The following example shows the steps of performing an update of a domain through the use of the NSDomain client interface and the sendSync() method:
try {
    domain.setTransId("ABC-12345-XYZ");
    domain.addDomainName("abcdefg.tv");
    domain.setSubProductID(NSSubProduct.TV);
    domain.setMonth(Calendar.JUNE);
    domain.setDay(15);

    response = domain.sendSync();
}

// end of try block

catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
}
// end of catch block
### 13.27.2 Host Mapping (NSHost Interface)

The Host Mapping is first handled by the `com.verisign.epp.interfaces.EPPHost` interface class, which is extended by the `com.verisign.epp.namestore.interfaces.NSHost` interface to add support for common extensions. Convenience methods are provided in `NSHost` to make managing hosts easier. For example, the method `setSubProductID` is provided instead of having to manually add the `EPPNamestoreExtNamestoreExt` with each action.

The `NSHost` interface has the following relevant methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>NSHost(EPPSession aNewSession)</code></td>
<td>This is the constructor method and it requires an EPP session object to be passed that has been authenticated (e.g. logged in).</td>
</tr>
<tr>
<td><code>void addHostName(java.lang.String)</code></td>
<td>This method adds a host name to the object for use with the action methods.</td>
</tr>
<tr>
<td><code>EPPResponse getResponse()</code></td>
<td>This method returns the EPP Response for the last executed command on the interface.</td>
</tr>
<tr>
<td><code>EPPResponse sendCheck()</code></td>
<td>This method sends the host check command to the server.</td>
</tr>
<tr>
<td><code>EPPResponse sendCreate()</code></td>
<td>This method sends the host create command to the server.</td>
</tr>
<tr>
<td><code>EPPResponse sendDelete()</code></td>
<td>This method sends the host delete command to the server.</td>
</tr>
<tr>
<td><code>EPPResponse sendInfo()</code></td>
<td>This method sends the host info command to the server.</td>
</tr>
<tr>
<td><code>EPPResponse sendUpdate()</code></td>
<td>This method sends the host update command to the server.</td>
</tr>
<tr>
<td><code>void addIPV4Address(java.lang.String)</code></td>
<td>This method adds an IPv4 address to the host object.</td>
</tr>
<tr>
<td><code>void removeIPV4Address(java.lang.String)</code></td>
<td>This method removes an IPv4 address from the host object.</td>
</tr>
<tr>
<td><code>void setName(java.lang.String)</code></td>
<td>This method sets a new value of the host name for use with the update method.</td>
</tr>
<tr>
<td><code>void addExtension(EPPCodecComponent)</code></td>
<td>This method sets the EPP extension for the host object.</td>
</tr>
</tbody>
</table>
### Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Parameters</th>
</tr>
</thead>
</table>
| Vector       | getExtensions()  
This method gets the EPP extensions for the host object. |
| void         | setTransId(java.lang.String)  
This method sets the client transaction identifier. |
| void         | setSubProductID(String)  
Sets the NameStore sub-product id associated with the action method. The NSSubProduct class includes a set of constant that can be used as the setSubProductID argument value. |
| void         | addOrgId(EPPOrgExtId)  
Adds an org identifier with role for use with sendCreate(). |
| Void         | addUpdateOrgId(int, EPPOrgExtId)  
Adds an org identifier with role for use with sendUpdate(). |

Action methods are prefixed with `send` and are shown in bold in the previous table. Each action method has a different set of pre-conditions defining what attributes need to be set with the `NSHost` setter methods. Each action method will return a response from the server and will throw an exception if any error occurs. If the exception is associated with an error response from the server, then the response can be retrieved from the exception with a call to `getResponse()`. The following sections describe and provide sample code for the action methods, the `NSHost` constructor and methods requiring additional explanation.

#### 13.27.2.1 `NSHost()` method

The `NSHost` constructor requires that an authenticated `EPPSession` object be passed upon creation. Once created, the `NSHost` object can perform multiple functions without reinitializing the `EPPSession` object. For example, you can use the same initialized `NSHost` object to create and info a host with the `sendCreate()` and `sendInfo()` commands.

13.27.2.1.1 Pre-Conditions

An authenticated session has been successfully established with an `EPPSession`.

13.27.2.1.2 Post-Conditions

The `NSHost` instance is ready for the execution of one or more operations.

13.27.2.1.3 Exceptions

None

13.27.2.1.4 Sample Code

The following example shows the steps of initializing an `EPPSession`, using the `EPPSession` to initialize the `NSHost` interface, then setting the extension.
```java
EPPSession session = new EPPSession();

// optional
session.setTransId("ABC-12345-XYZ");
session.setVersion("1.0");
session.setLang("en");

// required
session.setClientID("ClientXID");
session.setPassword("ClientXPass");

try {
    session.initSession();
} catch (EPPCommandException ex) {
    ex.printStackTrace();
    System.exit(1);
}

NSHost host = new NSHost(session);
```

### 13.27.2.2 sendCheck() method

The `sendCheck()` method sends the EPP check host command to check the allowable flag for one or more hosts.

#### 13.27.2.2.1 Pre-Conditions

The following method must be called to populate the host names:

- `addHostName(String)` – add a host name to the object in preparation for an action method.
- `setSubProductID(String)` – sub-product associated with operation. Use one of the constants from `com.verisign.epp.namestore.interfaces.NSSubProduct` or the TLD value going forward.

#### 13.27.2.2.2 Post-Conditions

On success, an `EPPHostCheckResp` is returned, with the following attributes:

- Results – the check results are returned in a vector containing one or more `EPPHostCheckResult` objects.

#### 13.27.2.2.3 Exceptions

An `EPPCommandException` will be returned that contains the `EPPResponse` returned from the server. The `getResponse()` method returns the associated `EPPResponse`. If the exception is thrown before reading the Server response, then `getResponse()` will return `null`.  

VeriSign Inc. Proprietary Information
Verisign Bundle EPP SDK Programmer's Guide • page 111
13.27.2.2.4 **Sample Code**

The following example shows the steps of performing a check on hosts through the use of the *Host* client interface and the *sendCheck()* method:

```java
try {
    // Check single host name
    host.setTransId("ABC-12345-XYZ");
    host.addHostName(“ns.myhost.net”);
    host.setSubProductID(NSSubProduct.CC);
    response = (EPPHostCheckResp) host.sendCheck();

    // Correct number of results?
    Assert.assertEquals(1, response.getCheckResults().size());

    // For each result
    for (int i = 0; i < response.getCheckResults().size(); i++) {
        EPPHostCheckResult currResult = (EPPHostCheckResult)
        response.getCheckResults().elementAt(i);

        if (currResult.isAvailable()){
            System.out.println("hostCheck: Host " +
            currResult.getName() + " is available");
        } else {
            System.out.println("hostCheck: Host " +
            currResult.getName() + " is not available");
        }
    }
} // end of try block
catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block
```

### 13.27.2.3 *sendCreate()* method

The *sendCreate()* method sends the EPP create host command to the server.

#### 13.27.2.3.1 Pre-Conditions

This method expects that the host object be populated with the name and address of the host to be created. The following methods must be called to populate the host name and address:

- `addHostName(String)` – add a host name to the object in preparation for an action method.
• setSubProductID(String) – sub-product associated with operation. Use one of the constants from com.verisign.epp.namestore.interfaces.NSSubProduct or the TLD value going forward.
• setIPV4Address(String) or setIPV6Address(String) – add an Ipv4 or Ipv6 address to the host object.

13.27.2.3.2 Post-Conditions
On success, an EPPHostCreateResp is returned, with the following attributes:

• Name – the host name that was successfully created is returned in the response.

13.27.2.3.3 Exceptions
An EPPCommandException will be returned that contains the EPPResponse returned from the server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the Server response, then getResponse() will return null.

13.27.2.3.4 Sample Code
The following example shows the steps of performing a create of a host through the use of the NSHost client interface and the sendCreate() method:

```java
try {
    host.setTransId("ABC-12345-XYZ");
    host.addHostName("ns.myhost.net");
    host.setIPV4Address("123.34.34.2");
    host.setSubProductID(NSSubProduct.NET);
    response = (EPPHostCreateResp) host.sendCreate();
    System.out.println("Host created: "+response.getName());
} // end of try block
catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block
```

13.27.2.4 sendDelete() method
The sendDelete() method sends the EPP delete host command to the server. This method requires that no domains are associated with the host prior to deletion. If there are domains associated with the host the delete will fail.
13.27.2.4.1 **Pre-Conditions**

This method expects that the host object be populated with the name of the host to be deleted. The following method must be called to populate the host name:

- `addHostName(String)` – add a host name to the object in preparation for an action method.
- `setSubProductID(String)` – sub-product associated with operation. Use one of the constants from `com.verisign.epp.namestore.interfaces.NSSubProduct` or the TLD value going forward.

13.27.2.4.2 **Post-Conditions**

On success, a standard `EPPResponse` is returned.

13.27.2.4.3 **Exceptions**

An `EPPCommandException` will be returned that contains the `EPPResponse` returned from the server. The `getResponse()` method returns the associated `EPPResponse`. If the exception is thrown before reading the Server response, then `getResponse()` will return `null`.

13.27.2.4.4 **Sample Code**

The following example shows the steps of performing a delete of a host through the use of the `NSHost` client interface and the `sendDelete()` method:

```java
try {
    host.setTransId("ABC-12345-XYZ");
    host.addHostName("ns.myhost.net");
    host.setSubProductID(NSSubProduct.NET);
    response = (EPPResponse) host.sendDelete();
    System.out.println("EPP Response: " + response);
} // end of try block
catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block
```
**13.27.2.5 sendInfo() method**

The *sendInfo()* method sends the EPP info host command to the server.

### 13.27.2.5.1 Pre-Conditions

This method expects that the host object be populated with a single host name of the host to be queried. The following method must be called to populate the host name:

- `addHostName(String)` – add a host name to the object in preparation for an action method.
- `setSubProductID(String)` – sub-product associated with operation. Use one of the constants from `com.verisign.epp.namestore.interfaces.NSSubProduct` or the TLD value going forward.

### 13.27.2.5.2 Post-Conditions

On success, an *EPPHostInfoResp* is returned, with the following attributes:

- `name` – the fully qualified host name.
- `address` – the Ipv4 or Ipv6 address of the host.
- `RegistrarId` – the identifier of the registrar the contact was created by.

### 13.27.2.5.3 Exceptions

An *EPPCommandException* will be returned that contains the *EPPResponse* returned from the server. The *getResponse()* method returns the associated *EPPResponse*. If the exception is thrown before reading the Server response, than *getResponse()* will return *null*.

### 13.27.2.5.4 Sample Code

The following example shows the steps of querying an host through the use of the *NSHost* client interface and the *sendInfo()* method:

```java
// Example code
```

```java
```
try {
    host.setTransId("ABC-12345-XYZ");
    host.addHostName("ns.myhost.net");
    host.setSubProductID(NSSubProduct.NET);
    response = (EPPHostInfoResp) host.sendInfo();
    System.out.println("hostInfo: name = " + response.getName());
    EPPHostAddress currAddress = (EPPHostAddress) response.getAddress();
    System.out.println("hostInfo: ip = ");
    // IPV4 Address?
    if (currAddress.getType() == EPPHostAddress.IPV4) {
        System.out.println(", type = IPV4");
    } // IPV6 Address?
    else if (currAddress.getType() == EPPHostAddress.IPV6) {
        System.out.println(", type = IPV6");
    }
    System.out.println("hostInfo: registrar = " +
            response.getRegistrar());
} // end of try block
catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block

13.27.2.6 sendUpdate() method

The sendUpdate() method sends the EPP update host command to the server.

13.27.2.6.1 Pre-Conditions

This method expects that the host object be populated with the name of the host to be updated and the Ipv4 or Ipv6 address to change. The following methods must be called to populate the host name with a new address:

- addHostName(String) – add a host name to the object in preparation for an action method.
- setSubProductID(String) – sub-product associated with operation. Use one of the constants from com.verisign.epp.namestore.interfaces.NSSubProduct or the TLD value going forward.
- addIPv4Address (String) or addIPv6Address (String) – add an Ipv4 or Ipv6 address to the host object.
- removeIPv4Address (String) or removeIPv4Address (String) – remove an Ipv4 or Ipv6 address from the host object.
13.27.2.6.2 **Post-Conditions**

On success, a standard `EPPResponse` is returned.

13.27.2.6.3 **Exceptions**

An `EPPCommandException` will be returned that contains the `EPPResponse` returned from the server. The `getResponse()` method returns the associated `EPPResponse`. If the exception is thrown before reading the Server response, then `getResponse()` will return `null`.

13.27.2.6.4 **Sample Code**

The following example shows the steps of performing an update of a host through the use of the `NSHost` client interface and the `sendUpdate()` method:

```java
try {
    host.setTransactionId("ABC-12345-XYZ");
    host.addHostName("ns.myhost.net");
    host.setSubProductID(NSSubProduct.NET);

    // set new IP address
    host.add IPV4Address("123.34.34.12");
    // remove old IP address
    host.remove IPV4Address("123.34.34.66");

    response = (EPPResponse) host.sendUpdate();

    System.out.println("Host updated: "+ response);
} // end of try block
catch (EPPCommandException cmdException) {
    // do something to handle the exception
    handleException(cmdException);
} // end of catch block
```

### 13.27.2.7 NSHost Support Classes

The Host package, `com.verisign.epp.codec.host`, contains additional classes required for host provisioning and maintenance. The package contains the following relevant classes:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPHostAddress</td>
<td>This class is used by the <code>sendInfo()</code> method for encapsulating IPv4 and IPv6 addresses.</td>
</tr>
<tr>
<td>EPPHostCheckResult</td>
<td>This class is used by the <code>sendCheck()</code> method for returning the allowable flag for multiple hosts being checked.</td>
</tr>
</tbody>
</table>
13.27.2.8 EPPHostAddress

The *EPPHostAddress* class is used by the *sendInfo()* method for encapsulating *Ipv4* and *Ipv6* addresses. The *getType()* method should be used for determining whether the address is an *Ipv4* or *Ipv6* address. The *getName()* and *getAddress()* methods return the host name and IP address, respectively, for the host queried. Please refer to the previous sections for sample code demonstrating the use of the *EPPHostAddress* class.

13.27.2.9 EPPHostCheckResult

The *EPPHostCheckResult* class is used by the *sendCheck()* method for returning the allowable flag for multiple hosts. The *isAvailable()* method of this class returns *true* if the host is available and *false* if the host is not available.
13.27.3 Contact Mapping (NSContact Interface)

This section is intended to provide users of the Extensible Provisioning Protocol (EPP) Software Development Kit (SDK) with an overview of the Contact, additions to the SDK. This document includes the following Contact information:

1. Definition of the Contact files (i.e. library, schema)
2. Description of the Contact interface class, including the pre-conditions, the post-conditions, the exceptions and sample code of each of the action methods.

The SDK provides detailed interface information in HTML Javadoc. This document does not duplicate the detailed interface information contained in the HTML Javadoc. Descriptions are provided for the Contact interface elements, the pre-conditions, the post-conditions, and example code.

It is assumed that the reader has reviewed the associated EPP Specifications and has a general understanding of the EPP concepts. Much of the EPP details are encapsulated in the SDK, but having a solid understanding of the EPP concepts will help in effectively using the SDK.

The Contact Mapping is first handled by the `com.verisign.epp.interfaces.EPPContact` interface class, which is extended by the `com.verisign.epp.namestore.interfaces.NSContact` interface to add support for common extensions. Convenience methods are provided in `NSContact` to make managing domains easier. For example, the method `setSubProductID` is provided instead of having to manually add the `EPPNamestoreExtNamestoreExt` with each action.
13.27.3.1 Contact Mapping Packages
Contact consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the SDK packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.contact</td>
<td>Contact EPP Encoder/Decoder package. All of the detail of encoding and decoding the Contact EPP messages are encapsulated in this package. The EPPContactMapFactory must be added to the EPP.MapFactories configuration parameter.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Addition of the EPPContact Client interface class, which is the primary class used by a Contact SDK client.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the com.verisign.epp.serverstub.ContactHandler class used to implement the EPP Contact Stub Server behavior. This class must be added to the EPP.ServerEventHandlers configuration parameter.</td>
</tr>
</tbody>
</table>
13.27.3.2 Contact Mapping XML Schema Files
The Contact EPP Mapping is defined using XML schema files. These files are located in the epp-contact.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contact-1.0.xsd</td>
<td>schemas</td>
<td>Contact XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.</td>
</tr>
</tbody>
</table>
13.27.3.3 Contact Client Interface

13.27.3.3.1 Overview

Figure 11 - EPPContact Class Diagram shows the class diagram for EPPContact, which is the primary interface used for managing Contact objects. Some classes out of the com.verisign.epp.codec.gen are used to support the interface. There is a reference from EPPContact to an EPPSession. The action methods are prefixed with send, and each action method has a different set of pre-conditions defining what attributes need to be set with the EPPContact setter methods. Each action method will return a response from the EPP Server and will throw an exception if any error occurs. If the exception is associated with an error response from the EPP Server, than the response can be retrieved from the exception with a call to getResponse().
13.27.3.2  Initialization

This section shows how an EPPContact is initialized. After each operation, EPPContact resets itself so that it can be used for more than one operation. For example, you can use the same initialized EPPContact to create a Contact with sendCreate(), than use it again to delete a Contact with sendDelete(). The attributes have to be set/reset before each call to an operation.

1.1.1.1.1  Pre-Conditions
An authenticated session has been successfully established with an EPPSession.

1.1.1.1.2  Post-Conditions
The EPPContact instance is ready for the execution of one or more operations.

1.1.1.1.3  Exceptions
None

1.1.1.1.4  Sample Code
Figure 12 - EPPContact Initialization Sample Code shows the steps of initializing an EPPSession, than using the EPPSession to initialize and EPPContact.

```java
EPPSession session = new EPPSession();
session.setClientID("ClientX");
session.setPassword("ClientXPass");
session.setTransId("ABC-12345");

try {
    session.initSession();
} catch (EPPCommandException ex) {
    ex.printStackTrace();
    System.exit(1);
}
EPPContact contact = EPPContact(session);
```

**Figure 12 - EPPContact Initialization Sample Code**
13.27.3.3  **sendCreate() Method**

Creates a Contact using this method. **sendCheck()** can be used to check the availability of a Contact before invoking **sendCreate()**.

### 1.1.1.1.1.5 Pre-Conditions

The following methods must be called:

- **addContactId(String)** – Contact ID.
- **addPostalInfo(EPPContactPostalDefinition)** – Sets the postal information of the contact.
- **setEmail(String)** – Sets the email address of the contact.
- **setAuthorizationId(String)** - Authorization string, which is provided by client.

The following methods can be called:

- **setVoicePhone(String)** – Sets the voice phone number for the contact.
- **setVoiceExt(String)** – Sets the extension for the voice phone number of the contact.
- **setFaxNumber(String)** – Sets the fax number for the contact
- **setFaxExt(String)** – Sets the extension for the fax number of the contact
- **setDisclose(EPPContactDisclose)** – Sets the disclose information about the contact.
- **addExtension(EPPCodecComponent)** – Sets the extension, if any
- **setTransId(String)** – Client transaction identifier, which is mirrored back in the response.

### 1.1.1.1.6 Post-Conditions

The Contact was successfully created. **EPPContactCreateResp** is returned, with the following attributes:

- `<epp:extension>` - EPPResponse.getExtension() if response contains extension
- `<epp:trID>` - EPPResponse.getTransId()
- `<contact:id>` – EPPContactCreateResp.getId()
- `<contact:crDate>` – EPPContactCreateResp.getCreationDate()

This response does not have Contact related information except for the creation date. The response may not even contain an extension.

### 1.1.1.1.7 Exceptions
**EPPCommandException** that contains the *EPPResponse* returned from the EPP Server. The *getResponse()* method returns the associated *EPPResponse*. If the exception is thrown before reading the EPP Server response, then *getResponse()* will return *null*.

### 1.1.1.1.8 Sample Code

Figure 13 - sendCreate() Sample Code shows the steps of creating a Contact. On success, *EPPContactCreateResp* is returned, which contains the contact Id and creation date. If an error occurs, *EPPCommandException* is thrown, and if the error was the result of an EPP Server response, than the response can be retrieved from the exception.
try {

    contact.setTransId("ABC-12345-XYZ");
    contact.setAuthorizationId("ClientXYZ");
    contact.addContactId("sh8013");
    contact.setVoicePhone("+1.7035555555");
    contact.setVoiceExt("123");
    contact.setFaxNumber("+1.7035555556");
    contact.setFaxExt("456");
    contact.setEmail("jdoe@example.com");

    // Streets
    Vector streets = new Vector();
    streets.addElement("123 Example Dr.");
    streets.addElement("Suite 100");
    streets.addElement("This is third line");

    EPPContactAddress address = new EPPContactAddress();
    address.setStreets(streets);
    address.setCity("Dulles");
    address.setStateProvince("VA");
    address.setPostalCode("20166-6503");
    address.setCountry("US");

    EPPContactPostalDefinition name = new EPPContactPostalDefinition(
        EPPContactPostalDefinition.ATTR_TYPE_LOC);

    name.setName("John Doe");
    name.setOrg("Example Inc.");
    name.setAddress(address);

    contact.addPostalInfo(name);

    // this is not a valid Example but it will do
    EPPContactAddress Intaddress = new EPPContactAddress();

    Intaddress.setStreets(streets);
    Intaddress.setCity("Dulles");
    Intaddress.setStateProvince("VA");
    Intaddress.setPostalCode("20166-6503");
    Intaddress.setCountry("US");

    EPPContactPostalDefinition Intname = new EPPContactPostalDefinition(
        EPPContactPostalDefinition.ATTR_TYPE_INT);

    Intname.setName("John Doe");
    Intname.setOrg("Example Inc.");
    Intname.setAddress(Intaddress);

    contact.addPostalInfo(Intname);

    // disclose names
    Vector names = new Vector();

    // names.addElement(new
// EPPContactDiscloseName(EPPContactDiscloseName.ATTR_TYPE_LOC));
names.addElement(new EPPContactDiscloseName(
EPPContactDiscloseName.ATTR_TYPE_INT));

// disclose orgs
Vector orgs = new Vector();
orgs.addElement(new EPPContactDiscloseOrg(
EPPContactDiscloseOrg.ATTR_TYPE_LOC));
orgs.addElement(new EPPContactDiscloseOrg(
EPPContactDiscloseOrg.ATTR_TYPE_INT));

// disclose addresses
Vector addresses = new Vector();
addresses.addElement(new EPPContactDiscloseAddress(
EPPContactDiscloseAddress.ATTR_TYPE_LOC));
addresses.addElement(new EPPContactDiscloseAddress(
EPPContactDiscloseAddress.ATTR_TYPE_INT));

// disclose
EPPContactDisclose disclose = new EPPContactDisclose();
disclose.setFlag("0");
disclose.setNames(names);
disclose.setOrgs(orgs);
disclose.setAddresses(addresses);
disclose.setVoice("");
disclose.setFax("");
disclose.setEmail("");

contact.setDisclose(disclose);
response = (EPPContactCreateResp) contact.sendCreate();

// -- Output all of the response attributes
System.out.println("contactCreate: Response = [" + response + "]\n\n");
System.out.println("Contact ID : " + response.getId());
System.out.println("Contact Created Date : " + response.getCreationDate());

Figure 13 - sendCreate() Sample Code

13.27.3.4 sendCheck() Method
Checks the availability of one or more Contact objects.

1.1.1.1.9 Pre-Conditions
The following methods must be previously called:

- addContactId(String) – Contact Id.
The following methods can be previously called:

- setTransId(String) – Client transaction identifier, which is mirrored back in the response.

1.1.1.1.10 Post-Conditions

The Contact was successfully checked. EPPContactCheckResp is returned with the following attributes:

- `<epp:extension>` - EPPResponse.getExtension() if response contains extension
- `<epp:trID>` - EPPResponse.getTransId()
- A Vector of EPPContactCheckResult objects – EPPContactInfoResp.getCheckResults(). Each EPPContactCheckResult object contains the following attributes:
  - `<contact:id>` - EPPContactCheckResult.getId()
  - `<contact:reason>` - EPPContactCheckResult.getContactReason()

1.1.1.1.11 Exceptions

_EPPCommandException_ that contains the EPPResponse returned from the EPP Server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the EPP Server response, than getResponse() will return null.

1.1.1.1.12 Sample Code

Figure 9 - sendCheck() Sample Code shows the steps of creating EPPContactCheckResp. If an error occurs, _EPPCommandException_ is thrown, and if the error was the result of an EPP Server response, than the response can be retrieved from the exception.

```java
try {
    contact.setTransId("ABC-12345");
    contact.addContactId("sh8013");
    EPPContactCheckResp response = contact.sendCheck();

    // For each result
    for (int i = 0; i < response.getCheckResults().size(); i++) {
        EPPContactCheckResult currResult = (EPPContactCheckResult) response.getCheckResults().elementAt(i);

        if (currResult.isAvailable()) {
            System.out.println("contactCheck: Contact "+ currResult.getId() + " is available");
        } else {
            System.out.println("contactCheck: Contact "+ currResult.getId() + " is unavailable");
        }
    }
} catch (EPPCommandException e) {
    e.printStackTrace();
}
```
catch (EPPCommandException ex) {
    EPPResponse response = (EPPResponse) ex.getResponse();
    if (response != null)
        System.err.println("Contact Check response error: " + ex + ", response = " + response);
    else
        System.err.println("Contact Check exception: " + ex);
}

Figure 9 - sendCheck() Sample Code

13.27.3.3.5 sendInfo() Method
Retrieves the Contact information.

1.1.1.1.13 Pre-Conditions
The following methods must be previously called:

- addContactId(String) – Contact Id.

The following methods can be previously called:

- setTransId(String) – Client transaction identifier, which is mirrored back in the response.

The Contact must exist.

1.1.1.1.14 Post-Conditions

The Contact information was successfully retrieved. EPPContactInfoResp contains the Contact information with the following attributes:

- <epp:extension> - EPPResponse.getExtension() if response contains extension
- <epp:trID> - EPPResponse.getTransId()
- <contact:id> - EPPContactInfoResp.getId() Gets the Contact ID
- <contact:roid> - EPPContactInfoResp.getRoid() Gets the Contact ROID
- <contact:status> - EPPContactInfoResp.getStatuses() Gets the vector of status
- <contact:postalinfo> - EPPContactInfoResp.getPostalInfo() Gets the Contact Postal Info
- <contact:voice> - EPPContactInfoResp.getVoice() Gets the Contact Voice number
• <contact:fax> - EPPContactInfoResp.getFax() Gets the Contact Fax number
• <contact:email> - EPPContactInfoResp.getEamil() Gets the Contact Email address
• <contact:clId> - EPPContactInfoResp.getClientId() Gets the Contact Client ID
• <contact:crDate> - EPPContactInfoResp.getCreatedDate() Gets the Contact Creation date
• <contact:trDate> - EPPContactInfoResp.getLastTransferDate() Gets the Contact last transfer date
• <contact:authInfo> - EPPContactInfoResp.getAuthInfo() Gets the Contact Authorization information
• <contact:disclose> - EPPContactInfoResp.getDisclose() Gets the Contact disclose information

1.1.1.1.15 Exceptions

EPPCommandException that contains the EPPResponse returned from the EPP Server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the EPP Server response, than getResponse() will return null.

1.1.1.1.16 Sample Code

Figure 14 - sendInfo() Sample Code shows the steps of creating EPPContactInfoResp. If an error occurs, EPPCommandException is thrown, and if the error was the result of an EPP Server response, than the response can be retrieved from the exception.

```java
try {
   EPPContactInfoResp contactResponse;
   System.out.println("JobsContact: Contact Info");
   contact.setTransId("ABC-12345-XYZ");
   contact.addContactId("helloworld");
   contactResponse = contact.sendInfo();
   System.out.println("contactInfo: id = " + response.getId());
   Vector postalContacts = null;
   if (response.getPostalInfo().size() > 0) {
      postalContacts = response.getPostalInfo();
      for (int j = 0; j < postalContacts.size(); j++) {
         // Name
         System.out.println("contactInfo:\t\tname = "
```
  + ((EPPContactPostalDefinition) postalContacts
     .elementAt(j)).getName());

  // Organization
  System.out.println("contactInfo:\t\torganization = "
      + ((EPPContactPostalDefinition) postalContacts
          .elementAt(j)).getOrg());

  EPPContactAddress address =
      ((EPPContactPostalDefinition)postalContacts
        .elementAt(j)).getAddress();

  for (int i = 0; i < address.getStreets().size(); i++) {
      System.out.println("contactInfo:\t\tstreet" + (i + 1) + " = " + address.getStreets().elementAt(i));
  }

  // Address City
  System.out.println("contactInfo:\t\tcity = " + address.getCity());

  // Address State/Province
  System.out.println("contactInfo:\t\tstate province = " + address.getStateProvince());

  // Address Postal Code
  System.out.println("contactInfo:\t\tpostal code = " + address.getPostalCode());

  // Address County
  System.out.println("contactInfo:\t\tcountry = " + address.getCountry());
}

  // Contact E-mail
  System.out.println("contactInfo:\temail = " + response.getEmail());

  // Contact Voice
  System.out.println("contactInfo:\tvoice = " + response.getVoice());

  // Contact Voice Extension
  System.out.println("contactInfo:\tvoice ext = " + response.getVoiceExt());

  // Contact Fax
  System.out.println("contactInfo:\tfax = " + response.getFax());

  // Contact Fax Extension
  System.out.println("contactInfo:\tfax ext = " + response.getFaxExt());

  // Client Id
  System.out.println("contactInfo: client id = "
+ response.getClientId();

// Created By
System.out.println("contactInfo: created by = "
    + response.getCreatedBy());

// Created Date
System.out.println("contactInfo: create date = "
    + response.getCreatedDate());

// -- Output optional response attributes using accessors
// Contact Fax
if (response.getFax() != null) {
    System.out.println("contactInfo: tfax = " +
        response.getFax());
}

// Contact Voice
if (response.getVoice() != null) {
    System.out.println("contactInfo: tVoice = "
        + response.getVoice());
}

// Last Updated By
if (response.getLastUpdatedBy() != null) {
    System.out.println("contactInfo: last updated by = "
        + response.getLastUpdatedBy());
}

// Last Updated Date
if (response.getLastUpdatedDate() != null) {
    System.out.println("contactInfo: last updated date = "
        + response.getLastUpdatedDate());
}

// Last Transfer Date
if (response.getLastTransferDate() != null) {
    System.out.println("contactInfo: last updated date = "
        + response.getLastTransferDate());
}

// Authorization Id
if (response.getAuthInfo() != null) {
    System.out.println("contactInfo: authorization info = "
        + response.getAuthInfo().getPassword());
}

// Disclose
if (response.getDisclose() != null) {
    System.out.println("contactInfo: disclose info = "
        + response.getDisclose());
}

Figure 14 - sendInfo() Sample Code
13.27.3.3.6 sendUpdate() Method

Updates the attributes of a Contact.

1.1.1.1.1.17 Pre-Conditions

The following methods must be previously called:

- \textit{addContactId(String)} – Contact Id.

The following methods can be previously called:

- \textit{setTransId(String)} – Client transaction identifier, which is mirrored back in the response.
- \textit{addExtension(EPPCodecComponent)} – Command extension, if any.
- \textit{addStatus(String)} – Contact status
- \textit{addPostalInfo(EPPContactPostalDefinition)} – Adding postal information
- \textit{setVoicePhone(String)} – Update voice number
- \textit{setFaxNumber(String)} – Update fax number
- \textit{setAuthorizationId(String)} – Update authorization information
- \textit{setDisclose(EPPContactDisclose)} – Updates the contact disclose information

The Contact must exist.

1.1.1.1.1.18 Post-Conditions

The Contact was successfully updated. \textit{EPPResponse} is returned, with the following attributes:

- \textit{<epp:extension>} - \textit{EPPResponse.getExtension()} if response contains extension
- \textit{<epp:trID>} - \textit{EPPResponse.getTransId()}

1.1.1.1.19 Exceptions

\textbf{EPPCommandException} that contains the \textit{EPPResponse} returned from the EPP Server. The \textit{getResponse()} method returns the associated \textit{EPPResponse}. If the exception is thrown before reading the EPP Server response, than \textit{getResponse()} will return \textit{null}.

1.1.1.1.20 Sample Code

Figure 15 - sendUpdate() Sample Code shows the steps of creating \textit{EPPResponse}. If an error occurs, \textit{EPPCommandException} is thrown, and if the error was the result of an EPP Server response, than the response can be retrieved from the exception.
try {

    contact.setTransId("ABC-12345-XYZ");
    contact.addContactId("sh8013");

    // Streets
    Vector streets = new Vector();
    streets.addElement("123 Example Dr.");
    streets.addElement("Suite 100");
    streets.addElement("This is third line");

    // Address
    EPPContactAddress address = new EPPContactAddress(streets,
                                                        "Dulles", "VA", "20166-6503", "US");

    EPPContactPostalDefinition postal = new EPPContactPostalDefinition(
                                                                              "Joe Brown", "Example Corp.",
                                                                              EPPContactPostalDefinition.ATTR_TYPE_LOC, address);

    // statuses
    contact.addStatus(EPPContact.STAT_PENDING_DELETE);
    contact.addPostalInfo(postal);
    contact.setVoicePhone("+1.7035555555");
    contact.setVoiceExt("456");
    contact.setFaxNumber("+1.7035555555");
    contact.setFaxExt("789");
    contact.setAuthorizationId("ClientXYZ");

    // disclose names
    Vector names = new Vector();

    // names.addElement(new
    // EPPContactDiscloseName(EPPContactDiscloseName.ATTR_TYPE_LOC));
    names.addElement(new EPPContactDiscloseName(
                                                        EPPContactDiscloseName.ATTR_TYPE_INT));

    // disclose orgs
    Vector orgs = new Vector();
    orgs.addElement(new EPPContactDiscloseOrg(
                                                        EPPContactDiscloseOrg.ATTR_TYPE_LOC));
    orgs.addElement(new EPPContactDiscloseOrg(
                                                        EPPContactDiscloseOrg.ATTR_TYPE_INT));

    // disclose addresses
    Vector addresses = new Vector();
    addresses.addElement(new EPPContactDiscloseAddress(
                                                        EPPContactDiscloseAddress.ATTR_TYPE_LOC));
    addresses.addElement(new EPPContactDiscloseAddress(
                                                        EPPContactDiscloseAddress.ATTR_TYPE_INT));

    // disclose
    EPPContactDisclose disclose = new EPPContactDisclose();
    disclose.setFlag("0");
    disclose.setNames(names);
    disclose.setOrgs(orgs);
}
disclose.setAddresses(addresses);
disclose.setVoice("");
disclose.setFax("");
disclose.setEmail("");

contact.setDisclose(disclose);

response = contact.sendUpdate();

    // -- Output all of the response attributes
    System.out.println("contactUpdate: Response = [" + response + "]\n\n");
} catch (EPPCommandException e) {
    handleException(e);
}

Figure 15 - sendUpdate() Sample Code
sendDelete() Method

Deletes a Contact.

1.1.1.1.1.21 Pre-Conditions

The following methods must be previously called:

- `addContactId(String)` – Contact Id.

The following methods can be previously called:

- `setTransId(String)` – Client transaction identifier, which is mirrored back in the response.
- `setExtension(EPPCodecComponent)` – Command extension to send with command.

The Contact must exist.

1.1.1.1.1.22 Post-Conditions

The Contact was successfully deleted. `EPPResponse` is returned, with the following attributes:

- `<epp:extension>` - `EPPResponse.getExtension()` if response contains extension
- `<epp:trID>` - `EPPResponse.getTransId()`

1.1.1.1.1.23 Exceptions

`EPPCommandException` that contains the `EPPResponse` returned from the EPP Server. The `getResponse()` method returns the associated `EPPResponse`. If the exception is thrown before reading the EPP Server response, than `getResponse()` will return `null`.

1.1.1.1.1.24 Sample Code

Figure 10 - `sendDelete()` Sample Code shows the steps of creating `EPPResponse`. If an error occurs, `EPPCommandException` is thrown, and if the error was the result of an EPP Server response, than the response can be retrieved from the exception.

```
try {
    contact.setTransId("ABC-12345");
    contact.addContactId("sh8013");
    EPPResponse response = contact.sendDelete();
    System.out.println("Contact Delete success response: " + response);
} catch (EPPCommandException ex) {
    EPPResponse response = (EPPResponse) ex.getResponse();
    if (response != null)
        System.err.println("Contact Delete response error: " + ex + ", response = " + response);
```
else  
    System.err.println("Contact Delete exception: "+ ex);
}

Figure 10 - sendDelete() Sample Code

13.27.3.3.8  sendTransfer() Method
Transfer a Contact.

1.1.1.1.1.25 Pre-Conditions
The following methods must be previously called:

- addContactId(String) – Contact ID.
- setAuthorizationId(String) – Authorization Information. This is required when a transfer is requested.
- setTransferOpCode(String) – Transfer operation as defined by one of the EPPContact.TRANSFER_ constants. For example, EPPContact.TRANSFER_REQUEST.

The following methods can be previously called:

- setTransId(String) – Client transaction identifier, which is mirrored back in the response.
- setExtension(EPPCodecComponent) – Command extension to send with command.

The Contact must exist.

1.1.1.1.1.26 Post-Conditions
The Contact transfer operation was successfully processed. EPPContactTransferResp is returned with the following attributes:

- <epp:extension> - EPPResponse.getExtension() if response contains extension
- <epp:trID> - EPPResponse.getTransId()
- <contact:id> – EPPContactTransferResp.getId()
- <contact:trStatus> – EPPContactTransferResp.getTransferStatus()
- <contact:reID> – EPPContactTransferResp.getRequestClient()
- <contact:reDate> – EPPContactTransferResp.getRequestDate()
- <contact:acID> – EPPContactTransferResp.getActionClient()
- <contact:acDate> – EPPContactTransferResp.getActionDate()
1.1.1.1.27 Exceptions

**EPPCommandException** that contains the **EPPResponse** returned from the EPP Server. The `getResponse()` method returns the associated **EPPResponse**. If the exception is thrown before reading the EPP Server response, than `getResponse()` will return null.

1.1.1.1.28 Sample Code

Figure 11 - `sendTransfer()` Sample Code shows the steps of creating `EPPContactTransferResp`. If an error occurs, **EPPCommandException** is thrown, and if the error was the result of an EPP Server response, than the response can be retrieved from the exception.

```java
try {
    contact.setTransferOpCode(EPPContact.TRANSFER_REQUEST);
    contact.setTransId("ABC-12345-XYZ");
    contact.setAuthorizationId("ClientX");
    contact.addContactId("sh8013");

    // Execute the transfer request
    response = contact.sendTransfer();

    // -- Output all of the response attributes
    System.out.println("contactTransfer: Response = [" + response + "]");

    // -- Output required response attributes using accessors
    System.out.println("contactTransfer: id = " + response.getId());
    System.out.println("contactTransfer: request client = " + response.getRequestClient());
    System.out.println("contactTransfer: action client = " + response.getActionClient());
    System.out.println("contactTransfer: transfer status = " + response.getTransferStatus());
    System.out.println("contactTransfer: request date = " + response.getRequestDate());
    System.out.println("contactTransfer: action date = " + response.getActionDate());
}

catch (EPPCommandException ex) {
    EPPResponse response = (EPPResponse) ex.getResponse();
    if (response != null)
        System.err.println("Contact Transfer response error: " + ex + ", response = " + response);
    else
        System.err.println("Contact Transfer exception: " + ex);
}
```

**Figure 11 - `sendTransfer()` Sample Code**


13.27.4 Registry Mapping

This section is intended to provide users of the Extensible Provisioning Protocol (EPP) Software Development Kit (SDK) with an overview of the Registry Mapping (Proprietary and IETF draft versions) additions to the SDK. This document includes the following Registry information:

1. Definition of the Registry SDK files (i.e. library, schema)
2. Description of the Registry interface classes, including the pre-conditions, the post-conditions, the exceptions, the EPP status codes, and sample code of each of the action methods.

The SDK provides detailed interface information in HTML Javadoc format. This document does not duplicate the detailed interface information contained in the HTML Javadoc. Descriptions are provided of the main Registry interface elements, the pre-conditions, the post-conditions, and example code.

It is assumed that the reader has reviewed the associated EPP specifications and has a general understanding of the EPP concepts. Much of the EPP details are encapsulated in the SDK, but having a solid understanding of the EPP concepts will help in effectively using the SDK.

13.27.4.1 Registry Tests

The Registry source distribution contains one test program for each Registry EPP Mapping the product uses. The tests are located in the suggestion/java directory in com.verisign.epp.interfaces package. The following table describes the test files:

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPRegistryTst.java</td>
<td>This is a sample program demonstrating the use of the EPPRegistry class.</td>
</tr>
<tr>
<td></td>
<td>The com.verisign.epp.interfaces.EPPRegistryTst is used to test the proprietary version of the Registry Mapping.</td>
</tr>
<tr>
<td></td>
<td>The com.verisign.epp.interfaces.v01.EPPRegistryTst is used to test the v01 of the Registry Mapping, which is defined in draft-gould-carney-regext-registry-02 and draft-gould-carney-regext-registry-03.</td>
</tr>
<tr>
<td></td>
<td>The com.verisign.epp.interfaces.v02.EPPRegistryTst is used to test the v02 of the Registry Mapping, which is defined in draft-gould-carney-regext-registry-04.</td>
</tr>
</tbody>
</table>
## 13.27.4.2 Registry Packages

The Registry portion of the Verisign Bundle EPP SDK consists of sub-packages and class additions to existing SDK packages. Figure 12 - Registry Packages provides an overview of the primary SDK packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.registry</td>
<td>Registry Encoder/Decoder package. All of the detail of encoding and decoding the Registry EPP messages are in this package. The EPPRegistryMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name. The <code>com.verisign.epp.codec.registry.EPPRegistryMapFactory</code> must be added to the <code>EPP.MapFactories</code> configuration parameter for the proprietary Registry Mapping. The <code>com.verisign.epp.codec.registry.v01.EPPRegistryMapFactory</code> must be added to the <code>EPP.MapFactories</code> configuration parameter for v01 of the Registry Mapping. The <code>com.verisign.epp.codec.registry.v02.EPPRegistryMapFactory</code> must be added to the <code>EPP.MapFactories</code> configuration parameter for v02 of the Registry Mapping.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the RegistryHandler class used to implement the EPP Registry Stub Server behavior.</td>
</tr>
</tbody>
</table>
The `com.verisign.epp.serverstub.RegistryHandler` must be added to the `EPP.ServerEventHandlers` configuration parameter for the proprietary Registry Mapping.

The `com.verisign.epp.serverstub.registry.v01.RegistryHandler` must be added to the `EPP.ServerEventHandlers` configuration parameter for v01 of the Registry Mapping.

The `com.verisign.epp.serverstub.registry.v02.RegistryHandler` must be added to the `EPP.ServerEventHandlers` configuration parameter for v02 of the Registry Mapping.

This package contains the Registry client interface classes. These classes provide the primary interfaces that map to the commands and objects of the Registry EPP Mapping.

The `com.verisign.epp.interfaces.EPPRegistry` is used for the proprietary Registry Mapping.

The `com.verisign.epp.interfaces.registry.v01.EPPRegistry` is used for v01 of the Registry Mapping.

The `com.verisign.epp.interfaces.registry.v02.EPPRegistry` is used for v02 of the Registry Mapping.

**Figure 12 - Registry Packages**

**13.27.4.3 Registry XML Schema files**

The Registry EPP Mapping is defined using XML schema files. These files are located in the `epp-verisign-bundle-{$BUILD_VER}.jar` in the `schemas` directory. You must un-jar the jar file in order to explicitly view them. The following table gives a brief description of these schema files:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>registry-1.0.xsd</td>
<td>schemas</td>
<td>Proprietary Registry XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.</td>
</tr>
<tr>
<td>registry-0.1.xsd</td>
<td>schemas</td>
<td>V01 of the IETF Registry XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.</td>
</tr>
</tbody>
</table>
Figure 13 - Registry Schema Files

13.27.4.4 Registry Client Interfaces

The Registry portion of the Verisign Bundle EPP SDK contains client interface classes for the Registry EPP Mapping. The interfaces provide mechanisms for querying suggestions. The following sections describe the client interface classes, supporting classes and their respective purposes.

13.27.4.4.1 Registry Interface

The Registry interface, EPPRegistry, is located in the com.verisign.epp.interfaces package or the versioned com.verisign.epp.interfaces.registry.vXX package, where XX is the version of the IETF draft XML schema namespace. This interface is used to query zone information for the system.
The \textit{EPPRegistry} interface has the following relevant methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPRegistry epic</td>
<td>\texttt{EPPRegistry(EPPSession aNewSession)} This is the constructor method and it requires an EPP session object to be passed that has been authenticated (e.g. logged in).</td>
</tr>
<tr>
<td>EPPRegistryCheck Resp</td>
<td>\texttt{sendCheck()} This method checks on the availability / existence of a zone. \textbf{This method is for future use.}</td>
</tr>
<tr>
<td>EPPRegistryInfoResp</td>
<td>\texttt{sendInfo()} This method is used to get registry zone information.</td>
</tr>
<tr>
<td>EPPRegistryCreateResp</td>
<td>\texttt{sendCreate()} This method creates a new zone. \textbf{This method is for future use.}</td>
</tr>
<tr>
<td>EPPResponse</td>
<td>\texttt{sendUpdate()} This method updates a new zone. \textbf{This method is for future use.}</td>
</tr>
<tr>
<td>EPPResponse</td>
<td>\texttt{sendDelete()} This method deletes a zone. \textbf{This method is for future use.}</td>
</tr>
</tbody>
</table>

\textbf{1.1.1.1.29 EPPRegistry() Constructor}

The \textit{EPPRegistry} constructor requires that an authenticated \textit{EPPSession} object be passed upon creation. Once created, the \textit{EPPRegistry} object can perform multiple functions without reinitializing the \textit{EPPSession} object. For example, you can use the same initialized \textit{EPPRegistry} object to call the \texttt{sendInfo()} method multiple times.

\textbf{1.1.1.1.29.1 Pre-Conditions}

An authenticated session has been successfully established with an \textit{EPPSession}.

\textbf{1.1.1.1.29.2 Post-Conditions}

The \textit{EPPRegistry} instance is ready for the execution of one or more operations.

\textbf{1.1.1.1.29.3 Exceptions}

None

\textbf{1.1.1.1.29.4 Sample Code}
The following example shows the steps of initializing an *EPPSession*, then using the *EPPSession* to initialize the *EPPRegistry* interface.

```java
EPPSession session = new EPPSession();
// optional
session.setTransId("ABC-12345-XYZ");
session.setVersion("1.0");
session.setLang("en");
// required
session.setClientID("ClientXID");
session.setPassword("ClientXPass");
try {
    session.initSession();
} catch (EPPCommandException ex) {
    ex.printStackTrace();
    System.exit(1);
}
EPPRegistry registry = new EPPRegistry(session);
```

### 1.1.1.1.30 `sendInfo()` method

The `sendInfo()` method sends the Registry EPP info command to the server. It has the following signature:

```
public EPPRegistryInfoResp sendInfo() throws EPPCommandException
```

#### 1.1.1.1.30.1 Pre-Conditions

**For proprietary version (com.verisign.epp.interfaces package):**

Either `setAllTlds(boolean)` is called with a value of `true`, `addTld(String)` is called, or `setSystem(boolean)` is called with a value of `true`. Both `setAllTlds(true)` and `addTld(String)` cannot be called prior to calling `sendInfo()`.

**For v01 version (com.verisign.epp.interfaces package.v01):**

Use of `setAllZones(boolean)` with a value of `true`, or `addZone(String)` / `addZone(EPPRegistryZoneName)`, or `setSystem(boolean)` with a value of `true`.

**For v02 version (com.verisign.epp.interfaces package.v02):**

Use of `setAllScope(EPPRegistryInfoCmd.Scope)` with a valid scope, or `addZone(String)` / `addZone(EPPRegistryZoneName)`, or `setInfoMode(EPPRegistryInfoCmd.Scope)` with a value of `EPPRegistryInfoCmd.Scope.system`. 
1.1.1.1.30.2 Post-Conditions
On success, an EPPRegistryInfoResp is returned.

1.1.1.1.30.3 Exceptions
An EPPCommandException will be returned that contains the EPPResponse returned from the server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the server response, than getResponse() will return null.

1.1.1.1.30.4 Sample Code
The following example shows the steps of performing a Registry info using the EPPRegistry client interface and the sendInfo() method:

For proprietary version (com.verisign.epp.interfaces package):

```java
EPPRegistryInfoResp response;
try {
    // Get all TLD summary information?
    if (all) {
        registry.setAllTlds(true);
    } else { // Get information for ".tld"
        registry.addTld(".tld");
    }
    response = registry.sendInfo();
}
catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
    e.printStackTrace();
}
```
For v01 version (com.verisign.epp.interfaces package.v01):

```java
EPPRegistryInfoResp response;

try {
  // Get all TLD summary information?
  if (all) {
    registry.setAllZones(true);
  }
  else if (system) {
    registry.setSystem(true);
  }
  else { // Get information for "EXAMPLE"
    registry.addZone("EXAMPLE");
  }
  response = registry.sendInfo();
}
catch (EPPCommandException e) {
  EPPResponse errorResponse = e.getResponse();
  Assert.fail(e.getMessage());
  e.printStackTrace();
}
```

For v02 version (com.verisign.epp.interfaces package.v02):

```java
EPPRegistryInfoResp response;

try {
  // Get all TLD summary information?
  if (all) {
    registry.setAllScope(EPPRegistryInfoCmd.Scope.both);
  }
  else if (system) {
    registry.setInfoMode(EPPRegistryInfoCmd.Mode.system);
  }
  else { // Get information for "EXAMPLE"
    registry.addZone("EXAMPLE");
  }
  response = registry.sendInfo();
}
catch (EPPCommandException e) {
  EPPResponse errorResponse = e.getResponse();
  Assert.fail(e.getMessage());
  e.printStackTrace();
}```
13.27.5 Balance Mapping
This section is intended to provide users of the Extensible Provisioning Protocol (EPP) Software Development Kit (SDK) with an overview of the Balance Mapping additions to the SDK. This document includes the following Balance information:

1. Definition of the Balance SDK files (i.e. library, schema)
2. Description of the Balance interface classes, including the pre-conditions, the post-conditions, the exceptions, the EPP status codes, and sample code of each of the action methods.

The SDK provides detailed interface information in HTML Javadoc format. This document does not duplicate the detailed interface information contained in the HTML Javadoc. Descriptions are provided of the main Balance interface elements, the pre-conditions, the post-conditions, and example code.

It is assumed that the reader has reviewed the associated EPP specifications and has a general understanding of the EPP concepts. Much of the EPP details are encapsulated in the SDK, but having a solid understanding of the EPP concepts will help in effectively using the SDK.

13.27.5.1 Balance Tests
The Verisign source distribution contains one test program for Balance EPP Mapping the product uses. The tests are located in the nsfinance/java directory in com.verisign.epp.interfaces package. The following table describes the test files:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPBalanceTst.java</td>
<td>This is a sample program demonstrating the use of the EPPBalance class.</td>
</tr>
</tbody>
</table>

13.27.5.2 Balance Packages
The Balance portion of the Verisign Bundle EPP SDK consists of sub-packages and class additions to existing SDK packages. Figure 14 - Balance Packages provides an overview of the primary SDK packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.balance</td>
<td>Balance Encoder/Decoder package. All of the detail of encoding and decoding the Balance EPP messages are in this package. The EPPBalanceFactory must be added to the EPP.MapFactories configuration parameter using the full</td>
</tr>
</tbody>
</table>
package and class name.

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.framework</td>
<td>Addition of Balance EPP Server Framework classes used by the Stub Server.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the BalanceHandler class used to implement the EPP Balance Stub Server behavior. This class must be added to the EPP.ServerEventHandlers configuration parameter using the full package and class names.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>This package contains the Balance client interface classes. These classes provide the primary interfaces that map to the commands and objects of the Balance EPP Mapping.</td>
</tr>
</tbody>
</table>

**Figure 14 - Balance Packages**

### 13.27.5.3 Balance XML Schema files

The Balance EPP Mapping is defined using XML schema files. These files are located in the `epp-verisign-bundle-{$BUILD_VER}.jar` in the `schemas` directory. You must un-jar the jar file in order to explicitly view them. The following table gives a brief description of these schema files:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>balance-1.0.xsd</td>
<td>schemas</td>
<td>Balance XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.</td>
</tr>
</tbody>
</table>

**Figure 15 - Balance Schema Files**

### 13.27.5.4 Balance Client Interfaces

The Balance portion of the Verisign Bundle EPP SDK contains client interface classes for the Balance EPP Mapping. The interfaces provide mechanisms for getting the account balance and other financial information. The following sections describe the client interface classes, supporting classes and their respective purposes.

#### 13.27.5.4.1 Balance Interface

The Balance interface, `EPPBalance`, is located in the `com.verisign.epp.interfaces` package. This interface is used to get the account balance and other financial information.
The `EPPBalance` interface has the following relevant methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPBalance(EPPSession aSession)</td>
<td>This is the constructor method and it requires an EPP session object to be passed that has been authenticated (e.g. logged in).</td>
</tr>
<tr>
<td>EPPBalanceInfpResp sendInfo()</td>
<td>This method is used to retrieve the account balance and other financial information.</td>
</tr>
</tbody>
</table>

The method on the `EPPBalance` takes request data that will be sent to the server. The only precondition that exists on the methods of this class is that a valid EPPSession is used to create the instance. There is no other state associated with this class so all data passed as arguments is sent to the server as is. The method will return a response from the Server and will throw an exception if any error occurs. If the exception is associated with an error response from the Server, then the response can be retrieved from the exception with a call to `getResponse()`. The following sections describe and provide sample code for the method and the `EPPBalance` constructor.

1.1.1.1.1.31 `EPPBalance()` Constructor
The `EPPBalance` constructor requires that an authenticated `EPPSession` object be passed upon creation. Once created, the `EPPBalance` object can perform multiple functions without reinitializing the `EPPSession` object. For example, you can use the same initialized `EPPBalance` object to get the balance information with the `sendInfo()` command.

1.1.1.1.1.31.1 Pre-Conditions
An authenticated session has been successfully established with an `EPPSession`.

1.1.1.1.1.31.2 Post-Conditions
The `EPPBalance` instance is ready for the execution of one or more operations.

1.1.1.1.1.31.3 Exceptions
None

1.1.1.1.1.31.4 Sample Code
The following example shows the steps of initializing an `EPPSession`, then using the `EPPSession` to initialize the `EPPBalance` interface.

```java
EPPSession session = new EPPSession();
```
// optional
session.setTransId("ABC-12345-XYZ");
session.setVersion("1.0");
session.setLang("en");

// required
session.setClientId("ClientXID");
session.setPassword("ClientXPass");

try {
    session.initSession();
}
catch (EPPCommandException ex) {
    ex.printStackTrace();
    System.exit(1);
}

EPPBalance balance = new EPPBalance(session);

1.1.1.1.32 sendInfo() method
The sendInfo() method sends the Balance EPP info command to the Server. It has the following signature:

public EPPBalanceInfoResp sendInfo() throws EPPCommandException

1.1.1.1.32.1 Pre-Conditions

1. The client transaction id should be set by calling setTransId(String aTransId) method.

1.1.1.1.32.2 Post-Conditions
On success, an EPPBalanceInfoResp is returned.

1.1.1.1.32.3 Exceptions
An EPPCommandException will be returned that contains the EPPResponse returned from the Server. The getResponse() method returns the associated EPPResponse. If the exception is thrown before reading the Server response, than getResponse() will return null.

1.1.1.1.32.4 Sample Code
The following example shows the steps of performing a Balance info using the EPPBalance client interface and the sendInfo() method:
EPPBalanceInfoResp response;

try {
    EPPBalance balance = new EPPBalance(session);

    balance.setTransId("ABC-12345");
    response = balance.sendInfo();

    System.out.println( "Credit Limit: " +
                        response.getCreditLimit() );

    System.out.println( "Balance: " +
                        response.getBalance() );

    System.out.println( "Available Credit: " +
                        response.getAvailableCredit() );

    System.out.println( "Credit Threshold (type, value): " +
                        response.getCreditThreshold().getType() + "," +
                        response.getCreditThreshold().getValue());
} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
13.27.6   EmailFwd Mapping
To be filled in. Please refer to the JavaDoc for detail on the EmailFwd Mapping support.
13.27.7 DefReg Mapping

To be filled in. Please refer to the JavaDoc for detail on the DefReg Mapping support.
13.27.8 **NameWatch Mapping**

To be filled in. Please refer to the JavaDoc for detail on the NameWatch Mapping support.
13.27.9  IDN Table Mapping
This section is intended to provide users of the Extensible Provisioning Protocol (EPP) Software Development Kit (SDK) with an overview of the Internationalized Domain Name (IDN) Table Mapping additions to the SDK. This document includes the following IDN Table information:

3. Definition of the IDN Table SDK files (i.e. library, schema)
4. Description of the IDN Table interface classes, including the pre-conditions, the post-conditions, the exceptions, the EPP status codes, and sample code of each of the action methods.

The SDK provides detailed interface information in HTML Javadoc format. This document does not duplicate the detailed interface information contained in the HTML Javadoc. Descriptions are provided of the main IDN Table interface elements, the pre-conditions, the post-conditions, and example code.

It is assumed that the reader has reviewed the associated EPP specifications and has a general understanding of the EPP concepts. Much of the EPP details are encapsulated in the SDK, but having a solid understanding of the EPP concepts will help in effectively using the SDK.

13.27.9.1 IDN Table Tests
The Verisign source distribution contains one test program for the IDN Table EPP Mapping. The tests are located in the idntable/java directory in com.verisign.epp.interfaces package. The following table describes the test files:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPIdnTableTst.java</td>
<td>This is a sample program demonstrating the use of the EPPIdnTable class.</td>
</tr>
</tbody>
</table>

13.27.9.2 IDN Table Packages
The IDN Table portion of the Verisign Bundle EPP SDK consists of sub-packages and class additions to existing SDK packages. Figure 14 - Balance Packages provides an overview of the primary SDK packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.idntable</td>
<td>IDN Table Encoder/Decoder package. All of the detail of encoding and decoding the IDN Table EPP messages are in this package. The EPPIdnTableFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
</tbody>
</table>
### com.verisign.epp.framework
Addition of IDN Table EPP Server Framework classes used by the Stub Server.

### com.verisign.epp.serverstub
Addition of the `IdnTableHandler` class used to implement the EPP IDN Table Stub Server behavior. This class must be added to the EPP.ServerEventHandlers configuration parameter using the full package and class names.

### com.verisign.epp.interfaces
This package contains the IDN Table client interface classes. These classes provide the primary interfaces that map to the commands and objects of the IDN Table EPP Mapping.

#### Figure 16 – IDN Table Packages

#### 13.27.9.3 IDN Table XML Schema files
The IDN Table EPP Mapping is defined using XML schema files. These files are located in the `epp-verisign-bundle-{BUILD_VER}.jar` in the `schemas` directory. You must un-jar the jar file in order to explicitly view them. The following table gives a brief description of these schema files:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idnTable-1.0.xsd</td>
<td>schemas</td>
<td>IDN Table XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.</td>
</tr>
</tbody>
</table>

#### Figure 17 – IDN Table Schema Files

#### 13.27.9.4 IDN Table Client Interfaces
The IDN Table portion of the Verisign Bundle EPP SDK contains client interface classes for the IDN Table EPP Mapping. The interfaces provide mechanisms for getting the IDN Table information. The following sections describe the client interface classes, supporting classes and their respective purposes.

##### 13.27.9.4.1 IDN Table Interface

The IDN Table interface, `EPPIdnTable`, is located in the `com.verisign.epp.interfaces` package. This interface is used to get the IDN Table information utilizing the forms defined in the IDN Table EPP Mapping.
The *EPPIdnTable* interface has the following relevant methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPIdnTable(aSession)</td>
<td>This is the constructor method and it requires an EPP session object to be passed that has been authenticated (e.g. logged in).</td>
</tr>
<tr>
<td>EPPIdnTableCheckResp</td>
<td>sendDomainCheck()</td>
</tr>
<tr>
<td></td>
<td>Sends an IDN Table Check Command in Domain Check Form.</td>
</tr>
<tr>
<td></td>
<td>Requires at least one domain name set with the <em>addDomain(String)</em> method.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <em>setTransId(String)</em> method.</td>
</tr>
<tr>
<td>EPPIdnTableCheckResp</td>
<td>sendTableCheck()</td>
</tr>
<tr>
<td></td>
<td>Sends an IDN Table Check Command in Table Check Form.</td>
</tr>
<tr>
<td></td>
<td>Requires at least one table identifier set with the <em>addTable(String)</em> method.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <em>setTransId(String)</em> method.</td>
</tr>
<tr>
<td>EPPIdnTableResp</td>
<td>sendDomainInfo()</td>
</tr>
<tr>
<td></td>
<td>Sends an IDN Table Info Command in Domain Info Form.</td>
</tr>
<tr>
<td></td>
<td>Requires one domain name set with the <em>addDomain(String)</em> method.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <em>setTransId(String)</em> method.</td>
</tr>
<tr>
<td>EPPIdnTableResp</td>
<td>sendTableInfo()</td>
</tr>
<tr>
<td></td>
<td>Sends an IDN Table Info Command in Table Info Form.</td>
</tr>
<tr>
<td></td>
<td>Requires one table identifier set with the <em>addTable(String)</em> method.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <em>setTransId(String)</em> method.</td>
</tr>
<tr>
<td>EPPIdnTableResp</td>
<td>sendListInfo()</td>
</tr>
<tr>
<td></td>
<td>Sends an IDN Table Info Command in List Info Form.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <em>setTransId(String)</em> method.</td>
</tr>
</tbody>
</table>
The methods on the `EPPIdnTable` takes request data that will be sent to the server. The methods will return a response from the Server and will throw an exception if any error occurs. If the exception is associated with an error response from the Server, then the response can be retrieved from the exception with a call to `getResponse()`. The following sections describe and provide sample code for the method and the `EPPIdnTable` constructor.

1.1.1.1.33 `EPPIdnTable ()` Constructor

The `EPPIdnTable` constructor requires that an authenticated `EPPSession` object be passed upon creation. Once created, the `EPPIdnTable` object can perform multiple functions without reinitializing the `EPPSession` object. For example, you can use the same initialized `EPPIdnTable` object to get the IDN Table information with any of the `send()` methods.

1.1.1.1.33.1 Pre-Conditions

An authenticated session has been successfully established with an `EPPSession`.

1.1.1.1.33.2 Post-Conditions

The `EPPIdnTable` instance is ready for the execution of one or more operations.

1.1.1.1.33.3 Exceptions

None

1.1.1.1.33.4 Sample Code

The following example shows the steps of initializing an `EPPSession`, then using the `EPPSession` to initialize the `EPPIdnTable` interface.

```java
EPPSession session = new EPPSession();

// optional
session.setTransId("ABC-12345-XYZ");
session.setVersion("1.0");
session.setLang("en");

// required
session.setClientID("ClientXID");
session.setPassword("ClientXPass");

try {
    session.initSession();
} catch (EPPCommandException ex) { 
```
ex.printStackTrace();
System.exit(1);

EPPIdnTable idnTable = new EPPIdnTable(session);
13.27.10  China Name Verification Mapping

This section is intended to provide users of the Extensible Provisioning Protocol (EPP) Software Development Kit (SDK) with an overview of the China Name Verification Mapping additions to the SDK. This document includes the following China Name Verification information:

1. Definition of the China Name Verification SDK files (i.e. library, schema)
2. Description of the China Name Verification interface classes, including the pre-conditions, the post-conditions, the exceptions, the EPP status codes, and sample code of each of the action methods.

The SDK provides detailed interface information in HTML Javadoc format. This document does not duplicate the detailed interface information contained in the HTML Javadoc. Descriptions are provided of the main China Name Verification interface elements, the pre-conditions, the post-conditions, and example code.

It is assumed that the reader has reviewed the associated EPP specifications and has a general understanding of the EPP concepts. Much of the EPP details are encapsulated in the SDK, but having a solid understanding of the EPP concepts will help in effectively using the SDK.

13.27.10.1 China Name Verification Tests

The Verisign source distribution contains one test program for the China Name Verification EPP Mapping. The tests are located in the vsp/java directory in com.verisign.epp.interfaces package. The following table describes the test files:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPNameVerificationTst.java</td>
<td>This is a sample program demonstrating the use of the EPPNameVerification class.</td>
</tr>
</tbody>
</table>

13.27.10.2 China Name Verification Packages

The China Name Verification portion of the Verisign Bundle EPP SDK consists of sub-packages and class additions to existing SDK packages. Figure 14 - Balance Packages provides an overview of the primary SDK packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.nv</td>
<td>China Name Verification Encoder/Decoder package. All of the detail of encoding and decoding the China Name Verification EPP messages are in this package. The EPPNameVerificationMapFactory must be added to the EPP.MapFactories configuration parameter using the</td>
</tr>
</tbody>
</table>
Addition of China Name Verification EPP Server Framework classes used by the Stub Server.

Addition of the NameVerificationHandler class used to implement the EPP China Name Verification Stub Server behavior. This class must be added to the EPP.ServerEventHandlers configuration parameter using the full package and class names.

This package contains the China Name Verification client interface classes. These classes provide the primary interfaces that map to the commands and objects of the China Name Verification EPP Mapping.

**Figure 18 – China Name Verification Packages**

### 13.27.10.3 China Name Verification XML Schema files

The China Name Verification EPP Mapping is defined using XML schema files. These files are located in the epp-verisign-bundle-\$BUILD_VER}.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them. The following table gives a brief description of these schema files:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nv-1.0.xsd</td>
<td>schemas</td>
<td>China Name Verification XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.</td>
</tr>
</tbody>
</table>

**Figure 19 – China Name Verification Schema Files**

### 13.27.10.4 China Name Verification Client Interfaces

The China Name Verification portion of the Verisign Bundle EPP SDK contains client interface classes for the China Name Verification EPP Mapping. The interfaces provide mechanisms for sending China Name Verification transform and query commands. The following sections describe the client interface classes, supporting classes and their respective purposes.

#### 13.27.10.4.1 China Name Verification Interface

The China Name Verification interface, EPPNameVerification, is located in the com.verisign.epp.interfaces package. This interface is used to get the China Name Verification information utilizing the forms defined in the China Name Verification EPP Mapping.
The `EPPNameVerification` interface has the following relevant methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPNameVerification(EPPSession aSession)</td>
<td>This is the constructor method and it requires an EPP session object to be passed that has been authenticated (e.g. logged in).</td>
</tr>
<tr>
<td>EPPNameVerificationChe ckResp sendCheck()</td>
<td>Sends a Name Verification Check Command to the server.</td>
</tr>
<tr>
<td></td>
<td>Requires at least one domain label set with the <code>addLabel(String)</code> method.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <code>setTransId(String)</code> method.</td>
</tr>
<tr>
<td>EPPNameVerificationInf oResp sendInfo()</td>
<td>Sends a Name Verification Info Command to the server.</td>
</tr>
<tr>
<td></td>
<td>Requires the verification code with the <code>setCode(String)</code> method and the info type with the <code>setType(EPPNameVerificationInfoCmd.Type)</code> method.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <code>setTransId(String)</code> method or set the authorization information with the <code>setAuthInfo(String)</code> method.</td>
</tr>
<tr>
<td>EPPNameVerificationCreateResp sendCreate()</td>
<td>Sends the Name Verification Create Command for either a Domain Name Verification (DNV) or Real Name Verification (RNV) object.</td>
</tr>
<tr>
<td></td>
<td>Requires either the Domain Name Verification (DNV) information to be set with <code>setDnv(EPPDomainNameVerification)</code> or the Real Name Verification (RNV) information to be set with <code>setRnv(EPPRealNameVerification)</code>, along with the authorization information with <code>setAuthInfo(String)</code>.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <code>setTransId(String)</code> method.</td>
</tr>
<tr>
<td>EPPResponse sendUpdate()</td>
<td>Sends the Name Verification Update Command to update the authorization information for a Name Verification object.</td>
</tr>
<tr>
<td></td>
<td>Requires the verification code with the <code>setCode(String)</code> method and</td>
</tr>
</tbody>
</table>
the new authorization information with the `setAuthInfo(String)` method.

Optionally the client transaction identifier set with the `setTransId(String)` method.

The methods on the `EPPNameVerification` takes request data that will be sent to the server. The methods will return a response from the Server and will throw an exception if any error occurs. If the exception is associated with an error response from the Server, then the response can be retrieved from the exception with a call to `getResponse()`. The following sections describe and provide sample code for the method and the `EPPNameVerification` constructor.

1.1.1.1.34.1 `EPPNameVerification()` Constructor

The `EPPNameVerification` constructor requires that an authenticated `EPPSession` object be passed upon creation. Once created, the `EPPNameVerification` object can perform multiple functions without reinitializing the `EPPSession` object. For example, you can use the same initialized `EPPNameVerification` object with the `send()` methods.

1.1.1.1.34.1 Pre-Conditions

An authenticated session has been successfully established with an `EPPSession`.

1.1.1.1.34.2 Post-Conditions

The `EPPNameVerification` instance is ready for the execution of one or more operations.

1.1.1.1.34.3 Exceptions

None

1.1.1.1.34.4 Sample Code

The following example shows the steps of initializing an `EPPSession`, then using the `EPPSession` to initialize the `EPPNameVerification` interface.

```java
EPPSession session = new EPPSession();
// optional
session.setTransId("ABC-12345-XYZ");
session.setVersion("1.0");
session.setLang("en");
// required
session.setClientID("ClientXID");
session.setPassword("ClientXPass");
```
try {
    session.initSession();
}

catch (EPPCommandException ex) {
    ex.printStackTrace();
    System.exit(1);
}

EPPNameVerification nameVerification = new EPPNameVerification(session);
13.27.11 Organization Mapping
This section is intended to provide users of the Extensible Provisioning Protocol (EPP) Software Development Kit (SDK) with an overview of the Organization Mapping additions to the SDK. This document includes the following Organization information:

3. Definition of the Organization SDK files (i.e. library, schema)
4. Description of the Organization interface classes, including the pre-conditions, the post-conditions, the exceptions, the EPP status codes, and sample code of each of the action methods.

The SDK provides detailed interface information in HTML Javadoc format. This document does not duplicate the detailed interface information contained in the HTML Javadoc. Descriptions are provided of the main Organization interface elements, the pre-conditions, the post-conditions, and example code.

It is assumed that the reader has reviewed the associated EPP specifications and has a general understanding of the EPP concepts. Much of the EPP details are encapsulated in the SDK, but having a solid understanding of the EPP concepts will help in effectively using the SDK.

13.27.11.1 Organization Tests
The Verisign source distribution contains one test program for the Organization EPP Mapping. The tests are located in the vsp/java directory in com.verisign.epp.interfaces package. The following table describes the test files:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPOrgTst.java</td>
<td>This is a sample program demonstrating the use of the EPPOrg class.</td>
</tr>
</tbody>
</table>

13.27.11.2 Organization Packages
The Organization portion of the Verisign Bundle EPP SDK consists of sub-packages and class additions to existing SDK packages. Figure 14 - Balance Packages provides an overview of the primary SDK packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.org</td>
<td>Organization Encoder/Decoder package. All of the detail of encoding and decoding the Organization EPP messages are in this package. The EPPOrgMapFactory must be added to the EPP.MapFactories configuration parameter using the full</td>
</tr>
</tbody>
</table>
package and class name.

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the OrgHandler class used to implement the EPP Organization Stub Server behavior. This class must be added to the EPP.ServerEventHandlers configuration parameter using the full package and class names.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>This package contains the Organization client interface classes. These classes provide the primary interfaces that map to the commands and objects of the Organization EPP Mapping.</td>
</tr>
</tbody>
</table>

**Figure 20 – Organization Packages**

**13.27.11.3 Organization XML Schema files**

The Organization EPP Mapping is defined using XML schema files. These files are located in the epp-verisign-bundle-{$BUILD_VER}.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them. The following table gives a brief description of these schema files:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>org-1.0.xsd</td>
<td>schemas</td>
<td>Organization Mapping XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.</td>
</tr>
</tbody>
</table>

**Figure 21 – Organization Schema Files**

**13.27.11.4 Organization Client Interfaces**

The Organization portion of the Verisign Bundle EPP SDK contains client interface classes for the Organization EPP Mapping. The interface provides a mechanism for sending Organization transform and query commands. The following sections describe the client interface classes, supporting classes and their respective purposes.

**13.27.11.4.1 Organization Interface**

The Organization interface, EPPOrg, is located in the com.verisign.epp.interfaces package. This interface is used to get the send Organization transform and query commands defined in the Organization EPP Mapping.
The *EPPOrg* interface has the following relevant methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EPPOrg(EPPSession aSession)</strong></td>
<td>This is the constructor method and it requires an EPP session object to be passed that has been authenticated (e.g. logged in).</td>
</tr>
<tr>
<td><strong>EPPOrgCheckResp</strong></td>
<td>sendCheck()</td>
</tr>
<tr>
<td></td>
<td>Sends a Organization Check Command to the server.</td>
</tr>
<tr>
<td></td>
<td>Requires at least one organization identifier set with the <code>addOrgId(String)</code> method.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <code>setTransId(String)</code> method.</td>
</tr>
<tr>
<td><strong>EPPOrganizationInfoResp</strong></td>
<td>sendInfo()</td>
</tr>
<tr>
<td></td>
<td>Sends a Organization Info Command to the server.</td>
</tr>
<tr>
<td></td>
<td>Requires the organization identifier with the <code>addOrgId(String)</code> method.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <code>setTransId(String)</code> method.</td>
</tr>
<tr>
<td><strong>EPPOrganizationCreateResp</strong></td>
<td>sendCreate()</td>
</tr>
<tr>
<td></td>
<td>Sends the Organization Create Command for a organization object.</td>
</tr>
<tr>
<td></td>
<td>Requires the organization identifier to be set with <code>addOrgId(String)</code> and at least one role be set with <code>addRole(EPPOrgRole)</code>.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <code>setTransId(String)</code> method along with the other reseller attributes.</td>
</tr>
<tr>
<td><strong>EPPResponse</strong></td>
<td>sendDelete()</td>
</tr>
<tr>
<td></td>
<td>Sends the Organization Delete Command to delete the organization object.</td>
</tr>
<tr>
<td></td>
<td>Requires the organization identifier to be set with <code>addOrgId(String)</code>.</td>
</tr>
<tr>
<td></td>
<td>Optionally the client transaction identifier set with the <code>setTransId(String)</code> method along with the other reseller attributes.</td>
</tr>
</tbody>
</table>
**EPPResponse**

**sendUpdate()**

Sends the Organization Update Command to update the organization object.

Requires the organization identifier to be set with `addOrgId(String)`.

Optionally the client transaction identifier set with the `setTransId(String)` method along with the other reseller attributes.

The methods on the `EPPOrg` takes request data that will be sent to the server. The methods will return a response from the Server and will throw an exception if any error occurs. If the exception is associated with an error response from the Server, then the response can be retrieved from the exception with a call to `getResponse()`. The following sections describe and provide sample code for the method and the `EPPOrg` constructor.

1.1.1.1.1.35 `EPPOrg()` Constructor

The `EPPOrg` constructor requires that an authenticated `EPPSession` object be passed upon creation. Once created, the `EPPOrg` object can perform multiple functions without reinitializing the `EPPSession` object. For example, you can use the same initialized `EPPOrg` object to send any of the Organization commands with the `send()` methods.

1.1.1.1.1.35.1 Pre-Conditions

An authenticated session has been successfully established with an `EPPSession`.

1.1.1.1.1.35.2 Post-Conditions

The `EPPOrganization` instance is ready for the execution of one or more operations.

1.1.1.1.1.35.3 Exceptions

None

1.1.1.1.1.35.4 Sample Code

The following example shows the steps of initializing an `EPPSession`, then using the `EPPSession` to initialize the `EPPOrganization` interface.

```java
EPPSession session = new EPPSession();
// optional
session.setTransId("ABC-12345-XYZ");
session.setVersion("1.0");
session.setLang("en");
```
// required
session.setClientID("ClientXID");
session.setPassphrase("ClientXPass");

try {
    session.initSession();
}
catch (EPPCommandException ex) {
    ex.printStackTrace();
    System.exit(1);
}

EPPOrg org = new EPPOrg(session);
Extensions

13.27.12 NamestoreExt Extension

The NamestoreExt is used for command routing to a logical service or registry using a sub-product property that is referred to as the SubProductID. Both Namestore (CTLD) and the SRS require the use of the NameStoreExt to uniquely identify the target registry or service. For example, when a host is created, the NamestoreExt element will specify the particular registry at which the host is created. Currently, the convention used for the SubProductID is to use the A-Label TLD, as in “CC” for the .CC TLD or “COM” for the .COM TLD.

13.27.12.1 EPPNamestoreExtNamestoreExt

The EPPNamestoreExtNamestoreExt class is used with the addExtension() method for specifying the subProduct. The constructor or the setSubProductID(java.lang.String) should be used for setting the SubProductID value.

Please refer to the following sections for sample code demonstrating the use of the EPPNamestoreExtNamestoreExt class.
13.27.13 **Whois Info Extension**

The Whois Info Extension defined in “Extensible Provisioning Protocol Extension Mapping: Whois Info” is used to get additional domain information that is provided in the Whois Server including the following attributes:

1. Registrar Name – Full name of the sponsoring Registrar
2. Whois Server – Whois Server of the sponsoring Registrar
3. Referral URL – Referral URL of the sponsoring Registrar
4. IRIS Server – IRIS Server of the sponsoring Registrar

To specify that the additional information is desired, either an `com.verisign.epp.codec.whois.EPPWhoisInf` instance needs to be added to the `com.verisign.epp.interfaces.EPPDomain` via the `addExtension(EPPCodecComponent)` method or via the `com.verisign.epp.interfaces.NSDomain.setWhoisInf(boolean)` method. If the flag is set to `true`, then the `com.verisign.epp.codec.whois.EPPWhoisInfData` instance can be retrieved from the returned `com.verisign.epp.codec.domain.EPPDomainInfoResp` via the `getExtension(Class)` method.
SecDNS Extension

The SecDNS Extension is used for provisioning and management of DNS security extensions in a shared central repository. This extension defines additional elements for EPP <create>, <update> commands and also for the EPP <info> response. There are two versions of the SecDNS Extension supported that include:

1. secDNS-1.0 – This is the term that refers to “RFC 4310 – EPP DNS Security Extensions Mapping”. The classes contained in the com.verisign.epp.codec.secdnsext package were moved to the com.verisign.epp.codec.secdnsext.v10 package to support more than one version of the SecDNS Extension.

2. secDNS-1.1 – This is the term that refers to “RFC 5910 – EPP DNS Security Extensions Mapping” that deprecates “RFC 4310 – EPP DNS Security Extensions Mapping”. It is recommended that secDNS-1.1 be used. The secDNS-1.1 classes are contained in the com.verisign.epp.codec.secdnsext.v11 package. There are some fundamental changes included in secDNS-1.1 that should be reviewed if the client is migrating from secDNS-1.0.

There are two approaches to setting the SecDNS Extension with a domain create or update command. The first approach sets the com.verisign.epp.codec.secdnsext.v10.EPPSecDNSExtCreate or com.verisign.epp.codec.secdnsext.v10.EPPSecDNSExtUpdate instances for secDNS-1.0 or sets the com.verisign.epp.codec.secdnsext.v11.EPPSecDNSExtCreate or com.verisign.epp.codec.secdnsext.v11.EPPSecDNSExtUpdate instances for secDNS-1.1 with the com.verisign.epp.interfaces.EPPDomain.addExtension(EPPCodecComponent) method. The second approach is to use the SecDNS Extension convenience methods with the com.verisign.epp.interfaces.NSDomain interface described below.

Some of the com.verisign.epp.namestore.interfaces.NSDomain methods support both secDNS1.0 and secDNS-1.1 by using reflection of the first element contained in the passed in List parameter. The List parameter contains com.verisign.epp.codec.secdnsext.v10.EPPSecDNSExtDsData instances for secDNS-1.0 and com.verisign.epp.codec.secdnsext.v11.EPPSecDNSExtDsData instances for secDNS-1.1. The exception to this is the setSecDNSUpdateForRem(List, Boolean) method where the List parameter contains Integer instances for secDNS-1.0 and com.verisign.epp.codec.secdnsext.v11.EPPSecDNSExtDsData instances for secDNS-1.1. The methods include the following:

1. setSecDNSCreate(List aDsData) – Set the DS to include with the sendCreate().

2. setSecDNSUpdateForAdd(List aAddDsData, boolean aUrgent) – Set the DS to add along with the urgent flag value. For secDNS-1.1 it is recommended to use
the `setSecDNSUpdate(List aAddDsData, List aRemDsData)` method instead. The `setSecDNSUpdateForAdd(List aAddDsData, boolean aUrgent)` method cannot be used in combination with any other `setSecDNSUpdate` methods.

3. `setSecDNSUpdateForRem(List aRemDsData, boolean aUrgent)` – Set the DS to remove along with the urgent flag value. For secDNS-1.1 it is recommended to use the `setSecDNSUpdate(List aAddDsData, List aRemDsData)` method instead. The `setSecDNSUpdateForRem(List aRemDsData, boolean aUrgent)` method cannot be used in combination with any other `setSecDNSUpdate` methods.

One method that only supports secDNS-1.1 is the following:

1. `setSecDNSUpdate(List aAddDsData, List aRemDsData)` – The method can be used to add, remove, remove all, and replace all DS. The constant `NSDomain.REM_ALL_DS` can be used for the `aRemDsData` parameter to remove all DS and to replace all DS by also including a non-null, non-empty `aAddDsData` parameter. `setSecDNSUpdate(List aAddDsData, List aRemDsData)` method cannot be used in combination with any other `setSecDNSUpdate` methods. The `aUrgent` parameter is not included with this method since the Verisign servers do not support setting the urgent flag to true.

One method that only supports secDNS-1.0 is the following:

1. `setSecDNSUpdateForChg(List aChgDsData, boolean aUrgent)` – This method replaces all of the DS according to secDNS-1.0. There is a different approach taken for secDNS-1.1 to replace all of the DS, so the use of the `<secDNS:chg>` with a list of DS is specific to secDNS-1.0.

### 13.27.14.1 Sample Code

The following is a sample of setting DS data on a domain create using secDNS-1.0.

```java
NSDomain theDomain = new NSDomain(session);
theDomain.addDomainName("example.com");
theDomain.setSubProductID(NSSubProduct.COM);
theDomain.setAuthString("ClientX");

// Add DS
List dsDataList = new ArrayList();
dsDataList.add(new com.verisign.epp.codec.secdnsext.v10.EPPSecDNSExtDsData(12345,
    EPPSecDNSAlgorithm.DSA,
    EPPSecDNSExtDsData.SHA1_DIGEST_TYPE,
    "49FD46E6C4B45C55D4AC");
theDomain.setSecDNSCreate(dsDataList);
```
EPPDomainCreateResp theResponse = theDomain.sendCreate();

The following is a sample of setting DS data on a domain create using secDNS-1.1.

```java
NSDomain theDomain = new NSDomain(session);
theDomain.addDomainName("example.com");
theDomain.setSubProductID(NSSubProduct.COM);
theDomain.setAuthString("ClientX");

// Add DS
List dsDataList = new ArrayList();
dsDataList.add(new com.verisign.epp.codec.secdnsext.v11.EPPSecDNSExtDsData(12345,
    EPPSecDNSAlgorithm.DSA, 
    EPPSecDNSExtDsData.SHA1_DIGEST_TYPE, 
    "49FD46E6C4B45C55D4AC"));
theDomain.setSecDNSCreate(dsDataList);
EPPDomainCreateResp theResponse = theDomain.sendCreate();
```

The following is a sample of adding DS data on a domain update using secDNS-1.0.

```java
NSDomain theDomain = new NSDomain(session);
theDomain.addDomainName("example.com");
theDomain.setSubProductID(NSSubProduct.COM);

// Add DS
List dsDataList = new ArrayList();
dsDataList.add(new com.verisign.epp.codec.secdnsext.v10.EPPSecDNSExtDsData(12345,
    EPPSecDNSAlgorithm.DSA, 
    EPPSecDNSExtDsData.SHA1_DIGEST_TYPE, 
    "49FD46E6C4B45C55D4AC"));
theDomain.setSecDNSUpdateForAdd(dsDataList);
```

The following is a sample of adding and removing DS data on a domain update using secDNS-1.1.

```java
NSDomain theDomain = new NSDomain(session);
theDomain.addDomainName("example.com");
theDomain.setSubProductID(NSSubProduct.COM);

// Add DS
List addDsDataList = new ArrayList();
addDsDataList.add(new com.verisign.epp.codec.secdnsext.v11.EPPSecDNSExtDsData(12345,
    EPPSecDNSAlgorithm.DSA, 
    EPPSecDNSExtDsData.SHA1_DIGEST_TYPE, 
    "49FD46E6C4B45C55D4AC"));
```
List remDsDataList = new ArrayList();
remDsDataList.add(new
\texttt{com.verisign.epp.codec.secdnsext.v11.EPPSecDNSExtDsData}(12345,
\texttt{EPPSecDNSAlgorithm.DSA},
\texttt{EPPSecDNSExtDsData.SHA1_DIGEST_TYPE},
"38EC35D5B3A34B44C39B "));
theDomain.setSecDNSUpdate(addDsDataList, remDsDataList);

\texttt{EPPDomainCreateResp theResponse = theDomain.sendUpdate();}

The following is a sample of replacing all DS data by removing all and then adding a list
of new DS using secDNS-1.1.

\texttt{NSDomain theDomain = new NSDomain(session);}
\texttt{theDomain.addDomainName("example.com");}
\texttt{theDomain.setSubProductID(NSSubProduct.COM);}

\texttt{// Add DS}
\texttt{List addDsDataList = new ArrayList();}
\texttt{addDsDataList.add(new
\texttt{com.verisign.epp.codec.secdnsext.v11.EPPSecDNSExtDsData}(12345,
\texttt{EPPSecDNSAlgorithm.DSA},
\texttt{EPPSecDNSExtDsData.SHA1_DIGEST_TYPE},
"49FD46E6C4B45C55D4AC"));}
\texttt{theDomain.setSecDNSUpdate(addDsDataList, \texttt{NSDomain.REM_ALL_DS});}

\texttt{EPPDomainCreateResp theResponse = theDomain.sendUpdate();}
13.27.15 COA Extension

The COA Extension is used for provisioning and management of Client Object Attribute extensions in a shared central repository. This extension defines additional elements for EPP <create>, <update> commands and also for the EPP <info> response.

There are two approaches to setting the COA Extension with a domain create or update command. The first approach sets the `com.verisign.epp.codec.coaext.EPPCoaExtCreate` or `com.verisign.epp.codec.coaext.EPPCoaExtUpdate` instances with the `com.verisign.epp.interfaces.EPPDomain.addExtension(EPPCodecComponent)` method. The second approach is to use the COA Extension convenience methods with the `com.verisign.epp.interfaces.NSDomain` interface described here.

1. `setCoaCreate(List aAttrs)` – Set the COAs to include with the `sendCreate()`.
   
   Code sample:
   ```java
   NSDomain theDomain = new NSDomain(session);
   theDomain.addDomainName("example.com");
   theDomain.setSubProductID(NSSubProduct.COM);
   theDomain.setAuthString("ClientX");
   
   // Client Object Attributes to be added
   EPPCoaExtAttr attr = new EPPCoaExtAttr("KEY1", "value1");
   List attrList = new ArrayList();
   attrList.add(attr);
   theDomain.setCoaCreate(attrList);
   
   EPPDomainCreateResp theResponse = theDomain.sendCreate();
   ```

2. `setCoaUpdateForPut(List aAttrs)` – Set the list of key / value pairs specifying the COAs to add or update.
   
   Code sample:
   ```java
   NSDomain theDomain = new NSDomain(session);
   theDomain.addDomainName("example.com");
   theDomain.setSubProductID(NSSubProduct.COM);
   theDomain.setAuthString("ClientX");
   
   // Client Object Attributes to be added
   EPPCoaExtAttr attr = new EPPCoaExtAttr("KEY1", "value1");
   List attrList = new ArrayList();
   attrList.add(attr);
   theDomain.setCoaUpdateForPut(attrList);
   
   EPPDomainCreateResp theResponse = theDomain.sendUpdate();
   ```
3. `setCoaUpdateForRem(List aKeys)` – Set the list of keys identifying which existing COAs to remove.

Code sample:

```java
NSDomain theDomain = new NSDomain(session);
theDomain.addDomainName("example.com");
theDomain.setSubProductID(NSSubProduct.COM);
theDomain.setAuthString("ClientX");

// Client Object Attributes to be removed
EPPCoaExtKey key = new EPPCoaExtKey("KEY1");
List attrList = new ArrayList();
attrList.add(key);
theDomain.setCoaUpdateForRem(attrList);

EPPDomainCreateResp theResponse = theDomain.sendUpdate();
```
13.27.16 Premium Domain Extension

The Premium Domain Extension defined in "Extensible Provisioning Protocol Extension Mapping: Premium Domain extension" is used to support premium features. This extension defines additional elements for EPP <check>, <update> commands and for the EPP <check> response.

Adding Premium Domain Extension to the EPP <check> command allows a client to retrieve premium information for a domain.

For this, a `com.verisign.epp.codec.premiumdomain.EPPPremiumDomainCheck(boolean)` instance needs to added to the `com.verisign.epp.interfaces.EPPDomain` via the `addExtension(EPPCodecComponent)` method. If the flag is set to true, then the `com.verisign.epp.codec.premiumdomain.EPPPremiumDomainCheckResp` instance can be retrieved from the returned `com.verisign.epp.codec.domain.EPPDomainCheckResp` via the `getExtension(Class)` method. `EPPPremiumDomainCheckResp.getCheckResults` returns the list of `EPPPremiumDomainCheckResult` instances. `EPPPremiumDomainCheckResult` holds the premium information.

Also, adding reassign premium domain Extension to the EPP <update> command allows a client to reassign a domain name to another registrar. For this, set the shortName of a registrar to whom this domain name needs to be reassigned using `setShortName(String)` method of `com.verisign.epp.codec.premiumdomain.EPPPremiumDomainReAssignCmd` instance. Send Premium Domain extension by adding `com.verisign.epp.codec.premiumdomain.EPPPremiumDomainReAssignCmd` instance to the `com.verisign.epp.interfaces.EPPDomain` via the `addExtension(EPPCodecComponent)` method. The reassign premium domain extension of domain update command returns the standard domain update response.
13.27.17  DotJobs Contact Extension

13.27.17.1  DotJobs Contact Packages
DotJobs Contact consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the SDK packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.jobscontact</td>
<td>DotJobs Contact EPP Encoder/Decoder package. All of the detail of encoding and decoding the DotJobs Contact EPP extension messages are encapsulated in this package. The EPPJobsContactExtFactory must be added to the EPP.CmdRspExtensions configuration parameter.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the com.verisign.epp.serverstub.JobsContactHandler class used to implement the EPP Contact Stub Server behavior. This class must be added to the EPP.ServerEventHandlers configuration parameter.</td>
</tr>
</tbody>
</table>

13.27.17.2  DotJobs Contact XML Schema Files
The DotJobs Contact EPP Extension Mapping is defined using XML schema files. These files are located in the epp-verisign-bundle-{$BUILD_VER}.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobsContact-1.0.xsd</td>
<td>schemas</td>
<td>DotJobs Contact XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.</td>
</tr>
</tbody>
</table>

13.27.17.3  DotJobs Contact Client Interface
13.27.17.3.1  Overview
DotJobs Contact is an extension to the Contact Object and hence EPPContact will be the client interface for DotJobs extension operations.

13.27.17.3.2  sendCreate() Method

Same as Contact sendCreate() method. However the extension part of create is explained using the sample listed below.
try {
    contact.setTransId("ABC-12345-XYZ");
    contact.setAuthorizationId("ClientXYZ");
    contact.addContactId("sh8013");
    contact.setVoicePhone("+1.7035555555");
    contact.setVoiceExt("123");
    contact.setFaxNumber("+1.7035555556");
    contact.setFaxExt("456");
    contact.setEmail("jdoe@example.com");

    // Streets
    Vector streets = new Vector();
    streets.addElement("123 Example Dr.");
    streets.addElement("Suite 100");
    streets.addElement("This is third line");

    EPPContactAddress address = new EPPContactAddress();
    address.setStreets(streets);
    address.setCity("Dulles");
    address.setStateProvince("VA");
    address.setPostalCode("20166-6503");
    address.setCountry("US");

    EPPContactPostalDefinition name = new EPPContactPostalDefinition(
        EPPContactPostalDefinition.ATTR_TYPE_LOC);
    name.setName("John Doe");
    name.setOrg("Example Inc.");
    name.setAddress(address);
    contact.addPostalInfo(name);

    // this is not a valid Example but it will do
    EPPContactAddress Intaddress = new EPPContactAddress();
    Intaddress.setStreets(streets);
    Intaddress.setCity("Dulles");
    Intaddress.setStateProvince("VA");
    Intaddress.setPostalCode("20166-6503");
    Intaddress.setCountry("US");

    EPPContactPostalDefinition Intname = new EPPContactPostalDefinition(
        EPPContactPostalDefinition.ATTR_TYPE_INT);
    Intname.setName("John Doe");
    Intname.setOrg("Example Inc.");
    Intname.setAddress(Intaddress);
    contact.addPostalInfo(Intname);

    // disclose names
    Vector names = new Vector();
    // names.addElement(new
// EPPContactDiscloseName(EPPContactDiscloseName.ATTR_TYPE_LOC));
names.addElement(new EPPContactDiscloseName(EPPContactDiscloseName.ATTR_TYPE_INT));

// disclose orgs
Vector orgs = new Vector();
orbs.addElement(new EPPContactDiscloseOrg(EPPContactDiscloseOrg.ATTR_TYPE_LOC));
orbs.addElement(new EPPContactDiscloseOrg(EPPContactDiscloseOrg.ATTR_TYPE_INT));

// disclose addresses
Vector addresses = new Vector();
adresses.addElement(new EPPContactDiscloseAddress(EPPContactDiscloseAddress.ATTR_TYPE_LOC));
adresses.addElement(new EPPContactDiscloseAddress(EPPContactDiscloseAddress.ATTR_TYPE_INT));

// disclose
EPPContactDisclose disclose = new EPPContactDisclose();
disclose.setFlag("0");
disclose.setNames(names);
disclose.setOrgs(orgs);
disclose.setAddresses(addresses);
disclose.setVoice("");
disclose.setFax("");
disclose.setEmail("");

contact.setDisclose(disclose);

EPPJobsContactCreateCmd createExt =
    new EPPJobsContactCreateCmd("SE", "www.verisign.com", "IT", "Yes", "No");

contact.addExtension(createExt);

response = (EPPContactCreateResp) contact.sendCreate();

// -- Output all of the response attributes
System.out.println("contactCreate: Response = [" + response + "]\n\n");
System.out.println("Contact ID : " + response.getId());
System.out.println("Contact Created Date : " +
    response.getCreationDate());
}
13.27.17.3.3 sendCheck() Method

DotJobs Contact is an extension to Contact Object and this method is not applicable to the extension.

13.27.17.3.4 sendInfo() Method

Same as Contact sendInfo() method. However the extension part of info is explained using the sample listed below.

```java
try {
    EPPContactInfoResp contactResponse = null;
    contactResponse = contact.sendInfo();
    System.out.println("contactInfo: id = " + response.getId());

    Vector postalContacts = null;
    if (response.getPostalInfo().size() > 0) {
        postalContacts = response.getPostalInfo();
        for (int j = 0; j < postalContacts.size(); j++) {
            EPPContactAddress address = ((EPPContactPostalDefinition) postalContacts.elementAt(j)).getAddress();
            System.out.println("street" + (i + 1) + " = " + address.getStreets().elementAt(i));
        }
    }

    System.out.println("city = " + address.getCity());
    System.out.println("state province = " + address.getStateProvince());
}
```
// Address Postal Code
System.out.println("contactInfo:\t\tpostal code = " 
 + address.getPostalCode());

// Address County
System.out.println("contactInfo:\t\tcountry = " 
 + address.getCountry());
}

// Contact E-mail
System.out.println("contactInfo:\temail = " + response.getEmail());

// Contact Voice
System.out.println("contactInfo:\tvoice = " + response.getVoice());

// Contact Voice Extension
System.out.println("contactInfo:\tvoice ext = " 
 + response.getVoiceExt());

// Contact Fax
System.out.println("contactInfo:\tfax = " + response.getFax());

// Contact Fax Extension
System.out.println("contactInfo:\tfax ext = " 
 + response.getFaxExt());

// Client Id
System.out.println("contactInfo: client id = " 
 + response.getClientId());

// Created By
System.out.println("contactInfo: created by = " 
 + response.getCreatedBy());

// Created Date
System.out.println("contactInfo: create date = " 
 + response.getCreatedDate());

// -- Output optional response attributes using accessors
// Contact Fax
if (response.getFax() != null) {
    System.out.println("contactInfo:\tfax = " + response.getFax());
}

// Contact Voice
if (response.getVoice() != null) {
    System.out.println("contactInfo:\tVoice = " 
 + response.getVoice());
}

// Last Updated By
if (response.getLastUpdatedBy() != null) {
    System.out.println("contactInfo: last updated by = " 
 + response.getLastUpdatedBy());
// Last Updated Date
if (response.getLastUpdatedDate() != null) {
    System.out.println("contactInfo: last updated date = "
       + response.getLastUpdatedDate());
}

// Last Transfer Date
if (response.getLastTransferDate() != null) {
    System.out.println("contactInfo: last updated date = 
       + response.getLastTransferDate());
}

// Authorization Id
if (response.getAuthInfo() != null) {
    System.out.println("contactInfo: authorization info = 
       + response.getAuthInfo().getPassword());
}

// Disclose
if (response.getDisclose() != null) {
    System.out.println("contactInfo: disclose info = 
       + response.getDisclose());
}

// -- Output extension attribute(s)
if (contactResponse.hasExtension(EPPJobsContactInfoResp.class)) {
    EPPJobsContactInfoResp ext = (EPPJobsContactInfoResp)
       contactResponse.getExtension(EPPJobsContactInfoResp.class);
    System.out.println("jobsContact: Title = " + ext.getTitle());
    System.out.println("jobsContact: Website = " + ext.getWebsite());
    System.out.println("jobsContact: industryType = " + 
       ext.getIndustryType());
    System.out.println("jobsContact: industryType = " + 
       ext.isAdminContact());
    System.out.println("jobsContact: associationMember = " + 
       ext.isAssociationMember());
} else {
    System.out.println("JobsContact: EPPJobsContact extn. NOT found");
}
13.27.17.4.1  sendUpdate() Method

Same as Contact sendUpdate() method. However the extension part of update is explained using the sample listed below.

```java
try {
    contact.setTransId("ABC-12345-XYZ");
    contact.addContactId("sh8013");

    // Streets
    Vector streets = new Vector();
    streets.addElement("123 Example Dr.");
    streets.addElement("Suite 100");
    streets.addElement("This is third line");

    // Address
    EPPContactAddress address = new EPPContactAddress(streets,
        "Dulles", "VA", "20166-6503", "US");

    EPPContactPostalDefinition postal = new EPPContactPostalDefinition(
        "Joe Brown", "Example Corp.",
        EPPContactPostalDefinition.ATTR_TYPE_LOC, address);

    // statuses
    contact.addStatus(EPPContact.STAT_PENDING_DELETE);
    contact.addPostalInfo(postal);
    contact.setVoicePhone("+1.7035555555");
    contact.setVoiceExt("456");
    contact.setFaxNumber("+1.7035555555");
    contact.setFaxExt("789");
    contact.setAuthorizationId("ClientXYZ");

    // disclose names
    Vector names = new Vector();
    names.addElement(new EPPContactDiscloseName(EPPContactDiscloseName.ATTR_TYPE_LOC));
    names.addElement(new EPPContactDiscloseName(EPPContactDiscloseName.ATTR_TYPE_INT));

    // disclose orgs
    Vector orgs = new Vector();
    orgs.addElement(new EPPContactDiscloseOrg(EPPContactDiscloseOrg.ATTR_TYPE_LOC));
    orgs.addElement(new EPPContactDiscloseOrg(EPPContactDiscloseOrg.ATTR_TYPE_INT));

    // disclose addresses
    Vector addresses = new Vector();
    addresses.addElement(new EPPContactDiscloseAddress(EPPContactDiscloseAddress.ATTR_TYPE_LOC));
    addresses.addElement(new EPPContactDiscloseAddress(EPPContactDiscloseAddress.ATTR_TYPE_LOC));
}
```
EPPContactDiscloseAddress.ATTR_TYPE_INT));

// disclose
EPPContactDisclose disclose = new EPPContactDisclose();
disclose.setFlag("0");
disclose.setNames(names);
disclose.setOrgs(orgs);
disclose.setAddresses(addresses);
disclose.setVoice(""):
disclose.setFax(""):
disclose.setEmail(""):

contact.setDisclose(disclose);

EPPJobsContactUpdateCmd updateExt = new EPPJobsContactUpdateCmd();

updateExt.setTitle("Customer Service");
updateExt.setWebsite("www.verisign.com");
updateExt.setIndustry("IT");
updateExt.setAdminContact("No");
updateExt.setAssociationMember("No");

contact.addExtension(updateExt);

response = contact.sendUpdate();

// -- Output all of the response attributes
System.out.println("contactUpdate: Response = [" + response + "]\n\n");
} catch (EPPCommandException e) {
handleException(e);
}

13.27.17.4.2  sendDelete() Method

DotJobs Contact is an extension to Contact Object and this method is not applicable to the extension.

13.27.17.4.3  sendTransfer() Method

DotJobs Contact is an extension to Contact Object and this method is not applicable to the extension.
13.27.18  Launch Extension
The Launch Extension defined in “Launch Phase Mapping for the Extensible Provisioning Protocol (EPP)” is used during the launch phases of new TLD’s. During the “sunrise” launch phase, trademark information is passed using the definitions in the “Mark and Signed Mark Objects Mapping”. The Launch Extension includes extensions to the Domain Check Command, Domain Info Command, Domain Create Command, Domain Update Command, and Domain Delete Command. The majority of the SDK support for the Launch Extension is handled by using the Launch CODEC classes, in the com.verisign.epp.codec.launch package, with the com.verisign.epp.interfaces.EPPLaunchExtFactory method or the com.verisign.epp.codec.launch.EPPLaunchExtFactory method. The com.verisign.epp.interfaces.EPPLaunchClient interfaces class was created to simplify sending the Launch Check Command (Claims Check Form or Availability Check Form).

13.27.18.1  Launch Extension Packages
The Launch Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Launch Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.launch</td>
<td>Launch Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Launch Extension EPP messages are encapsulated in this package. The com.verisign.epp.codec.launch.EPPLaunchExtFactory must be added to the EPP.CmdRspExtensions configuration parameter.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Addition of the EPPLaunch Client Interface class, which provides a simplified interface for sending the Claims Check Command.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the com.verisign.epp.serverstub.LaunchDomainHandler class and the com.verisign.epp.serverstub.LaunchPollHandler used to implement the EPP Launch Stub Server behavior. The com.verisign.epp.serverstub.LaunchDomainHandler class must be added to the EPP.ServerEventHandlers configuration parameter and the com.verisign.epp.serverstub.LaunchPollHandler class must be added to the EPP.PollHandlers configuration parameter to simulate a server that supports the Launch Extension.</td>
</tr>
</tbody>
</table>

13.27.18.2  Launch XML Schema Files
The Launch Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the epp-verisign-bundle-{$BUILD_VER}.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them.
### 13.27.18.3 Launch Client Interface

The Launch Extension is an extension to the Domain Object and so the Domain Client Interface classes that include `com.verisign.epp.interfaces.EPPDomain` and `com.verisign.epp.namestore.interfaces.NSDomain` are used along with the Launch Extension specific `com.verisign.epp.interfaces.EPPLaunch` class for the Claims Check Command. A set of examples is provided for various Launch Extension cases in the following sections.

#### 13.27.18.3.1 sendCheck() Method

The Launch Extension includes two forms of the Check Command, which include the Claims Check Form and the Availability Check Form. Both forms may be handled using the `com.verisign.epp.interfaces.EPPLaunch.sendCheck()` method.

“Figure 22 - Launch Claims Check Example” shows an example of executing a Launch Extension Claims Check Command using the `com.verisign.epp.interfaces.EPPLaunch.sendCheck()` method.

#### Figure 22 - Launch Claims Check Example

```java
EPPResponse response;
try {
    EPPLaunch launch = new EPPLaunch(session);
```
launch.setTransId("ABC-12345");
launch.addDomainName("example1.tld");
launch.setPhase(EPPLaunch.PHASE_CLAIMS);

response = launch.sendCheck();

if (response.hasExtension(EPPLaunchChkData.class)) {
    EPPLaunchChkData ext = (EPPLaunchChkData)
    response.getExtension(EPPLaunchChkData.class);

    List<EPPLaunchCheckResult> results = ext.getCheckResults();
    for (EPPLaunchCheckResult result : results) {
        if (result.isExists()) {
            System.out.println(result.getName() + ", mark

exists, claimsKey = [" + result.getClaimKey() + "]");
        } else {
            System.out.println(result.getName() + ", mark DOES NOT

exist");
        }
    }
}

} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}

“Figure 23 - Launch Availability Check Example” shows an example of executing a Launch Extension Availability Check Command using the
com.verisign.epp.interfaces.EPPLaunch.sendCheck() method for the custom “idn-releases” phase.

**Figure 23 - Launch Availability Check Example**

```java
EPPDomainCheckResp response;
try {
    EPPLaunch launch = new EPPLaunch(session);
```
launch.setTransId("ABC-12345");
launch.addDomainName("example1.tld");
launch.setPhase(EPPLaunch.PHASE_CUSTOM);
launch.setPhaseName("idn-release");
launch.setType(EPPLaunch.TYPE_AVAILABILITY);

response = (EPPDomainCheckResp) launch.sendCheck();

System.out.println("Launch Availability Response = [" + response + "]");
}
}

13.27.18.3.2  sendInfo() Method

The Launch Extension utilizes the com.verisign.epp.interfaces.EPPDomain.sendInfo() method, that is extended by the com.verisign.epp.namestore.interfaces.NSDomain class, to send a Domain Info Command with the Launch Extension.

“Figure 24 – Domain Info for Launch Application Example” shows an example of executing a Domain Info Command for a Sunrise Application with the Application Identifier of “abc123”.

**Figure 24 – Domain Info for Launch Application Example**

```java
EPPDomainInfoResp response;
try {
    NSDomain domain = new NSDomain(session);
```
domain.setTransId("ABC-12345");
domain.addDomainName("example1.tld");

domain.addExtension(new EPPLaunchInfo(new
EPPLaunchPhase(EPPLaunchPhase.PHASE_SUNRISE), "abc123");

response = domain.sendInfo();

if (response.hasExtension(EPPLaunchInfData.class)) {
    EPPLaunchInfData ext =
    response.getExtension(EPPLaunchInfData.class);
    System.out.println("Phase = " + ext.getPhase().getPhase());
    System.out.println("Id = " + ext.getApplicationId());
    System.out.println("Status = " + ext.getStatus().getStatus());
    System.out.println("Mark = " + ext.getMark());
}
}

catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}

“Figure 25 – Domain Info for Launch Registration Example” shows an example of executing a
Domain Info Command for a Sunrise Registration.

**Figure 25 – Domain Info for Launch Registration Example**

```java
EPPDomainInfoResp response;
try {
    NSDomain domain = new NSDomain(session);
```
domain.setTransId("ABC-12345");
domain.addDomainName("example1.tld");

EPPLaunchInfo infExt = new EPPLaunchInfo(new
EPPLaunchPhase(EPPLaunchPhase.PHASE_SUNRISE));
infExt.setIncludeMark(true);
domain.addExtension(infExt);
response = domain.sendInfo();

if (response.hasExtension(EPPLaunchInfData.class)) {
    EPPLaunchInfData ext = response.getExtension(EPPLaunchInfData.class);
    System.out.println("Phase = " + ext.getPhase().getPhase());
    System.out.println("Mark = " + ext.getMark());
}
}

catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getErrorResponse();
    Assert.fail(e.getMessage());
}

13.27.18.3.3 sendCreate() Method

The Launch Extension utilizes the com.verisign.epp.interfaces.EPPDomain.sendCreate() method, that is extended by the com.verisign.epp.namestore.interfaces.NSDomain class, to send a Domain Create Command with the Launch Extension. The Launch Extension supports all four forms (Sunrise Create Form, Claims Create Form, General Create Form, Mix Create Form).

“Figure 26 – Domain Create in Sunrise Create Form Example” shows an example of executing a Domain Create Command for a Sunrise Application with an Encoded Signed Mark.

**Figure 26 – Domain Create in Sunrise Create Form Example**

```java
EPPResponse response;
try {
    NSDomain domain = new NSDomain(session);

    // Define Mark
    EPPMark mark = new EPPMark();
    ...

    // Define Issuer
    EPPIssuer issuer = new EPPIssuer("1", "Example Inc.",
    support@example.tld);
    ...

    // Define Encoded Signed Mark and sign it
    EPPEncodedSignedMark signedMark =
```
```java
new EPPEncodedSignedMark("1-2", issuer,
    new GregorianCalendar(2013, 1, 1).getTime(),
    new GregorianCalendar(2014, 1, 1).getTime(),
    mark);
signedMark.sign(privateKey, certChain);

domain.addExtension(new EPPLaunchCreate(new
    EPPLaunchPhase(EPPLaunchPhase.PHASE_SUNRISE), signedMark,
    EPPLaunchCreate.TYPE_APPLICATION));

domain.setTransId("ABC-12345");
domain.addDomainName("example1.tld");
domain.setAuthString("ClientX");

response = domain.sendCreate();
if (response.hasExtension(EPPLaunchCreData.class)) {
    EPPLaunchCreData ext = response.getExtension(EPPLaunchCreData.class);
    System.out.println("Phase = " + ext.getPhase().getPhase());
    System.out.println("Id = " + ext.getApplicationId());
}
}
catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
```

“Figure 27 – Domain Create in Claims Create Form Example” shows an example of executing a Domain Create Command for a domain registration with the claims notice information.

**Figure 27 – Domain Create in Claims Create Form Example**

```java
EPPResponse response;

try {
    NSDomain domain = new NSDomain(session);

    domain.setTransId("ABC-12345");
    domain.addDomainName("example1.tld");
    domain.setAuthString("ClientX");

    domain.addExtension(new EPPLaunchCreate(new
        EPPLaunchPhase(EPPLaunchPhase.PHASE_CLAIMS), new
        EPPLaunchNotice("49FD46E6C4B45C55D4AC", new Date(), new Date()),
        EPPLaunchCreate.TYPE_REGISTRATION);
```
```java
response = domain.sendCreate();
}
catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
```

“Figure 28 – Domain Create in General Create Form Example” shows an example of executing a Domain Create Command in General Create Form to explicitly specify the phase and the type of object to create.

![Figure 28 – Domain Create in General Create Form Example](image)

```java
response = domain.sendCreate();
if (response.hasExtension(EPPLaunchCreData.class)) {
    EPPLaunchCreData ext = response.getExtension(EPPLaunchCreData.class);
    System.out.println("Phase = " + ext.getPhase().getPhase());
    System.out.println("Id = " + ext.getApplicationId());
}
catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
```

“Figure 29 – Domain Create in Mix Create Form Example” shows an example of executing a Domain Create Command in Mix Create Form to create a non-TMCH sunrise application with a mark.

![Figure 29 – Domain Create in Mix Create Form Example](image)

```java
response = domain.sendCreate();
if (response.hasExtension(EPPLaunchCreData.class)) {
    EPPLaunchCreData ext = response.getExtension(EPPLaunchCreData.class);
    System.out.println("Phase = " + ext.getPhase().getPhase());
    System.out.println("Id = " + ext.getApplicationId());
}
catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
```
// Define Mark
EPPMark mark = new EPPMark();

... 

domain.setTransId("ABC-12345");
domain.addDomainName("example1.tld");
domain.setAuthString("ClientX");

EPPLaunchCreate creExt = new EPPLaunchCreate(new
EPPLaunchPhase(EPPLaunchPhase.CUSTOM, "non-tmch-sunrise"),
EPPLaunchCreate.TYPE_APPLICATION);

creExt.addCodeMark(EPPLaunchCodeMark(mark));
creExt.setNotice(new EPPLaunchNotice("49FD46E6C4B45C55D4AC", new
Date(), new Date());
domain.addExtension(creExt);

response = domain.sendCreate();

if (response.hasExtension(EPPLaunchCreData.class)) {
    EPPLaunchCreData ext = response.getExtension(EPPLaunchCreData.class);
    System.out.println("Phase = " + ext.getPhase().getPhase());
    System.out.println("Id = " + ext.getApplicationId());
}

} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}

13.27.18.3.4  sendUpdate() Method

The Launch Extension utilizes the com.verisign.epp.interfaces.EPPDomain.sendUpdate() method, that is extended by the com.verisign.epp.namestore.interfaces.NSDomain class, to send a Domain Update Command with the Launch Extension. The Launch Extension enables a client to update a launch application referenced by the Application Id returned in the Domain Create Response. “Figure 30 – Domain Update of Sunrise Application Example” shows an example of updating a sunrise application.
Figure 30 – Domain Update of Sunrise Application Example

```java
EPPResponse response;
try {
    NSDomain domain = new NSDomain(session);
    domain.setTransId("ABC-12345");
    domain.addDomainName("example1.tld");
    domain.setUpdateAttrib(EPPDomain.HOST, "ns2.example.tld", EPPDomain.ADD);
    domain.setUpdateAttrib(EPPDomain.HOST, "ns1.example.tld", EPPDomain.REMOVE);

    // Add extension
    domain.addExtension(new EPPLaunchUpdate(new EPPLaunchPhase(EPPLaunchPhase.PHASE_SUNRISE), "abc123");
    response = domain.sendUpdate();
} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
```

13.27.18.3.5 sendDelete() Method

The Launch Extension utilizes the `com.verisign.epp.interfaces.EPPDomain.sendDelete()` method, that is extended by the `com.verisign.epp.namestore.interfaces.NSDomain` class, to send a Domain Delete Command with the Launch Extension. The Launch Extension enables a client to delete a launch application referenced by the Application Id returned in the Domain Create Response. “Figure 30 – Domain Update of Sunrise Application Example” shows an example of updating a sunrise application.
Figure 31 – Domain Delete of Sunrise Application Example

```java
EPPResponse response;
try {
    NSDomain domain = new NSDomain(session);
    domain.setTransId("ABC-12345");
    // Add extension
    domain.addExtension(new EPPLaunchDelete(new EPPLaunchPhase(EPPLaunchPhase.PHASE_SUNRISE), "abc123"));
    response = domain.sendDelete();
} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
```
13.27.19 PersReg Extension

The Personal Registration Extension defined in "Extensible Provisioning Protocol Extension Mapping: <Personal Registration>" includes extensions to the create (Domain, Email Forwarding, and Defensive Registration) response, an extension to the info (Domain, Email Forwarding) response, and an extension to the create (Domain, Email Forwarding) command to pass a consent identifier to authorize the create with a conflicting Defensive Registration.

13.27.19.1 PersReg Extension Packages

The Personal Registration Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Personal Registration Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.persreg</td>
<td>Personal Registration Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Personal Registration Extension EPP messages are encapsulated in this package. The com.verisign.epp.codec.persreg.EPPPersRegExtFactory must be added to the EPP.CmdRspExtensions configuration parameter.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes the com.verisign.epp.interfaces.EPPPersRegTst test class for sending the Personal Registration Extension on a Domain and Email Forwarding create and receiving responses with the Personal Registration Extension.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the com.verisign.epp.serverstub.PersRegDomainHandler class and the com.verisign.epp.serverstub.PersRegEmailFwdHandler used to implement the EPP Personal Registration Stub Server behavior. These classes must be added to the EPP.ServerEventHandlers configuration to simulate a server that supports the Personal Registration Extension.</td>
</tr>
</tbody>
</table>

13.27.19.2 PersReg XML Schema Files

The Personal Registration Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the epp-verisign-bundle-{$BUILD_VER}.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>persReg-1.0.xsd</td>
<td>schemas</td>
<td>Personal Registration Extension XML Schema</td>
</tr>
</tbody>
</table>
13.27.19.3 PersReg Client Interface

The Personal Registration Extension is an extension to the Domain and Email Forwarding objects, so the Domain (`com.verisign.epp.interfaces.EPPDomain`) and Email Forwarding (`com.verisign.epp.interfaces.EPPEmailFwd`) Client Interface classes are used with the Personal Registration Extension.

“Figure 32 – Personal Registration Create with Consent Example” shows an example of executing a Domain Create Command with consent using the Personal Registration Create Extension `com.verisign.epp.codec.persreg.EPPPersRegCreate` class.

```
EPPDomainCreateResp response;

try {
    EPPDomain domain = new EPPDomain(session);
    domain.setTransId("ABC-12345");
    domain.addDomainName("example1.tld");

    domain.addExtension(new EPPPersRegCreate("ID:12345"));
    response = domain.sendCreate();

    if (response.hasExtension(EPPPersRegCreateData.class)) {
        EPPPersRegCreateData ext = (EPPPersRegCreateData) response.getExtension(EPPPersRegCreateData.class);
        System.out.println("bundle rate = " + ext.isBundleRate());
    }
}

catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    if (errorResponse.hasExtension(EPPPersRegCreateErrData.class)) {
        EPPPersRegCreateErrData ext = (EPPPersRegCreateErrData) errorResponse.getExtension(EPPPersRegCreateErrData.class);
        System.out.println("PersReg error = " + ext);
    }
    Assert.fail(e.getMessage());
}
```
13.27.20 Related Domain Extension

The Related Domain Extension defined in “Extensible Provisioning Protocol Extension Mapping: <Related Domain>” includes an extension to the domain name mapping for managing client-side and server-side domain name relationships.

13.27.20.1 Related Domain Extension Packages

The Related Domain Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Related Domain Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.relateddomainext</td>
<td>Related Domain Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Related Domain Extension EPP messages are encapsulated in this package. The com.verisign.epp.codec.relateddomainext.EPPRelatedDomainExtFactory must be added to the EPP.CmdRspExtensions configuration parameter.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes the com.verisign.epp.interfaces.EPPRelatedDomain class to send both forms of the info command (Domain Info Form and Related Info Form) and the multiple related domain transform commands (Create, Update, Delete, Renew, and Transfer), and com.verisign.epp.interfaces.EPPRelatedDomainTst test class for testing the com.verisign.epp.interfaces.EPPRelatedDomain class.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the com.verisign.epp.serverstub.RelatedDomainHandler class used to implement the EPP Related Domain Stub Server behavior. These classes must be added to the EPP.ServerEventHandlers configuration to simulate a server that supports the Related Domain Extension.</td>
</tr>
</tbody>
</table>

13.27.20.2 Related Domain XML Schema Files

The Related Domain Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the epp-verisign-bundle-{$BUILD_VER}.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relatedDomain-1.0.xsd</td>
<td>schemas</td>
<td>Related Domain Extension XML Schema</td>
</tr>
</tbody>
</table>
13.27.20.3 Related Domain Client Interface

The Related Domain Extension is an extension to the Domain object, so the `com.verisign.epp.interfaces.EPPRelatedDomain` class extends the `com.verisign.epp.interfaces.EPDomain`, and is a superclass of the `com.verisign.epp.namestore.interfaces.NSDomain` class. The `com.verisign.epp.interfaces.EPPRelatedDomain` Client Interface class methods can be used to send the info command with the two types/forms (`EPPRelatedDomain.TYPE_DOMAIN` for the Domain Info Form and `EPPRelatedDomain.TYPE_RELATED` for the Related Info Form) or the multiple related domain transform commands (create, update, delete, renew, and transfer). The `EPPRelatedDomain.sendRelatedInfo() : EPPResponse` is used to get the `com.verisign.epp.codec.domain.EPPDomainInfoResp`, for the Domain Info Form, and the `com.verisign.epp.codec.gen.EPPResponse`, for the Related Info Form, with the `com.verisign.epp.codec.relateddomainext.EPPRelatedDomainExtInfoData` extension.

“Figure 33 – Related Domain Info Command in Domain Info Form Example” shows an example of executing a Domain Info Command in the Domain Info Form to receive a response that includes the Related domain information for “example1.tld” along with the related domain information in the `com.verisign.epp.codec.relateddomainext.EPPRelatedDomainExtInfoData` class.

**Figure 33 – Related Domain Info Command in Domain Info Form Example**

```java
EPPDomainInfoResp response;
try {
    EPPRelatedDomain relDomain = new EPPRelatedDomain(session);
```

relDomain.setTransId("ABC-12345");
relDomain.addDomainName("example1.tld");
relDomain.setInfoForm(EPPRelatedDomain.DOMAIN_INFO_FORM);

response = (EPPDomainInfoResp) relDomain.sendRelatedInfo();

if (response.hasExtension(EPPRelatedDomainExtInfData.class)) {
    EPPRelatedDomainExtInfData ext = (EPPRelatedDomainExtInfData)
            response.getExtension(EPPRelatedDomainExtInfData.class);
    System.out.println("related domain info = " + ext);
}

// Get the domain info for example1.tld
System.out.println("Domain ROID = " + response.getRoid());

} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}

“Figure 34 – Related Domain Info Command in Related Info Form Example” shows an example of executing a Domain Info Command in the Related Info Form to receive a response that includes the related domain information of "example1.tld" in the com.verisign.epp.codec.relateddomainext.EPPRelatedDomainExtInfData class.

Figure 34 – Related Domain Info Command in Related Info Form Example

EPPResponse response;

try {
    EPPRelatedDomain relDomain = new EPPRelatedDomain(session);

    relDomain.setTransId("ABC-12345");
    relDomain.addDomainName("example1.tld");

    relDomain.setInfoForm(EPPRelatedDomain.RELATED_INFO_FORM);

    response = relDomain.sendRelatedInfo();

    if (response.hasExtension(EPPRelatedDomainExtInfData.class)) {
        EPPRelatedDomainExtInfData ext = (EPPRelatedDomainExtInfData)
                response.getExtension(EPPRelatedDomainExtInfData.class);
        System.out.println("related domain info = " + ext);
    }

} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
“Figure 35 – Related Domain Create Command Example” shows an example of executing a Related Domain Create Command to create all of the domains “example.com”, “domain1.com”, “domain2.com”, and “xn—idn.com” at once.

**Figure 35 – Related Domain Create Command Example**

```
EPPDomainCreateResp response;
try {
    EPPRelatedDomain relDomain = new EPPRelatedDomain(session);
    relDomain.setTransId("ABC-12345");
    relDomain.addDomainName("example1.tld");
    relDomain.setAuthString("2fooBAR");

    EPPRelatedDomainExtAuthInfo authInfo = new 
    EPPRelatedDomainExtAuthInfo("relDom123!");
    EPPRelatedDomainExtPeriod period = new EPPRelatedDomainExtPeriod(5);
    relDomain.addRelatedDomain(new 
    EPPRelatedDomainExtDomain("domain1.com", authInfo, period));
    relDomain.addRelatedDomain(new 
    EPPRelatedDomainExtDomain("domain2.com", authInfo, period));
    relDomain.addRelatedDomain(new EPPRelatedDomainExtDomain("xn-idn.com", 
    authInfo, period, "CHI");

    response = relDomain.sendRelatedCreate();
    if (response.hasExtension(EPPRelatedDomainExtCreateResp.class)) {
        EPPRelatedDomainExtCreateResp ext = 
        (EPPRelatedDomainExtCreateResp) 
        response.getExtension(EPPRelatedDomainExtCreateResp.class);
        System.out.println("related domain create = " + ext);
    }
} 
catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
```

“Figure 36 – Related Domain Delete Command Example” shows an example of executing a Related Domain Delete Command to delete all of the domains “example.com”, “domain1.com”, “domain2.com”, and “xn—idn.com” at once.

**Figure 36 – Related Domain Delete Command Example**

```
EPPResponse response;
try {
    EPPRelatedDomain relDomain = new EPPRelatedDomain(session);
```

VeriSign Inc. Proprietary Information
Verisign Bundle EPP SDK Programmer's Guide ● page 204
relDomain.setTransId("ABC-12345");
relDomain.addDomainName("example1.tld");
relDomain.addRelatedName("domain1.com");
relDomain.addRelatedName("domain2.com");
relDomain.addRelatedName("xn--idn.com");

response = relDomain.sendRelatedDelete();
if (response.hasExtension(EPPRelatedDomainExtDeleteResp.class)) {
    EPPRelatedDomainExtDeleteResp ext =
    (EPPRelatedDomainExtDeleteResp)
    response.getExtension(EPPRelatedDomainExtDeleteResp.class);
    System.out.println("related domain delete = " + ext);
}

} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}

“Figure 37 – Related Domain Update Command Example” shows an example of executing a Related Domain Update Command to update all of the domains “example.com”, “domain1.com”, “domain2.com”, and “xn—idn.com” by adding the clientHold status.

**Figure 37 – Related Domain Update Command Example**

```java
EPPResponse response;
try {
    EPPRelatedDomain relDomain = new EPPRelatedDomain(session);
    relDomain.setTransId("ABC-12345");
    relDomain.addDomainName("example1.tld");
    relDomain.addRelatedName("domain1.com");
    relDomain.addRelatedName("domain2.com");
    relDomain.addRelatedName("xn--idn.com");
    relDomain.setUpdateAttrib( EPPDomain.STATUS, new EPPDomainStatus(EPPDomain.STATUS_CLIENT_HOLD, EPPDomain.ADD);
    response = relDomain.sendRelatedUpdate();
}
catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
```
“Figure 38 – Related Domain Renew Command Example” shows an example of executing a Related Domain Renew Command to renew all of the domains “example.com”, “domain1.com”, “domain2.com”, and “xn—idn.com” for 5 years.

**Figure 38 – Related Domain Renew Command Example**

```java
EPPDomainRenewResp response;
try {
    EPPRelatedDomain relDomain = new EPPRelatedDomain(session);
    Date  currExpDate = new GregorianCalendar(2013, 9, 10).getTime();
    relDomain.setTransId("ABC-12345");
    relDomain.addDomainName("example1.tld");
    relDomain.setPeriodLength(5);
    EPPRelatedDomainExtPeriod period = new EPPRelatedDomainExtPeriod(5);
    relDomain.addRelatedDomainExtDomain("domain1.com", currExpDate, period);
    relDomain.addRelatedDomainExtDomain("domain2.com", currExpDate, period);
    relDomain.addRelatedDomainExtDomain("xn—idn.com", currExpDate, period);
    response = relDomain.sendRelatedRenew();
    if (response.hasExtension(EPPRelatedDomainExtRenewResp.class)) {
        EPPRelatedDomainExtRenewResp ext = (EPPRelatedDomainExtRenewResp)
            response.getExtension(EPPRelatedDomainExtRenewResp.class);
        System.out.println("related domain renew = " + ext);
    }
} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
```

“Figure 39 – Related Domain Transfer Command Example” shows an example of executing a Related Domain Transfer Command to request transfer of all the domains “example.com”, “domain1.com”, “domain2.com”, and “xn—idn.com”.

**Figure 39 – Related Domain Transfer Command Example**

```java
EPPResponse response;
try {
    EPPRelatedDomain relDomain = new EPPRelatedDomain(session);
```
relDomain.setTransId("ABC-12345");
relDomain.addDomainName("example1.tld");
relDomain.setTransferOpCode(EPPDomain.TRANSFER_REQUEST);
relDomain.setAuthString("2fooBAR");
relDomain.setPeriodLength(1);

EPPRelatedDomainExtPeriod period = new EPPRelatedDomainExtPeriod(1);
EPPRelatedDomainExtAuthInfo authInfo = new EPPRelatedDomainExtAuthInfo("relDom123!");

relDomain.addRelatedDomain(new EPPRelatedDomainExtDomain("domain1.com", authInfo, period));
relDomain.addRelatedDomain(new EPPRelatedDomainExtDomain("domain2.com", authInfo, period));
relDomain.addRelatedDomain(new EPPRelatedDomainExtDomain("xn—idn.com", authInfo, period));

response = relDomain.sendRelatedTransfer();

if (response.hasExtension(EPPRelatedDomainExtTransferResp.class)) {
    EPPRelatedDomainExtTransferResp ext = (EPPRelatedDomainExtTransferResp) response.getExtension(EPPRelatedDomainExtTransferResp.class);
    System.out.println("related domain transfer = " + ext);
}

} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}
13.27.21  Change Poll Extension
The Change Poll Extension defined in “Change Poll Extension for the Extensible Provisioning Protocol (EPP)” includes an extension to any object mapping for notifying clients of operations on client sponsored objects that were not initiated by the client through EPP.

13.27.21.1  Change Poll Extension Packages
The Change Poll Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Change Poll Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.changepoll</td>
<td>Change Poll Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Change Poll Extension EPP messages are encapsulated in this package. The com.verisign.epp.codec.changepoll.EPPChangePollExtFactor must be added to the EPP.CmdRspExtensions configuration parameter.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes com.verisign.epp.interfaces.EPPChangePollDomainTest class for testing the Change Poll Extension against the Stub Server.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the com.verisign.epp.serverstub.ChangePollDomainHandler class used to implement the EPP Change Poll Domain Stub Server behavior. These classes must be added to the EPP.ServerEventHandlers configuration to simulate a server that supports the Change Poll Extension for domain objects.</td>
</tr>
</tbody>
</table>

13.27.21.2  Change Poll Extension XML Schema Files
The Change Poll Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the epp-verisign-bundle-{$BUILD_VER}.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>changePoll-1.0.xsd</td>
<td>schemas</td>
<td>Change Poll Extension XML Schema</td>
</tr>
</tbody>
</table>
13.27.22 Registry Fee Extension

The Registry Fee Extension defined in “Registry Fee Extension for the Extensible Provisioning Protocol (EPP)” that provides a mechanism by which EPP clients may query the fees and credits associated with various billable transactions and also obtain their current account balance.

13.27.22.1 Registry Fee Extension Packages

The Registry Fee Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Registry Fee Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.fee</td>
<td>Registry Fee Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Registry Fee EPP messages are encapsulated in this package.</td>
</tr>
<tr>
<td></td>
<td>The com.verisign.epp.codec.fee.v06.EPFeeExtFactory must be added to the EPP.CmdRspExtensions configuration parameter to support draft-brown-epp-fees-03.</td>
</tr>
<tr>
<td></td>
<td>The com.verisign.epp.codec.fee.v07.EPFeeExtFactory must be added to the EPP.CmdRspExtensions configuration parameter to support draft-brown-epp-fees-04.</td>
</tr>
<tr>
<td></td>
<td>The com.verisign.epp.codec.fee.v08.EPFeeExtFactory must be added to the EPP.CmdRspExtensions configuration parameter to support draft-brown-epp-fees-05.</td>
</tr>
<tr>
<td></td>
<td>The com.verisign.epp.codec.fee.v09.EPFeeExtFactory must be added to the EPP.CmdRspExtensions configuration parameter to support draft-brown-epp-fees-06.</td>
</tr>
<tr>
<td></td>
<td>The com.verisign.epp.codec.fee.v11.EPFeeExtFactory must be added to the EPP.CmdRspExtensions configuration parameter to support draft-brown-epp-fees-07.</td>
</tr>
<tr>
<td></td>
<td>The com.verisign.epp.codec.fee.v23.EPFeeExtFactory must be added to the EPP.CmdRspExtensions configuration parameter to support draft-ietf-regext-epp-fees-08.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes com.verisign.epp.interfaces.v06.EPPFeeDomainTst test class for testing the Registry Fee Extension against the Stub Server for draft-brown-epp-fees-03</td>
</tr>
<tr>
<td></td>
<td>Includes com.verisign.epp.interfaces.v07.EPPFeeDomainTst test class for testing the Registry Fee Extension against the</td>
</tr>
</tbody>
</table>
Includes `com.verisign.epp.interfaces.v08.EPPFeeDomainTst` test class for testing the Registry Fee Extension against the Stub Server for draft-brown-epp-fees-05.

Includes `com.verisign.epp.interfaces.v09.EPPFeeDomainTst` test class for testing the Registry Fee Extension against the Stub Server for draft-brown-epp-fees-06.

Includes `com.verisign.epp.interfaces.v11.EPPFeeDomainTst` test class for testing the Registry Fee Extension against the Stub Server for draft-brown-epp-fees-07.

Includes `com.verisign.epp.interfaces.v23.EPPFeeDomainTst` test class for testing the Registry Fee Extension against the Stub Server for draft-ietf-regext-epp-fees-08.

Includes `com.verisign.epp.interfaces.v1_0.EPPFeeDomainTst` test class for testing the Registry Fee Extension against the Stub Server for draft-ietf-regext-epp-fees-10 and later.

**13.27.22.2 Registry Fee Extension XML Schema Files**

The Registry Fee Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the `epp-verisign-bundle-{$BUILD_VER}.jar` in the `schemas` directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fee-0.6.xsd</td>
<td>schemas</td>
<td>draft-brown-epp-fees-03 Registry Fee Extension XML Schema</td>
</tr>
<tr>
<td>fee-0.7.xsd</td>
<td>schemas</td>
<td>draft-brown-epp-fees-04 Registry Fee Extension XML Schema</td>
</tr>
<tr>
<td>fee-0.8.xsd</td>
<td>schemas</td>
<td>draft-brown-epp-fees-05 Registry Fee Extension XML Schema</td>
</tr>
<tr>
<td>fee-0.9.xsd</td>
<td>schemas</td>
<td>draft-brown-epp-fees-06 Registry Fee Extension XML Schema</td>
</tr>
<tr>
<td>fee-0.11.xsd</td>
<td>schemas</td>
<td>draft-brown-epp-fees-07 Registry Fee Extension XML Schema</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>-----------------------------------------------------------</td>
</tr>
<tr>
<td>fee-0.23.xsd</td>
<td>schemas</td>
<td>draft-ietf-regext-epp-fees-08 Registry Fee Extension XML Schema</td>
</tr>
<tr>
<td>fee-1.0.xsd</td>
<td>schemas</td>
<td>draft-ietf-regext-epp-fees-10 and later Registry Fee Extension XML Schema</td>
</tr>
</tbody>
</table>
13.27.23 **Allocation Token Extension**
The Allocation Token Extension defined in “Allocation Token Extension for the Extensible Provisioning Protocol (EPP)” that is used to support including an allocation token or code for allocating an object like a domain name to the client.

13.27.23.1 **Allocation Token Extension Packages**
The Allocation Token Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Allocation Token Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.allocationToken</td>
<td>Allocation Token Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Allocation Token EPP messages are encapsulated in this package.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes com.verisign.epp.interfaces.EPPAllocationTokenDomainTst test class for testing the Allocation Token Extension against the Stub Server.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the com.verisign.epp.serverstub.AllocationTokenDomainHandler class used to implement the EPP Allocation Token Domain Stub Server. These classes must be added to the EPP.ServerEventHandlers configuration to simulate a server that supports the Allocation Token Extension.</td>
</tr>
</tbody>
</table>

13.27.23.2 **Allocation Token Extension XML Schema Files**
The Allocation Token Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the `epp-verisign-bundle-${BUILD_VER}.jar` in the `schemas` directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allocationToken -1.0.xsd</td>
<td>schemas</td>
<td>Allocation Token Extension XML Schema</td>
</tr>
</tbody>
</table>
13.27.24  **IDN Map Extension**
The IDN Map Extension defined in “Internationalized Domain Name Mapping Extension for the Extensible Provisioning Protocol (EPP)” that is used to pass the Internationized Domain Name (IDN) table identifier for IDN domain names.

13.27.24.1  **IDN Map Extension Packages**
The IDN Map Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the IDN Map Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.idnmap</td>
<td>IDN Map Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the IDN Map EPP messages are encapsulated in this package.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes com.verisign.epp.interfaces.EPPIdnMapDomainTst test class for testing the IDN Map Extension against the Stub Server.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the com.verisign.epp.serverstub.IdnMapDomainHandler class used to implement the EPP IDN Map Domain Stub Server. These classes must be added to the EPP.ServerEventHandlers configuration to simulate a server that supports the IDN Map Extension.</td>
</tr>
</tbody>
</table>

13.27.24.2  **IDN Map Extension XML Schema Files**
The IDN Map Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the epp-verisign-bundle-{$BUILD_VER}.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idn-1.0.xsd</td>
<td>schemas</td>
<td>IDN Map Extension XML Schema</td>
</tr>
</tbody>
</table>
13.27.25 Verification Code Extension

The Verification Code Extension, defined in “Verification Code Extension for the Extensible Provisioning Protocol (EPP)”, provides support for including a verification code for making the data for a transform command as being verified by a 3rd party, which is referred to as the Verification Service Provider (VSP). The verification code is digitally signed by the VSP using XML Signature and is “base64” encoded”. The extension also supports an extension to the info command and response to receive the verification compliance status of the domain name along with the relevant information like the verification codes that have been set and the verification codes that are missing.

13.27.25.1 Verification Code Extension Packages

The Verification Code Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Verification Code Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.verificationcode</td>
<td>Verification Code Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Verification Code EPP messages are encapsulated in this package.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes com.verisign.epp.interfaces.EPPVerificationCodeDomainTest test class for testing the Verification Code Extension against the Stub Server.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the com.verisign.epp.serverstub.VerificationCodeDomainHandler class used to implement the EPP Verification Code Domain Stub Server. These classes must be added to the EPP.ServerEventHandlers configuration to simulate a server that supports the Verification Code Extension.</td>
</tr>
</tbody>
</table>

13.27.25.2 Verification Code Extension XML Schema Files

The Verification Code Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the epp-verisign-bundle-$BUILD_VER.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>verificationCode-1.0.xsd</td>
<td>schemas</td>
<td>Verification Code Extension XML Schema</td>
</tr>
</tbody>
</table>
13.27.26 Organization Extension

The Organization Extension, defined in “Organization Extension for the Extensible Provisioning Protocol (EPP)”, provides support for assigning a organization with a role to existing object (domain, host, contact) as well as any future objects.

13.27.26.1 Organization Extension Packages

The Organization Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Organization Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.orgext</td>
<td>Organization Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Organization Extension EPP messages are encapsulated in this package.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes <code>com.verisign.epp.interfaces.EPPOrgExtDomainTst</code> test class for testing the Organization Extension against the Stub Server.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the <code>com.verisign.epp.serverstub.OrgExtDomainHandler</code> class used to implement the EPP Organization Extension Domain Stub Server. These classes must be added to the <code>EPP.ServerEventHandlers</code> configuration to simulate a server that supports the Organization Extension.</td>
</tr>
</tbody>
</table>

13.27.26.2 Organization Extension XML Schema Files

The Organization Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the `epp-verisign-bundle-{$BUILD_VER}.jar` in the `schemas` directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orgext-1.0.xsd</td>
<td>schemas</td>
<td>Organization Extension XML Schema</td>
</tr>
</tbody>
</table>
### 13.27.27 Login Security Extension

The Login Security Extension defined in “Login Security for the Extensible Provisioning Protocol (EPP)” that provides a enhances the security of the EPP connection and login by supporting passwords greater than 16 characters, supporting the passing of client user agent information to the server for identifying current and future issues, and supporting the return of security events (errors and warnings) in the login response for the client to address.

#### 13.27.27.1 Login Security Extension Packages

The Login Security Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Login Security Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.loginsec</td>
<td>Login Security Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Login Security EPP messages are encapsulated in this package. The com.verisign.epp.codec.fee.v01.EPPLoginSecExtFactory must be added to the EPP.CmdRspExtensions configuration parameter to support draft-gould-regext-login-security-00. The com.verisign.epp.codec.fee.v02.EPPLoginSecExtFactory must be added to the EPP.CmdRspExtensions configuration parameter to support draft-gould-regext-login-security-01.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the com.verisign.epp.serverstub.LoginSecGenHandler class that extends the com.verisign.epp.serverstub.GenHandler class for adding support for the Login Security Extension with the login command and inclusion of the Login Security Extension in the login response. The LoginSecGenHandler will route requests to the version specific LoginSecSubGenHandler to implement the behavior for draft-gould-regext-login-security-00 and future versions of the extension. The com.verisign.epp.serverstub.LoginSecGenHandler class</td>
</tr>
</tbody>
</table>
must be added to the EPP.ServerEventHandlers in place of com.verisign.epp.serverstub.GenHandler to simulate a server that supports the different versions of the Login Security Extension.

13.27.27.2 Login Security Extension XML Schema Files

The Login Security Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the epp-verisign-bundle-{$BUILD_VER}.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loginSec-0.1.xsd</td>
<td>schemas</td>
<td>draft-gould-regext-login-security-00 Login Security Extension XML Schema</td>
</tr>
<tr>
<td>loginSec-0.2.xsd</td>
<td>schemas</td>
<td>draft-gould-regext-login-security-01 Login Security Extension XML Schema</td>
</tr>
</tbody>
</table>
13.27.28  **Launch Policy Extension**

The Launch Policy Extension in “Launch Phase Policy Extensions Mapping for the Extensible Provisioning Protocol (EPP)” is an extension of the Registry Mapping that defines the server policy of the Launch Phase EPP extension.

13.27.28.1  **Launch Policy Extension Packages**

The Launch Policy Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Launch Policy Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.launchpolicy</td>
<td>Launch Policy Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Login Security EPP messages are encapsulated in this package. The com.verisign.epp.codec.launchpolicy.v01.EPPLaunchPolicyExtFactory must be added to the EPP.CmdRspExtensions configuration parameter to support draft-gould-regext-launch-policy-00.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes com.verisign.epp.interfaces.launchpolicy.v01.EPPLaunchPolicyTst test class for testing the Launch Policy Extension against the Stub Server for draft-gould-regext-launch-policy-00.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>For the Launch Policy Extension to be supported by the Stub Server, the com.verisign.epp.serverstub.registry.v01.RegistryHandler or later must be added to the EPP.ServerEventHandlers configuration parameter, and the desired Launch Policy Extension adapter must be added to the EPP.RegistryPolicyAdapters configuration parameter. The com.verisign.epp.codec.launchpolicy.v01.EPPLaunchPolicyAdapter must be added to EPP.RegistryPolicyAdapters configuration parameter to support draft-gould-regext-launch-policy-00.</td>
</tr>
</tbody>
</table>
13.27.28.2 Launch Policy Extension XML Schema Files

The Launch Policy Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the `epp-verisign-bundle-{$BUILD_VER}.jar` in the `schemas` directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>launchPolicy-0.1.xsd</td>
<td>schemas</td>
<td>draft-gould-regext-launch-policy-00 Launch Policy Extension XML Schema</td>
</tr>
</tbody>
</table>
13.27.29 Login Security Policy Extension


13.27.29.1 Login Security Extension Packages

The Login Security Policy Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Login Security Policy Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.loginsecpolicy</td>
<td>Login Security Policy Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Login Security Policy EPP messages are encapsulated in this package. The <code>com.verisign.epp.codec.loginsecpolicy.v01.EPPLoginSecPolicyExtFactory</code> must be added to the <code>EPP.CmdRspExtensions</code> configuration parameter to support draft-gould-regext-login-security-policy-00.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes <code>com.verisign.epp.interfaces.loginsecpolicy.v01.EPPLLoginSecPolicyTst</code> test class for testing the Login Security Policy Extension against the Stub Server for draft-gould-regext-login-security-policy-00.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>For the Login Security Policy Extension to be supported by the Stub Server, the <code>com.verisign.epp.serverstub.registry.v01.RegistryHandler</code> or later must be added to the <code>EPP.ServerEventHandlers</code> configuration parameter, and the desired Login Security Policy Extension adapter must be added to the <code>EPP.RegistryPolicyAdapters</code> configuration parameter. The <code>com.verisign.epp.codec.loginsecpolicy.v01.EPPLLoginSecPolicyAdapter</code> must be added to <code>EPP.RegistryPolicyAdapters</code> configuration parameter to support draft-gould-regext-login-security-policy-00.</td>
</tr>
</tbody>
</table>
13.27.29.2 Login Security Policy Extension XML Schema Files

The Login Security Policy Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the `epp-verisign-bundle-{$BUILD_VER}.jar` in the `schemas` directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>loginSecPolicy-0.1.xsd</td>
<td>schemas</td>
<td>draft-gould-regext-login-security-policy-00 Login Security Policy Extension XML Schema</td>
</tr>
</tbody>
</table>
13.27.30 Validate Extension

The Validate Extension in “Validate Mapping for the Extensible Provisioning Protocol (EPP)” describes a mapping for the validation of contact and eligibility data.

13.27.30.1 Validate Extension Packages

The Validate Extension consists of sub-packages of the SDK packages and class additions to existing SDK packages. The following table provides an overview of the Validate Extension packages.

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.verisign.epp.codec.validate</td>
<td>Validate Extension EPP Encoder/Decoder package. All of the detail of encoding and decoding the Validate EPP messages are encapsulated in this package. The com.verisign.epp.codec.validate.v02.EPPValidateMapFactory must be added to the EPP.MapFactories configuration parameter to support draft-ietf-regext-validate-04.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes com.verisign.epp.interfaces.validate.v02.EPPValidateTst test class for testing the Validate Extension against the Stub Server for draft-ietf-regext-validate-04.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>For the Validate Extension to be supported by the Stub Server, the com.verisign.epp.serverstub.validate.v02.ValidateHandler or later must be added to the EPP.ServerEventHandlers configuration parameter. The com.verisign.epp.serverstub.validate.v02.ValidateHandler is used to support draft-ietf-regext-validate-04.</td>
</tr>
</tbody>
</table>

13.27.30.2 Validate Extension XML Schema Files

The Validate Extension is defined using an XML schema file and is dependent on a set of XML schema files. These files are located in the epp-verisign-bundle-{$BUILD_VER}.jar in the schemas directory. You must un-jar the jar file in order to explicitly view them.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>validate-0.2.xsd</td>
<td>schemas</td>
<td>draft-ietf-regext-validate-04 Validate Extension XML Schema</td>
</tr>
</tbody>
</table>