# Change Log

<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</td>
<td>Initial Revision</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>8/2009</td>
<td></td>
</tr>
<tr>
<td>Scott Hollenbeck</td>
<td>Extensible Provisioning Protocol (EPP) Transport Over TCP</td>
<td>8/2009</td>
<td></td>
</tr>
<tr>
<td>Scott Hollenbeck</td>
<td>Extensible Provisioning Protocol (EPP) Domain Name Mapping</td>
<td>8/2009</td>
<td></td>
</tr>
<tr>
<td>Scott Hollenbeck</td>
<td>Extensible Provisioning Protocol (EPP) Host Mapping</td>
<td>8/2009</td>
<td></td>
</tr>
<tr>
<td>Scott Hollenbeck</td>
<td>Extensible Provisioning Protocol (EPP) Contact Mapping</td>
<td>8/2009</td>
<td></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>Venkat Munuswamy</td>
<td>Extensible Provisioning Protocol Extension Mapping: IDN Language Tag</td>
<td>1.1</td>
<td></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>Mahendra Jain</td>
<td>Premium Domain Extension Mapping</td>
<td>2.0</td>
<td>3/18/2010</td>
</tr>
<tr>
<td>Deepak Deshpande</td>
<td>WhoWas Mapping</td>
<td>1.0</td>
<td>3/31/2010</td>
</tr>
<tr>
<td>Developer(s)</td>
<td>Mapping 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>Client Object Attribute Extension Mapping</td>
<td>1.0</td>
<td>2/15/2011</td>
</tr>
<tr>
<td>James Gould</td>
<td>Balance Mapping</td>
<td>1.0</td>
<td>11/7/2011</td>
</tr>
<tr>
<td></td>
<td>Defensive Registration Mapping</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Email Forwarding Mapping</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>James Gould</td>
<td>Launch Phase Extension Mapping</td>
<td>11</td>
<td>5/17/2013</td>
</tr>
<tr>
<td>Wil Tan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gavin Brown</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Personal Registration Extension Mapping</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Registry Mapping</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Related Domain Extension Mapping</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Definitions, Acronyms, and Abbreviations

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPP</td>
<td>Extensible Provisioning Protocol</td>
</tr>
<tr>
<td>RFC</td>
<td>Request for Comments</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>XML</td>
<td><a href="http://www.w3c.org/XML/">http://www.w3c.org/XML/</a></td>
</tr>
<tr>
<td>RGP</td>
<td>Registry Grace Period</td>
</tr>
<tr>
<td>CTLD</td>
<td>Consolidated Top-Level Domain</td>
</tr>
<tr>
<td>DS</td>
<td>Delegation Signer</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
</tbody>
</table>
# Contents

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

2. CHANGES FROM PREVIOUS VERSION ................................................................................. 3
   2.1 VERSION 1.0.0 .................................................................................................................. 3

3. SUPPORTED TRANSPORTS .................................................................................................... 4

4. QUICK START INSTRUCTIONS .............................................................................................. 5
   4.1 RUNNING SDK TESTS VIA STUB SERVER ........................................................................... 6
   4.2 CHANGES REQUIRED TO INTERFACE WITH VERISIGN SERVERS ........................................ 6
   4.3 SET THE CLASSPATH OF THE APPLICATION ....................................................................... 8
   4.4 EXAMPLE CODE TO INITIALIZE THE VERISIGN BUNDLE EPP SDK AND START SESSION ........ 8
   4.5 EXAMPLE CODE TO INITIALIZE THE VERISIGN BUNDLE EPP SDK AND USING SESSION POOL ....... 9
   4.6 EXAMPLE / TEST PROGRAMS ........................................................................................... 11

5. SDK OVERVIEW .................................................................................................................. 15
   5.1 SDK DIRECTORIES .......................................................................................................... 16
   5.2 SDK PACKAGES .............................................................................................................. 16

6. SDK CONFIGURATION ........................................................................................................... 18
   6.1 CONFIGURATION FILE .................................................................................................... 18
   6.2 LIBRARIES .................................................................................................................... 27
   6.3 DIAGNOSTIC AND ERROR LOGGING .............................................................................. 28
       6.3.1 Basic Configuration Mode (EPP.LogLevel and EPP.LogFile) ........................................... 28
       6.3.2 Log4J Configuration File Mode (EPP.LogCfgFile and EPP.LogCfgFileWatch) .................. 29
       6.3.3 Custom Mode ............................................................................................................ 30
       6.3.4 SDK Log Categories ............................................................................................... 30
   6.4 ADDING AN EPP COMMAND MAPPING TO THE SDK .................................................... 31

7. GENERIC EPP CLIENT INTERFACES ..................................................................................... 33
   7.1 EPPAPPLICATION .......................................................................................................... 33
   7.2 EPPSESSION .................................................................................................................. 34
       7.2.1 Overview .................................................................................................................. 34
       7.2.2 Sample Code ............................................................................................................ 36
       7.2.3 initSession() Method ............................................................................................. 38
       7.2.4 endSession() Method ............................................................................................. 39
       7.2.5 hello() Method ......................................................................................................... 40
       7.2.6 sendPoll() Method ................................................................................................. 41
   7.3 EPPENV .......................................................................................................................... 42

8. XML PARSER POOL ............................................................................................................. 45

9. EXTENDING THE SDK ......................................................................................................... 47
   9.1 TRANSPORT ................................................................................................................... 47

10. STUB SERVER ...................................................................................................................... 50

VeriSign Inc. Proprietary Information
Verisign Bundle EPP SDK Programmer's Guide • page vii
# 11. CLIENT IMPLEMENTATION NOTES

## 11.1 Pooling

### 11.1.1 Multiple Session Pools

### 11.1.2 Separate SSL Configuration Per Session Pool

## 11.2 Threading

## 11.3 Pipelining

# 12. POLL MESSAGES

# 13. MAPPINGS AND EXTENSIONS

## 13.4 Namestore Client Interfaces

## 13.5 Mappings

### 13.5.1 Domain Mapping (NSDomain Interface)

### 13.5.2 Host Mapping (NSHost Interface)

### 13.5.3 Contact Mapping (NSContact Interface)

### 13.5.4 Registry Mapping

### 13.5.5 Suggestion Mapping

### 13.5.6 WhoWas Mapping

### 13.5.7 Balance Mapping

### 13.5.8 EmailFwd Mapping

### 13.5.9 DefReg Mapping

### 13.5.10 NameWatch Mapping

## Extensions

### 13.5.11 NamestoreExt Extension

### 13.5.12 Whois Info Extension

### 13.5.13 SecDNS Extension

### 13.5.14 COA Extension

### 13.5.15 Premium Domain Extension

### 13.5.16 DotJobs Contact Extension

### 13.5.17 Launch Extension

### 13.5.18 PersReg Extension

### 13.5.19 Related Domain Extension
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, .NET, and IDN .COM / .NET TLDs. The SRS commands require the NameStore Extension for specifying the target logical 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 logical 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.
   b. Name Suggestion - Provides highly related domain suggestion.
   c. WhoWas – Provides history records of entities.

3. Dotname Registry System, hereon referred to as Dotname, supports the .NAME TLD.

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 convinience sub-classes 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 1.0.0

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 |
4.1 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
```

4.2 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></td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------</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>com.verisign.epp.codec.host.EPPHostMapFactory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>com.verisign.epp.codec.hostname.EPPNamestoreExt.EPPNamestoreExtFactory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPP.SSLKeyName</td>
<td>JSSE keystore file name that contains the CA issued certificate chain and the associated private key.</td>
<td>./.../lib/keystore/testkeys</td>
</tr>
<tr>
<td>EPP.SSLPassPhrase</td>
<td>Password needed to access EPP.SSLKeyName file.</td>
<td>passphrase</td>
</tr>
<tr>
<td>EPP.SSLKeyPassPhrase</td>
<td>Password needed to access the private key defined in the EPP.SSLKeyName 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</td>
<td>TLS</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/testkeys</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>Variable Name</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>EPP.ClientSocketName</td>
<td>Class used to make client connections.</td>
<td>com.verisign.epp.transporthandler.EPPClientSocket</td>
</tr>
<tr>
<td></td>
<td><strong>Set this to</strong> com.verisign.epp.transport.client.EPPSSLClientSocket for SSL.</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>
<tr>
<td></td>
<td>1. NameStore OTE – otessl.verisign-grs.com</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. NameStore Production – namestoressl.verisign-grs.com</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. SRS OTE – epp-ote.verisign-grs.com</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. SRS Production – epp.epersign-grs.net</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5. NAME OTE - nameeppote.verisign-grs.com</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6. NAME Production - nameepp.verisign-grs.net</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Recommend configuring and using the session pooling feature defined in section 11.1 to interface with multiple Verisign servers with a single client.</strong></td>
<td></td>
</tr>
<tr>
<td>EPP.ServerPort</td>
<td>Set to the Verisign server port number, which should be 700.</td>
<td>1700</td>
</tr>
</tbody>
</table>

**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.

### 4.3 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

### 4.4 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.
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.setPassword("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);
        }
    }
}

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

The example provided in section 4.4 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 11.1 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.
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();
4.6 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>
<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 domain create with COA, and adding, changing and removing COAs using domain updates. Also tests using the domain info command to retrieve a COA from an existing owned domain.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces.EPPContactTst</td>
<td>Test of using the EPPContact interface for all of the RFC 5733 contact commands. This test creates an individual EPP session without the SDK session pool.</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 the RFC 5733 contact commands along with the</td>
</tr>
<tr>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</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 NameStore Extension (EPPNamestoreExtNamestoreExt). 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.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>com.verisign.epp.interfaces.EPPSyncDomainTst</td>
<td>Test of using ConsoliDate extension (EPPSyncExtUpdate) to synchronize the expiration date of a domain. This test creates an individual EPP session without the SDK session pool.</td>
</tr>
<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>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</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 (default and “test”).</td>
</tr>
</tbody>
</table>
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 9.1 - 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 6.3 - Diagnostic and Error Logging.
5.1 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>

5.2 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 Packages provides an overview of the SDK packages.

Table 5 - Verisign Bundle EPP SDK 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>com.verisign.epp.interfaces</td>
<td>Client interface classes including EPPApplication and</td>
</tr>
<tr>
<td>Package</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</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

6.1 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.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>Variable</th>
<th>Access</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPP.LogLevel</td>
<td>Both</td>
<td>No</td>
<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>
</tr>
<tr>
<td>EPP.LogFile</td>
<td>Both</td>
<td>No</td>
<td>Log4J Log File Name. Logs will be appended to the log file. The default file is “epp.log”.</td>
</tr>
<tr>
<td>EPP.LogCfgFile</td>
<td>Both</td>
<td>No</td>
<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>
</tr>
<tr>
<td>EPP.LogCfgFileWatch</td>
<td>Both</td>
<td>No</td>
<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>
</tr>
<tr>
<td>EPP.ConTimeOut</td>
<td>Both</td>
<td>Yes</td>
<td>Connection and read timeout in milliseconds. A setting of 0 indicates no timeout. The default setting is 500000 milliseconds (500 seconds).</td>
</tr>
<tr>
<td>EPP.ClientSocketName</td>
<td>Client</td>
<td>No</td>
<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: com.verisign.epp.transport.client.EPPPlainClientSocket - Plain TCP/IP socket connection(s) com.verisign.epp.transport.client.EPPSSLClientSocket - SSL TCP/IP socket connection(s) com.verisign.epp.transport.client.EPPPlainProxyClientSocket - Plain TCP/IP socket connection(s) that connects through an Apache Proxy Server (mod_proxy).</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| EPP.ProxyServersLocator  | Client  | No      | 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.                                                            |
| EPP.ProxyServers         | Client  | No      | 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: `(proxy server):<port number>,<proxy server>:<port number>...`<proxy server> ::= '[?<ip address> | <host name>]'
<p>|                          |         |         | An example value of for connecting to the local Apache Server using the host name, IPv4 address, and IPv6 address is “localhost:80,127.0.0.1:80,::1:80”.                                                             |
| EPP.ProxyServersRandomize| Client  | No      | 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. |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>EPP.SSLProtocol</td>
<td>Both</td>
<td>Yes</td>
<td>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</td>
</tr>
<tr>
<td>EPP.SSLKeyStore</td>
<td>Both</td>
<td>No</td>
<td>JSSE KeyStore format. The default setting is “JKS”.</td>
</tr>
<tr>
<td>EPP.SSLKeyFileName</td>
<td>Both</td>
<td>No</td>
<td>JSSE KeyStore file used for authentication. The SDK</td>
</tr>
</tbody>
</table>
includes a self-signed certificate in the KeyStore “testkeys”.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPP.SSLPassPhrase</td>
<td>Both</td>
<td>No</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>No</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>No</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 of protocols include:</td>
</tr>
</tbody>
</table>
|                                 |          |         | • SSL - Supports some version of SSL  
• SSLv2 - Supports SSL version 2 or higher  
• SSLv3 - Supports SSL version 3  
• TLS - Supports some version of TLS  
• TLSv1 - Supports TLS version 1 |
| EPP.SSLEnabledCipherSuites     | Both     | No      | Enabled Cipher Suites. Space delimited list of cipher suites. Examples include:                                                                                                                            |
|                                 |          |         | • SSL_RSA_WITH_RC4_128_MD5  
• SSL_RSA_WITH_RC4_128_SHA |
| EPP.SSLTrustStoreFileName      | Both     | No      | 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 |
|                                 |          |         | It is recommended to comment out or not define this property when connecting to the NameStore or SRS Servers.                                                                                                  |
| EPP.SSLTrustStorePassPhrase    | Both     | No      | JSSE TrustStore pass-phrase. This property is required if EPP.SSLTrustStoreFileName is defined.                                                                                                           |
| javax.net.debug                | Both     | No      | JSSE debug options. This is very useful for debugging SSL handshaking issues. The possible values include:                                                                                                   |
|                                 |          |         | • none – No debug  
• all – All debug |
<p>|                                 |          |         | This property will set the javax.net.debug System property.                                                                                                                                             |
| PoolMan.[Client/Server].logFile| Client/Server | No     | Log file to write debug messages, if PoolMan.Client.debugging is true. Default value is System.out.                                                                                                        |
| PoolMan.[Client/Server].initialObjects | Client/Server | No     | Initial number of objects to create in the pool. Default value is 1.                                                                                                                                       |
| PoolMan.[Client/Server].      | Client/Server | No     | Minimum number of objects that can be in the pool. Default value is 1.                                                                                                                                      |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimumSize</td>
<td>er</td>
<td>value is 0.</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].maximumSize</td>
<td>Client/Serv</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].maximumSoft</td>
<td>Client/Serv</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].shrinkBy</td>
<td>Client/Serv</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].debugging</td>
<td>Client/Serv</td>
<td>No</td>
<td>Write debug messages? Debug messages are written to the file specified by the PoolMan.Client.logFile parameter. Default is false.</td>
</tr>
<tr>
<td>PoolMan.[Client/Server].objectTimeout</td>
<td>Client/Serv</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].userTimeout</td>
<td>Client/Serv</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/Serv</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/Serv</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>EPP.MaxPacketSize</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>EPP.SessionPool.poolableClassName</td>
<td>Client</td>
<td>No</td>
<td>Fully qualified class name of object pooled by the EPPSessionPool. Set to</td>
</tr>
<tr>
<td>Value</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.clientId</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>EPP.SessionPool.password</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>EPP.SessionPool.absolute</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>EPP.SessionPool.idleTime</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>EPP.SessionPool.minIdle</td>
<td>Client</td>
<td>No</td>
<td>Minimum number of idle sessions in the session pool. Default is 5</td>
</tr>
<tr>
<td>EPP.SessionPool.maxIdle</td>
<td>Client</td>
<td>No</td>
<td>Maximum number of idle sessions in the session pool. Default is 10</td>
</tr>
<tr>
<td>EPP.SessionPool.maxActive</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>EPP.SessionPool.initMaxActive</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>EPP.SessionPool.borrowRetries</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 EPP.SessionPool.initMaxActive is true. Default is 0</td>
</tr>
<tr>
<td>EPP.SessionPool.maxWait</td>
<td>Client</td>
<td>No</td>
<td>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>Client</td>
<td>No</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>Client</td>
<td>No</td>
<td>Default the set of system session pools in a comma separated list of names (i.e. srs, namestore). The system</td>
</tr>
</tbody>
</table>
name “default” initializes a default pool that uses the EPP.SessionPool.<param> property along with other properties like EPP.ServerName and EPP.ServerPort. The default pool uses the EPPSessionPool methods that don’t take a <System> parameter.

<table>
<thead>
<tr>
<th>Property</th>
<th>Client</th>
<th>No</th>
<th>System specific setting of EPP.SessionPool.&lt;system&gt;, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.poolableClassName</td>
<td>Client</td>
<td>No</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;.serverName</td>
<td>Client</td>
<td>No</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>Client</td>
<td>No</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>Client</td>
<td>No</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>Client</td>
<td>No</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>Client</td>
<td>No</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>Client</td>
<td>No</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>Client</td>
<td>No</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>Client</td>
<td>No</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>Client</td>
<td>No</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>EPP.SessionPool.&lt;system&gt;.maxActive</td>
<td>Client</td>
<td>No</td>
<td>System specific setting of EPP.SessionPool.maxActive, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.initMaxActive</td>
<td>Client</td>
<td>No</td>
<td>System specific setting of EPP.SessionPool.initMaxActive, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.borrowRetries</td>
<td>Client</td>
<td>No</td>
<td>System specific setting of EPP.SessionPool.borrowRetries, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.maxWait</td>
<td>Client</td>
<td>No</td>
<td>System specific setting of EPP.SessionPool.maxWait, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.timeBetweenEvictionRunsMillis</td>
<td>Client</td>
<td>No</td>
<td>System specific setting of EPP.SessionPool.timeBetweenEvictionRunsMillis, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore)</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLProtocol</td>
<td>Client</td>
<td>No</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 TLS</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLKeyStore</td>
<td>Client</td>
<td>No</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>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLKeyFileName</td>
<td>Client</td>
<td>No</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>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLPassPhrase</td>
<td>Client</td>
<td>No</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>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLEnabledProtocols</td>
<td>Client</td>
<td>No</td>
<td>Optional space delimited list of enabled SSL protocols, where &lt;system&gt; is replaced with the system name (i.e. srs, namestore). Examples include: SSL_RSA_WITH_RC4_128_MD5, SSL_RSA_WITH_RC4_128_SHA</td>
</tr>
<tr>
<td>EPP.SessionPool.&lt;system&gt;.SSLEnabledCipherSuites</td>
<td>Client</td>
<td>No</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: SSL_RSA_WITH_RC4_128_MD5, SSL_RSA_WITH_RC4_128_SHA</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 then 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</td>
</tr>
</tbody>
</table>
property when connecting to the NameStore or SRS Servers.

EPP.SessionPool.<system>.SSLTrustStorePassPhrase
Client No Pool passphrase/password to access the Trust Store file defined by the pool SSLTrustStoreFileName property.

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

Table 6 - SDK Configuration File Parameters

6.2 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 1.7.1 – .jar files in epp-verisign-$\{BUILD_VER\}/lib/ant
• junit-3.8.1.jar – Library used for SDK tests
• log4j-1.2.8.jar – Library used for SDK diagnostic and error logging
• XercesJ 2.11.0 – Library used for XML parsing. This includes xercesImpl-2.11.0.jar and xmlParserAPIs-2.11.0.jar
• poolman-2.1-b1.jar – Library used for pooling the XML Parsers and Transformers
• Jalopy 1.0b10 – Library used for code formatting. .jar files in epp-verisign-${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 2.5.1 – 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.

6.3 Diagnostic and Error Logging
The Verisign Bundle EPP SDK uses Log4J(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.

6.3.1 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{yyyyMMMd HHmmss} %c %-5p %m%n", which will log the date in the
format “yyyyMMdd HHmms”, the category as the fully qualified class name, the priority (DEBUG, INFO, WARN, ERROR, or FATAL), and the log message.

6.3.2 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 HHmms”, 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 HHmms} %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" />
</log4j:configuration>
```
6.3.3 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.

6.3.4 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.EPPSchemaCachingEntityResolver</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</td>
</tr>
<tr>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</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>
<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>

Table 7 - SDK Log Categories

6.4 Adding an EPP Command Mapping to the SDK
The SDK can be easily extended to support new EPP Command Mappings. As described in section 6.2, 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.ServerEventHandlers` 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.

7.1 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

7.2 EPPSession

7.2.1 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.
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.

### 7.2.2 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);
        }
    }
}

Figure 5 - EPPSession Life Cycle Sample Code
7.2.3 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

7.2.4 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>
</table>
| EPPResult.SUCCESS_END_SESSION          | 1500           | Session has successfully been closed. endSession will not throw an exception, and the successful response can be retrieved by calling `getResponse()`.
| EPPResult.COMMAND_SYNTAX_ERROR         | 2001           | Malformed EPP message.                                                      |
| EPPResult.COMMAND_USE_ERROR            | 2002           | Session has not been established.                                           |
| EPPResult.PARAM_OUT_OF_RANGE           | 2004           | Input attribute not valid                                                   |
| EPPResult.COMMAND_FAILED               | 2500           | Internal EPP Server error                                                   |
| EPPResult.TIMEOUT_END                  | 2501           | Command or session timeout has occurred. The EPP Server closes the connection.|

Table 9 - EPPSession.initSession EPP Status Code Matrix

### 7.2.5 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.

**7.2.6 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>
EPPResult.SUCCESS_POLL_MSG 1301 Successful Poll request command. A poll and a queue size is returned.

EPPResult.COMMAND_SYNTAX_ERROR 2001 Malformed EPP message.

EPPResult.COMMAND_USE_ERROR 2002 Session has not been established.

EPPResult.PARAM_OUT_OF_RANGE 2004 Input attribute not valid

EPPResult.COMMAND_FAILED 2500 Internal EPP Server error

EPPResult.TIMEOUT_END 2501 Command or session timeout has occurred. The EPP Server closes the connection.

Table 10 - EPPSession.sendPoll EPP Status Code Matrix

7.3 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.
Figure 6 - EPPEnv Class Diagram

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</td>
<td>Class</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>
getServerSocketName() | EPP.ServerSocketName
---|---
getSSLEnabledCipherSuites() | EPP.SSLEnabledCipherSuites
getSSLEnabledProtocol() | EPP.SSLEnabledProtocols
getSSLKeyFileName() | EPP.SSLKeyFileName
getSSLKeyManager() | EPP.SSLKeyManager
getSSLKeyPassPhrase() | EPP.SSLKeyPassPhrase
getSSLPassPhrase() | EPP.SSLPassPhrase
getSSLPassPhrase() | EPP.SSLPassPhrase
getSSLProtocol() | EPP.SSLProtocol
getSSLTrustStoreFileName() | EPP.SSLTrustStoreFileName
getSSLTrustStorePassPhrase() | EPP.SSLTrustStorePassPhrase
getValidating() | EPP.Validating

### Table 11 - EPPEnv Method Mappings

#### 8. XML Parser Pool

The XML Parser included in the SDK is XercesJ ([http://xml.apache.org/xerces-j/index.html](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/](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

9.1 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.

```java
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;

} // End class EPPClientConSample

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.

10.1 **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.

10.2 **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

11.1 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 {
```

VeriSign Inc. Proprietary Information

Verisign Bundle EPP SDK Programmer’s Guide
if (theSession != null)
    EPPSessionPool.getInstance().returnObject(theSession);

11.1.1 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”.

```
EPP.SessionPool.systemPools=default,test
EPP.SessionPool.test.poolableClassName=com.verisign.epp.pool.EPPSessionPoolableFactory
EPP.SessionPool.test.serverName=localhost
EPP.SessionPool.test.serverPort=1700
EPP.SessionPool.test.clientId=username
EPP.SessionPool.test.password=password
EPP.SessionPool.test.absoluteTimeout=8280000
EPP.SessionPool.test.idleTimeout=480000
EPP.SessionPool.test.minIdle=0
EPP.SessionPool.test.maxIdle=-1
EPP.SessionPool.test.maxActive=10
EPP.SessionPool.test.initMaxActive=true
EPP.SessionPool.test.borrowRetries=3
EPP.SessionPool.test.maxWait=60000
EPP.SessionPool.test.timeBetweenEvictionRunsMillis=500
```

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

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

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

The example below illustrates the same session pool sample using the “default” system instead of using the equivalent default system methods of EPPSessionPool.
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);
}

11.1.2 Separate SSL Configuration Per Session Pool

Section 11.1.1 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/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

11.2 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.

11.3 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 11.1, 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);
    }
}
theSession = null;
}
}
finally {
    if (theSession != null) {
        theSession.setMode(EPPSession.MODE_SYNC);
        EPPSessionPool.getInstance().returnObject(theSession);
    }
}

12. Poll Messages
The servers can send different kinds 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>Namestore</th>
<th>NAME</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>Class Name</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:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>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</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>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,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cancelled, approved, rejected, and auto approved.</td>
<td></td>
</tr>
<tr>
<td>com.verisign.epp.codec.rgpoll.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 the 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 three Verisign EPP systems (Namestore Platform - Namestore, COM/NET Shared Registry System - SRS, and Dotname Registry System - NAME) along with a short description. The name of the Mapping and Extension corresponds to the directory name included in the source distribution.

Table 13 - Mapping and Extension System Support

<table>
<thead>
<tr>
<th>Mapping / Extension</th>
<th>Namestore</th>
<th>SRS</th>
<th>NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coa</td>
<td></td>
<td>X</td>
<td></td>
<td>Client Object Attribute Extension to allow the creation and maintenance of key/value pairs associated with Objects.</td>
</tr>
<tr>
<td>contact</td>
<td></td>
<td>X</td>
<td>X</td>
<td>Standard IETF Contact Mapping</td>
</tr>
<tr>
<td>defreg</td>
<td></td>
<td></td>
<td>X</td>
<td>Defensive Registration Mapping</td>
</tr>
<tr>
<td>domain</td>
<td></td>
<td>X</td>
<td>X</td>
<td>Standard IETF</td>
</tr>
<tr>
<td>Domain Mapping</td>
<td>emailfwd</td>
<td>host</td>
<td>idn</td>
<td>jobsContact</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------</td>
<td>-------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>Email Forwarding Mapping</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard IETF Host Mapping</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Extension for the International Domain Name Tag required for IDN domain creates.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extension for .JOBS specific contact attributes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard IETF Launch Phase EPP Extension</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Namestore Extension needed to specify the target sub-product for a command.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NameWatch Mapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low Balance Poll Mapping and Balance Mapping</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Personal Registration Extension</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>premiumdomain</td>
<td>X</td>
<td></td>
<td></td>
<td>Extensions to the domain check command, domain check response and domain update command to support premium features.</td>
</tr>
<tr>
<td>---------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>registry</td>
<td>X</td>
<td>X*</td>
<td>X*</td>
<td>Registry Mapping</td>
</tr>
<tr>
<td>relateddomain</td>
<td></td>
<td></td>
<td>X*</td>
<td>Related Domain Extension</td>
</tr>
<tr>
<td>rgp</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Standard IETF Redemption Grace Period Extension and RGP poll message.</td>
</tr>
<tr>
<td>secdns</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Domain Name System Security Extension to provide additional features required for the provisioning of DNS</td>
</tr>
<tr>
<td>suggestion</td>
<td>X</td>
<td></td>
<td></td>
<td>Name Suggestion Mapping</td>
</tr>
<tr>
<td>sync</td>
<td>X</td>
<td>X</td>
<td>X*</td>
<td>Extension to support</td>
</tr>
</tbody>
</table>
The domain sync command defined in the ConsoliDat e Mapping Extension to the domain info command and domain info response to specify if whois information is desired and the whois attributes, respectively. The whois attributes include the registrar name, the whois server name, the registrar referral URL, and the IRIS server name.

<table>
<thead>
<tr>
<th>whois</th>
<th>X</th>
<th>X</th>
<th>the domain sync command defined in the ConsoliDat e Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>whoas</td>
<td>X</td>
<td></td>
<td>WhoWas Mapping</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</td>
</tr>
<tr>
<td></td>
<td>execute the test programs.</td>
</tr>
<tr>
<td>build.sh</td>
<td>This is a Bash shell script that executes Ant to compile and</td>
</tr>
<tr>
<td></td>
<td>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 6.1 for information</td>
</tr>
<tr>
<td></td>
<td>on configuring the SDK.</td>
</tr>
<tr>
<td>common-</td>
<td>Common Ant targets used by both the source and binary</td>
</tr>
<tr>
<td>targets.xml</td>
<td>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.coaext</td>
<td>The COA Extension Encoder/Decoder package. All of the detail of encoding</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and decoding the COA Extension is in this package.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The EPPCoaExtFactory must be added to the EPP.CmdRspExtensions configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
<td>encoding and decoding the EPP Contact messages are in this package.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The EPPContactMapFactory must be added to the EPP.MapFactories configuration</td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameter using the full package and class name.</td>
</tr>
<tr>
<td>defreg</td>
<td>com.verisign.epp.codec.defreg</td>
<td>The EPP Defensive Registration Encoder/Decoder</td>
</tr>
<tr>
<td>Package</td>
<td>Class Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>g</td>
<td></td>
<td>All the detail of encoding and decoding the EPP 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>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.idnnext</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.jobscontact</td>
<td>The .JOBS Contact Extension Encoder/Decoder package. All of the detail of encoding and decoding the .JOBS Contact 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>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</td>
</tr>
</tbody>
</table>
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.

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

<table>
<thead>
<tr>
<th>Package</th>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namestore Ext</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 EPPLNamestoreExtExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>NameWatch Ext</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 EPPLNameWatchMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>Balance Ext</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 EPPLBalanceMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>Low Balance Poll Ext</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 EPPLLowBalancePollMapFactory must be added to the EPP.MapFactories configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>PersReg Ext</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 EPPLPersRegExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.</td>
</tr>
<tr>
<td>Package</td>
<td>Full Package Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------</td>
<td>-----------------------------------------------------------------------------</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>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 Balance 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.rgpoll</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.secdnsnext</td>
<td>The SecDNS Extension Encoder/Decoder package. All of the detail of encoding and decoding the SecDNS Extension is in this package.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Component</th>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>sync</td>
<td>com.verisign.epp.codec.sync</td>
<td>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.</td>
</tr>
<tr>
<td>whois</td>
<td>com.verisign.epp.codec.whois</td>
<td>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.</td>
</tr>
<tr>
<td>whowas</td>
<td>com.verisign.epp.codec.whowas</td>
<td>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.</td>
</tr>
<tr>
<td>File Name</td>
<td>Location</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------</td>
<td>-------------------------------------------------</td>
</tr>
<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>emailfwd-1.0.xsd</td>
<td>emailfwd/schemas</td>
<td>Email Forwarding XML Schema.</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>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>xmlmdsig-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>persReg-1.0.xsd</td>
<td>persreg/schemas</td>
<td>Personal Registration Extension XML Schema.</td>
</tr>
<tr>
<td>relatedDomain-1.0.xsd</td>
<td>relateddomain/schemas</td>
<td>Related Domain Extension XML Schema.</td>
</tr>
<tr>
<td>rgp-poll-1.0.xsd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sync-1.0.xsd</td>
<td>sync/schemas</td>
<td>Standard EPP ConsoliDate Mapping XML Schema.</td>
</tr>
</tbody>
</table>
**13.4 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 13.5.1.

- `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 13.5.2.

- `com.verisign.epp.namestoreinterfaces.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 13.5.3.

- `com.verisign.epp.namestore.interfaces.NSSubProduct` – Set of constants that can be used for the SubProductID in the Namestore Extension. For new TLD’s, the TLD name can be used for the SubProductID.
13.5 Mappings

13.5.1 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>NSDomain(EPPSession aNewSession)</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>void</td>
<td>addDomainName(java.lang.String)</td>
</tr>
<tr>
<td>void</td>
<td>addExtension(EPPCodecComponent)</td>
</tr>
<tr>
<td>void</td>
<td>addHostName(java.lang.String)</td>
</tr>
<tr>
<td>void</td>
<td>addContact(String, String)</td>
</tr>
<tr>
<td>EPPResponse</td>
<td>getResponse()</td>
</tr>
<tr>
<td>Date</td>
<td>getExpirationDate()</td>
</tr>
<tr>
<td>Vector</td>
<td>getExtensions()</td>
</tr>
<tr>
<td>Boolean</td>
<td>hasExtension(Class)</td>
</tr>
<tr>
<td>EPPResponse</td>
<td>sendCheck()</td>
</tr>
<tr>
<td>EPPResponse</td>
<td>sendCreate()</td>
</tr>
<tr>
<td>EPPResponse</td>
<td>sendDelete()</td>
</tr>
<tr>
<td>EPPResponse</td>
<td>sendInfo()</td>
</tr>
<tr>
<td>EPPResponse</td>
<td>sendUpdate()</td>
</tr>
</tbody>
</table>

This method returns the EPP Response for the last executed command on the interface.

Returns the expiration date of the domain.

Gets all set command extensions.

Does a command extension exist with the specified Class.

This method sends the domain check command to the server.

This method sends the domain create command to the server.

This method sends the domain delete command to the server.

This method sends the domain info command to the server.

This method sends the domain update command to the server.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPResp  onse sendRenew()</td>
<td>This method sends the domain renewal command to the server.</td>
</tr>
<tr>
<td>EPPResp  onse sendTransfer()</td>
<td>This method sends the domain transfer command to the server.</td>
</tr>
<tr>
<td>EPPResp  onse sendRestoreRequest()</td>
<td>This method sends the Registry Grace Period (RGP) restore request command.</td>
</tr>
<tr>
<td>EPPResp  onse sendRestoreReport()</td>
<td>This method sends the Registry Grace Period (RGP) restore report command.</td>
</tr>
<tr>
<td>EPPResp  onse sendSync()</td>
<td>This method sends the ConsoliDate sync command.</td>
</tr>
<tr>
<td>void setTransferOpCode(String)</td>
<td>Sets the operation type for future transfer commands. Valid values are “TRANSFER_REQUEST”, “TRANSFER_APPROVE” or “TRANSFER_REJECT”</td>
</tr>
<tr>
<td>void setTransId(String)</td>
<td>This methods sets the client transaction identifier.</td>
</tr>
<tr>
<td>void setDay(int)</td>
<td>Sets the target day for a call to sendSync.</td>
</tr>
<tr>
<td>void setMonth(int)</td>
<td>Sets the target month using the Calendar month constants (Calendar.JANUARY to Calendar.DECEMBER) for a call to sendSync.</td>
</tr>
<tr>
<td>void setIDNLangTag(String)</td>
<td>Sets the IDN language tag for a call to sendCreate() of an IDN domain.</td>
</tr>
<tr>
<td>void setSubProductID(String)</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 setSubProductID argument value.</td>
</tr>
<tr>
<td>void setPeriodLength(int)</td>
<td>Sets the registration period for a create, renew, or transfer command. The default value is 1 year.</td>
</tr>
<tr>
<td>void setPeriodUnit(String)</td>
<td>Sets the unit of the registration period as defined by setPeriodLength(int) according to the EPP specification. The servers currently only support the default value of “y” for years.</td>
</tr>
<tr>
<td>void setExpirationDate(Date)</td>
<td>Sets the current expiration date for a call to sendRenew().</td>
</tr>
<tr>
<td>void setUpdateAttrib(...)</td>
<td>Sets attribute to update for a call to sendUpdate().</td>
</tr>
<tr>
<td>void setAuthString(String)</td>
<td>Sets authorization string for a call to sendCreate(), sendTransfer(), or sendInfo()</td>
</tr>
<tr>
<td>void setRegistrant(String)</td>
<td>Sets the domain registrant for a call to sendCreate() or sendUpdate().</td>
</tr>
<tr>
<td>void setHosts(String)</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.
```

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.

### 13.5.1.1 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.

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

#### 13.5.1.1.2 Post-Conditions
The NSDomain instance is ready for the execution of one or more operations.

#### 13.5.1.1.3 Exceptions
None

#### 13.5.1.1.4 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);
}

NSDomain domain = new NSDomain(session);
```
13.5.1.2  sendCheck() method

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

13.5.1.2.1  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)

13.5.1.2.2  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.

13.5.1.2.3  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.

13.5.1.2.4  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:
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++) {
            EPPPremiumDomainCheckResult currResult =
                (EPPPremiumDomainCheckResult) 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");
            }
        }
    }
}
13.5.1.3 sendCreate() method

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

13.5.1.3.1 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 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.

13.5.1.3.2 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.
13.5.1.3.3 **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`.

13.5.1.3.4 **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()
        .setFlags(EPPSecDNSExtKeyData.FLAGS_ZONE_KEY_SEP)
        .setAlg(EPPSecDNSAlgorithm.RSASHA1)
        .setPubKey("AQPmsXx3Q1ngNSzsH1lrx63mR1htwkkK+5zj" + "vxykBCViNYne83+8RXkBElGb/YJln4TacMUs" + "po7a7caj7MdOaADKmzb2ic0vwpNyW0t2" + "AnaQqy1ce+07Y8RkbTC6xCeEw1UQ273PzIO" + "0vJDdjmFwWa09F7zSxnGpt0WtulTQ==");

    // instantiate another secDNS:KeyData object
    EPPSecDNSExtKeyData keyData2 = new EPPSecDNSExtKeyData()
        .setFlags(EPPSecDNSExtKeyData.FLAGS_ZONE_KEY_SEP)
        .setAlg(EPPSecDNSAlgorithm.RSASHA1)
        .setPubKey("AQOxXpFbRp7+zPBoTt6zL7Af0aEKzpS4JbVB" + "5ofk5E5HpXuUmU+Hnt9hm2Km6LZdEEL4" + "qo0HrgiETPCsN/YM4Zn+meKELLipCG93Cu/H" + "hwvxfazNeUAA6VB9FwXQlEMYRW05k/gh2Ge" + "w5Sk/0o6Ev7DKG2Y1D7JA17QsaZtFw==");

    // instantiate a secDNS:DsData object
    EPPSecDNSExtDsData dsData = new EPPSecDNSExtDsData()
        .setKeyTag(34095)
        .setAlg(EPPSecDNSAlgorithm.RSASHA1)
        .setDigestType(EPPSecDNSExtDsData.SHA1_DIGEST_TYPE)
        .setDigest("6BD4FFFF11566D6E6A5BA44ED0018797564AAA289")
        .setMaxSigLife(604800)
        .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

13.5.1.4 sendDelete() method

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

13.5.1.4.1 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.

13.5.1.4.2 Post-Conditions

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

13.5.1.4.3 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.

13.5.1.4.4 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
```

13.5.1.5 sendInfo() method

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

13.5.1.5.1 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.veriisign.epp.codec.whois.EPPWhoisInfData class.

13.5.1.5.2 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.veriisign.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.veriisign.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.

13.5.1.5.3 **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`.

13.5.1.5.4 **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:

```java
```
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:
            
        }
    }
}
13.5.1.6 sendUpdate() method

The sendUpdate() method sends the EPP update domain command to the server.
13.5.1.6.1 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 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.)

13.5.1.6.2 Post-Conditions

On success, an EPPResponse is returned, with no attributes.
13.5.1.6.3 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`.

13.5.1.6.4 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
```

13.5.1.7 sendRenew() method

The `sendRenew()` method sends the EPP renew domain command to the server.
13.5.1.7.1  **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

13.5.1.7.2  **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.

13.5.1.7.3  **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.

13.5.1.7.4  **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

13.5.1.8 **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.

13.5.1.8.1 **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”

13.5.1.8.2  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.

13.5.1.8.3  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.

13.5.1.8.4  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

13.5.1.9 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.

13.5.1.9.1 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.

13.5.1.9.2 Post-Conditions

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

13.5.1.9.3 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.

13.5.1.9.4 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

13.5.1.10 sendRestoreReport() method

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

13.5.1.10.1 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.

13.5.1.10.2 Post-Conditions

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

13.5.1.10.3 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.
13.5.1.10.4 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:

```java
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();
```
13.5.11 sendSync() method

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

13.5.11.1 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.

13.5.11.2 Post-Conditions

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

13.5.11.3 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.

13.5.11.4 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 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
### 13.5.2 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>
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.

### 13.5.2.1 `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.

#### 13.5.2.1.1 Pre-Conditions

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

#### 13.5.2.1.2 Post-Conditions

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

#### 13.5.2.1.3 Exceptions

None

#### 13.5.2.1.4 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");
```
13.5.2.2  sendCheck() method

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

13.5.2.2.1  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.

13.5.2.2.2  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.

13.5.2.2.3  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 Namestore server response, then `getResponse()` will return `null`.

13.5.2.2.4  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:
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

13.5.2.3 sendCreate() method

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

13.5.2.3.1 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.
13.5.2.3.2 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.

13.5.2.3.3 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 Namestore server response, then getResponse() will return null.

13.5.2.3.4 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 cmdExc) {
    // do something to handle the exception
    handleException(cmdExc);
} // end of catch block
```

13.5.2.4 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.

13.5.2.4.1 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.

13.5.2.4.2 Post-Conditions

On success, a standard EPPResponse is returned.

13.5.2.4.3 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 Namestore server response, then getResponse() will return null.

13.5.2.4.4 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
```

13.5.2.5 sendInfo() method

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

13.5.2.5.1 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.

13.5.2.5.2 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.

13.5.2.5.3 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 Namestore server response, than `getResponse()` will return `null`.

13.5.2.5.4 Sample Code

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

13.5.2.6 sendUpdate() method

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

13.5.2.6.1 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.
13.5.2.6.2 Post-Conditions

On success, a standard EPPResponse is returned.

13.5.2.6.3 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 Namestore server response, then getResponse() will return null.

13.5.2.6.4 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.setTransId("ABC-12345-XYZ");
    host.addHostName("ns.myhost.net");
    host.setSubProductID(NSSubProduct.NET);

    // set new IP address
    host.addIPV4Address("123.34.34.12");
    // remove old IP address
    host.removeIPV4Address("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
```

13.5.2.7 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 sendInfo() method for encapsulating Ipv4 and Ipv6 addresses.</td>
</tr>
<tr>
<td>EPPHostCheckResult</td>
<td>This class is used by the sendCheck() method for returning the allowable flag for multiple hosts being checked.</td>
</tr>
</tbody>
</table>
13.5.2.8 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.

13.5.2.9 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.
13.5.3 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.
13.5.3.1 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&gt;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>
13.5.3.2 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>
13.5.3.3 Contact Client Interface

13.5.3.3.1 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().

```
EPPContact

+ EPPContac{newSession: EPPSession};
+ addExtension: (EPPCodeComponent) void
+ setExtension: (EPPCodeComponent) void
+ addPostInfo: (java.util.Vector) void
+ setPostInfo: (java.util.Vector) void
+ addPostalContact: (EPPPostalContact) void
+ setPostalContact: (EPPPostalContact) void
+ addEPPContact: (EPPContact) void
+ setName: (java.lang.String) void
+ addAttribute: (EPPAttribute) void
+ removeAttribute: (EPPAttribute) void
+ addAuthorization: (EPPAuthorization) void
+ removeAuthorization: (EPPAuthorization) void
+ addTransfer: (EPPTransfer) void
+ removeTransfer: (EPPTransfer) void
+ getHost(): String
+ getPort(): String
+ getTimeout(): long
+ getSendTimeout(): long
+ getTrace(): boolean
+ getTraceFile(): String
+ getLog(): String
+ getEncoding(): String
+ getTraceEncoding(): String
+ getLogEncoding(): String
+ getResponse(): EPPResponse
+ sendCreate(): EPPCreateResp
+ sendUpdate(): EPPUpdateResp
+ sendDelete(): EPPDeleteResp
+ sendInfo(): EPPInfoResp
+ sendDelete(): EPPDeleteResp
+ sendCreate(): EPPCreateResp
+ reset(): void
```

Figure 11 - EPPContact Class Diagram
13.5.3.3.2 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
13.5.3.3 sendCreate() Method

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

1.1.1.1.5 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.setMail("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();
adresses.addElement(new EPPContactDiscloseAddress(EPPContactDiscloseAddress.ATTR_TYPE_LOC));
adresses.addElement(new EPPContactDiscloseAddress(EPPContactDiscloseAddress.ATTR_TYPE_INT));

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

13.5.3.3.4 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 ");
        }
    }
} catch (EPPCommandException e) {
    // Handle exception
}
```
+ currResult.getId() + " is unavailable");
}
}

} 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

13.5.3.3.5 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 statuses
- `<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.getEmail()` 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, then `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++) {
            // Process postal contact information
        }
    }
```
// 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)).etAddress();

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:\t\temail = " + response.getEmail());

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

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

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

// Contact Fax Extension
System.out.println("contactInfo:\t\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
13.5.3.3.6  sendUpdate() Method

Updates the attributes of a Contact.

1.1.1.1.17 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.
- `addExtension(EPPCodecComponent)` – Command extension, if any.
- `addStatus(String)` – Contact status
- `addPostalInfo(EPPContactPostalDefinition)` – Adding postal information
- `setVoicePhone(String)` – Update voice number
- `setFaxNumber(String)` – Update fax number
- `setAuthorizationId(String)` – Update authorization information
- `setDisclose(EPPContactDisclose)` – Updates the contact disclose information

The Contact must exist.

1.1.1.1.18 Post-Conditions
The Contact was successfully updated. `EPPResponse` is returned, with the following attributes:

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

1.1.1.1.19 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`.

1.1.1.1.20 Sample Code

Figure 15 - `sendUpdate()` 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-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(""犍
disclose.setFax(""犍
disclose.setEmail(""犍
c
contact.setDisclose(disclose);
response = contact.sendUpdate();

// -- Output all of the response attributes
System.out.println("contactUpdate: Response = [" + response + "]
handleException(e);
}
13.5.3.3.7 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.

```java
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

13.5.3.3.8 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 + "]\n\n");

    // -- 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**
13.5.4 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 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.

13.5.4.1 Registry Tests

The Registry source distribution contains one test program for 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>Directory</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>
</tbody>
</table>

13.5.4.2 Suggestion Packages

The Registry portion of the Verisign Bundle EPP SDK consists of sub-packages and class additions to existing SDK packages. Figure 14 - Suggestion 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</td>
</tr>
</tbody>
</table>
Addition of Registry EPP Server Framework classes used by the Stub Server.

Addition of the RegistryHandler class used to implement the EPP Registry 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 Registry client interface classes. These classes provide the primary interfaces that map to the commands and objects of the Registry EPP Mapping.

### Figure 12 - Registry Packages

#### 13.5.4.3 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>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

#### 13.5.4.4 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.

13.5.4.4.1 Registry Interface

The Registry interface, `EPPRegistry`, is located in the `com.verisign.epp.interfaces` package. This interface is used to query zone information for the system.
The *EPPRegistry* interface has the following relevant methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPRegistry</td>
<td>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>sendCheck() This method checks on the availability / existence of a zone. This method is for future use.</td>
</tr>
<tr>
<td>EPPRegistryInfoResp</td>
<td>sendInfo() This method is used to get registry zone information.</td>
</tr>
<tr>
<td>EPPRegistryCreateResp</td>
<td>sendCreate() This method creates a new zone. This method is for future use.</td>
</tr>
<tr>
<td>EPPResponse</td>
<td>sendUpdate() This method updates a new zone. This method is for future use.</td>
</tr>
<tr>
<td>EPPResponse</td>
<td>sendDelete() This method deletes a zone. This method is for future use.</td>
</tr>
</tbody>
</table>

### 1.1.1.1.29 EPPRegistry () Constructor

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

#### 1.1.1.1.29.1 Pre-Conditions

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

#### 1.1.1.1.29.2 Post-Conditions

The *EPPRegistry* instance is ready for the execution of one or more operations.

#### 1.1.1.1.29.3 Exceptions

None

#### 1.1.1.1.29.4 Sample Code
The following example shows the steps of initializing an \textit{EPPSession}, then using the \textit{EPPSession} to initialize the \textit{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 \textit{sendInfo()} method sends the Registry EPP info command to the server. It has the following signature:

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

1.1.1.1.30.1 Pre-Conditions

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

1.1.1.1.30.2 Post-Conditions

On success, an \textit{EPPRegistryInfoResp} is returned.

1.1.1.1.30.3 Exceptions

An \textit{EPPCommandException} will be returned that contains the \textit{EPPResponse} returned from the server. The \textit{getResponse()} method returns the associated \textit{EPPResponse}. If the exception is thrown before reading the server response, than \textit{getResponse()} will return \textit{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:

```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();
}
```
13.5.5 Suggestion Mapping

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

3. Definition of the Suggestion SDK files (i.e. library, schema)
4. Description of the Suggestion 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 Suggestion 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.

13.5.5.1 Suggestion Tests

The Namestore source distribution contains one test program for Suggestion EPP Mapping the product uses. The tests are located in the suggestion/java directory in \textit{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>EPPSuggestionTst.java</td>
<td>This is a sample program demonstrating the use of the EPPSuggestion class.</td>
</tr>
</tbody>
</table>

13.5.5.2 Suggestion Packages

The Suggestion portion of the Verisign Bundle EPP SDK consists of sub-packages and class additions to existing SDK packages. \textbf{Figure 14 - Suggestion 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.suggestion</td>
<td>Name Suggestion Encoder/Decoder package. All of the detail of encoding and decoding the Suggestion EPP messages are in this package.</td>
</tr>
<tr>
<td></td>
<td>The EPPSuggestionMapFactory must be added to</td>
</tr>
</tbody>
</table>
the EPP.MapFactories configuration parameter using the full 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 Suggestion EPP Server Framework classes used by the Stub Server.</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the SuggestionHandler class used to implement the EPP Suggestion 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 Suggestion client interface classes. These classes provide the primary interfaces that map to the commands and objects of the Suggestion EPP Mapping.</td>
</tr>
</tbody>
</table>

**Figure 14 - Suggestion Packages**

### 13.5.5.3 Suggestion XML Schema files

The Suggestion 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>suggestion-1.1.xsd</td>
<td>schemas</td>
<td>Suggestion 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 - Suggestion Schema Files**

### 13.5.5.4 Suggestion Client Interfaces

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

#### 13.5.5.4.1 Suggestion Interface

The Suggestion interface, EPP.Suggestion, is located in the com.verisign.epp.interfaces package. This interface is used to query suggestions in the Suggestion system.
The *EPPSuggestion* interface has the following relevant methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPSuggestion(EPPSession aNewSession)</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>EPPSuggestionInfoResp</td>
<td>sendInfo() This method is used to retrieve suggestions.</td>
</tr>
</tbody>
</table>

The method on the *EPPSuggestion* 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 Namestore server and will throw an exception if any error occurs. If the exception is associated with an error response from the Namestore 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 *EPPSuggestion* constructor.

1.1.1.1.31  *EPPSuggestion () Constructor*

The *EPPSuggestion* constructor requires that an authenticated *EPPSession* object be passed upon creation. Once created, the *EPPSuggestion* object can perform multiple functions without reinitializing the *EPPSession* object. For example, you can use the same initialized *EPPSuggestion* object to info a suggestions with the *sendInfo()* command.

1.1.1.1.31.1 **Pre-Conditions**

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

1.1.1.1.31.2 **Post-Conditions**

The *EPPSuggestion* instance is ready for the execution of one or more operations.

1.1.1.1.31.3 **Exceptions**

None

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 *EPPSuggestion* 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);
}

EPPSuggestion suggestion = new EPPSuggestion(session);

### 1.1.1.1.32 sendInfo() method

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

```java
public EPPSuggestionInfoResp sendInfo() throws EPPCommandException
```

#### 1.1.1.1.32.1 Pre-Conditions

The `setCommand(com.verisign.epp.codec.suggestion.EPPSuggestionInfoCmd)` is called with a `com.verisign.epp.codec.suggestion.EPPSuggestionInfoCmd` filled in with the suggestion input. See the Javadoc for more detail on the options for `com.verisign.epp.codec.suggestion.EPPSuggestionInfoCmd`. The `com.verisign.epp.codec.suggestion.EPPSuggestionInfoCmd` supports many options including:

1. Input suggestion key
2. Suggestion filter
3. Desired suggestion language

#### 1.1.1.1.32.2 Post-Conditions

On success, an `EPPSuggestionInfoResp` 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, then *getResponse()* will return *null*.

### 1.1.1.1.1.32.4 Sample Code

The following example shows the steps of performing a Suggestion info using the *EPPSuggestion* client interface and the *sendInfo()* method:

```java
EPPSuggestionInfoResp response;
try {
    EPPSuggestionInfoCmd cmd = new EPPSuggestionInfoCmd("51364-CLI");
    cmd.setKey("soccerteam.com");

    EPPSuggestionFilter filter = new EPPSuggestionFilter();
    filter.setContentFilter(false);
    filter.setCustomFilter(false);
    filter.setMaxLength(63);
    filter.setMaxResults(20);
    filter.setUseHyphens(false);
    filter.setUseNumbers(true);
    filter.setTableView();

    cmd.setFilter(filter);

    // Set the desired language to Spanish "ESP"
    // English "ENG" is the default language.
    cmd.setLanguage(EPPSuggestionConstants.SPANISH_CODE);

    suggestion.setCommand(cmd);
    response = suggestion.sendInfo();
} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
    e.printStackTrace();
}
```
13.5.6 WhoWas Mapping

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

1. Definition of the WhoWas SDK files (i.e. library, schema)
2. Description of the WhoWas 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 WhoWas 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.

13.5.6.1 WhoWas Tests

The Namestore source distribution contains one test program for WhoWas EPP Mapping the product uses. The tests are located in the whowas/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>EPPWhoWasTst.java</td>
<td>This is a sample program demonstrating the use of the EPPWhoWas class.</td>
</tr>
</tbody>
</table>

13.5.6.2 WhoWas Packages

The WhoWas portion of the Verisign Bundle EPP SDK consists of sub-packages and class additions to existing SDK packages. Figure 16 - WhoWas PackagesFigure 14 - Suggestion Packagesprovides 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.whowas</td>
<td>WhoWas Encoder/Decoder package. All of the detail of encoding and decoding the WhoWas EPP messages are in this package.</td>
</tr>
<tr>
<td></td>
<td>The <code>EPPWhoWasMapFactory</code> must be added to the EPP.MapFactories configuration parameter using the full</td>
</tr>
</tbody>
</table>
The WhoWas EPP Mapping is defined using XML schema files. These files are located 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><code>whoWas-1.0.xsd</code></td>
<td>schemas</td>
<td>WhoWas XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.</td>
</tr>
</tbody>
</table>

### 13.5.6.4 WhoWas Client Interfaces

The WhoWas portion of the Verisign Bundle EPP SDK contains client interface classes for the WhoWas EPP Mapping. The interfaces provide mechanisms for getting history records. The following sections describe the client interface classes, supporting classes and their respective purposes.

#### 13.5.6.4.1 WhoWas Interface

The WhoWas interface, `EPPWhoWas`, is located in the `com.verisign.epp.interfaces` package. This interface is used to get history records from the WhoWas system.
The EPPWhoWas interface has the following relevant methods:

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPPWhoWas(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>EPPWhoWasInfoResp sendInfo()</td>
<td>This method is used to retrieve whowas history records.</td>
</tr>
</tbody>
</table>

The method on the EPPWhoWas 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 Namestore server and will throw an exception if any error occurs. If the exception is associated with an error response from the Namestore 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 EPPWhoWas constructor.

1.1.1.1.33 EPPWhoWas() Constructor

The EPPWhoWas constructor requires that an authenticated EPPSession object be passed upon creation. Once created, the EPPWhoWas object can perform multiple functions without reinitializing the EPPSession object. For example, you can use the same initialized EPPWhoWas object to info whowas history with the sendInfo() command.

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 EPPWhoWas 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 EPPWhoWas 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);
}

EPPWhoWas whowas = new EPPWhoWas(session);

1.1.1.1.34 sendInfo() method
The sendInfo() method sends the WhoWas EPP info command to the Namestore server. It has
the following signature:

    public EPPWhoWasInfoResp sendInfo() throws EPPCommandException

1.1.1.1.34.1 Pre-Conditions

1. The client transaction id should be set by calling setTransId(String aTransId) method.
2. The type can be set by calling setType(String aType) method. If type is not set then by
default “domain” type will be set. The constant
   com.verisign.epp.codec.whowas.EPPWhoWasConstants.TYPE_DOMAIN can be used in
   place of “domain”.
3. Either name or roid should be set by calling setName(String aName) or setRoid(String
   aRoid) methods.

1.1.1.1.34.2 Post-Conditions
On success, an EPPWhoWasInfoResp is returned.

1.1.1.1.34.3 Exceptions
An EPPCommandException will be returned that contains the EPPResponse returned from the
Namestore server. The getResponse() method returns the associated EPPResponse. If the
exception is thrown before reading the Namestore server response, than getResponse() will
return null.
1.1.1.1.34.4 Sample Code
The following example shows the steps of performing a WhoWas info using the \textit{EPPWhoWas} client interface and the \texttt{sendInfo()} method:

```java
EPPWhoWasInfoResp response;
try {
    EPPWhoWas whowas = new EPPWhoWas(session);

    whowas.setTransId("ABC-12345");
    whowas.setType(EPPWhoWasConstants.TYPE_DOMAIN); // optional
    whowas.setName("abc.com");

    response = whowas.sendInfo();

    EPPWhoWasHistory eppWhoWasHistory = response.getHistory();
    List historyRecords = eppWhoWasHistory.getRecords();
    for ( int i = 0; i < historyRecords.size(); i++ ) {
        System.out.println( "Record Name:[" + i + "]" +
                            historyRecord.getName() );

        System.out.println( "Record Roid:[" + i + "]" +
                            historyRecord.getRoid() );

        System.out.println( "Record Operation:[" + i + "]" +
                            historyRecord.getOperation() );

        System.out.println( "Record Transaction Date:[" + i + "]" +
                            historyRecord.getTransactionDate() );

        System.out.println( "Record Client ID:[" + i + "]" +
                            historyRecord.getClientID() );

        System.out.println( "Record Client Name:[" + i + "]" +
                            historyRecord.getClientName() );
    }
}
catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
e.printStackTrace();
}
```
13.5.7 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 WhoWas information:

3. Definition of the Balance SDK files (i.e. library, schema)
4. 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 WhoWas 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.

13.5.7.1 Balance Tests

The Namestore source distribution contains one test program for WhoWas 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>

13.5.7.2 Balance Packages

The Balance portion of the Verisign Bundle EPP SDK consists of sub-packages and class additions to existing SDK packages. Figure 18 - 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 package and class name.</td>
</tr>
</tbody>
</table>
Addition of Balance EPP Server Framework classes used by the Stub Server.

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.

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.

**Figure 18 - Balance Packages**

### 13.5.7.3 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 19 - Balance Schema Files**

### 13.5.7.4 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.

#### 13.5.7.4.1 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 Namestore server and will throw an exception if any error occurs. If the exception is associated with an error response from the Namestore 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.

### 1.1.1.1.1.35 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.

#### 1.1.1.1.1.35.1 Pre-Conditions

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

#### 1.1.1.1.1.35.2 Post-Conditions

The `EPPBalance` 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 `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);

1.1.1.1.36 sendInfo() method
The sendInfo() method sends the Balance EPP info command to the Namestore server. It has the following signature:

    public EPPBalanceInfoResp sendInfo() throws EPPCommandException

1.1.1.1.36.1 Pre-Conditions

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

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

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

1.1.1.1.36.4 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());
}
13.5.8 EmailFwd Mapping

To be filled in. Please refer to the JavaDoc for detail on the EmailFwd Mapping support.
13.5.9 DefReg Mapping
To be filled in. Please refer to the JavaDoc for detail on the DefReg Mapping support.
13.5.10 NameWatch Mapping
To be filled in. Please refer to the JavaDoc for detail on the NameWatch Mapping support.
Extensions

13.5.11 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 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 prefix the TLD with “dot”, as in “dotCC” for the .CC TLD or “dotCOM” for the .COM TLD. Going forward the convention will be to use the TLD without the “dot” prefix for the SubProductID. For example, for .TLD, the SubProductID is “TLD” and for IDN TLDs the A-label will be the SubProductID. The existing TLDs, including .CC, .TV, .JOB, .COM, and .NET, with support both the old convention and the new convention for backward compatibility. New TLDs will only support the new convention. The class com.verisign.epp.namestore.interfaces.NSSubProduct includes a set of constants of SubProductID values based on the old convention. For example, NSSubProduct.TV can be used to specify the sub-product for .TV. The com.verisign.epp.namestore.interfaces.NSDomain, com.verisign.epp.namestore.interfaces.NSHost, and com.verisign.epp.namestore.interfaces.NSContact classes provide a setSubProductID(String) utility method in place of using the NamestoreExt directly.

13.5.11.1 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.
13.5.12 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 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, than 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.
13.5.13  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.

### 13.5.13.1 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,
    EPPSecDNSAlgorithms.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.

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.

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);

EPPDomainCreateResp theResponse = theDomain.sendUpdate();

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

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"));

VeriSign Inc. Proprietary Information

Verisign Bundle EPP SDK Programmer's Guide • page 156
List remDsDataList = new ArrayList();
remDsDataList.add(new 
com.verisign.epp.codecsecdnext.v11.EPPSecDNSExtDsData(12345,
EPPSecDNSAlgorithm.DSA,
EPPSecDNSExtDsData.SHA1_DIGEST_TYPE,
"38EC35D5B3A34B44C39B ");
theDomain.setSecDNSUpdate(addDsDataList, remDsDataList);
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.

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.codecsecdnext.v11.EPPSecDNSExtDsData(12345,
EPPSecDNSAlgorithm.DSA,
EPPSecDNSExtDsData.SHA1_DIGEST_TYPE,
"49FD46E6C4B45C55D4AC");
theDomain.setSecDNSUpdate(addDsDataList, NSDomain.REM_ALL_DS);
EPPDomainCreateResp theResponse = theDomain.sendUpdate();
13.5.14 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();
```
13.5.15 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.
13.5.16  DotJobs Contact Extension

13.5.16.1 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>

13.5.16.2 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>

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

13.5.16.3.2 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();
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);

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.getContactID());
System.out.println("Contact Created Date : " +
    response.getCreationDate());
}
13.5.16.3.3  sendCheck() Method

DotJobs Contact is an extension to Contact Object and this method is not applicable to the extension.

13.5.16.3.4  sendInfo() Method

Same as Contact sendInfo() method. However the extension part of info is explained using the sample listed below.

```java
try {
    EPPContactInfoResp contactResponse;
    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\ctCity = " +
                address.getCity());
            // Address State/Province
            System.out.println("contactInfo:\t\tState Province = "
```
public void showEppResponse() {
    // Output basic response information
    System.out.println("Contact ID: " + response.getId());

    // Output contact information
    System.out.println("Contact Name: " + response.getContactName());
    System.out.println("Contact Email: " + response.getEmail());
    System.out.println("Contact Voice: " + response.getVoice());
    System.out.println("Contact Voice Extension: " + response.getVoiceExt());
    System.out.println("Contact Fax: " + response.getFax());
    System.out.println("Contact Fax Extension: " + response.getFaxExt());

    // Output address information
    System.out.println("Address: " + address.getAddress());
    System.out.println("Address: " + address.getStateProvince());
    System.out.println("Address: " + address.getPostalCode());
    System.out.println("Address: " + address.getCountry());

    // Output optional response attributes using accessors
    // Contact Fax
    if (response.getFax() != null) {
        System.out.println("Contact Fax: " + response.getFax());
    }
    // Contact Voice
    if (response.getVoice() != null) {
        System.out.println("Contact Voice: " + response.getVoice());
    }
    // Last Updated By
    if (response.getLastUpdatedBy() != null) {
        System.out.println("Last Updated By: " + response.getLastUpdatedBy());
    }
}

// Output basic response information
System.out.println("Contact ID: " + response.getId());

// Output contact information
System.out.println("Contact Name: " + response.getContactName());
System.out.println("Contact Email: " + response.getEmail());
System.out.println("Contact Voice: " + response.getVoice());
System.out.println("Contact Voice Extension: " + response.getVoiceExt());
System.out.println("Contact Fax: " + response.getFax());
System.out.println("Contact Fax Extension: " + response.getFaxExt());
System.out.println("Address: " + address.getAddress());
System.out.println("Address: " + address.getStateProvince());
System.out.println("Address: " + address.getPostalCode());
System.out.println("Address: " + address.getCountry());

// Output optional response attributes using accessors
// Contact Fax
if (response.getFax() != null) {
    System.out.println("Contact Fax: " + response.getFax());
}
// Contact Voice
if (response.getVoice() != null) {
    System.out.println("Contact Voice: " + response.getVoice());
}
// Last Updated By
if (response.getLastUpdatedBy() != null) {

```java
System.out.println("Contact: address = " + address.getAddress());
System.out.println("Contact: state/province = " + address.getStateProvince());
System.out.println("Contact: postal code = " + address.getPostalCode());
System.out.println("Contact: country = " + address.getCountry());
```
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");
    }
}
13.5.16.4 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_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);
}

13.5.16.4.2 sendDelete() Method

DotJobs Contact is an extension to Contact Object and this method is not applicable to the extension.

13.5.16.4.3 sendTransfer() Method

DotJobs Contact is an extension to Contact Object and this method is not applicable to the extension.
13.5.17 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.EPPDomain.addExtension(EPPCodecComponent) method or the com.verisign.epp.namestore.interfaces.NSDomain.addExtension(EPPCodecComponent) method. The com.verisign.epp.interfaces.EPPLaunch interfaces class was created to simplify sending the Launch Check Command (Claims Check Form or Availability Check Form).

13.5.17.1 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>

13.5.17.2 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.
<table>
<thead>
<tr>
<th>File Name</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>launch-1.0.xsd</td>
<td>schemas</td>
<td>Launch Extension XML Schema. This XML Schema is dependent on the mark-1.0.xsd and the signedMark-1.0.xsd.</td>
</tr>
<tr>
<td>mark-1.0.xsd</td>
<td>schemas</td>
<td>Trademark definition XML Schema.</td>
</tr>
<tr>
<td>signedMark-1.0.xsd</td>
<td>schemas</td>
<td>Signed Mark and Encoded Signed Mark XML Schema. This XML Schema is dependent on mark-1.0.xsd and xmldsig-core-schema.xsd.</td>
</tr>
</tbody>
</table>

### 13.5.17.3 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.

#### 13.5.17.3.1 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 20 - 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 20 - 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 21 - 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 21 - Launch Availability Check Example**

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 + "]");
} catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}

13.5.17.3.2 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 22 – 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 22 – Domain Info for Launch Application Example

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 23 – Domain Info for Launch Registration Example” shows an example of executing a Domain Info Command for a Sunrise Registration.

**Figure 23 – 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.getResponse();
    Assert.fail(e.getMessage());
}

13.5.17.3.3 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 24 – 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 24 – 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
```
EPFSignedMark signedMark = new EPFSignedMark("1-2", issuer,
new GregorianCalendar(2013, 1, 1).getTime(),
new GregorianCalendar(2014, 1, 1).getTime(),
mark);
signedMark.sign(privateKey, certChain);
signedMark.setBase64Encode(true);
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());
}
}

Figure 25 – 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 25 – Domain Create in Claims Create Form Example

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(),
    new Date(),
    EPPLaunchCreate.TYPE_REGISTRATION);
response = domain.sendCreate();
}
catch (EPPCommandException e) {
    EPPResponse errorResponse = e.getResponse();
    Assert.fail(e.getMessage());
}

“Figure 26 – 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 26 – Domain Create in General 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_LANDRUSH),
                                             EPPLaunchCreate.TYPE_APPLICATION);
    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 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 27 – Domain Create in Mix Create Form Example**

```java
EPPResponse response;
try {
    NSDomain domain = new NSDomain(session);
```
// 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());
}

13.5.17.3.4 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 28 – Domain Update of Sunrise Application Example” shows an example of updating a sunrise application.
13.5.17.3.5 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 28 – Domain Update of Sunrise Application Example” shows an example of updating a sunrise application.
Figure 29 – 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());
}
```
13.5.18 PersReg Extension

The Personal Registration Extension defined in “Extensible Provisioning Protocol Extension Mapping: &lt;Personal Registration&gt;” 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.

13.5.18.1 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>

13.5.18.2 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-{SBUILD_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>
13.5.18.3 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 30 – 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.

Figure 30 – Personal Registration Create with Consent Example

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(EPPERSRegCreateErrData.class)) {
        EPPERSRegCreateErrData ext = (EPPERSRegCreateErrData)
        errorResponse.getExtension(EPPERSRegCreateErrData.class);
        System.out.println("PersReg error = " + ext);
    } Assert.fail(e.getMessage());
}
13.5.19 Related Domain Extension

The Related Domain Extension defined in “Extensible Provisioning Protocol Extension Mapping: <Related Domain>” includes extensions to the Domain Info Command and the Domain Info Response to retrieve information on the related domains to a domain name.

13.5.19.1 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 <code>com.verisign.epp.codec.relateddomainext.EPPRelatedDomainExtFactory</code> must be added to the <code>EPP.CmdRspExtensions</code> configuration parameter.</td>
</tr>
<tr>
<td>com.verisign.epp.interfaces</td>
<td>Includes the <code>com.verisign.epp.interfaces.EPPRelatedDomainTst</code> test class for sending the Domain Info Command with the Related Domain Extension (<code>com.verisign.epp.codec.relateddomainext.EPPRelatedDomainExtInfo</code>) to receive the Domain Info Response with the Related Domain Extension (<code>com.verisign.epp.codec.relateddomainext.EPPRelatedDomainExtInfData</code>).</td>
</tr>
<tr>
<td>com.verisign.epp.serverstub</td>
<td>Addition of the <code>com.verisign.epp.serverstub.RelatedDomainHandler</code> class used to implement the EPP Related Domain Stub Server behavior. These classes must be added to the <code>EPP.ServerEventHandlers</code> configuration to simulate a server that supports the Related Domain Extension.</td>
</tr>
</tbody>
</table>

13.5.19.2 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-</td>
<td>schemas</td>
<td>Related Domain Extension XML Schema</td>
</tr>
</tbody>
</table>
13.5.19.3 Related Domain Client Interface

The Related Domain Extension is an extension to the Domain object, so the Domain 
(com.verisign.epp.interfaces.EPPDomain) Client Interface class, as well as the derived 
com.verisign.epp.namestore.interfaces.NSDomain class, is used with the Related Domain 
Extension.

“Figure 31 – Related Domain Info Command Example” shows an example of executing a 
Domain Info Command with the Related Domain Extension 
com.verisign.epp.codec.relateddomainext.EPPRelatedDomainExtInfo class to receive a response 
that includes the Related Domain information in the 
com.verisign.epp.codec.relateddomainext.EPPRelatedDomainExtInfData class.

Figure 31 – Related Domain Info Command Example

```java
EPPDomainInfoResp response;
try {
    EPPDomain domain = new EPPDomain(session);
    domain.setTransId("ABC-12345");
    domain.addDomainName("example1.tld");
    domain.addExtension(new EPPRelatedDomainExtInfo());
    response = domain.sendInfo();
    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());
}
```