NAMESTORE AND SRS SDK PROGRAMMER'S GUIDE

Version 6.21

November 7, 2011

Change Log

Author(s)	Date	Revision	Description
Colin Lloyd	09/01//2002	1.0	Initial Revision
Colin Lloyd	11/01//2002	2.0	To SMP version 2.0
Chris Bason	12/05/2002	2.1	Addition of Namestore Regional ccTLD Product.
Colin Lloyd	01/09/2003	2.2	Updates and corrections to SMP section
Colin Lloyd	02/14/2003	2.3	More updates and corrections to SMP section
Srinivas Sunkara	03/14/2003	2.4	Updates to ccTLD section to add Poll Queue information and corrections.
John Fraser	04/14/2003	2.5	Updates to include dotTV and dotCC in ccTld section.
Damon Miller	07/01/2003	2.6	Updates to include Backorder product.
Srinivas Sunkara	07/07/2003	2.7	Updates to Backorder section
Srinivas Sunkara	07/11/2003	2.8	Updates to Backorder section to fix defects
Colin Lloyd	11/11/2003	2.9	Updates for the description of the HTTP Protocol
James Gould	2/2/2004	3.0	Updated for EPP 1.0 update and restructuring of the packaging to support an open source and binary distribution. Included in this update is: Removal of SMP Removal of Namestore Domain Billing Extension
James Gould	3/8/2004	3.1	Removed Backorder product and updated SSL, HTTPS configuration
John Fraser	3/10/2004	3.2	Updated ccTld section for updated Rcctld product.
James Gould	4/2/2004	3.3	Added reference to http://www.verisign.com/nds/naming/namestore/techdocs.html for document updates.
James Gould	4/9/2004	3.4	Updated EPP references to the RFC's
James Gould	6/13/2004	4.0	Updated for Consolidated NameStore 3.0.0 by including the new extensions and mappings. The new components include: RGP Sync IDN Language Tag Low Balance Notification
James Gould	7/5/2004	4.1	Added updates for NSHost and NSDomain.

James Gould	8/12/2004	4.2	Updated the description of addDomainName(String) to indicate correctly if a domain name or a domain id is required.
James Gould	12/17/2004	4.3	Updated to add setHosts(String) to EPPDomain an option to use setAuthString with sendInfo().
James Gould	12/28/2004	4.4	Updated configuration properties to reflect the properties in the 3.0.0.8 release. Specifically, the javax.net.ssl.keyStore, javax.net.ssl.keyStorePassword, and javax.net.ssl.trustStore properties have been removed and the EPP.SSLTrustStoreFileName, EPP.SSLTrustStorePassPhrase, and EPP.SSLKeyPassPhrase properties have been added. Some property descriptions were updated and the configuration file search order was added for clarity with different client environments.
James Gould	3/21/2005	4.5	Small updates to RCC Domain create and NSDomain check commands.
James Gould	3/31/2005	4.6	Added description of result of running a successful test-client-server test since it will include a stack trace on successful executions.
Colin Lloyd	8/10/2005	4.7	Updates for the CLS product.
John Fraser	8/12/2005	4.8	Added description of available RCC job:info status codes.
James Gould	9/5/2005	5.0	Revamped the document to reference both COM/NET SRS and Namestore, update the SSL configuration, and removed old/incorrect information. Renamed the document to include SRS along with Namestore. Removed references to the eppsdk/sample directory based on SR 446925. Add documentation of using system session pools.
Javier Juarez	10/21/2005	5.1	Added RCCTLD EU Sunrise Extension
James Gould	11/12/2005	5.2	Added Whois Info Extension and updated documentation for the 3.3 release.
John Fraser	12/19/2005	5.3	Updated the RCC tables detailing poll response errors.
David Liu	12/20/2005	5.4	Added RCCTLD EU Literal Extension
James F. Gould	12/21/2005	5.5	Added change log information for the 3.4 release
James F. Gould	1/19/2006	5.6	Made some documentation corrections based on testing review, changed the default settings for absoluteTimout and idleTimeout properties, and added change log information for the 3.4.0.1 release.
Mahendra Jain	04/10/2006	5.7	Added Name Suggestion Product
Mahendra Jain	05/23/2006	5.8	Added change log information for the 3.5.1.0 release
John Fraser	09/20/2006	5.9	Added RCCTLD Transfers support and removed RCCTLD EU Sunrise extension

James F. Gould	09/25/2006	6.0	Made some significant documentation updates in preparation for the 3.6.1.0 release
James F. Gould	5/25/2007	6.1	Added the EPP.MaxPacketSize property
Rupert Fernando	09/13/2007	6.2	Added Contact and dotJobs Contact product
Rupert Fernando	09/20/2007	6.3	Modified the EPP.CmdRspExtensions property for DotJobs section.
Rupert Fernando	10/02/2007	6.4	Updated RFC references. Made changes to the Contact SDK to handle Contact update and the optional fields.
James F. Gould	10/27/2008	6.5	Added new Session Pool initMaxActive and borrowRetries properties. Added section on “11.3 Pipelining” to cover the use of pipelining with the SDK.
Leela Koneru	12/05/2008	6.6	Added secdns (DNSSEC) Extension and updated documentation for the 3.8 release.
Mahendra Jain	12/18/2009	6.7	Added support for Premium domain Extension
James F. Gould	12/23/2009	6.8	Revamped the quick start instructions to include instructions for using the session pool and to include references to the client interface tests. Also updated the Name Suggestion section to provide additional information.
Mahendra Jain	02/23/2010	6.9	Addressed DR # 24811 and #24835
James F. Gould	03/19/2010	6.10	Small editorial updates.
James F. Gould	03/23/2010	6.11	Added support for secDNS-1.1 (RFC 5910).
Mahendra Jain	03/26/2010	6.12	Updated Premium domain section
Deepak Deshpande	03/31/2010	6.13	Added WhoWas Product
Mahendra Jain	04/12/2010	6.14	Added optional renewalPrice element to Premium Domain extension of Domain Check command response.
James F. Gould	5/2/2010	6.15	Small editorial updates.
James F. Gould	5/17/2010	6.16	Updated references with published RFC 5910 (secDNS-1.1).
Jeff Faust	3/1/2011	6.17	Added Client Object Attribute extension.
James F. Gould	3/2/2011	6.18	Removed HTTP Extension
James F. Gould	3/28/2011	6.19	Added notes about addition of EPPCommand.getExtension(Class, boolean), EPPResponse.getExtension(Class, boolean), and EPPDuplicateExtensionException.
James F. Gould	4/22/2011	6.20	Added addition document related to the new EPPSessionPool.close method.
James F. Gould	11/7/2011	6.21	Added support for connecting through an Apache Proxy (mod_proxy). Also added support for the Balance Object

			Mapping.

References

Author(s)	Title	Revision	Date
Scott Hollenbeck	Extensible Provisioning Protocol		8/2009
Scott Hollenbeck	Extensible Provisioning Protocol (EPP) Transport Over TCP		8/2009
Scott Hollenbeck	Extensible Provisioning Protocol (EPP) Domain Name Mapping		8/2009
Scott Hollenbeck	Extensible Provisioning Protocol (EPP) Host Mapping		8/2009
Scott Hollenbeck	Extensible Provisioning Protocol (EPP) Contact Mapping		8/2009
Scott Hollenbeck	Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol		4/16/2004
Scott Hollenbeck	ConsoliDate Mapping for the Extensible Provisioning Protocol	01	3/22/2005
Scott Hollenbeck	Domain Name System (DNS) Security Extensions Mapping for the Extensible Provisioning Protocol (EPP)		11/2005
Mahendra Jain	EPP RGP Poll Mapping Guide	1.1	5/23/2004
Mahendra Jain	EPP Low Balance Mapping Guide	1.1	5/23/2004
Venkat Munuswamy	Extensible Provisioning Protocol Extension Mapping: IDN Language Tag	1.1	
James F. Gould	Extensible Provisioning Protocol Extension Mapping: Whois Info	1.0	11/12/2005
John Colosi	Suggestion Mapping	1.1	5/23/2006
Rupert Fernando	Jobs Contact Extension Mapping	1.1	09/07/2007
Mahendra Jain	Premium Domain Extension Mapping	2.0	3/18/2010
Scott Hollenbeck	RFC 4310 – Domain Name System (DNS) Security Extensions Mapping for the Extensible Provisioning Protocol		11/2005
James Gould and Scott Hollenbeck	RFC 5910 – Domain Name System (DNS) Security Extensions Mapping for the Extensible Provisioning Protocol		5/2010
Deepak Deshpande	WhoWas Mapping	1.0	3/31/2010

Jeff Faust	Client Object Attribute Extension Mapping	1.0	2/15/2011
James Gould	Balance Mapping	1.0	11/7/2011

Definitions, Acronyms, and Abbreviations

Term	Description
SSL	http://home.netscape.com/eng/ssl3/ssl-toc.html
TLS	http://www.ietf.org/rfc/rfc2246.txt?number=2246
EPP	Extensible Provisioning Protocol
IETF	http://www.ietf.org/
RFC	Request for Comments
SDK	Software Development Kit
SRS	Shared Registry System
VNS	Verisign Naming Services
XML	http://www.w3c.org/XML/
RGP	Registry Grace Period
CTLD	Consolidated Top-Level Domain
DS	Delegation Signer
DNS	Domain Name System

Contents

1. Introduction

This document provides instructions on how to use the Namestore Software Development Kit (SDK) to communicate with the Verisign Naming Services (VNS) Consolidated Namestore, hereon referred to as Namestore, or the COM/NET Shared Regisry System (SRS), hereon referred to as SRS, using the Extensible Provisioning Protocol (EPP). The SDK includes all of the API's required to interface with the products in Namestore and SRS. The products supported by Namestore include:

1. Consolidated Top-Level Domain (CTLD) - Is a platform of domain products that follow the dotCOM and dotNET business rules. CTLD commands require the Namestore Extension for specifying the target logical Registry, which currently includes the possible values dotCC, dotTV, or dotJOBS.
2. Contact – Thick registry has all of the information associated with registered entities.
3. DotJobs Contact Extension – In addition to the thick registry information other contact information specific to dotJOBS contacts are registered using this extension.
4. Name Suggestion - Provides highly related domain name suggestions.
5. WhoWas – Provides history records of entities.

The SRS supports Domain and Host commands for dotCOM and dotNET, and only supports the SSL transport. The Namestore Extension is required for the SRS. Currently, the SRS bundles dotCOM and dotNET, so the value of the Namestore Extension can be dotCOM, dotNET, or dotEDU. The same API's are used to interface with CTLD. The Namestore Extension can be dotCC, dotTV, or dotJOBS for CTLD.

The SDK includes a full implementation of the EPP specifications independent of the services supported by the Registry services (i.e. Namestore and SRS). The `com.verisign.epp.interfaces.EPPDomain` and `com.verisign.epp.interfaces.EPPHost` fully support the IETF Domain and Host mappings, while `com.verisign.namestore.interfaces.NSDomain` and `com.verisign.namestore.interfaces.NSHost` provide convinence 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 poll messages produced by Namestore and the SRS.

The instructions provided include an overview of the Namestore and SRS SDK, how to configure it, how to use it, and how to extend it. Please see http://www.verisign.com/information-services/naming-services/page_001081.html for updates to the Programmer's Guide between Namestore and SRS SDK releases.

The Namestore and SRS 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 3.12 to Version 3.13

1. Added missing getXmlSchemas() : Set method to preload the dependent XML schemas by the com.verisign.epp.util.EPPSchemaCachingParser class to com.verisign.epp.codec.secdnsext.v10.EPPSecDNSExtFactory and com.verisign.epp.codec.secdnsext.v10.EPPSecDNSExtFactory.
2. Added support for connecting through an Apache Proxy (mod_proxy) by creating the com.verisign.epp.transport.client.EPPPlainProxyClientSocket class for TCP and com.verisign.epp.transport.client.EPPSSLProxyClientSocket for SSL/TLS. Additional customization classes and properties were created for defining the list of proxy servers (com.verisign.epp.transport.client.EPPProxyServersLocator interface) and the concrete com.verisign.epp.transport.client.EPPConfigProxyServersLocator class for using the EPP.ProxyServers property for the list. The EPP.ProxyServersRandomize property was defined to enable and disable the selection of the proxy servers to use.
3. Removed the pre-condition check of passing a non-null EPPSSLContext parameter to the constructor of com.verisign.epp.transport.client.EPPPlainClientSocket.
4. Added support for the new Balance Object Mapping that defines an info command and response to retrieve the account balance and other balance information.
5. Bumped up the default maximum packet size (EPP.MaxPacketSize) in com.verisign.epp.util.EPPXMLStream from 10000 to 355000 for WhoWas.
6. Removed the schema-based 32-character length restriction for the input key for NameSuggestion.
7. Added constants for French language for NameSuggestion.

2.2 Version 3.11 to Version 3.12

8. Added EPPSessionPool.close and EPPSystemSessionPool.close methods to cleanly close the sessions from the underlying GenericObjectPool instance(s).
9. Added Client Object Attribute extension to the NameStore bundle.
10. Removed the HTTP / HTTPS transport support from the NameStore bundle.
11. Added EPPCommand.getExtension(Class aExtensionClass, boolean aFailOnDuplicate) method, EPPResponse.getExtension(Class aExtensionClass, boolean aFailOnDuplicate) method, and exception com.verisign.epp.codec.gen.EPPDuplicateExtensionException.

2.3 Version 3.10 to Version 3.11

1. Changed EPPRgpExtReport.encode(Document) to use preData and postData instead of preWhois and postWhois to be compliant with RFC 3915.
2. Added WhoWas extension to the NameStore bundle.

3. Added EPPDomainAddRemove.isEmpty() : boolean method to easily determine if all of the attributes are null.
4. Added support for secDNS-1.1 as defined in RFC 5910 and moved the secDNS-1.0 support to codec sub-package. Also updated NSDomain to support secDNS-1.1. This included adding dependency to dnsjava 2.0.8 for creating DS data from Key data.
5. Added Premium Domain Check command to the premiumdomain extension and to the NameStore bundle.
6. Added WhoWas command to the NameStore bundle.

2.4 Version 3.9 to Version 3.10

1. Added the Premium Domain (premiumdomain) extension to the NameStore bundle.
2. Updated NSDomainHandler, NSDomainTst, RgpDomainHandler, EPPRgpDomainTst to return both grace period and pending period statuses.
3. Updated the client interface tests to drive off of the EPP.Test.clientId and EPP.Test.password epp.config properties with default hardcoded values based on DR #17146.
4. Upgraded from Ant 1.6.0 to 1.7.1 to resolve an issue with an exception raised after running the test-client-server when Ant is stopping the Stub Server. This is based on DR #21252.
5. Added comment in the epp.config related to commenting out the EPP.SessionPool.<pool>.SSLProtocol property to test against the Stub Server over TCP. This is based on DR #21253.
6. Removed the jvmarg setting of javax.net.ssl.trustStore in all of the common-target.xml files and added the EPP.TrustStoreFileName and EPP.TrustStorePassPhrase epp.config properties to be able to override the setting more easily. This is based on DR #21274.
7. Updated the Name Suggestion section of the Programmer's Guide to include instructions for setting the language and better describing the setting of the command. This is based on DR #21276.
8. Replaced all of the XML entity includes with standard Ant import tasks. This is based on DR #23114.
9. Revamped the quick start instructions in the Programmer's Guide to include instructions for using the session pool and to include references to the client interface tests as running examples. .

2.5 Version 3.8 to Version 3.9

1. Added new ant target "jar-src" to creating source jar file with version number.
2. Modify "jar" ant target to include version number in jar file name.
3. Added dependency and configuration of the Host mapping to secdns to remove XML schema errors when testing from the secdns build by the domain XML schema reference host XML schema elements.

4. Replaced the XML entity includes of common-targets.xml with Ant import tasks to remove Ant warning messages.
5. Fixed the decoding of the EPPSecDNSExtUpdate urgent flag to accept "0" and "1" values, where previously it only supported "false" and "true".
6. Changed the default availability flag returned on a Domain Check by NSDomainHandler to be true instead of false.
7. Added EPPLoginCmd.mergeServicesAndExtensionServices(EPPGreeting) method and added a call from EPPSession.login to merge the default services and extension services included in the EPP Login with the services and extension services return in the EPP Greeting. This will ensure protocol compliance of the EPP Login even when there are many more services configured on the client side.
8. Added EPPRgpExtInfData.setStatuses(Vector) and EPPRgpExtInfData.getStatuses() : Vector to match the latest version of RFC 3915, where multiple statuses can be returned.
9. Added EPPRgpExtReport getPreData, getPostData, setPreData, and setPostData along deprecating getPreWhois, getPostWhois, setPreWhois, and setPostWhois to be in line with the latest version of RFC 3915. Also updated rgp-1.0.xsd and EPPRgpExtReport.decode to support passing either (pre|post)Whois or (pre|post)Data to simulate a Stub Server that accepts either one during a transition period to the latest version of RFC 3915. The EPPRgpExtTst, EPPRgpDomainTst, NSDomainTst, and RgpDomainHandler were updated not to use deprecated methods.

2.6 Version 3.7 to Version 3.8

1. Added the initMaxActive EPPSessionPool property to set if at initialization whether maxActive sessions should be created. The default behavior is "false" which will create sessions on demand.
2. Added the for borrowRetries EPPSessionPool property to set the number of retries that should be executed if on call to borrowObject() or borrowObject(String) there is a failure getting/creating a session. The default value is 0 and setting the property to a positive value will result in one attempt plus the number of borrowRetries attempts.
3. Added Name Suggestion language element.
4. In com.verisign.epp.codec.domain.EPPDomainTst, changed to load the EPPHostMapFactory prior to the EPPDomainMapFactory based on the dependency of the domain-1.0.xsd to the host-1.0.xsd.
5. Fixed issue that the XML Parser Pool needs to be initialized after the CODEC to properly pre-load all of the dependency XML schemas.
6. Changed (!theResponse instanceof CLASS) calls in EPPDomain.java, EPPHost.java, and EPPContact.java to use the new EPPSession.processDocument(EPPCommand, Class) : EPPResponse calls to allow for the new EPPSession.MODE_ASYNC mode where null can be returned from EPPSession.processDocument(EPPCommand) and EPPSession.processDocument(EPPCommand, Class).

7. Added testAsyncCommands() to NSDomainTst.java to test use of the MODE_ASYNC with EPPSession.
8. Added doAsyncPoll() to EPPSessionTst.java to test poll messaging with the EPPSession mode set to MODE_ASYNC.
9. Added processDocument(aCommand : EPPCommand, aExpectedResponse Class) method, and isModeSupported(int) : boolean method to EPPHttpSession.java to support the new EPPSession API but to specify to the client that EPPHttpSession does NOT support MODE_ASYNC mode.
10. Added MODE_SYNC constant, MODE_ASYNC constant, setMode(int) : int method, getMode() : int method, isModeSupported(int) : boolean method, processDocument(aCommand : EPPCommand, aExpectedResponse Class) method, readResponse() : EPPResponse to support asynchronous mode for implementing pipelining.
11. Added missing com.verisign.epp.serverstub.ContactPollHandler from the EPP.PollHandlers property of epp-http.config.
12. Added secdns (DNSSEC) extension to the bundle.

2.7 Version 3.6 to Version 3.7

1. Removed RCC entries from the NameStore Bundle default configuration and test code. Removed the RCC Mapping Factories, Extension Factories, and Handlers from the default NameStore Bundle epp.config and epp-http.config files. Removed inclusion of RCC CODEC and Interfaces tests from the NameStore Bundle common-targets.xml. Removed RCC poll messages from NSPollTst and NSDomainHandler
2. Removed cls from the bundle.
3. Added the jobsContact extension to the bundle
4. Added contact to bundle
5. Made the default log level of EPPSession and EPPLoginCmd to be error level in logconfig.xml
6. Added the EPPSession.getClientCon() method and made the EPPSSLClientSocket.getSocket() method public to support SSL Re-negotiation
7. Added the EPP.MaxPacketSize configuration property to override the EPPXMLStream maximum packet size that was previous defined by the constant MAX_PACKET_SIZE. The EPPXMLStream.MAX_PACKET_SIZE has been renamed to EPPXMLStream.DEFAULT_MAX_PACKET_SIZE and used if the EPP.MaxPacketSize property is not set
8. Added support for specifying a period of 0 to match the EPP specification as identified in SR 1288144.
9. Fixed EPPDomainInfoCmd.HOSTS_NONE constant to have a value of "none" instead of "del"

10. Added "getXmlSchemas() : Set" method to EPPFactory, EPPMapFactory, EPPExtFactory, and the derived classes of EPPMapFactory and EPPExtFactory to specify the XML schemas that need to be pre-loaded. The schemaLocation attribute is optional with EPP RFC 4930, so dynamically loading the XML schemas will not consistently work. All XML schemas will be pre-loaded by EPPSchemaCachingParser based on the configured EPP.MapFactories and EPP.CmdRspExtensions. The order of the factories is signifant, where EPPHostMapFactory should precede EPPDomainMapFactory since the domain-1.0.xsd depends on the host-1.0.xsd. (James F. Gould)
11. Updated the gen, domain, host, contact, and tcp specifications and XML schemas to match the May 2007 RFC's. (James F. Gould)
12. Updated the copyright dates in license.txt to include 2004-2007.
13. Updated the EPPContactAddChange validation of optional fields for Contact update
14. Updated the EPPContactAddress to handle the street address correctly

2.8 Version 3.5 to Version 3.6

1. Added support for separate SSL configurations per Session Pool. The SSL configuration for HTTPS cannot be configured on a per Session Pool basis.
2. Added SSL properties for specifying the supported protocols and cipher suites.
3. Added support for RCCTLD Transfers
4. Removed the RCCTLD EU Sunrise Extension

2.9 Version 3.4 to Version 3.5

1. Added the Name Suggestion Product.

2.10 Version 3.3 to Version 3.4

1. Added the RCCLiteralEU Extension.
2. Fixed an issue with initializing Apache HTTP Client from EPPHttpSession that resulted in failure to connect with two-way SSL.
3. Added com.verisign.epp.codec.rccsunriseeu.RCCSunriseEUExtFactory to epp.config and epp-http.config EPP.CmdRspExtensions property and replaced com.verisign.epp.serverstub.RccDomainHandler with com.verisign.epp.serverstub.RCCExtensionsDomainHandler in the EPP.ServerEventHandlers property.
4. Added handling an empty <epp:extValue> reason element in EPPExtValue.decode().
5. Fixed a bug with the EPPUtil.encodeBigDecimal and EPPClsBidPrice.encode that resulted in localization issues, where the decimal point could be represented as a different character. This is related to SR #661639.
6. Fixed a bug in com.verisign.epp.codec.gen.EPPValue.decode where it required the namespace definition as an attribute of the <value> sub-element. The namespace can now be defined anywhere in the XML.

7. Changed log level of services mismatch from error to warn in com.verisign.epp.codec.gen.EPPLoginCmd based on SR 652869.
8. Fixed a bug with SSL not getting initialized in EPPHttpSession(String) which is required for the session pool to work with https. This was identified in SR #737277.
9. Changed default settings for the absoluteTimeout and idleTimeout to be 82800000 ms (23 hours) and 480000 ms (8 minutes), respectively based on SR #743545.
10. Added the EPP.Test.clientId, EPP.Test.password, and EPP.Test.stubServer epp.config and epp-http.config properties along with EPPSessionTst to configure the tests for different environments. More tests and potentially properties will be added to parameterize the tests to run against more than the Stub Server.
11. Added the Name Store and SRS (COM/NET) OT&E and Production host names to epp.config, and the Name Store OT&E and Production URL's to epp-http.config for easier configuration.

2.11 Version 3.2 to Version 3.3

This section highlights the changes made to the SDK from the previous version, version 3.2. The following have been added or changed from the prior version:

1. Fixed an issue with EPPDomainInfoCmd when the hosts attribute is not set, which is the default, that caused the com.verisign.epp.codec.domain.EPPDomainTst test to fail.
2. Fixed incorrect reference of EPPCLSBidInterfaceTst.suite() to EPPCLSItemInterfaceTst.suite() in EPPCLSItemInterfaceTst.java.
3. Added the Whois Info Extension contained under the whois SDK directory.
4. Added the ability to set the client host name or IP address when creating a TCP or SSL connection. This ability was also added to the pools with the EPP.ClientHost or EPP.SessionPool.<pool>.clientHost properties. Many files were modified to accommadate this feature.
5. Added the EDU constants to NSSubProduct.
6. Changed the EPPDomainPendActionMsg isSuccess and setSuccess methods to isPASuccess and setPASuccess so that they don't incorrectly override the same methods in the EPPResponse base class. Added additional codec and interfaces test code.
7. Fixed adding the svcExtension elements in EPPLoginCmd. The extservices attributes had to be set with EPPFactory.getInstance().getExtensions().
8. Added insertion of the EPPRgpExtInfData extension in NSDomainHandler.doDomainInfo() when the domain name is pendingrestore.com for testing the use of the extension.
9. Added convenience constructor to EPPRgpextInfData that takes an EPPRgpExtStatus parameter.
10. Add inserting an EPPLowBalancePollResponse in NSDomainHandler.doDomainCreate when the domain name is "lowbalancepoll.com".
11. Added handling of the new EPPAssemblerException constant objects types COMMANDNOTFOUND, RESPONSENOTFOUND, and EXTENSIONNOTFOUND.
12. Added handling of EPPComponentNotFoundException in EPPXMLAssembler by mapping the kind attribute to EPPAssemblerException constant object types.

13. Added COMMANDNOTFOUND, RESPONSENOTFOUND, and EXTENSIONNOTFOUND constant objects to EPPAssemblerException to identify when a specific compoonent can not be found in EPPAssembler.toEvent.
14. Added EPPComponentNotFoundException to be thrown by EPPCodec.decode, EPPCommand.decode, and EPPResponse.decode when a mapping or extension component can not be found.
15. Fixed text included in error log and EPPConException when EPPSSLClientSocket can not establish a connection.
16. Added new Log4J category called com.verisign.epp.util.EPPXMLStream.packet for logging packets sent and received for routed to a seperate log file. Added new appender in logconfig.xml to demonstrate creating a seperate log. This is based on SR 472609.
17. Updated com.verisign.epp.codec.rcccontact.EPPContactUpdateCmd to not default the legal contact to the value of the general contact. This is based on SR #464229.
18. Added negative test scenario change to NSDomainHandler to return PARAM_VALUE_POLICY_ERROR with a EPPNamestoreExtNSExtErrData extension when the subProductID is "BAD".

2.12 Version 3.1 to Version 3.2

This section highlights the changes made to the SDK from the previous version, version 3.1. The following have been added or changed from the prior version:

1. Support for JDK 1.3 with JSSE 1.2 has been dropped to utilize the JDK 1.4 JSSE API's. This will greatly ease the use of different JSSE providers with the SDK.
2. Added the ability to use multiple session pools with a system name as a key (i.e. srs, namestore). The EPPSessionPool class added overloaded borrowObject, returnObject, and invalidateObject methods that include the system name as a parameter. The configuration parameter EPP.SessionPool.systemPools was added along with the ability to specify the serverName and serverPort along with the other standad pool parameters using the parameter name format EPP.SessionPool.<system>.<param>. If EPPSessionPool.systemPools is not defined, the EPP.SessionPool.<prop> will be used and setting EPP.SessionPool.systemPools=default is equavalent to not defining the property.
3. Added handing of `com.verisign.epp.codec.domain.EPPDomainPendActionMsg` to `NSPollTst` and `NSPollHandler` in support of JOBS. Also added handling of `EPPClsPollResponse` in `NSPollHandler`.
4. Added JOBS and NAME constants to `NSSubProduct`, and added checking for null in the `setSubProductID` methods of `NSDomain` and `NSHost`.
5. Added CLS product.
6. Added encode/decode methods for `BigDecimal` to `EPPUtil` in the codec package in gen.
7. In `EPPSSLImpl` and `EPPHttpSession` removed Sun JSSE 1.0.2 references and used default algorithm for both the `KeyManagerFactory` and the `TrustManagerFactory`, and deprecated use of `EPPEnv.getSSLKeyManager`. This will allow the SDK to easily

use different JSSE providers. These changes require JDK 1.4 to compile and run the SDK with the default JSSE implementation.

8. Added `EPPSession(String, int)` method and dependent methods to allow for a client connection to made to a different host and port than what is defined in epp.config. This is useful if the client needs to connect to multiple servers.
9. Added `EPPDomain.setAuthRoid(String)` and `EPPDomain.getAuthRoid()` for use with thick registries and the `sendTransfer` and `sendInfo` methods.
10. Fixed bug with setting the roid in `EPPAuthInfo` where the roid was placed in the `authInfo` element instead of the `pw` element. Updated associated `EPPDomainTst's`.
11. Updated the default configuration settings for bundles/namestore/epp.config and bundles/namestore/epp-http.config to work better when testing in SSL mode.
12. Added the ability to run with both TCP and HTTP in `EPPIdnDomainTst` and `EPPClsBidInterfaceTst` similar to the other interfaces tests.
13. Changed `EPPHttpSession` service comparison check to log a warning when there is a mismatch in the same way as `EPPSession`.
14. Removed unused `EPPHTTPClientCon` and `EPPHTTPSClientCon` since `EPPHTTPSession` utilizes Jakarta HTTP Client in place of these classes.

2.13 Version 2.5 to Version 3.1

This section highlights the changes made to the SDK from the previous version, version 2.5. The following have been added or changed from the prior version:

1. Addition of session pool, both HTTP and TCP, which incorporates automatic management of idle timeouts, absolute timeouts, and other general pool features like minimum pool size and maximum pool size. Apache Common Pool (http://jakarta.apache.org/commons/pool/) is used with the configuration parameters added to the SDK config for customization. The kind of session stored in the pool EPPHttpSession and EPPSession is specified with the SDK config property `EPP.SessionPool.poolableClassName,` with EPPSession as the default. The use of the session pool is optional, but the source and test code can be used as a sample for implementation of a session pool against the Namestore EPP server. Look in the `com.verisign.epp.pool` Java package in the gen SDK for more details.
2. Creation of a `com.verisign.epp.namestore.interfaces.NSDomain` interface that extends the `com.verisign.epp.interfaces.EPPDomain` interface for providing a simplier interface to Namestore specific operations like RGP, Sync, and specifying the target sub-product id. Without NSDomain, the use of command extensions to the EPPDomain interface would have to be used along with seamily unrelated operations like *sendUpdate*. Look to `com.verisign.epp.namestore.interfaces.NSDomainTst` for a sample of using the NSDomain interface along with the use of the session pool described above.
3. Changed the handling of a mismatched login and greeting services in `com.verisign.epp.interfaces.EPPSession.initSession()` to log a warning

log message instead of raising an exception. The server will return an error if the requested services is not valid.

4. Added `com.verisign.epp.codec.gen.EPPResponse.getResult() : EPPResult` to return the first of the list of results associated with the response. Most if not all responses contain a single result, so using getResult() instead of getResults() will ease in the processing of responses.
5. Addition of the *nsfinance* SDK component, which initially contains the specification and interface for the low balance EPP poll notification.
6. Addition of the *idn* SDK component, which initially contains the Domain IDN Language Tag extension used to specify the language tag of IDN domains. NSDomain provides a utility method called setIDNLangTag(String) for specifying the value on calls to sendCreate().
7. Addition of *rgp* SDK component, which contains specification and interfaces for sending domain restore requests, domain restore reports, response extension for RGP statuses, and RGP notification message.
8. Addition of the *sync* SDK component, which contains the specification and interface for sending a domain sync command.

3. Namestore and SRS Supported Transports

Namestore and the dotCOM and dotNET (SRS) environments SSL as the EPP transport per RFC 5734. 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 Namestore and SRS 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 Namestore and SRS SDK. With the source distribution, the Unix filename is ***epp-namestore--${BUILD_VER}-src.tar.gz*** and the Windows filename is ***epp-namestore-${BUILD_VER}-src.zip***. With the binary distribution, the Unix filename is ***epp-namestore-${BUILD_VER}-bin.tar.gz*** and the Windows filename is ***epp-namestore-${BUILD_VER}-bin.zip***. ***${BUILD_VER}*** is the version number for the release (e.g 1.0.0).
2. Change to the Namestore and SRS SDK directory: ***eppsdk/bundles/namestore***
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 - NameStore build.xml Targets.

Table 1 - NameStore build.xml Targets

Target	**Distribution (src, bin, or both)**	**Description**
`clean`	`both`	`Cleans the built files and directories`
`compile`	`src`	`Compiles the source files`
`dist`	`src`	`Creates the distributions (-Dbuild.version required)`
`dist-bin`	`src`	`Creates the binary distribution (-Dbuild.version required)`
`dist-src`	`src`	`Creates the source distribution (-Dbuild.version required)`
`doc`	`src`	`Creates the HTML API documentation`
`format`	`src`	`Formats the source code`
`init`	`both`	`Initializes the build for rest of targets`

`jar`	`src`	`Creates the jar file (epp-namestore.jar)`
		`Default for src distribution`

`start-server`	`both`	`Starts the TCP Stub Server`

`test`	`both`	`Runs all tests (CODEC and TCP).`
		`Default for bin distribution`

`test-client`	`both`	`Runs tests over TCP against TCP Stub Server`
`test-client-server`	`both`	`Runs full client server test over TCP.`
`test-codec`	`both`	`Runs CODEC unit tests`

4.1 Running SDK Tests via Stub Server

The SDK works with JDK 1.4 and JDK 1.5. 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 NameStore or SRS Server

The SDK configuration has to be changed to communicate with the real NameStore or SRS Server, since the transport is SSL for NameStore and for SRS. The NameStore client mappings/extensions that are used might have to be changed. Set the properties in "Table 2 - Changes Required to Interface with NameStore or SRS via SSL" in ***epp.config***.

Table 2 - Changes Required to Interface with NameStore or SRS via SSL

Property	Update To	Default
EPP.MapFactories	MapFactories for products that will be provisioned. For example, if .tv and .cc domains are only provisioned, set `EPP.MapFactories` to: • `com.verisign.epp.codec.domain.EPPDomainMapFactory` • `com.verisign.epp.codec.host.EPPHostMapFactory`	All NameStore and SRS map factories. **Use of the default map factories is recommended**
EPP.CmdRspExtensi	Dependent command/response extensions	All NameStore and SRS

ons	needed by the products that will be provisioned. For example, if .tv and .cc domains are only provisioned, set `EPP.CmdRspExtensions` to: • `com.verisign.epp.codec.namestoreext.EPPNamestoreExtExtFactory`	extension factories **Use of the default extension factories**
EPP.SSLKeyFileName	JSSE keystore file name that contains the private key and associated certificate.	../../lib/keystore/testkeys
EPP.SSLPassPhrase	Password needed to access `EPP.SSLKeyFileName` file.	passphrase
EPP.SSLKeyPassPhrase	Password needed to access the private key defined in the `EPP.SSLKeyFileName` file. If this property is not defined, `EPP.SSLPassPhrase` will be used for accessing both the keystore and the private key.	Not Defined
EPP.SSLProtocol	SSL protocol of the configured provider	TLS
EPP.SSLKeyStore	SSL keystore file type	JKS
EPP.SSLTrustStoreFileName	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 will be used.	../../lib/keystore/testkeys **It is recommended to comment out or not define this property when interfacing with NameStore or SRS**
EPP.SSLTrustStorePassPhrase	Password for accessing the trust store defined by the `EPP.SSLTrustStoreFileName` property. This property is required if `EPP.SSLTrustStoreFileName` is defined.	passphrase
EPP.ClientSocketName	Class used to make client connections. Set this to `com.verisign.epp.transport.client.EPPSSLClientSocket` for SSL.	com.verisign.epp.transport.client.EPPPlainClientSocket
EPP.ServerName	Set to the VNS NameStore or SRS server name or IP address. The following are possible values: 1. NameStore OTE - otessl.verisign-grs.com 2. NameStore Production – namestoressl.verisign-grs.com 3. SRS OTE – epp-ote.verisign-grs.com	localhost

	4. SRS Production - epp.verisign-grs.net	
EPP.ServerPort	Set to the VNS NameStore server port number, which should be 700.	1700

The SDK uses Sun's reference implementation of JSSE in the JDK (1.4 and above). Use the EPP.SSL properties of the SDK to configure JSSE. Please consult Sun's 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:

- eppsdk/lib/epp/epp-namestore.jar
  This .jar file includes all of the NameStore EPP mappings and extensions
- eppsdk/lib/*.jar
  These are dependent third party .jar files including JUnit, Log4j, PoolMan, and XercesJ

4.4 Example Code to initialize the Namestore and SRS SDK and Start Session

The following code can be used to initialize a session with a 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 Namestore and SRS 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 then 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 Namestore and SRS 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 Namestore and SRS SDK to review the source of the tests included in Table 3 - SDK Interface Test Classes.

Table 3 - SDK Interface Test Classes

Test Classes	Description
com.verisign.epp.interfaces.EPPCoaDomainTst	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.
com.verisign.epp.interfaces.EPPContactTst	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.
com.verisign.epp.interfaces.EPPDomainTst	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.
com.verisign.epp.interfaces.EPPHostTst	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.
com.verisign.epp.interfaces.EPPIdnDomainTst	Tests send a domain create command with the IDN language extension (EPPIdnLangTag). This test creates an individual EPP session without the SDK session pool.
com.verisign.epp.interfaces.EPPJobsContactTst	Test of using the the RFC 5733 contact commands along with the Jobs Contact Extension. This test creates an individual EPP session without the SDK session pool .
com.verisign.epp.interfaces.EPPLowBalanceDomainTst	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.
com.verisign.epp.interfaces.EPPNamestoreExtDomainTst	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.
com.verisign.epp.interfaces.EPPNamestoreExtHostTst	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.

com.verisign.epp.interfaces.EPPPremiumDomainTst	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.
com.verisign.epp.interfaces.EPPRgpDomainTst	Test of using the Domain Registry Grace Period (RGP) extension defined in RFC 3915 to restore a domain and to retrieve RGP statuses from the Stub Server. This test creates an individual EPP session without the SDK session pool.
com.verisign.epp.interfaces.EPPSecDNSDomainTst	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.
com.verisign.epp.interfaces.EPPSessionTst	Test of doing the RFC 5730 commands (login, hello, poll, and logout).
com.verisign.epp.interfaces.EPPSuggestionTst	Test of sending a set of randomized Name Suggestion commands. This test creates an individual EPP session without the SDK session pool.
com.verisign.epp.interfaces.EPPSyncDomainTst	Test of using ConsoliDate extension (EPPSyncExtUpdate) to synchronize the experation date of a domain. This test creates an individual EPP session without the SDK session pool.
com.verisign.epp.interfaces.EPPWhoisDomainTst	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.
com.verisign.epp.namestore.interfaces.NSDomainTst	Test of using NSDomain for all of the domain commands. This test also utilizes the SDK session pool.
com.verisign.epp.namestore.interfaces.NSHostTst	Test of using NSHost for all of the host commands. This test also utilizes the SDK session pool.
com.verisign.epp.namestore.interfaces.NSPollTst	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.
com.verisign.epp.pool.EPPSessionPoolTst	Test of using the EPPSessionPool for an individual session pool. It utializes the session pool to send a hello and poll command. It also tests the absolute and idle timeout features of the pool.
com.verisign.epp.pool.EPPSystemSessionPoolTst	Test using two session pools (default and "test").
com.verisign.epp.interfaces.EPPWhoWasTst	Test of sending a set of WhoWas commands. This test creates an individual EPP session without the SDK session pool.
com.verisign.epp.interfaces.EPPBalanceTst	Test of sending a Balance Info Command.

5. SDK Overview

The Namestore and SRS 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 Pipelining is sending multiple commands and processing the responses asynchronously.

The generic EPP client interface classes are described in Section 5 -

Generic EPP Client Interfaces. 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 Namestore and SRS SDK will have the following directories:

Table 4 - Namestore and SRS SDK Directories

Directory	Description
eppsdk/bundles/namestore	Main directory for the NameStore EPP SDK bundle. 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 - NameStore build.xml Targets.
eppsdk/bundles/namestore/doc	This directory contains bundled documentation for the Namestore and SRS 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.
eppsdk/bundles/namestore/doc /html	This directory contains the bundled interface specification in Javadoc format.
eppsdk/lib	This directory contains dependent JAR files including JUnit, Log4j, PoolMan, and XercesJ.
eppsdk/lib/epp	This directory contains the bundled JAR and WAR files for the Namestore and SRS SDK (epp-namestore.jar and namestore.war).

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 - Namestore and SRS SDK Packages provides an overview of the SDK packages.

Table 5 - Namestore and SRS SDK Packages

Package	Description
com.verisign.epp.codec	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).
com.verisign.epp.exception	General EPP SDK exception classes
com.verisign.epp.interfaces	Client interface classes including EPPApplication and

	EPPSession
com.verisign.epp.framework	EPP Server Framework classes used by the Stub Server
com.verisign.epp.serverstub	Stub Server classes including handlers for each of the supported EPP Command Mappings.
com.verisign.epp.util	Set of SDK utility classes including EPPEnv.
com.verisign.epp.pool	Session Pool classes

6. SDK Configuration

6.1 Configuration File

The Namestore and SRS 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.getSystemResourceAsStream(*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. 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.

Parameter	**Process**	**Required**	**Description**
EPP.LogMode	Both	Yes	Log Configuration Mode. The mode controls the way by which the logging facility (Log4J) is initialized. There are three different modes: • BASIC - Initialize logging using EPP.LogLevel and EPP.LogFile • CFGFILE - Initialize logging using EPP.LogCfgFile and optionally EPP.LogCfgFileWatch • 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.

EPP.LogLevel	Both	No	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
EPP.LogFile	Both	No	Log4J Log File Name. Logs will be appended to the log file. The default file is “epp.log”.
EPP.LogCfgFile	Both	No	Log4J XML Configuration File Name used to define the log levels and the appenders (file, syslog, etc.). The default file is “logconfig.xml”.
EPP.LogCfgFileWatch	Both	No	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).
EPP.ConTimeOut	Both	Yes	Connection and read timeout in milliseconds. A setting of 0 indicates no timeout. The default setting is 500000 milliseconds (500 seconds).
EPP.ClientSocketName	Client	No	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). The EPP.ProxyServersLocator property must be set and the EPP.ProxyServers and EPP.ProxyServersRandomize should be set.**

			com.verisign.epp.transport.client.EPPSSLProxyClientSocket - SSL TCP/IP socket connection(s) that connects through an Apache Proxy Server. The EPP.ProxyServersLocator property must be set and the EPP.ProxyServers and EPP.ProxyServersRandomize should be set.
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>']'?** **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.**
EPP.ClientHost	Client	No	Host name or IP Address that the client will connect from. If not defined the client host will default to the loopback address.
EPP.ServerName	Both	Yes	Host name or IP Address that the server will listen on and that the client will connect to. The default setting is “localhost”.
EPP.ServerPort	Both	Yes	Port that the server will listen on and that the client will connect to.
EPP.ServerSocketName	Server	Yes	Concrete server socket class used by the Stub Server. The classes provided in the SDK include: com.verisign.epp.transport.server.EPPPlainServer - Plain TCP/IP socket connection(s)

			com.verisign.epp.transport.server.EPPSSLServer - SSL TCP/IP socket connection(s)
EPP.MapFactories	Client	Yes	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.
EPP.ServerAssembler	Server	No	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.
EPP.ProtocolExtensions	Client	No	Space separated list of fully qualified EPP Protocol Extension factory class names. There is one EPP Protocol Extension Factory Mapping per EPP Protocol Extensions. See the Programmer Guide for the desired EPP Command Mapping for more details.
EPP.CmdRspExtensions	Client	No	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
EPP.ServerEventHandler s	Server	Yes	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.
EPP.PollHandlers	Server	No	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.
EPP.SSLProtocol	Both	Yes	JSSE Protocol used. The possible values include: TLS - Supports some version of TLS SSL - Supports some version of SSL SSLv2 - Supports SSL version 2 or higher SSLv3 - Supports SSL version 3 TLSv1 - Supports TLS version 1
EPP.SSLKeyStore	Both	No	JSSE KeyStore format. The default setting is “JKS”.
EPP.SSLKeyFileName	Both	No	JSSE KeyStore file used for authentication. The SDK

			includes a self-signed certificate in the KeyStore “testkeys”.
EPP.SSLPassPhrase	Both	No	JSSE KeyStore pass-phrase. The SDK provided KeyStore has a pass-phrase of “passphrase”.
EPP.SSLKeyPassPhrase	Both	No	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.
EPP.SSLEnabledProtocols	Both	No	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 delimeters. Examples of protocols include: • 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 delimeted 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 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/Serv	No	Minimum number of objects that can be in the pool. Default

minimumSize	er		value is 0.
PoolMan.[Client/Server]. maximumSize	Client/Server	No	Maximum number of objects that can be in the pool. Default value is Integer.MAX_VALUE
PoolMan.[Client/Server]. maximumSoft	Client/Server	No	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.
PoolMan.[Client/Server]. skimmerFrequency	Client/Server	No	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)
PoolMan.[Client/Server]. shrinkBy	Client/Server	No	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.
PoolMan.[Client/Server]. debugging	Client/Server	No	Write debug messages? Debug messages are written to the file specified by the PoolMan.Client.logFile parameter. Default is false.
PoolMan.[Client/Server]. objectTimeout	Client/Server	No	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)
PoolMan.[Client/Server]. userTimeout	Client/Server	No	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).
EPP.Validating	Client/Server	No	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.
EPP.FullSchemaChecking	Client/Server	No	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.
EPP.MaxPacketSize	Client/Server	No	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.
EPP.SessionPool.poolableClassName	Client	No	Fully qualified class name of object pooled by the EPPSessionPool. Set to

			com.verisign.epp.pool.EPPSessionPoolableFactory to TCP session pooling. Default is com.verisign.epp.pool.EPPSessionPoolableFactory
EPP.SessionPool.clientId	Client	No	Client id/name used to authenticate session in the session pool. Required if using the EPPSessionPool.
EPP.SessionPool.password	Client	No	Password used to authenticate session in the session pool. Required if using the EPPSessionPool.
EPP.SessionPool.absoluteTimeout	Client	No	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)
EPP.SessionPool.idleTimeout	Client	No	Idle timeout of session in millisonds of a sessin 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)
EPP.SessionPool.minIdle	Client	No	Minimum number of idle sessions in the session pool. Default is 5
EPP.SessionPool.maxIdle	Client	No	Maximum number of idle sessions in the session pool. Default is 10
EPP.SessionPool.maxActive	Client	No	Maximum number of active sessions borrowed from the session pool. Default is 10
EPP.SessionPool.initMaxActive	Client	No	Boolean value that will pre-initialize maxActive sessions in the pool. Default is false
EPP.SessionPool.borrowRetries	Client	No	Number of retries when there is a failure in borrowing a new session from the pool. This eliminiates the client having to implement its own retry loop on a call to EPPSessionPool.borrowObject() and also applies to pre-initalizing the sessions when EPP.SessionPool.initMaxActive is true. Default is 0
EPP.SessionPool.maxWait	Client	No	The maximum number of milliseconds a client will block waiting for a session from the session pool. Default is 60000 (1 minute)
EPP.SessionPool.timeBetweenEvictionRunsMillis	Client	No	Frequency in milliseconds to scan idle sessions in the session pool for timeouts. Default is 500 (1/2 second)
EPP.SessionPool.systemPools	Client	No	Default the set of system session pools in a comma separated list of names (i.e. srs,namestore). The system

			name “default” initializes a default pool that uses the EPP.SessioinPool.<param> property along with other properties like EPP.ServerName and EPP.ServerPort. The default pool uses the EPPSessionPool methods that don’t take a aSystem parameter.
EPP.SessionPool.<system>.poolableClassName	Client	No	System specific setting of EPP.SessionPool.poolableClassName, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.serverName	Client	No	System specific setting of EPP.ServerName, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.serverPort	Client	No	System specific setting of EPP.ServerPort, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.clientHost	Client	No	TCP client host name. If not defined, the loopback will be used.
EPP.SessionPool.<system>.clientId	Client	No	System specific setting of EPP.SessionPool.clientId, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.password	Client	No	System specific setting of EPP.SessionPool.password, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.absoluteTimeout	Client	No	System specific setting of EPP.SessionPool.absoluteTimeout, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.idleTimeout	Client	No	System specific setting of EPP.SessionPool.idleTimeout, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.minIdle	Client	No	System specific setting of EPP.SessionPool.minIdle, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.maxIdle	Client	No	System specific setting of EPP.SessionPool.maxIdle, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.maxActive	Client	No	System specific setting of EPP.SessionPool.maxActive, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.initMaxActive	Client	No	System specific setting of EPP.SessionPool.initMaxActive, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.borrowRetries	Client	No	System specific setting of EPP.SessionPool.borrowRetries, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.maxWait	Client	No	System specific setting of EPP.SessionPool.maxWait, where <system> is replaced with the system name (i.e. srs, namestore)

EPP.SessionPool.<system>.timeBetweenEvictionRunsMillis	Client	No	System specific setting of EPP.SessionPool.timeBetweenEvictionRunsMillis, where <system> is replaced with the system name (i.e. srs, namestore)
EPP.SessionPool.<system>.SSLProtocol	Client	No	Protocol to use for pool, where <system> 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
EPP.SessionPool.<system>.SSLKeyStore	Client	No	Type of identity KeyStore, where <system> is replaced with the system name (i.e. srs, namestore). Required if SSLProtocol is defined for pool. Default is JKS
EPP.SessionPool.<system>.SSLKeyFileName	Client	No	Name of the identity KeyStore file, where <system> is replaced with the system name (i.e. srs, namestore). Required if SSLProtocol is defined for pool. Default is ../../lib/keystore/testkeys
EPP.SessionPool.<system>.SSLPassPhrase	Client	No	Passphrase/password to access the identity KeyStore file defined by SSLKeyFileName pool property, where <system> is replaced with the system name (i.e. srs, namestore). Required if SSLProtocol is defined for pool. Default is passphrase
EPP.SessionPool.<system>.SSLEnabledProtocols	Client	No	Optional space delimited list of enabled SSL protocols, where <system> is replaced with the system name (i.e. srs, namestore).
EPP.SessionPool.<system>.SSLEnabledCipherSuites	Client	No	Optional space delimited list of SSL cipher suites, where <system> is replaced with the system name (i.e. srs, namestore). Examples include: • SSL_RSA_WITH_RC4_128_MD5 • SSL_RSA_WITH_RC4_128_SHA
EPP.SessionPool.<system>.SSLKeyPassPhrase	Client	No	Optional passphrase/password for the private key stored in the identity KeyStore, where <system> is replaced with the system name (i.e. srs, namestore). If undefined the pool SSLPassPhrase will be used.
EPP.SessionPool.<system>.SSLTrustStore	Client	No	Optional Type of the Trust Store, where <system> 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
EPP.SessionPool.<system>.SSLTrustStoreFileName	Client	No	Pool trust store file that contains the list of Certificate Authorities that should be trusted. If not set than the pool defaults to the keystore that comes with the JDK which is: $JAVA_HOME/jre/lib/security/cacerts. It is recommended to comment out or not define this

			property when connecting to the NameStore or SRS Servers.
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 Namestore and SRS SDK library, ***epp-namestore.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 1.6.0 – .jar files in eppsdk/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 2.6.0 – Library used for XML parsing. This includes xercesImpl-2.6.0.jar and xmlParserAPIs-2.6.0.jar

- poolman-2.1-b1.jar – Library used for pooling the XML Parsers
- Jalopy 1.0b10 – Library used for code formatting. .jar files in eppsdk/lib/jalopy
- Apache Common Pool 1.1 – Library used for implementing the EPP session pool.

Each set of product specific EPP Command Mappings are included in the *epp-namestore.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 Namestore and SRS 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{yyyyMMdd HHmmss}  %c %-5p %m\n`", which will log the date in the format "`yyyyMMdd HHmmss`", the category as the fully qualified class name, the priority (DEBUG, INFO, WARN, ERROR, or FATAL), and the log message.

6.3.2 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 HHmmss”, the category as the fully qualified class name, the priority (DEBUG, INFO, WARN, ERROR, or FATAL), and the log message. The root category is set with a priority of “debug”, so that all SDK messages will be logged.

```
<log4j:configuration>

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

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

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

</log4j:configuration>
```

Figure 1 - Default SDK Log4J Configuration File

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.

Category	**Description**
com.verisign.epp.util.EPPXMLStrea m	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.
com.verisign.epp.util. EPPSchemaCachingEntityResolver	Class that handles the loading and caching of XML schemas. The loading is done from the CLASSPATH, where the schemas need to reside in the directory "schemas".
com.verisign.epp.transport	Package that contains the SDK transport classes. When the "debug" priority level is enabled, the trace of the primary classes will be logged.
com.verisign.epp.interfaces	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.
com.verisign.epp.codec	Package that contains the EPP encoding/decoding classes. Currently, these classes don't log messages, but it is likely that

Category	Description
	logs will be added in future releases.
com.verisign.epp.pool	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.

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 eppsdk/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*, respectively. 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.

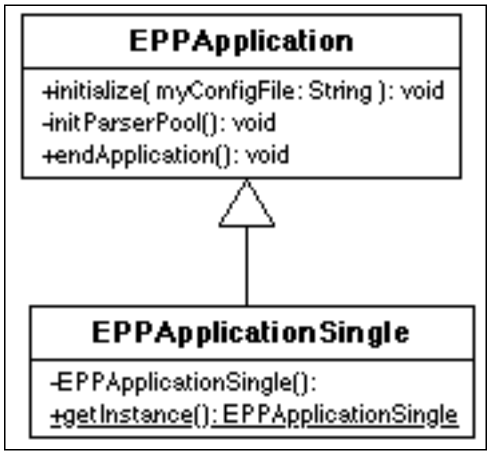


Figure 2 - EPPApplication Class Diagram

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.

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

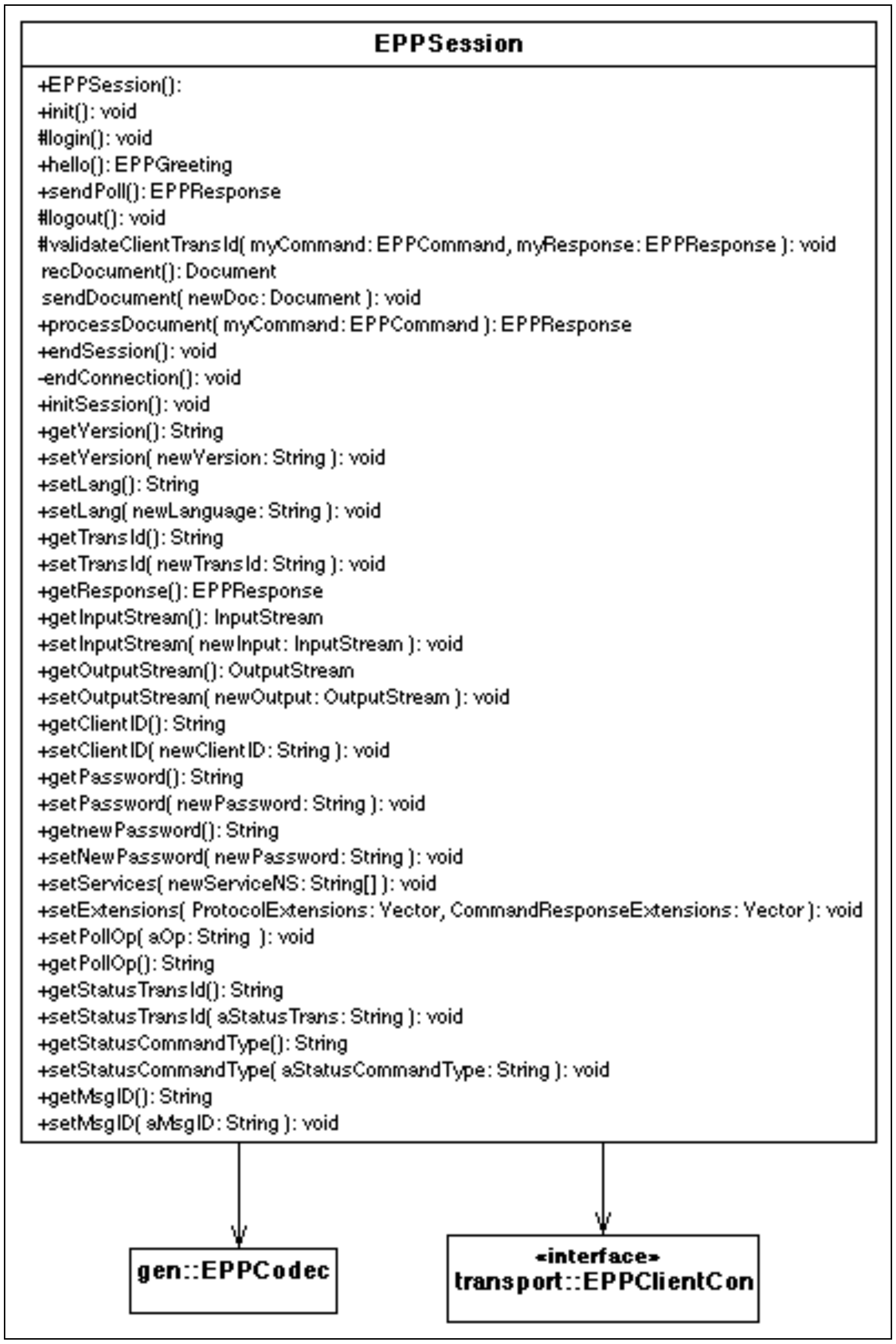
```
// Create one or more EPP Server sessions.
EPPSession session = new EPPSession();
```

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.



```
EPPSession
+EPPSession():
+init(): void
#login(): void
+hello(): EPPGreeting
+sendPoll(): EPPResponse
#logout(): void
#validateClientTransId( myCommand: EPPCommand, myResponse: EPPResponse ): void
 recDocument(): Document
 sendDocument( newDoc: Document ): void
+processDocument( myCommand: EPPCommand ): EPPResponse
+endSession(): void
-endConnection(): void
+initSession(): void
+getVersion(): String
+setVersion( newVersion: String ): void
+setLang(): String
+setLang( newLanguage: String ): void
+getTransId(): String
+setTransId( newTransId: String ): void
+getResponse(): EPPResponse
+getInputStream(): InputStream
+setInputStream( newInput: InputStream ): void
+getOutputStream(): OutputStream
+setOutputStream( newOutput: OutputStream ): void
+getClientID(): String
+setClientID( newClientID: String ): void
+getPassword(): String
+setPassword( newPassword: String ): void
+getnewPassword(): String
+setNewPassword( newPassword: String ): void
+setServices( newServiceNS: String[] ): void
+setExtensions( ProtocolExtensions: Vector, CommandResponseExtensions: Vector ): void
+setPollOp( aOp: String ): void
+getPollOp(): String
+getStatusTransId(): String
+setStatusTransId( aStatusTrans: String ): void
+getStatusCommandType(): String
+setStatusCommandType( aStatusCommandType: String ): void
+getMsgID(): String
+setMsgID( aMsgID: String ): void
```

Figure 4 - EPPSession Class Diagram

Figure 5 - EPPSession Life Cycle shows the sequence in creation, usage, and closing of an *EPPSession*.. The following is a further description of the sequence:

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()*. 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()*. This is useful when an EPP Server does not support all of the services or the extensions loaded in the SDK, with the use of the *EPP.MapFactories, EPP.ProtocolExtensions, EPP.CmdRspExtensions* configuration parameters.
4. The *EPPSession* will read the EPP Greeting from the EPP Server, will create an EPP Login, validate that the server services are compatible with the client services, 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.

Figure 5 - EPPSession Life Cycle Sequence

7.2.2 Sample Code

Figure 6 - EPPSession Life Cycle Sample Code shows the code associated with the **Figure 5 - EPPSession Life Cycle Sequence**. 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.

```
// 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 seesion
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 6 - 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 VNS 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 VNS 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. This is useful when a specific EPP Server supports a subset of the services loaded in the SDK.
- *addExtensions(Vector,Vector)* – Set the Extensions to use with the session. The default extensions are derived from the classes defined by the *EPP.ProtocolExtensions ,*

EPP.CmdRspExtensions configuration parameter. This is useful when a specific EPP Server supports a subset of the extensions loaded in the SDK.

- *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 asynchronousnly.

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:

Constant Name	**Constant Value**	**Description**
EPPResult.SUCCESS	1000	Session has successfully been initialized. initSession 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 already been established, which could be due to initSession being called more than once.
EPPResult.PARAM_OUT_OF_RANGE	2004	Input attribute (i.e. client identifier, password) not valid
EPPResult.COMMAND_FAILED	2500	Internal EPP Server error
EPPResult.AUTHENTICATION_ERROR	2200	Not a valid user
EPPResult.AUTHORIZATION_ERROR	2201	User is not authorized to login.
EPPResult.TIMEOUT_END	2501	Command timeout has occurred. The EPP Server

		closes the connection.

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:

Constant Name	Constant Value	Description
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:

Constant Name	Constant Value	Description
EPPResult. SUCCESS	1000	Poll acknowledgement command was successful
EPPResult.SUCCESS_POLL_NO_MSGS	1300	Successful Poll request command. There are no messages in the queue.
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 7 - EPPEnv Class Diagram** shows the class diagram for *EPPEnv* and *EPPEnvSingle*.

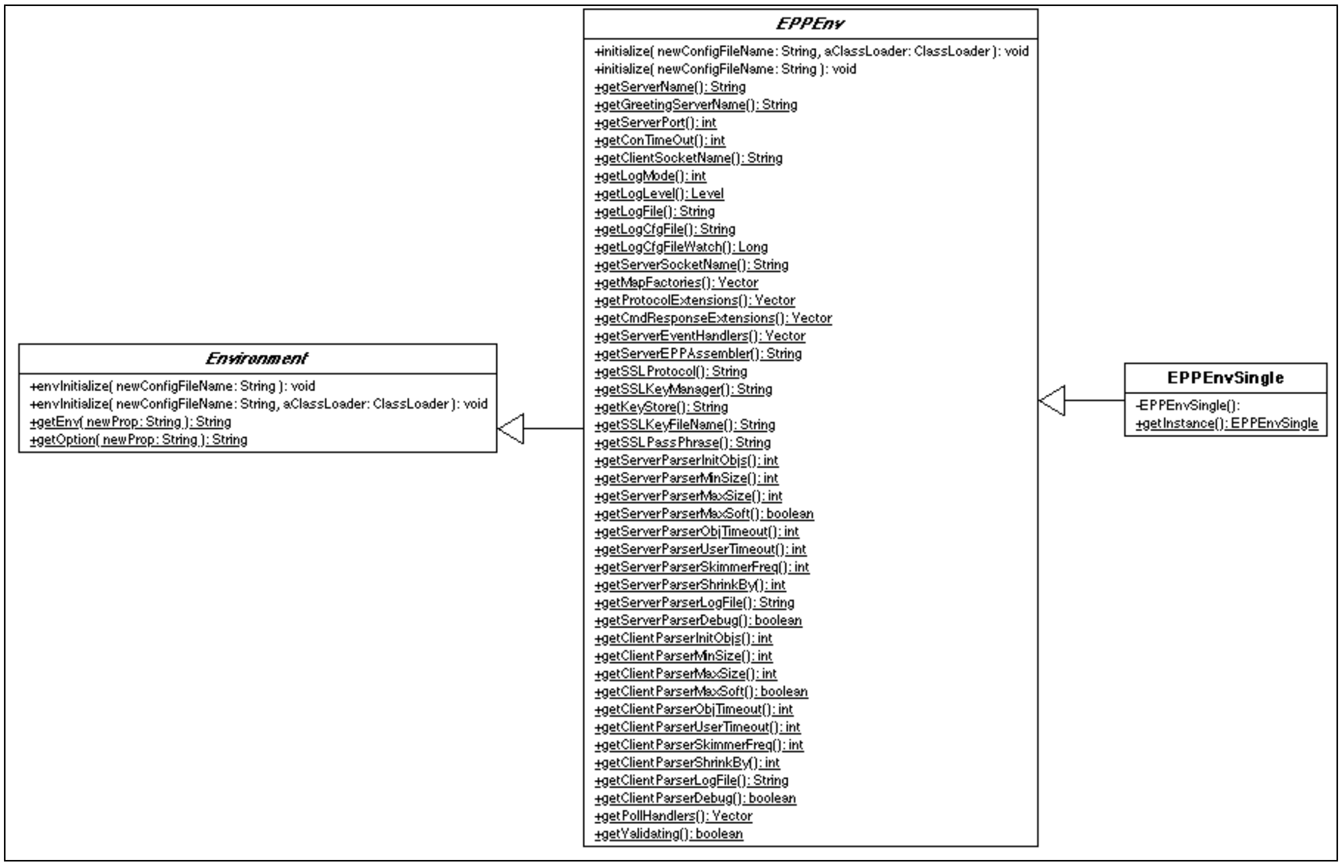

EPPEnv
+initialize(newConfigFileName: String, aClassLoader: ClassLoader): void
+initialize(newConfigFileName: String): void
+getServerName(): String
+getGreetingServerName(): String
+getServerPort(): int
+getConTimeOut(): int
+getClientSocketName(): String
+getLogMode(): int
+getLogLevel(): Level
+getLogFile(): String
+getLogCfgFile(): String
+getLogCfgFileWatch(): Long
+getServerSocketName(): String
+getMapFactories(): Vector
+getProtocolExtensions(): Vector
+getCmdResponseExtensions(): Vector
+getServerEventHandlers(): Vector
+getServerEPPAssembler(): String
+getSSLProtocol(): String
+getSSLKeyManager(): String
+getKeyStore(): String
+getSSLKeyFileName(): String
+getSSLPassPhrase(): String
+getServerParserInitObjs(): int
+getServerParserMinSize(): int
+getServerParserMaxSize(): int
+getServerParserMaxSoft(): boolean
+getServerParserObjTimeout(): int
+getServerParserUserTimeout(): int
+getServerParserSkimmerFreq(): int
+getServerParserShrinkBy(): int
+getServerParserLogFile(): String
+getServerParserDebug(): boolean
+getClientParserInitObjs(): int
+getClientParserMinSize(): int
+getClientParserMaxSize(): int
+getClientParserMaxSoft(): boolean
+getClientParserObjTimeout(): int
+getClientParserUserTimeout(): int
+getClientParserSkimmerFreq(): int
+getClientParserShrinkBy(): int
+getClientParserLogFile(): String
+getClientParserDebug(): boolean
+getPollHandlers(): Vector
+getValidating(): boolean
Environment
+envInitialize(newConfigFileName: String): void
+envInitialize(newConfigFileName: String, aClassLoader: ClassLoader): void
+getEnv(newProp: String): String
+getOption(newProp: String): String
EPPEnvSingle
-EPPEnvSingle():
+getInstance(): EPPEnvSingle

Figure 7 - 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.

Method Name	Configuration Parameter
getServerName()	EPP.ServerName
getServerPort()	EPP.ServerPort
getConTimeOut	EPP.ConTimeOut
getClientSocketName()	EPP.ClientSocketName
getServerSocketName()	EPP.ServerSocketName
getMapFactories()	EPP.MapFactories
getProtocolExtensions()	EPP.ProtocolExtensions

Method Name	**Configuration Parameter**
getCmdResponseExtensions()	EPP.CmdRspExtensions
getSSLProtocol()	EPP.SSLProtocol
getSSLKeyManager()	EPP.SSLKeyManager
getKeyStore()	EPP.SSLKeyStore
getSSLKeyFileName()	EPP.SSLKeyFileName
getSSLPassPhrase()	EPP.SSLPassPhrase
getLogMode()	EPP.LogMode
getLogLevel()	EPP.LogLevel
getLogFile()	EPP.LogFile
getLogCfgFile()	EPP.LogCfgFile
getLogCfgFileWatch()	EPP.LogCfgFileWatch
getServerEventHandlers()	EPP.ServerEventHandlers
getPollHandlers()	EPP.PollHandlers
get[Client/Server]ParserInitObjs()	EPP.PoolMan.[Client/Server].initialObjects
get[Client/Server]ParserMinSize()	EPP.PoolMan.[Client/Server].minimumSize
get[Client/Server]ParserMaxSize()	EPP.PoolMan.[Client/Server]..maximumSize
get[Client/Server]ParserMaxSoft()	EPP.PoolMan.[Client/Server].maximumSoft
get[Client/Server]ParserObjTimeout()	EPP.PoolMan.[Client/Server].objectTimeout
get[Client/Server]ParserUserTimeout()	EPP.PoolMan.[Client/Server].userTimeout
get[Client/Server]ParserSkimmerFreq()	EPP.PoolMan.[Client/Server].skimmerFrequency
get[Client/Server]ParserShrinkBy	EPP.PoolMan.[Client/Server].shrinkBy
get[Client/Server]ParserLogFile()	EPP.PoolMan.[Client/Server].logFile

Method Name	Configuration Parameter
get[Client/Server]ParserDebug()	EPP.PoolMan.[Client/Server].debugging
getValidating()	EPP.Validating
GetFullSchemaChecking	EPP.FullSchemaChecking

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), which is not thread-safe. There are three approaches to overcoming this limitation, which include:

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

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

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

9. Extending the SDK

9.1 Transport

The SDK allows for the replacement of the transport layer with the setting of the 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 8 - EPPClientCon Class Diagram** shows the class diagram for *EPPClientCon*.

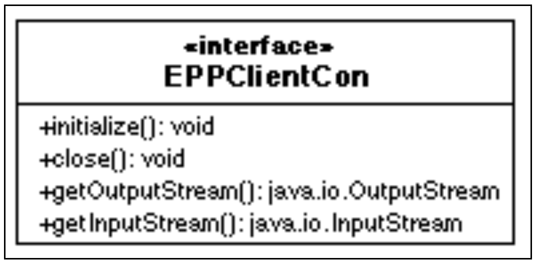

«interface»
EPPClientCon
+initialize(): void
+close(): void
+getOutputStream(): java.io.OutputStream
+getInputStream(): java.io.InputStream

Figure 8 - EPPClientCon Class Diagram

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

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

public class EPPClientConSample implements EPPClientCon {

        public EPPClientConSample() throws EPPConException {
                try {
                        _hostName = EPPEnv.getServerName();
                        _portNum  = EPPEnv.getServerPort();
                        _conTimeout = EPPEnv.getConTimeOut();
```

```
		}
		catch (EPPEnvException ex) {
			throw new EPPConException("Error initializing
			attributes: " + ex);
		}

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

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

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

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

		return _input;
	}

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

		return _output;
	}

	public void close() throws EPPConException {

		if (_socket != null) {

			try {
				_socket.close();
```

```
                        }
                        catch (IOException ex) {
                                throw new EPPConException("Closing socket: "
                                + ex);
                        }

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

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

} // End class EPPClientConSample
```

Figure 9 - SDK TCP/IP Transport Sample Code

10. Stub Server

The Namestore and SRS 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 a 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 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. Currently, there is no means of having an *EPPSession* use a connection pool. 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:

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

```
            EPPSessionPool.getInstance().returnObject(theSession);
}
```

11.1.1 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 initalize a pool for each system name specified in the comma separated list o 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.EPPSessionPoolab
leFactory

EPP.SessionPool.test.serverName=localhost
EPP.SessionPool.test.serverPort=1700
EPP.SessionPool.test.clientId=username
EPP.SessionPool.test.password=password
EPP.SessionPool.test.absoluteTimeout=82800000
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();
      ….
}
```

```
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 referencing the "default" system instead of using the equavalent default system methods of `EPPSessionPool`.

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

      EPPSessionPool.getInstance().invalidateObject(EPPSesionPool.DEFAULT,
theSession);
```

```
                  theSession = null;
            }
      }
      else if (theSession != null) {

      EPPSessionPool.getInstance().invalidateObject(EPPSesionPool.DEFAULT,
theSession);
            theSession = null;
      }
}
finally {
      if (theSession !- null)

      EPPSessionPool.getInstance().returnObject(EPPSessionPool.DEFAULT,theSes
sion);
}
```

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. `bundes/namestore/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. `bundes/namestore/epp-server.config` – SDK configuration to run the Stub Server using "../../lib/keystore/testkeys" as the identity keystore and "../../lib/keystore/server-truststore.jks" as the truststore. "../../lib/keystore/server-truststore.jks" contains the certificates for "../lib/keystore/client1-identity.jks" and "../lib/keystore/client2-identity.jks" so that both client session pools can establish a two-way SSL connection.
3. `lib/keystore/client1-identity.jks` – Identity keystore used by the first session pool ("default").

4. `lib/keystore/client2-identity.jks` – Identity keystore used by the second session pool ("test").
5. `lib/keystore /client-truststore.jks` – Truststore used by clients containing the server certificate from "lib/keystore/testkeys".
6. `lib/keystore/server-truststore.jks` – Truststore used by server containing the certificates from "lib/keystore/client1-identity.jks" and "lib/keystore/client2-identity.jks".

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

1. `cd eppsdk/bundles/namestore`
2. `build.sh –DEPP.ConfigFile=epp-server.config start-server`
3. In a separate Window:
   a. `cd eppsdk/bundles/namestore`
   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 contain its own *EPPSession*.

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 line 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:

```
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. Consolidated Top Level Domain (CTLD) Product (dotCC/dotTV/ dotJOBS/dotCOM/dotNET)

This section is intended to provide users of the Extensible Provisioning Protocol (EPP) Software Development Kit (SDK) with an overview of the Consolidated Top Level Domain Product, hereon referred to as CTLD, additions to the SDK. This document includes the following CTLD information:

1. Definition of the CTLD SDK files (i.e. library, schema)
2. Description of the CTLD 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 CTLD products are thin top-level domains that follow similar business rules with exception to dotJOBS subproduct. Not all of the CTLD products need to be available from the same gateway or services. In the case of SRS, dotCOM and dotNET use consistent interfaces similar to CTLD products, but are hosted by a separate set of gateways. On Namestore dotCC, dotJOBS, and dotTV are all available from the same set of gateways, but different backend servers and databases could host them. The Namestore Extension is required for the gateways to route requests to the appropriate backend server.

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 CTLD 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.

12.1 CTLD Directories

The CTLD product uses the standard EPP Domain and Host objects. In addition, it requires that all commands be passed with an EPP NamestoreExt extension element specifying the particular sub-product that the command should be operated on. The directories that make up CTLD are defined in Table 12 - CTLD Directories.

Table 12 - CTLD Directories

Directory	Description
domain	Standard IETF Domain Mapping
host	Standard IETF Host Mapping
namestoreext	Namestore Extension needed to specify the target sub-product for a command.

Directory	Description
rgp	Standard IETF Redemption Grace Period Extension and RGP poll message. The CTLD products fully support the RGP Restore and Restore Report.
sync	Extension to support the domain sync command defined in the ConsoliDate Mapping
idn	Extension for the International Domain Name Tag required for IDN domain creates.
whois	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.
secdns	Domain Name System Security Extension to provide additional features required for the provisioning of DNS
coa	Client Object Attribute Extension to allow the creation and maintenance of key/value pairs associated with Objects.
premiumdomai n	Extensions to the domain check command, domain check response and domain update command to support premium features.

Table 13 - CTLD Directory Standard Files defines the standard set of files that reside in the CTLD directories defined in Table 12 - CTLD Directories. The CTLD mappings and extensions can be worked on in isolation by using the files in the CTLD directories, but most of the time the Namestore bundle directory should be used, which is the bundle/namestore directory.

Table 13 - CTLD Directory Standard Files

Directory	Description
build.bat	This is a Windows batch file that executes Ant to compile and execute the test programs.
build.sh	This is a Bash shell script that executes Ant to compile and execute the test programs.
build.xml	This is the Ant XML configuration file.
epp.config	This is a sample EPP configuration file. See section 6.1 for information on configuring the SDK.
common-targets.xml	Common Ant targets used bye both the source and binary distributions.
logconfig.xml	Log4J XML configuration file.

12.2 CTLD Packages

The CTLD portion of the Namestore and SRS SDK consists of sub-packages and class additions to existing SDK packages. The following table provides an overview of the primary SDK packages.

Table 14 - CTLD Packages

Package	**Description**
com.verisign.epp.codec.domain	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.
com.verisign.epp.codec.host	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.
com.verisign.epp.codec.namestoreext	NamestoreExt Extension Encoder/Decoder package. All of the detail of encoding and decoding the EPP NamestoreExt messages are in this package. The EPPNamestoreExtExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.
com.verisign.epp.codec.rgpext	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.
com.verisign.epp.codec.rgppoll	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.

Package	**Description**
com.verisign.epp.codec.syncext	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.
com.verisign.epp.codec.idnext	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.
com.verisign.epp.codec.whois	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.
com.verisign.epp.codec.secdnsext	The SecDNS Extension Encoder/Decoder package. All of the detail of encoding and decoding the SecDNS Extension is in this package. The EPPSecDNSExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.
com.verisign.epp.namestore.interfaces	NameStore interface extensions like NSDomain and NSHost. NSDomain provides utility methods and extends com.verisign.epp.interfaces.EPPDomain to include NameStore supported operations including sync, restore request, and restore report.
com.verisign.epp.codec.premiumdomain	The Premium Domain Extension Encoder/Decoder package. All of the detail of encoding and decoding the Premium Domain Extension is in this package.

Package	Description
	The EPPPremiumDomainExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.
com.verisign.epp.codec.coaext	The COA Extension Encoder/Decoder package. All of the detail of encoding and decoding the COA Extension is in this package. The EPPCoaExtFactory must be added to the EPP.CmdRspExtensions configuration parameter using the full package and class name.

12.3 CTLD XML Schema files

The CTLD EPP Mappings and Extensions are defined using XML schema files. These files are located in the ***schemas*** directory of *epp-namestore.jar*. Extract the *epp-namestore.jar* in the binary distribution to view the schema files or look to the location column defined in Table 15 - CTLD 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 15 - CTLD XML Schema Files

File Name	Location	Description
domain-1.0.xsd	domain/schemas	Standard EPP Domain XML Schema.
host-1.0.t.xsd	host/schemas	Standard EPP Contact XML Schema.
namestoreExt-1.0.xsd	namestoreext/schemas	CTLD NamestoreExt XML Schema.
rgp-1.0.xsd	rgp/schemas	Standard EPP Domain Registry Grace Period Mapping XML Schema.
rgp-poll-1.0.xsd	rgp/schemas	EPP Registry Grace Period Poll Mapping XML Schema.
sync-1.0.xsd	sync/schemas	Standard EPP ConsoliDate Mapping XML Schema.
idnLang-1.0.xsd	idn/schemas	IDN Language Tag Extension XML Schema.
whoisInf-1.0.xsd	whois/schemas	Whois Info Extension XML Schema.
secDNS-1.0.xsd	secdns/schemas	Domain Name System Security Extension XML Schema.
premiumdomain-	premiumdomain	Premium Domain Extension XML Schema.

File Name	Location	Description
1.0.xsd	/schemas	
coa-1.0.xsd	coa/schemas	Client Object Attribute Extension XML Schema

12.4 CTLD Client Interfaces

12.4.1 NamestoreExt Support Classes

The CTLD products require that a NamestoreExt element be passed with every action. This is to specify the particular sub-product that the action will operate on. For example, when a host is created, the NamestoreExt element will specify the particular registry at which the host is created. Currently, valid values for the SubProductID are “dotCC”, “dotTV” and “dotJOBS”. The class *com.verisign.epp.namestore.interfaces.NSSubProduct* includes a set of constants of supported sub-products. For example, *NSSubProduct.TV* can be used to specify the sub-product for .tv. The *com.verisign.epp.namestore.interfaces.NSDomain* and *com.verisign.epp.namestore.interfaces.NSHost* classes provide a *setSubProductID(String)* utility method in place of using the NamestoreExt directly.

12.4.1.1 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 subproduct value.

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

12.4.2 Whois Info Support Classes

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.

12.4.3 SecDNS Extension Support Classes

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.v10.EPPSecDNSExtCreate* or *com.verisign.epp.codec.secdnsext.v10.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.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.

12.4.3.1 Sample Code

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

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

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

EPPDomainCreateResp theResponse = theDomain.sendCreate();
```

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

```
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"));
List remDsDataList = new ArrayList();
remDsDataList.add(new
com.verisign.epp.codec.secdnsext.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.codec.secdnsext.v11.EPPSecDNSExtDsData(12345,
                  EPPSecDNSAlgorithm.DSA,
                  EPPSecDNSExtDsData.SHA1_DIGEST_TYPE,
                  "49FD46E6C4B45C55D4AC"));
theDomain.setSecDNSUpdate(addDsDataList, NSDomain.REM_ALL_DS);

EPPDomainCreateResp theResponse = theDomain.sendUpdate();
```

12.4.4 COA Extension Support Classes

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:

```
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:

```
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:

```
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();
```

12.4.5 Premium Domain Extension Support Classes

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.

12.4.6 CTLD Host Interface

The CTLD Host interface, *NSHost*, is located in the *com.verisign.epp.namestore.interfaces* package. This interface extends the *com.verisign.epp.interface.EPPHost* interface and is used to check, create, query, modify, and delete hosts. Convenience methods are provided in *NSHost* to make managing hosts in NameStore 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:

Return Value	Parameters
	`NSHost(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).
`void`	`addHostName(java.lang.String)` This method adds a host name to the object for use with the action methods.
`EPPResponse`	`getResponse()` This method returns the EPP Response for the last executed command on the interface.
`EPPResponse`	**`SendCheck()`** This method sends the host check command to the server.
`EPPResponse`	**`sendCreate()`** This method sends the host create command to the server.
`EPPResponse`	**`sendDelete()`** This method sends the host delete command to the server.

Return Value	Parameters
`EPPResponse`	**`sendInfo()`** This method sends the host info command to the server.
`EPPResponse`	**`sendUpdate()`** This method sends the host update command to the server.
`void`	`addIPV4Address(java.lang.String)` This method adds an Ipv4 address to the host object.
`void`	`removeIPV4Address(java.lang.String)` This method removes an Ipv4 address from the host object.
`void`	`setNewName(java.lang.String)` This method sets a new value of the host name for use with the update method.
`void`	`addExtension(EPPCodecComponent)` This method sets the EPP extension for the host object.
`Vector`	`getExtensions()` This method gets the EPP extensions for the host object.
`void`	`setTransId(java.lang.String)` This methods sets the client transaction identifier.
`void`	`setSubProductID(String)` Sets the NameStore sub-product id associated with the action method. The *NSSubProduct* class includes a set of constant that can be used as the *setSubProductID* argument value.

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 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 action methods, the *NSHost* constructor and methods requiring additional explanation.

12.4.6.1 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.

12.4.6.1.1 Pre-Conditions

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

12.4.6.1.2 Post-Conditions

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

12.4.6.1.3 Exceptions

None

12.4.6.1.4 Sample Code

The following example shows the steps of initializing an *EPPSession*, using the *EPPSession* to initialize the *NSHost* interface, then setting the extension. This example will create a host specifically at the dotCC registry.

```
EPPSession session = new EPPSession();

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

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

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

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

12.4.6.2 sendCheck() method

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

12.4.6.2.1 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,* which currently include CC, COM, EDU, NET, and TV.

12.4.6.2.2 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.

12.4.6.2.3 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, then *getResponse()* will return *null*.

12.4.6.2.4 Sample Code

The following example shows the steps of performing a check on CTLD 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
```

12.4.6.3 sendCreate() method

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

12.4.6.3.1 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,* which currently include CC, COM, EDU, NET, and TV.
- setIPV4Address(String) or setIPV6Address(String) – add an Ipv4 or Ipv6 address to the host object.

12.4.6.3.2 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.

12.4.6.3.3 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, then *getResponse()* will return *null*.

12.4.6.3.4 Sample Code

The following example shows the steps of performing a create of a CTLD host through the use of the *Host* client interface and the *sendCreate()* method:

```
try {

        host.setTransId("ABC-12345-XYZ");
        host.addHostName(“ns.myhost.net”);
        host.setIPV4Address(“123.34.34.2”);
        host.setSubProductID(NSSubProduct.NET);

        response = (EPPHostCreateResp) host.sendCreate();

        System.out.println(“Host created: “ + response.getName());

} // end of try block
catch (EPPCommandException cmdException) {

        // do something to handle the exception
        handleException(cmdException);

} // end of catch block
```

12.4.6.4 sendDelete() method

The *sendDelete()* method sends the EPP delete host command to the Namestore 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.

12.4.6.4.1 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,* which currently include CC, COM, EDU, NET, and TV.

12.4.6.4.2 Post-Conditions

On success, a standard *EPPResponse* is returned.

12.4.6.4.3 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, then *getResponse()* will return *null*.

12.4.6.4.4 Sample Code

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

```
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
```

12.4.6.5 sendInfo() method

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

12.4.6.5.1 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,* which currently include CC, COM, EDU, NET, and TV.

12.4.6.5.2 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.

12.4.6.5.3 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*.

12.4.6.5.4 Sample Code

The following example shows the steps of querying an CTLD host through the use of the *Host* 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.IPV4) {
                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
```

12.4.6.6 sendUpdate() method

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

12.4.6.6.1 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,* which currently include CC, COM, EDU, NET, and TV.
- 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.

12.4.6.6.2 Post-Conditions

On success, a standard *EPPResponse* is returned.

12.4.6.6.3 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, then *getResponse()* will return *null*.

12.4.6.6.4 Sample Code

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

```
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
```

12.4.7 CTLD Host Support Classes

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

Class	Description
EPPHostAddress	This class is used by the *sendInfo()* method for encapsulating Ipv4 and Ipv6 addresses.
EPPHostCheckResult	This class is used by the *sendCheck()* method for returning the allowable flag for multiple hosts being checked.

12.4.7.1 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.

12.4.7.2 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.

12.4.8 CTLD Domain Interface

The CTLD Domain interface, *NSDomain*, is located in the *com.verisign.epp.namestore.interfaces* package. This interface extends the *com.verisign.epp.interfaces.EPPDomain* interface and is used to check, create, query, modify, delete, sync, and restore domains. Convenience methods are provided in *NSDomain* to make managing domains in NameStore 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:

Return Value	Parameters
	`NSDomain(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).
`void`	`addDomainName(java.lang.String)` This method adds a domain name to the object for use with the action methods.
`void`	`addExtension(EPPCodecComponent)` Adds a extension to be sent with the command.
`void`	`addHostName(java.lang.String)` Adds a host name to be associated with the domain.
`void`	`addContact(String, String)` Adds contact for call to sendCreate() or sendUpdate(). CTLD currently includes only thin Registries, so the setting of contacts is not supported.
`EPPResponse`	`getResponse()` This method returns the EPP Response for the last executed command on the interface.
`Date`	`getExpirationDate()` Returns the expiration date of the domain.
`Vector`	`getExtensions()` Gets all set command extensions.

Boolean	`hasExtension(Class)` Does a command extension exist with the specified Class.
EPPResponse	`sendCheck()` This method sends the domain check command to the server.
EPPResponse	`sendCreate()` This method sends the domain create command to the server.
EPPResponse	`sendDelete()` This method sends the domain delete command to the server.
EPPResponse	`sendInfo()` This method sends the domain info command to the server.
EPPResponse	`sendUpdate()` This method sends the domain update command to the server.
EPPResponse	`sendRenew()` This method sends the domain renewal command to the server.
EPPResponse	`sendTransfer()` This method sends the domain transfer command to the server.
EPPResponse	`sendRestoreRequest()` This method sends the Registry Grace Period (RGP) restore request command.
EPPResponse	`sendRestoreReport()` This method sends the Registry Grace Period (RGP) restore report command.
EPPResponse	`sendSync()` This method sends the ConsoliDate sync command.
void	`setTransferOpCode(java.lang.String)` Sets the operation type for future transfer commands. Valid values are "TRANSFER_REQUEST", "TRANSFER_APPROVE" or "TRANSFER_REJECT"
void	`setTransId(java.lang.String)` This methods sets the client transaction identifier.
void	`setDay(int)` Sets the target day for a call to sendSync.
void	`setMonth(int)` Sets the target month using the Calendar month constants (Calendar.JANUARY to Calendar.DECEMBER) for a call to sendSync.
void	`setIDNLangTag(String)` Sets the IDN language tag for a call to sendCreate() of an IDN domain.
void	`setSubProductID(String)` Sets the NameStore sub-product id associated with the action method. The NSSubProduct class includes a set of constant that can be used as the setSubProductID argument value.
void	`setPeriodLength(int)` Sets the registration period for a create, renew, or transfer command. The default value is 1 year.
void	`setPeriodUnit(String)` Sets the unit of the registration period as defined by setPeriodLength(int) according to the EPP specification. CTLD only supports the default value of "y" for years.

void	`setExpirationDate(Date)` Sets the current expiration date for a call to sendRenew().
void	`setUpdateAttrib(…)` Sets attribute to update for a call to sendUpdate().
void	`setAuthString(String)` Sets authorization string for a call to sendCreate() , sendTransfer(), or sendInfo()
void	`setRegistrant(String)` Sets the domain registrant for a call to sendCreate() or sendUpdate().CTLD currently includes only thin Registries, so the setting of registrant is not supported.
void	`setHosts(String)` 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.
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.
void	`setSecDNSCreate(List<EPPSecDNSExtDsData>)` Sets the list of <`EPPSecDNSExtDsData` > instances in order to create delegation signer information.
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.
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.
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.
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.
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.

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 *EPPDomain* setter methods. Each action 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 action methods, the *EPPDomain* constructor and methods requiring additional explanation.

12.4.8.1 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.

12.4.8.1.1 Pre-Conditions

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

12.4.8.1.2 Post-Conditions

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

12.4.8.1.3 Exceptions

None

12.4.8.1.4 Sample Code

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

```
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);
```

12.4.8.2 sendCheck() method

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

12.4.8.2.1 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,* which currently include CC, COM, EDU, NET, and TV.

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

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

12.4.8.2.2 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.

12.4.8.2.3 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, then *getResponse()* will return *null*.

12.4.8.2.4 Sample Code

The following example shows the steps of performing a check on domains through the use of the *Domain* 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");
                }
           }
        }
} // end of try block
catch (EPPCommandException cmdException) {

        // do something to handle the exception
        handleException(cmdException);

} // end of catch block
```

12.4.8.3 sendCreate() method

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

12.4.8.3.1 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,* which currently include CC, COM, EDU, NET, and TV.

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.
- setSubProductID(String) – sub-product associated with operation. Use one of the constants from *com.verisign.epp.namestore.interfaces.NSSubProduct,* which currently include CC, COM, EDU, NET, and TV.
- 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.

12.4.8.3.2 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.

12.4.8.3.3 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, then *getResponse()* will return *null*.

12.4.8.3.4 Sample Code

The following example shows the steps of performing a create of an CTLD domain through the use of the *Domain* client interface and the *sendCreate()* method:

```
try {

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

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

         // -- Add Delegation Signer Information
        // instantiate a secDNS:keyData object
        EPPSecDNSExtKeyData keyData = new EPPSecDNSExtKeyData();
        keyData.setFlags( EPPSecDNSExtKeyData.FLAGS_ZONE_KEY_SEP );
        keyData.setProtocol( EPPSecDNSExtKeyData.DEFAULT_PROTOCOL );
        keyData.setAlg( EPPSecDNSAlgorithm.RSASHA1 );
        keyData.setPubKey( "AQPmsXk3Q1ngNSzsH1lrX63mRIhtwkkK+5Zj"
                       + "vxykBCV1NYne83+8RXkBElGb/YJ1n4TacMUs"
                       + "poZap7caJj7MdOaADKmzB2ci0vwpubNyW0t2"
                       + "AnaQqpy1ce+07Y8RkbTC6xCeEw1UQZ73PzIO"
                       + "OvJDdjwPxWaO9F7zSxnGpGt0WtuItQ==" );

        // instantiate another secDNS:keyData object
        EPPSecDNSExtKeyData keyData2 = new EPPSecDNSExtKeyData(
                       EPPSecDNSExtKeyData.FLAGS_ZONE_KEY_SEP,
                       EPPSecDNSExtKeyData.DEFAULT_PROTOCOL,
                       EPPSecDNSAlgorithm.RSASHA1,
                       "AQOxXpFbRp7+zPBoTt6zL7Af0aEKzpS4JbVB"
                       + "5ofk5E5HpXuUmU+Hnt9hm2kMph6LZdEEL142"
                       + "nq0HrgiETFCsN/YM4Zn+meRkELLpCG93Cu/H"
                       + "hwvxfaZenUAAA6Vb9FwXQ1EMYRW05K/gh2Ge"
                       + "w5Sk/0o6Ev7DKG2YiDJYA17QsaZtFw==" );

        // instantiate a secDNS:dsData object
        EPPSecDNSExtDsData dsData = new EPPSecDNSExtDsData();
        dsData.setKeyTag( 34095 );
        dsData.setAlg( EPPSecDNSAlgorithm.RSASHA1 );
        dsData.setDigestType( EPPSecDNSExtDsData.SHA1_DIGEST_TYPE );
        dsData.setDigest( "6BD4FFFF11566D6E6A5BA44ED0018797564AA289" );
        dsData.setMaxSigLife( 604800 );
        dsData.setKeyData( keyData );

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

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

         // Call only if server supports creating delegation signer
         // information
        theDomain.setSecDNSCreate( dsDataRecords );

        response = (EPPDomainCreateResp) domain.sendCreate();

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

        System.out.println("domainCreate: CreationDate = " +
                response.getCreationDate());

        System.out.println("domainCreate: ExpirationDate = " +
                response.getExpirationDate());


} // end of try block
catch (EPPCommandException cmdException) {

        // do something to handle the exception
        handleException(cmdException);

} // end of catch block
```

12.4.8.4 sendDelete() method

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

12.4.8.4.1 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,* which currently include CC, COM, EDU, NET, and TV.

12.4.8.4.2 Post-Conditions

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

12.4.8.4.3 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, then *getResponse()* will return *null*.

12.4.8.4.4 Sample Code

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

```
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
```

12.4.8.5 sendInfo() method

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

12.4.8.5.1 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,* which currently include CC, COM, EDU, NET, and TV.

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

- setAuthString(String) – sets the authorization string for getting full domain information if not sponsoring Registrar.
- setHosts(String) – Sets the desired level of host information. Use one of the *HOSTS_* constant values:

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

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

12.4.8.5.2 Post-Conditions

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

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

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

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

12.4.8.5.3 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, then *getResponse()* will return *null*.

12.4.8.5.4 Sample Code

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

```
try {

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

        response = (EPPDomainInfoResp) domain.sendInfo();

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

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

       //-- Output the secDNS:infData extension if returned
       if (response.hasExtension(EPPSecDNSExtInfData.class)) {
                EPPSecDNSExtInfData infData =(EPPSecDNSExtInfData)
                 response.getExtension(EPPSecDNSExtInfData.class);

                Collection dsDataVec = infData.getDsData();
                EPPSecDNSExtDsData dsData = null;
                if (dsDataVec == null) {
                        System.out.println("domainInfo: secDNS:infData
                                dsDataVec = " + dsDataVec);
                }
                else {
                        int i = 0;
                        Iterator iter = dsDataVec.iterator();
                        while (iter.hasNext()) {
                                dsData = (EPPSecDNSExtDsData)iter.next();

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

```
                                       = "  + dsData.getDigest());
                               System.out.println("domainInfo:
                                       secDNS:infData/dsData[" + i +
                                       "]/maxSigLife = "  +
                                       dsData.getMaxSigLife());

                               EPPSecDNSExtKeyData keyData =
                                       dsData.getKeyData();
                               if (keyData == null) {
                                       System.out.println("domainInfo:
                                               secDNS:infData/dsData[" + i +
                                               "]/keyData = "  + keyData);
                               }
                               else {
                                       System.out.println("domainInfo:
                                               secDNS:infData/dsData[" + i +
                                               "]/keyData/flags = "
                                               + keyData.getFlags());
                                       System.out.println("domainInfo:
                                               secDNS:infData/dsData[" + i +
                                               "]/keyData/protocol = "
                                               + keyData.getProtocol());
                                       System.out.println("domainInfo:
                                               secDNS:infData/dsData[" + i +
                                               "]/keyData/alg = "
                                               + keyData.getAlg());
                                       System.out.println("domainInfo:
                                               secDNS:infData/dsData[" + i +
                                               "]/keyData/pubKey = "
                                               + keyData.getPubKey());
                               }

                               i++;

                       } // end while
               }

       }
} // end of try block
catch (EPPCommandException cmdException) {

       // do something to handle the exception
       handleException(cmdException);

} // end of catch block
```

12.4.8.6 sendUpdate() method

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

12.4.8.6.1 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,* which currently include JOBS, CC, COM, NET, and TV.

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.)

12.4.8.6.2 Post-Conditions

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

12.4.8.6.3 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, then *getResponse()* will return *null*.

12.4.8.6.4 Sample Code

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

```
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
```

12.4.8.7 sendRenew() method

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

12.4.8.7.1 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,* which currently include CC, COM, EDU, NET, and TV.
- 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

12.4.8.7.2 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.

12.4.8.7.3 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, then *getResponse()* will return *null*.

12.4.8.7.4 Sample Code

The following example shows the steps of performing an renewal of an domain through the use of the *Domain* client interface and the *sendRenew()* method:

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

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

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


        response = (EPPDomainRenewResp) domain.sendRenew();

        //-- Output response attributes using accessors
        System.out.println("domainRenew: name = " +
                response.getName());

        System.out.println("domainRenew: ExpirationDate = " +
                response.getExpirationDate());

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

12.4.8.8 sendTransfer() method

The sendTransfer*()* method sends the EPP transfer domain command to the Namestore 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.

12.4.8.8.1 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,* which currently include CC, COM, EDU, NET, and TV.

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

12.4.8.8.2 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 attributres are set.

12.4.8.8.3 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, then *getResponse()* will return *null*.

12.4.8.8.4 Sample Code

The following example shows the steps of performing a transfer request of a domain through the use of the *Domain* 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 *Domain* 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 *Domain* 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
```

12.4.8.9 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.

12.4.8.9.1 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,* which currently include CC, COM, EDU, NET, and TV.

12.4.8.9.2 Post-Conditions

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

12.4.8.9.3 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, then *getResponse()* will return *null*.

12.4.8.9.4 Sample Code

The following example shows the steps of performing an update of an domain through the use of the *Domain* 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
```

12.4.8.10 sendRestoreReport() method

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

12.4.8.10.1 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,* which currently include CC, COM, EDU, NET, and TV.
- setReport(EPPRgpExtReport) – Sets the details of the restore report.

12.4.8.10.2 Post-Conditions

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

12.4.8.10.3 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, then *getResponse()* will return *null*.

12.4.8.10.4 Sample Code

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

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

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


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

```
              + " Agreement"));


         theReport.setOther("other stuff");


         // Execute restore report

         domain.setReport(theReport);

         theResponse = domain.sendRestoreReport();

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

12.4.8.11 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.

12.4.8.11.1 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,* which currently include CC, COM, EDU, NET, and TV.
- setMonth(int) – Sets the month using a *Calendar* constant of the target expiration date.
- setDay(int) – Sets the day of the target expiration date.

12.4.8.11.2 Post-Conditions

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

12.4.8.11.3 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, then *getResponse()* will return *null*.

12.4.8.11.4 Sample Code

The following example shows the steps of performing an update of an domain through the use of the *Domain* 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
```

12.5 Namestore Poll Messages

Namestore will send the following kinds of poll messages:

- com.verisign.epp.codec.domain.EPPDomainTransferResp – Used for transfer notifications for the transfer actions: request, cancel, approve, reject, auto and approve.
- com.verisign.epp.codec.lowbalancepoll.EPPLowBalancePollResponse – Used for account low balance notifications. Low balance poll notification is not technically CTLD specific.
- com.verisign.epp.codec.rgppoll.EPPRgpPollResponse – Used for Registry Grace Period (RGP) pending restore notifications.
- com.verisign.epp.codec.domain.EPPDomainPendActionMsg – Used for domain pending action notifications.

The test *com.verisign.epp.namestore.interface.NSPollTst* provides a sample of processing the Namestore 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");
        } // Tranfer auto approved?
        else if
(theStatus.equals(EPPDomainTransferResp.TRANSFER_SERVER_APPROVED)) {
                System.out.println("testPoll: Got transfer auto approve
notification");
        } // Tranfer 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. Name Suggestions Product

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

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

13.1 Name Suggestions Tests

The Namestore source distribution contains one test program for Name Suggestions 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:

Directory	Description
EPPSuggestionTst.java	This is a sample program demonstrating the use of the EPPSuggestion class.

13.2 Name Suggestions Packages

The Name Suggestions portion of the Namestore and SRS SDK consists of sub-packages and class additions to existing SDK packages. **Figure 10 - Name Suggestions Packages** provides an overview of the primary SDK packages.

Package	Description
com.verisign.epp.codec.suggestion	Name Suggestion Encoder/Decoder package. All of the detail of encoding and decoding the Name Suggestions EPP messages are in this package. The EPPSuggestionMapFactory must be added to the EPP.MapFactories configuration parameter using the full

	package and class name.
com.verisign.epp.framework	Addition of Name Suggestions EPP Server Framework classes used by the Stub Server.
com.verisign.epp.serverstub	Addition of the SuggestionHandler class used to implement the EPP Name Suggestions Stub Server behavior. This class must be added to the EPP.ServerEventHandlers configuration parameter using the full package and class names.
com.verisign.epp.interfaces	This package contains the Name Suggestions client interface classes. These classes provide the primary interfaces that map to the commands and objects of the Name Suggestions EPP Mapping.

Figure 10 - Name Suggestions Packages

13.3 Name Suggestions XML Schema files

The Name Suggestions EPP Mapping is defined using XML schema files. These files are located in the *epp-namestore.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:

File Name	Location	Description
suggestion-1.1.xsd	schemas	Name Suggestions XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.

Figure 11 - Name Suggestions Schema Files

13.4 Name Suggestions Client Interfaces

The Name Suggestions portion of the Namestore and SRS SDK contains client interface classes for the Name Suggestions EPP Mapping. The interfaces provide mechanisms for querying suggestions. The following sections describe the client interface classes, supporting classes and their respective purposes.

13.4.1 Name Suggestions Interface

The Name Suggestions interface, *EPPSuggestion*, is located in the *com.verisign.epp.interfaces* package. This interface is used to query suggestions in the Name Suggestions system.

The *EPPSuggestion* interface has the following relevant methods:

Return Value	Parameters
	EPPSuggestion`(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).
`EPPSuggestionInfoResp`	sendInfo`()` This method is used to retrieve suggestions.

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.

13.4.1.1 *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.

13.4.1.1.1 Pre-Conditions

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

13.4.1.1.2 Post-Conditions

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

13.4.1.1.3 Exceptions

None

13.4.1.1.4 Sample Code

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

```
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);
```

13.4.1.2 sendInfo() method

The sendI*nfo()* method sends the Name Suggestions EPP info command to the Namestore server. It has the following signature:

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

13.4.1.2.1 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

13.4.1.2.2 Post-Conditions

On success, an *EPPSuggestionInfoResp* is returned.

13.4.1.2.3 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*.

13.4.1.2.4 Sample Code

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

```
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();
}
```

14. Contact

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 SDK 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.

14.1 Contact 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.

Package	**Description**
com.verisign.epp.codec.contact	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.
com.verisign.epp.interfaces	Addition of the *EPPContact* Client interface class, which is the primary class used by a Contact SDK client.
com.verisign.epp.serverstub	Addition of the *com.verisign.epp.serverstub.ContactHandler* class used to implement the EPP Contact Stub Server behavior. This class must be added to the *EPP.ServerEventHandlers* configuration parameter.

14.2 Contact XML Schema Files

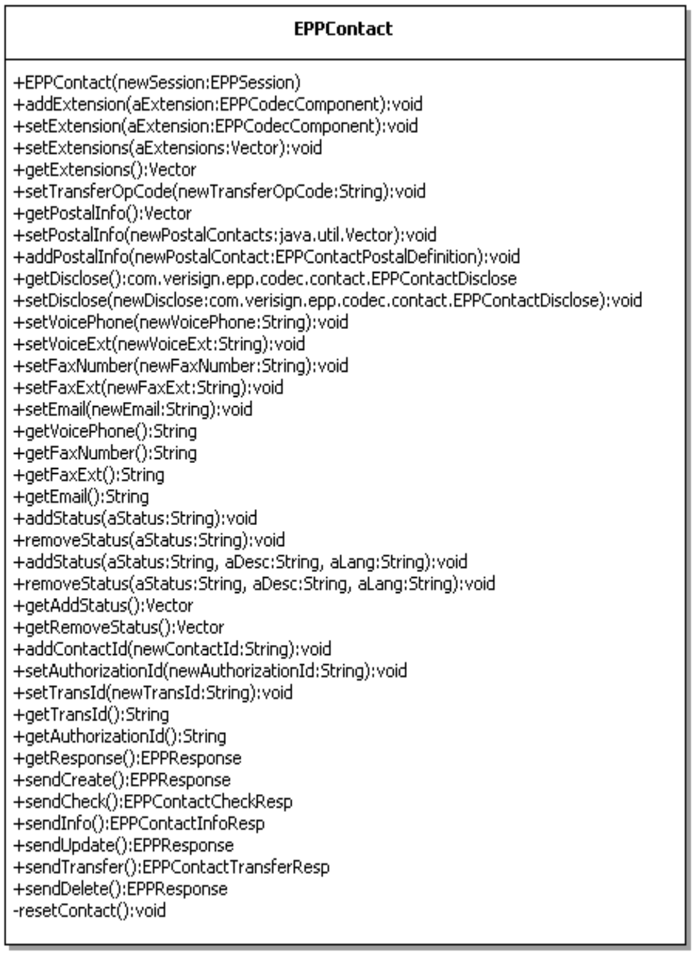
The Contact EPP Mapping is defined using XML schema files. These files are located in the *epp-contact.jar* in the ***schemas*** directory. You must un-jar the jar file in order to explicitly view them.

File Name	Location	Description
contact-1.0.xsd	schemas	Contact XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.

14.3 Contact Client Interface

14.3.1 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

+EPPContact(newSession:EPPSession)
+addExtension(aExtension:EPPCodecComponent):void
+setExtension(aExtension:EPPCodecComponent):void
+setExtensions(aExtensions:Vector):void
+getExtensions():Vector
+setTransferOpCode(newTransferOpCode:String):void
+getPostalInfo():Vector
+setPostalInfo(newPostalContacts:java.util.Vector):void
+addPostalInfo(newPostalContact:EPPContactPostalDefinition):void
+getDisclose():com.verisign.epp.codec.contact.EPPContactDisclose
+setDisclose(newDisclose:com.verisign.epp.codec.contact.EPPContactDisclose):void
+setVoicePhone(newVoicePhone:String):void
+setVoiceExt(newVoiceExt:String):void
+setFaxNumber(newFaxNumber:String):void
+setFaxExt(newFaxExt:String):void
+setEmail(newEmail:String):void
+getVoicePhone():String
+getFaxNumber():String
+getFaxExt():String
+getEmail():String
+addStatus(aStatus:String):void
+removeStatus(aStatus:String):void
+addStatus(aStatus:String, aDesc:String, aLang:String):void
+removeStatus(aStatus:String, aDesc:String, aLang:String):void
+getAddStatus():Vector
+getRemoveStatus():Vector
+addContactId(newContactId:String):void
+setAuthorizationId(newAuthorizationId:String):void
+setTransId(newTransId:String):void
+getTransId():String
+getAuthorizationId():String
+getResponse():EPPResponse
+sendCreate():EPPResponse
+sendCheck():EPPContactCheckResp
+sendInfo():EPPContactInfoResp
+sendUpdate():EPPResponse
+sendTransfer():EPPContactTransferResp
+sendDelete():EPPResponse
-resetContact():void
```

Figure 11 - EPPContact Class Diagram

14.3.2 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.

14.3.2.1 Pre-Conditions

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

14.3.2.2 Post-Conditions

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

14.3.2.3 Exceptions

None

14.3.2.4 Sample Code

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

```
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

14.3.3 sendCreate() Method

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

14.3.3.1 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.

14.3.3.2 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.

14.3.3.3 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*.

14.3.3.4 Sample Code

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

```
try {

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

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

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

        EPPContactPostalDefinition name = new EPPContactPostalDefinition(

        EPPContactPostalDefinition.ATTR_TYPE_LOC);

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

        contact.addPostalInfo(name);

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

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

        EPPContactPostalDefinition Intname = new EPPContactPostalDefinition(

        EPPContactPostalDefinition.ATTR_TYPE_INT);

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

        contact.addPostalInfo(Intname);

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

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

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

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

        EPPContactDiscloseAddress.ATTR_TYPE_LOC));
        addresses.addElement(new EPPContactDiscloseAddress(

        EPPContactDiscloseAddress.ATTR_TYPE_INT));

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

        contact.setDisclose(disclose);

        response = (EPPContactCreateResp) contact.sendCreate();

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

}
```

Figure 13 - sendCreate() Sample Code

14.3.4 sendCheck() Method

Checks the availability of one or more Contact objects.

17.3.4.1 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.

14.3.4.1 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()*

14.3.4.2 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*.

14.3.4.3 Sample Code

Figure 12 - 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.

```
try {
   contact.setTransId("ABC-12345");
   contact.addContactId("sh8013");

   EPPContactCheckResp response = contact.sendCheck();

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

        if (currResult.isAvailable()) {
                System.out.println("contactCheck: Contact "
                        + currResult.getId() + " is available");
        }
        else {
                System.out.println("contactCheck: Contact "
                        + currResult.getId() + " is unavailable");
```

```
         }
      }
}
catch (EPPCommandException 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 12 - sendCheck() Sample Code

14.3.5 sendInfo() Method

Retrieves the Contact information.

14.3.5.1 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.

14.3.5.2 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

14.3.5.3 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*.

14.3.5.4 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.

```
try {
        EPPContactInfoResp contactResponse;

        System.out.println("JobsContact: Contact Info");

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

        contactResponse = contact.sendInfo();

        System.out.println("contactInfo: id = " + response.getId());

        Vector postalContacts = null;

        if (response.getPostalInfo().size() > 0) {
                postalContacts = response.getPostalInfo();

        for (int j = 0; j < postalContacts.size(); j++) {
```

```
        // Name
        System.out.println("contactInfo:\t\tname = "
           + ((EPPContactPostalDefinition) postalContacts
                .elementAt(j)).getName());

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


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

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

        }

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


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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
}
```

Figure 14 - sendInfo() Sample Code

14.3.6 sendUpdate() Method

Updates the attributes of a Contact.

14.3.6.1 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.

14.3.6.2 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()*

14.3.6.3 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*.

14.3.6.4 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("");
        disclose.setFax("");
        disclose.setEmail("");

        contact.setDisclose(disclose);

        response = contact.sendUpdate();

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

Figure 15 - sendUpdate() Sample Code

14.3.7 sendDelete() Method

Deletes a Contact.

14.3.7.1 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.

14.3.7.2 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()*

14.3.7.3 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*.

14.3.7.4 Sample Code

Figure 13 - sendDelete() Sample Code shows the steps of creating EPPResponse. If an error occurs, *EPPCommandException* is thrown, and if the error was the result of an EPP Server response, than the response can be retrieved from the exception.

```
try {
        contact.setTransId("ABC-12345");
        contact.addContactId("sh8013");

        EPPResponse response = contact.sendDelete();

        System.out.println("Contact Delete success response: " + response);
}
catch (EPPCommandException ex) {
        EPPResponse response = (EPPResponse) ex.getResponse();

        if (response != null)
                System.err.println("Contact Delete response error: " + ex +
                        ", response = " + response);
        else
                System.err.println("Contact Delete exception: " + ex);
}
```

Figure 13 - sendDelete() Sample Code

14.3.8 sendTransfer() Method

Transfer a Contact.

14.3.8.1 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.

14.3.8.2 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()*

14.3.8.3 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*.

14.3.8.4 Sample Code

Figure 14 - 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.

```
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 14 - sendTransfer() Sample Code

15. DotJobs Contact Extension

15.1 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.

Package	Description
com.verisign.epp.codec.jobscontact	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.
com.verisign.epp.interfaces	Addition of the *EPPContact* Client interface class, which is the primary class used by a DotJobs Contact SDK client.
com.verisign.epp.serverstub	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.

15.2 DotJobs Contact XML Schema Files

The DotJobs Contact EPP Extension Mapping is defined using XML schema files. These files are located in the *epp-jobsContact.jar* in the ***schemas*** directory. You must un-jar the jar file in order to explicitly view them.

File Name	Location	Description
jobsContact-1.0.xsd	schemas	DotJobs Contact XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.

15.3 DotJobs Contact Client Interface

15.3.1 Overview

DotJobs Contact is an extension to the Contact Object and hence EPPContact will be the client interface for DotJobs extension operations.

15.3.2 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.getId());
        System.out.println("Contact Created Date : " +
                response.getCreationDate());

}
```

15.3.3 sendCheck() Method

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

15.3.4 sendInfo() Method

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

```
try {
        EPPContactInfoResp contactResponse;

        System.out.println("JobsContact: Contact Info");

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

        contactResponse = contact.sendInfo();

        System.out.println("contactInfo: id = " + response.getId());

        Vector postalContacts = null;

        if (response.getPostalInfo().size() > 0) {
                postalContacts = response.getPostalInfo();

        for (int j = 0; j < postalContacts.size(); j++) {

                // Name
                System.out.println("contactInfo:\t\tname = "
                   + ((EPPContactPostalDefinition) postalContacts
                        .elementAt(j)).getName());

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


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

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

                }

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


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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        // -- 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");
        }

}
```

15.3.5 sendUpdate() Method

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

```
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);
}
```

15.3.6 sendDelete() Method

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

15.3.7 sendTransfer() Method

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

16. WhoWas Product

This section is intended to provide users of the Extensible Provisioning Protocol (EPP) Software Development Kit (SDK) with an overview of the WhoWas Product 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.

16.4 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:

Directory	**Description**
EPPWhoWasTst.java	This is a sample program demonstrating the use of the EPPWhoWas class.

16.5 WhoWas Packages

The WhoWas portion of the Namestore and SRS SDK consists of sub-packages and class additions to existing SDK packages. Figure 15 - WhoWas PackagesFigure 10 - Name Suggestions Packagesprovides an overview of the primary SDK packages.

Package	**Description**
com.verisign.epp.codec.whowas	WhoWas Encoder/Decoder package. All of the detail of encoding and decoding the WhoWas EPP messages are in this package. The *EPPWhoWasMapFactory* must be added to the EPP.MapFactories configuration parameter using the full

	package and class name.
com.verisign.epp.framework	Addition of WhoWas EPP Server Framework classes used by the Stub Server.
com.verisign.epp.serverstub	Addition of the *WhoWasHandler* class used to implement the EPP WhoWas Stub Server behavior. This class must be added to the EPP.ServerEventHandlers configuration parameter using the full package and class names.
com.verisign.epp.interfaces	This package contains the WhoWas client interface classes. These classes provide the primary interfaces that map to the commands and objects of the WhoWas EPP Mapping.

Figure 15 - WhoWas Packages

16.6 WhoWas XML Schema files

The WhoWas EPP Mapping is defined using XML schema files. These files are located in the *epp-namestore.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:

File Name	Location	Description
whowas-1.0.xsd	schemas	WhoWas XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.

Figure 16 - WhoWas Schema Files

16.7 WhoWas Client Interfaces

The WhoWas portion of the Namestore and SRS 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.

16.7.1 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:

Return Value	Parameters
	`EPPWhoWas(EPPSession aSession)` This is the constructor method and it requires an EPP session object to be passed that has been authenticated (e.g. logged in).
`EPPWhoWasInfoResp`	`sendInfo()` This method is used to retrieve whowas history records.

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.

16.7.1.1 *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.

16.7.1.1.1 Pre-Conditions

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

16.7.1.1.2 Post-Conditions

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

16.7.1.1.3 Exceptions

None

16.7.1.1.4 Sample Code

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

```
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);
```

16.7.1.2 sendInfo() method

The sendI*nfo()* method sends the WhoWas EPP info command to the Namestore server. It has the following signature:

```
public EPPWhoWasInfoResp sendInfo() throws EPPCommandException
```

16.7.1.2.1 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.

16.7.1.2.2 Post-Conditions

On success, an *EPPWhoWasInfoResp* is returned.

16.7.1.2.3 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*.

16.7.1.2.4 Sample Code

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

```
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 = aEPPWhoWasHistory.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();
}
```

17. 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.

17.8 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:

Directory	Description
EPPBalanceTst.java	This is a sample program demonstrating the use of the EPPBalance class.

17.9 Balance Packages

The Balance portion of the Namestore and SRS SDK consists of sub-packages and class additions to existing SDK packages. Figure 17 - Balance Packages provides an overview of the primary SDK packages.

Package	Description
com.verisign.epp.codec.balance	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.
com.verisign.epp.framework	Addition of Balance EPP Server Framework classes used by the Stub Server.
com.verisign.epp.serverstub	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.
com.verisign.epp.interfaces	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 17 - Balance Packages

17.10 Balance XML Schema files

The Balance EPP Mapping is defined using XML schema files. These files are located in the *epp-namestore.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:

File Name	Location	Description
balance-1.0.xsd	schemas	Balance XML Schema. This file must reside in the current directory of the client application and the EPP Stub Server.

Figure 18 - Balance Schema Files

17.11 Balance Client Interfaces

The Balance portion of the Namestore and SRS 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.

17.11.1 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:

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

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.

17.11.1.1 *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.

17.11.1.1.1 Pre-Conditions

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

17.11.1.1.2 Post-Conditions

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

17.11.1.1.3 Exceptions

None

17.11.1.1.4 Sample Code

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

```
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);
```

17.11.1.2 sendInfo() method

The sendI*nfo()* method sends the Balance EPP info command to the Namestore server. It has the following signature:

```
public EPPBalanceInfoResp sendInfo() throws EPPCommandException
```

17.11.1.2.1 Pre-Conditions

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

17.11.1.2.2 Post-Conditions

On success, an *EPPBalanceInfoResp* is returned.

17.11.1.2.3 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*.

17.11.1.2.4 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());
        e.printStackTrace();
}
```