NEUSTAR SHARED REGISTRY SYSTEM DEVELOPERS GUIDE

REGISTRARS TOOLKIT (RTK) 0.4.1 FOR NEW GENERIC TLDS

October 8, 2013

neustar®
Real Intelligence. Better Decisions.™

Table of Contents

Table of Figures and Tables

1 Introduction

1.1 Purpose

This document explains how to use Neustar's Shared Registry System (SRS) Registrars Toolkit to interface with new generic TLD domain applications. .EXAMPLE is used as the TLD to illustrate examples. This document supplements the HTML documentation provided with the SRS Toolkit. This document has supplements for First Come, First Served (FCFS) Sunrise, Multi-Application (MULTI) Sunrise, Landrush with Claims, and GA with Claims phases, and International Domain Names (IDN).

The SRS Registrars Toolkit (RTK) 0.4.1 Developer's Guide and HTML documentation can be found online at https://registry.neustar.biz/regadm/index.jsp and at http://epp-rtk.sourceforge.net/.

1.2 Intended Audience & Reading Suggestions

This document is targeted toward those desiring an understanding of the SRS Registrars Toolkit (RTK), especially those involved in the design, development, integration, and testing of the RTK. The reader should have a strong understanding of either Java or C++ programming languages and should be familiar with domain name registration in a shared registry environment.

Knowledge of and expertise with the following languages and protocols is recommended in order to make full use of the EPP Java API:

- Java and/or C++ Programming Languages
- Extensible Provisioning Protocol (EPP)
- Extensible Markup Language (XML): required to work with the internals of the RTK.

1.3 Reference Documents

The following documents are the key references for this document.

- License Agreement
- IETF RFCs
- DNS and BIND

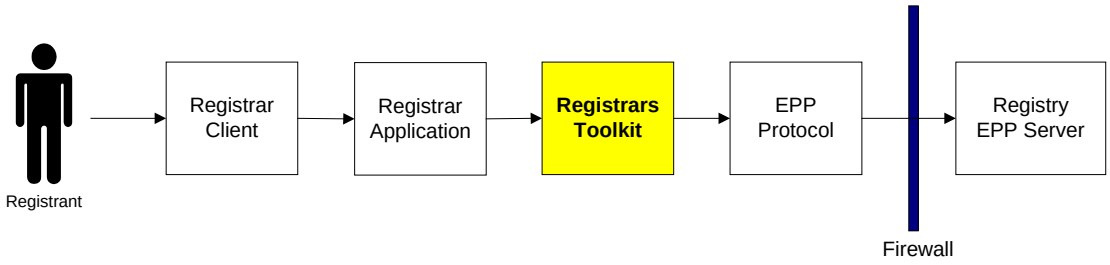
2 Getting Started

This section provides a high-level view of the developers' toolkit and is the best place to gain the overall picture of the Shared Registry System.

2.1 Introduction to a Shared Registry System (SRS)

A Shared Registry System (SRS) enables multiple registrars to register domain information on behalf of a registrant with the Neustar Registry. Registrars connect to the Neustar Registry using the Extensible Provisioning Protocol (EPP). EPP is the mechanism used by Registrars to manage domain information. The Neustar EPP servers interface with a series of application servers to register a domain name under a top-level domain (TLD) that is administered by Neustar.

The Neustar Registrars Toolkit enables registrars to use a simple high-level API to interface with the Neustar Registry. This toolkit sits in the middle of the registration process and takes input from the registrar's client registrant application software and the EPP servers residing at the registry. The toolkit manages the transport layer, the session layer and the protocol interface through a simple API.



2.2 Where to Start

2.2.1 Choose a Toolkit (Java or C++)

The first step to getting starting is choosing the programming language of choice for the toolkit. Currently Java and C++ are supported. The Java and C++ APIs provide equivalent interfaces to all supported EPP objects and extensions. Each Java class and method has an equivalent C++ counterpart. With the following exception, each C++ method operates in the same way as its corresponding Java method.

- In the Java API, the method **EppResult::getValue()** returns a vector of String and/or EppValueResult objects.
- In the C++ API, the method **EppResult::getValue()** returns a vector of EppValueResult objects; a String object is encapsulated in an EppValueResult object without a reason text and the language attribute.

2.2.2 Download Required Third Party Software

Based on which toolkit you choose, you must download, install and configure several third party applications required by the toolkit.

2.2.3 JAVA Toolkit Required Software

Download and install the following required third party modules (refer to the URLs stated in section 3.2):

- JDK 1.6
- Apache Xalan-Java 2.7.1
  - xalan.jar
  - xercesImpl.jar

2.2.4 C++ Toolkit Required Software

Download and install the required third party modules (refer to the URLs stated in section 3.3)

- XERCES-C++ Version 1.5
- OPENSSL Version 1.0.1e
- GNU Make Version 3.81
- DOC++ Version 3.4.8

2.3 Installing the Toolkit

This toolkit includes two sets of the RTK API, one in C++ and the other in Java. The RTK API provides registrar developers with a well-defined, clearly documented set of calls to generate XML conformant with EPP 1.0 as well as the underlying facilities for session and connection management. The toolkit can be found at:
http://sourceforge.net/projects/epp-ver-10/?source=directory

Each set of APIs comes with HTML documentation (Javadoc or DOC++), test clients, and the actual API classes themselves. The following subsections provide brief instructions for detailed installation.

Neustar's EPP toolkit also includes sample test clients and dummy EPP servers in Java and C++ for testing purpose. The source and binary files are located in the following directories:

```
java/test
c++/test
```

For instructions on building and running the test clients and dummy servers, please refer to the files included in the toolkit.

```
java/test/RunTestTcp
c++/test/RunTestTcp
```

2.3.1 Java Installation Instructions

2.3.1.1 Common Java Installation Instructions

Use the provided top-level makefile. This will build all the API classes and generate a JAR file along with the Javadoc documentation for the API classes. You can use the following make targets:

- To build the EPP APIs JAR file only, run the make with the jars target: make jars
- To build the APIs JAR file with optimization, run make with the JAVAOPT option: make JAVAOPT=-O
- From the top of the EPP Java tree, type the following command to build the documentation: make docs apidocs deployment as a cold stand-by.

Once you have installed the required third party modules, modify your CLASSPATH environment variable to include the xerces jar file. Note that your CLASSPATH environment variable must not end with a '\', this would cause the build to fail. Your CLASSPATH environment variable should include the following (replace d:/lib with the location of your jar files):

2.3.1.2 CLASSPATH='d:/lib/xerces.jar;'Special Java Instructions For Unix Platform

For the Unix platform, make sure that your CLASSPATH environment variable includes the xerces jar file as follows (replace /lib with the location of your jar files):

CLASSPATH=/lib/xerces.jar:

2.3.2 C++ Installation Instructions – needs validation

2.3.2.1 Common C++ Toolkit Installation Instructions

Depending on your build platform, select one correct Makefile for that platform and copy it to Makefile.plat. These Makefiles are found in the src directory, which includes the platform dependent information. Currently, there are 4 platforms supported by EPP C++ API 0.2.5:

- Makefile.linux - for G++: 2.96
- Makefile.solaris - for Solaris 2.7/2.8 with G++ 2.95.2 or SC 4.2
- Makefile.hpux - for aCC: HP ANSI C++ B3910B A.03.13
- Makefile.win32 - for MS VC++: 6.0 (CL 12.00.8168)

For example if you are building for a Linux platform, you need to perform the following:

```
cp Makefile.linux Makefile.plat
```

Next you need to edit Makefile.plat to specify the paths and libraries for XERCES-C++ and OPENSSL as follows:

XERCES_INC_DIR - include directory for XERCES-C++

XERCES_LIB_DIR - the absolute path of the XERCES-C++ library

OPENSSL_INC_DIR - include directory for OPENSSL

OPENSSL_LIB_DIR - the absolute path of the OPENSSL libraries

You can also edit the Makefile.plat for any other customized needs that you may have.

The top level Makefile will build everything, including the documentation. If you only want to compile the source code and make the library files, run the following command from the top of the EPP C++ tree:

```
make compile_src lib
```

Make options:

To compile the class files with optimization, add CXXFLAGS=-O in the make command line.

For example: make CXXFLAGS=-O

To build the EPP C++ Documentation only, run the following: make docs from the top of the EPP C++ tree.

2.3.2.2 Special C++ Instructions For Windows Platform

In order to use the provided Makefile to build the API jar file, you first need a copy of Cygnus's Cygwin, MinGW or some other utility that allows you to run Make on windows. Refer to http://sourceware.cygnus.com/cygwin or http://www.mingw.org/ or http://gnuwin32.sourceforge.net/packages/make.htm for more details on how to download the correct version for your Windows platform.

Finally, from the top of the EPP C++ tree, run make.

2.3.2.3 Special C++ Instructions For Unix Platform

Once you have installed all the required third party libraries, and copied the appropriate Makefile for your platform, follow the General build instructions to create the API library and documentation.

2.4 Setup for Authentication

Table 1 Java Properties

Property	Description
ssl.client.authentication	The property for enabling client side authentication, default true
ssl.keymanager.algorithm	The property for key manager algorithm, default "SUNX509"
ssl.keymanager.provider	The property for key manager provider, default null
ssl.keystore.type	The property for key store type, default "JKS"
ssl.keystore.provider	The property for key store provider, default null
ssl.keystore.format	The property for key store format, default "file" *(only "file" is supported currently.*
ssl.keystore.name	The property for key store name. If the key store format is file", the value of this property is the file name of the key store file.
ssl.keystore.storepass	The property for key store password, default null
ssl.keystore.keypass	The property for key password for recovering keys, default as the value of the key store password
ssl.trustmanager.algorithm	The property for trust manager algorithm, default "SUNX509"
ssl.trustmanager.provider	The property for trust manager provider, default null
ssl.truststore.type	The property for trust store type, default "JKS"
ssl.truststore.provider	The property for trust store provider, default null
ssl.truststore.format	The property for trust store format, default "file" *(only "file" is supported currently*.
ssl.truststore.name	The property for trust store name. If the key store format is "file", the value of this property is the file name of the trust store file.
ssl.truststore.storepass	The property for trust store password, default null

2.4.1 C++ Initialization

EppSession created must be initialized prior to setting up connection. EppSession initializes the run-time parameters related to an EPP Session. The configuration object can be a Properties object if the implementation chooses to use Java Property file, or a DOM_Element object, if the configuration file is in XML format, or an const char that represents the name of the configuration file in any format.

*Note: currently the object is type of (char * argv[3]), containing* the names of three files in SSL_FILETYPE_PEM format

argv[0] - client private key file

argv[1] - client public key file

argv[2] - server public key file

Verify the Installation

The installation can be verified by running sample server and client for a successful result. The instructions and sample applications are available in the test directory of the toolkit in Java and C++.

Please refer to the text file *RunTestTcp* for instructions on how to configure and run the test Server and Client applications. This file can be found in the *test* directory for C++ and Java respectively.

2.5 Write a Client Application

Step 1: Setup the session and connection to the server. Setup a connection (EppSession) with the Registry server, using the server name and the port number. Session Management Operations provide APIs for establishing a connection. The established session needs to be initialized with the security information if chosen to use TLS. The Toolkit supports direct TCP with or without TLS

Step 2: Establish a communication channel between your client and the server. Once a connection is established, establish communication channels between the client and the server for data exchange (EppChannel). A single session supports multiple channels over which requests are sent and responses are received.

Step 3: Establish an EPP Session. Establish an EPP Session over an established data exchange channel. A data exchange channel supports one EPP Session at a time. An EPP session is established by logging in to the registry server using valid credentials using EppCommandLogin.

Step 4: Submit requests to the server. Send the supported Commands over the established channel using send() method. The toolkit implements a synchronous messaging between client and the server returning a response for every request Error occurred at transport layer results in a null response and respective error messages can be obtained by accessing exceptions on EppSession object.

Step 5: Access and Process the Response. Every response received includes a result element and the response data for a successful response. The returned result is either success or error with result codes and respective messages. The response data includes requested data from the server.

Step 6: Terminate the EPP session. Terminating an EPP session will end the EPP session and the exchange of data over the channel is terminated until establishment of another EPP session over that channel, using valid credentials.

Step 7: Terminate the Communication Channel. Terminating the communication channel will close the channel and any established EPP session over the channel.

Step 8: Terminate the Connection. Terminating the connection will end the connection at the transport level which closes all the open channels and open EPP sessions.

3 Object Attributes and Description

3.1 Domain Name Attributes and Description

The following table lists the Domain Object attributes and the associated values. The Registrar provides some of these attributes; the Registry generates the others.

Table 2 Domain Object Attributes and Values

Attribute Name	Description	Data Type	Source
Domain Name	The name of the domain to be registered. The syntax rules for domain names follow the conventions as set forth in RFC952 and RFC1123. A summary of the rules is as follows: The name is to be unique to the Registry. The name may contain only letters, numbers, or the special character hyphen. However, a hyphen may not begin or end a name, nor have two or more consecutive hyphens. No other special characters are permitted. (Until the rules for internationalization of domain names have been approved and put into use.) The name label must have a length greater than 2 characters and less than or equal to 63 RFC 1035 valid characters in the second level domain (SLD). Once created, the name and its ROID cannot be modified.	String(64) + TLD/SLD suffix	Registrar
ROID	The globally unique identifier of the domain. This is populated by the registry.	String(89)	Registry
Status	One of the various statuses or states a domain object can be. Set to “OK” once registration is successful. The registrar sets some of the statuses; others are set by the Registry. A domain object can have more that one status associated with it. For a complete list of domain statuses and their descriptions, see Section 13.2.1.	String(50)	Registrar or Registry
Sponsoring Registrar	ID of the Registrar the domain is currently registered to in the Registry system.	String(16)	Registry
Registrant	The ID of the Domain Registrant Object. The domain Registrant Contact Object must already exist in the	String(16)	Registrar

Attribute Name	Description	Data Type	Source
	Registry.		
Expiration Date	The date the domain name's registration period ends. This date is calculated based on the domain registration date, registration period, and any renewal periods that might have been added.	Date in GMT	Registry
Last Transfer Date	The date the domain name was last transferred successfully.	Date in GMT	Registry
Authorization Information (Password)	Authorization information to be associated with the domain object.	String(16)	Registrar
Variable Set of the Registrar (NAME = Value)	This is a list of Registrar provided name/value pair. This is a free format list of attributes and may vary from one Registrar to another. A maximum of 10 name/value pairs can be set for each domain object.	Name-String (128) Value String (1000)	Registrar
Technical Contact	The ID of the technical contact for the domain. The contact must have already been entered in the Registry prior to being used. A minimum of one technical contact must be provided with an unlimited maximum. The Registry recommends providing two technical contacts per domain name.	String(16)	Registrar
Admin Contact	The ID of the administrative contact for the domain. The contact must already exist in the Registry prior to being used. A minimum of one admin contact must be provided with an unlimited maximum. The Registry recommendation is to provide two admin contacts per domain.	String(16)	Registrar
Billing Contact	The ID of the billing contact for the domain. The contact must already be entered in the Registry prior to use. A minimum of one billing contact must be provided with an unlimited maximum. The Registry recommends providing two billing contacts per domain name.	String(16)	Registrar
Registration Date	The date the domain name was entered in the Registry system.	Date in GMT	Registry
Registration Period	The registration period for the domain. Values of 1 to 5, inclusive may be provided. However, regardless of the value provided an awarded domain will be	Integer	Registrar

Attribute Name	Description	Data Type	Source
	registered for 1 year.		
Created Date	Set to the date and time the domain was created in the registry system.	Date in GMT	Registry
Created By	Set to the ID of the registrar creating domain name.	String(16)	Registry
Updated Date	Date and time the domain name was modified.	Date in GMT	Registry
Updated By	Entity that modified the domain name.	String(16)	Registry
Last Transfer Date	The date the domain was most recently transferred.	Date in GMT	Registry

3.2 Contact Attributes and Description

The following table lists the Contact Object attributes and the associated values. The Registrar provides some of these attributes; the Registry generates the others.

Attribute Name	Description	Data Type	Source
ID	A registrar desired identifier that uniquely identifies the object in the Registry.	String(16)	Registrar
ROID	The globally unique identifier of the contact. This is populated by the registry.	String(89)	Registry
Status	One of the various statuses or states a contact be assigned. This is set to “OK” once registration is successful. For a complete list of contact statuses and their descriptions, please see Section 10.0.	String(50)	Registrar or Registry
Name	The name (includes first and last name) of the contact	String(255)	Registrar
Organization	The organization the contact belongs to.	String(255)	Registrar
Address1	The first line of the contact’s address	String(255)	Registrar
Address2	The second line of the contact’s address	String(255)	Registrar
Address3	The third line of the contact’s address	String(255)	Registrar
City	The city of the contact’s address	String(255)	Registrar
State/Province	The state or province of the contact’s address	String(255)	Registrar
Postal Code	The postal code of the contact’s address	String(16)	Registrar
Country Code	The two letter code assigned to the country of the contact’s address	String(2)	Registrar
I15d Name	Internationalized contact name (includes first and last names)	String(255)	Registrar

Attribute Name	Description	Data Type	Source
I15d Organization	Internationalized contact organization.	String(255)	Registrar
I15d Address 1	Internationalized contact address 1	String(255)	Registrar
I15d Address 2	Internationalized contact address 2	String(255)	Registrar
I15d Address 3	Internationalized contact address 3	String(255)	Registrar
I15d City	Internationalized contact city	String(255)	Registrar
I15d State/Province	Internationalized contact state or province	String(255)	Registrar
I15d Postal Code	Internationalized postal code	String(16)	Registrar
I15d Country	Internationalized contact country	String(2)	Registrar
Telephone Number	The telephone number of the contact	String(17)	Registrar
Telephone Extension	The telephone extension of the contact	String(10)	Registrar
Fax Number	The fax number of the contact	String(17)	Registrar
Fax Extension	The fax extension of the contact	String(10)	Registrar
Email Address	The email address of the contact	String(80)	Registrar
Sponsoring Registrar	ID of the Registrar the contact is currently registered to in the Registry system.	String(16)	Registry
Registrar name/value pairs (Name=value string)	This is a list of registrar provided name/value pair. This is a free format list of attributes and may vary from one registrar to another. A maximum of 10 name/value pairs can be set for each contact object.	Name – String(128) Value-String(1000)	Registrar
Authorization Information (type &	Authorization information to be associated with the contact object.	Type – String Roid _	Registrar

Attribute Name	Description	Data Type	Source
roid)		String(89)	
Last Transfer Date	Date and timestamp the contact was last transferred	Date in GMT	Registry
Created Date	This is set to the date and time the contact was created in the registry system.	Date in GMT	Registry
Created By	ID of the registrar creating the contact.	String(16)	Registry
Updated Date	Date and timestamp when the contact was last updated.	Date in GMT	Registry
Updated By	ID of the entity updating the contact whether Registrar or Registry.	String(16)	Registry
Last Transfer Date	The date the domain was most recently transferred.	Date in GMT	Registry

3.3 Host Object Attributes and Description

The following table lists the Host Object attributes and the associated values. Note: by policy name servers must NOT be included with the sunrise application.

Attribute Name	Description	Data Type	Source
Host Name	The name of a name server.	String(255)	Registrar
ROID	The globally unique identifier of the host. This is populated by the registry.	String(89)	Registry
Status	One of the various statuses or states a name server can be assigned. This is set to “OK” once registration is successful. For a complete list and description, please see Section 10.0.	String(50)	Registrar or Registry
IP Address (Type & Address)	The type of IP Address of the name server, either IPv4 or IPv6. A .CO host must have only one IP address associated with it.	Type – String(2) Address-String(45)	Registrar
Sponsoring Registrar	ID of the Registrar the Host currently registered to in the Registry system.	String(16)	Registry
Registrar name/value pairs (Name=value string)	This is a list of registrar provided name/value pair. This is a free format list of attributes and may vary from one registrar to another. A maximum of 10 name/value pairs can be set for each contact object.	Name-String(128) Value-String(1000)	Registrar
Created Date	Date and timestamp when the Host was created in the registry system.	Date in GMT	Registry
Created By	ID of the Registrar creating the Host.	String(16)	Registry
Updated Date	Date and timestamp when the Host record was updated.	Date in GMT	Registry
Updated By	Entity that updated the Host’s record.	String(16)	Registry
Last Transfer Date	The date the domain was most recently transferred.	Date in GMT	Registry

4 Toolkit API Classes

This section describes the classes provided with the Toolkit API. Classes are categorized by the following functional categories (click to follow link).

Specific details on the fields and methods associated with each class are provided in the HTML Documentation that accompanies the Toolkit. Section 5 of this Guide provides additional information and examples on implementing commonly-used classes.

4.1 General Classes

4.1.1 EppAuthoInfo

This class is used to capture authorization information.

4.1.2 EppCreds

This class allows for the creation of credentials given the registrar's id and password.

4.1.3 EppCredsOptions

Creates an EppCredsOptions object, given the version string and language id.

4.1.4 EppUnspec

This class creates an Unspec object with a Name/value pair string.

4.1.5 EppStatus

This class is used to implement the status code and a descriptive message. The default for status is "ok" and for language is "en."

4.1.6 EppServiceMenu

This class implements the svcMenuType entity.

4.1.7 EppTransactionID

This class is used to create an EppTransactionID object given only the service transaction id or the client transaction id and service transaction id.

4.1.8 EppPeriod

This EppPeriod class implements EPP Domain periodType entity.

4.1.9 EppPollable

The EppPollable interface is designed for pollable EppEntity, such as EppCommandTransfer objects, that can be included as the data element in the response of an EPP Poll command.

4.1.10 EppParser

This class is designed to parse and validate EPP XML messages against EPP XML Schema. Apache Xerces 1.4.2 or later for Java is required.

4.1.11 EppUtil

This class provides a set of “general-utility” methods used by various components.

4.2 Session Management Classes

4.2.1 EppSession

Abstract class used to establish a connection between the client and the server.

4.2.2 EppSessionTcp

An EppSession that uses TCP as the transport for connection establishment.

4.2.3 EppChannel

Once a connection is created with the server. Use this class to create a dedicated communication channel. The created channel needs to be authenticated to establish an EppSession by using start method with a valid EppCommandLogin. This class is also used to send and retrieve requests and responses to and from the server. Sending an EppCommandLogin request will log the user out of the system and will terminate the channel.

4.2.4 EppChannelTcp

This class is designed to handle a session connection to the EPP server via TCP/TLS.

4.2.5 EppGreeting

When a connection is established, this class implements the greetings returned from the server. Greetings include the parameters agreed upon in the session such as the language of the session, the services provided, etc.

4.2.6 EppDataCollectionPolicy

This class implements the EppDataCollectionPolicy object, which describes the server's privacy policy for data collection and management.

4.2.7 EppDataCollectionStatement

This class implements the EppDataCollectionStatement object for the EppDataCollectionPolicy, which is used to provide information on data collection purposes, data recipients, and data retention practices.

4.2.8 EppMessageUtil

This class contains methods for handling message exchanges between an EPP Server and an EPP Client.

4.3 Command Classes

4.3.1 EppCommand

Abstract class implements commands.

4.3.2 EppCommandLogin

This class is used to login to the server and authenticate the connection.

4.3.3 EppCommandLogout

This command class is used to logout of the server, however, it does not close the connection between the client and the server.

4.3.4 EppCommandPoll

This command class is used to poll the server for any pending **EppPollable** messages.

4.4 Object Management Base Classes

4.4.1 EppObject

This EppObject class is the base class for all objects registered in the registry via the EPP Protocol.

4.4.2 EppCommandCheck

This class is the base class for all classes that implement methods for registry <check> commands.

4.4.3 EppComandInfo

This class is the base class for all classes that implement methods for registry <info> commands.

4.4.4 EppCommandRenew

This class is the base class for all classes that implement methods for registry <renew> commands.

4.4.5 EppCommandTransfer

This class is the base class for all classes that implement methods for registry <transfer> commands.

4.4.6 EppCommandUpdate

This class is the base class for all classes that implement methods for registry <update> commands.

4.5 Domain Management Classes

4.5.1 EppDomain

This class represents the domain entity.

4.5.2 EppCommandCreate

This class is used for the registration of a new domain.

4.5.3 EppCommandCheckDomain

This class checks for the availability of a domain.

4.5.4 EppCommandDeleteDomain

This class deletes the registration of a domain name in the system.

4.5.5 EppCommandUpdateDomain

This class is used to update the attributes of an existing domain.

4.5.6 EppCommandInfoDomain

This class is used to query the attributes and state of an existing domain.

4.5.7 EppCommandRenewDomain

This class is used to renew a domain registration by adding a specified number of years to the registration period of an existing domain name.

4.5.8 EppCommandTransferDomain

This class controls the movement of domain names from one registrar to another whether it is a request to transfer a domain name or to approve/disapprove a transfer.

4.6 Host Management Classes

4.6.1 EppHost

This class represents a host entity.

4.6.2 EppIpAddress

This class creates the IP address object with an address type of “v4” (default) or “v6”.

4.6.3 EPPHostAttribute

This class is used to support the “host-attribute” model used by some registries.

4.6.4 EppCommandCreate

This class is used for the creation of a new host.

4.6.5 EppCommandCheckHost

This class checks for the availability of a host, (i.e. whether a host is already registered or not).

4.6.6 EppCommandDeleteHost

This class deletes an already existing host from the registry.

4.6.7 EppCommandUpdateHost

This class is used to update the attributes of an existing host.

4.6.8 EppCommandInfoHost

This class is used to query the attributes and state of an existing host.

4.7 Contact Management Classes

4.7.1 EppContact

This class represents the contact entity in the registry.

4.7.2 EppContactType

This class creates the contact type object with the id of a contact object and a type string. This class is used to define the type of contact when a contact entity is associated with a domain. Contact types can be Admin, Billing, Technical, or Registrant.

4.7.3 EppContactData

This class is used to populate the social data of a contact entity (the contact address information).

4.7.4 EppContactDisclose

This class is used to store and retrieve contact data disclosure information. It contains Boolean flags that are used to control disclosure of certain contact data elements. Note that this information is NOT required by the Registry.

4.7.5 EppCommandCreate

This class is used to create a new contact in the registry.

4.7.6 EppCommandCheckContact

This class is used to check whether a contact is already registered in the registry.

4.7.7 EppCommandDeleteContact

This class is used to delete an already existing contact from the registry.

4.7.8 EppCommandUpdateContact

This class is used to modify the attributes of an already existing contact.

4.7.9 EppCommandInfoContact

This class is used to query the attributes and state of an already existing contact entity.

4.7.10 EppCommandTransferContact

This class controls the movement of contacts from one registrar to another whether it is a request to transfer a contact or approve/disapprove a transfer.

4.7.11 EppAddress

This class is used to create an address.

4.7.12 EppE164

This class implements the EPP e164Type, which is the ITU E.164 format for telephone numbers, plus extension.

4.8 Response Processing Classes

4.8.1 EppError

This class provides a layer of abstraction that helps shield client applications from future changes to error message numbering. For example, to check whether a command has

completed successfully, refer to EppError.CODE_NO_ERROR rather than to error code 1000. In this way, should error messages be re-numbered in the future, applications that implement this class will not be impacted.

4.8.2 EppResponse

This class represents the response that is returned from the server when a command is executed.

4.8.3 EppResponseData

This class contains the data that is returned from some of the commands, for example the EppCommandInfoDomain will return the domain information in the EppResponseData.

4.8.4 EppResponseDataCheck

This class and its subclasses implement methods for response data associated with the <check> command.

- **EppResponseDataCheckContact** implements the data returned from an EppCommandCheckContact.
- **EppResponseDataCheckDomain** implements the data returned from an EppCommand CheckDomain.
- **EppResponseDataCheckHost** implements the data returned from an EppCommandCheckHost.

4.8.5 EppResponseDataCreate

This class and its subclasses implement the EppResponseDataCreate object.

- **EppResponseDataCreateContact** implements the data returned from EppCommandCreate. When creating a new contact, the data returned is the ID of the newly created contact.
- **EppResponseDataCreateDomain** implements the data returned from EppCommandCreate. When creating a new domain the data returned is the expiration date of the created domain.
- **EppResponseDataCreateHost** implements the data returned from EppCommandCreate. When creating a new host, the data returned is the name of the newly created host.

4.8.6 EppResponseDataInfo

This class implements the data returned from the server after successful execution of an EppCommandInfo.

4.8.7 EppResponseDataPending

This class and its subordinate classes are not currently implemented.

4.8.8 EppResponseDataRenew

This class implements methods for the renewal of registry objects.

4.8.9 EppResponseDataRenewDomain

This class implements methods for renewal of domain objects.

4.8.10 EppResponseDataTransfer

This class implements the data returned from the server for an EppCommandTransfer.

4.8.11 EppResponseDataTransferContact

This class is used to retrieve information from an EppCommandTransferContact. The information returned includes the status of the transfer, the ID of the requesting client, the ID of the transferred contact object, etc.

4.8.12 EppResponseDataTransferDomain

This class is used to retrieve information from an EppCommandTransferDomain. The information returned includes the status of the transfer, the ID of the requesting client, the new Expiration Date of the transferred domain, etc.

4.8.13 EppResponseDataPoll

This class is used to retrieve pending EppPollable message from an EppCommandPoll.

4.8.14 EppResult

This class contains the error code (also called a result code) and message returned from the server when a command is executed. Note the following differences between the Java and C++ APIs:

- In the Java API, the method **EppResult::getValue()** returns a vector of String and/or EppValueResult objects.
- In the C++ API, the method **EppResult::getValue()** returns a vector of EppValueResult objects; a String object is encapsulated in an EppValueResult object without a reason text and the language attribute.

4.8.15 EppResultMessage

This class contains the message ID, language type, and the message text returned from the server for an EppResult.

4.8.16 EppValueReason

This class is used to store diagnostic information associated with an EppResult object.

5 Operations

The primary operations supported by the toolkit are classified into following sets:

- Session Management Operations
- Data Exchange Operations
- Object Management Operations
- Response Processing Operations
- Error Processing
- Logging Operations

This section describes these operations in detail with an example. The examples in this section are in Java. Corresponding C++ code can be found in the sample C++ files provided along with the toolkit.

5.1 Session Management Operations

5.1.1 Establishing a TCP connection

A connection needs to be established with the server using *connect()* method in EppSessionTCP class prior to accessing services on the server.

An EppSessionTCP can be created with or without using TLS profile.

An EppSessionTCP with TLS profile after creation needs to be initialized with properties file or public/private keys.

Server on successful connection establishment returns an EppGreeting element with server name, date and menu of supported services.

Required Attributes

- Server name String
- Port number Integer

Data Returned

- on Success EppGreeting
- on Failure null

Exceptions

On null response, check exceptions and error messages using EppSession getException() and getMessage() methods.

Result Codes

- None

Usage

```
EppSession session = null;
// Instantiate an appropriate been session according to the
// value of the first argument which defines the underlying
// transport protocol (tcp+tls, tcp).
if (protocol.equals("tcp"))
{
    // create an EppSessionTcp without using TLS profile
    session = new EppSessionTcp(false);
}
else if (protocol.equals("tcp-tls"))
{
    // create an EppSessionTcp using TLS profile
    session = new EppSessionTcp();
    session.init("testkeys.client.prop");
}
else
{
    // error out, unsupported transport
}
// connect to the server
EppGreeting greeting = session.connect(host, port);
if (greeting == null)
{
// failed to get greeting message
System.out.println("Cannot connect to " + host + ":" +             port);
System.out.println("Error Message: " + session.getMessage());
    // error out ...
}
System.out.println("Successfully connected to " + host + ":" + port);
```

5.1.2 Establishing a data communication channel

A data communication channel needs to be established over an established connection for data exchange between Registry client and the server. A *getchannel()* method in EppSession class will create a channel for request response exchange.

Required Attributes

None

Data Returned

- on Success EppChannel
- on Failure null

Exceptions

- On null response, check exceptions and error messages using EppSession getException() and getMessage() methods.

Result Codes

- None

Usage

```
// creating the channel to send message to the server
EppChannel channel = session.getChannel();
// if we cannot create a channel to communicate with report the error and fail
if (channel == null)
{
System.out.println("Cannot establish a channel : "
+ session.getMessage());
// error processing ...
}
System.out.println("Cannot establish a channel : "
```

5.1.3 Establishing an EPP session

An Epp session is created on starting a channel with a valid EppCommandLogin, with valid server supported service menu and valid credentials.

Required Attributes

None

Data Returned

- EppResponse

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Success
- Authentication error – invalid credentials (userid, password, language or version)
- Unimplemented object services – invalid services in Login command
- Command use error – using Login command on an existing session
- 2502 Session limit exceeded, server closing connection

Usage

```
EppCommandLogin login = new EppCommandLogin(greeting.getServiceMenu());
// set the login's credential information
login.setClientId(userid);
login.setPassword(password);
// if new password is not null, change it
if (newPassword != null)
{
    login.setNewPassword(newPassword);
}
// set transaction id, if any
login.setClientTransactionId("MY-LOGIN-TRANSACTION-ID");
// send the actual login command and get the server's response
EppResponse res = channel.start(login);
// check for response received from the server
if (res == null)
{
    System.out.println("failed to login : " + session.getMessage());
    // error Processing ...
}
else if ( ! res.success() )
{
    System.out.println("failed to login : " + res);
    // error processing ...
}
// login command has been executed successfully
```

5.1.4 Terminating an EPP session

An Epp session is terminated on transmission of a EppCommandLogout on an established EppChannel

Required Attributes

- None

Data Returned

- EppResponse

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Success
- 2002 Command use error – Trying to Logout without logging in

Usage

```
EppCommandLogout logout = new EppCommandLogout();
// set transaction id, if any
logout.setClientTransactionId("MY-LOGOUT-TRANSACTION-ID");
// send the logout command to the server
EppResponse res = channel.send(logout);
// optionally, check for responses received ...
```

5.1.5 Terminating the Channel

The terminate() method in EPPChannel terminates the data communication channel.

Required Attributes

- None

Data Returned

- EppResponse

Exceptions

- On null response, check exceptions and error messages using EppSession getException() and getMessage() methods.

Result Codes

- 1000 Success
- 2002 Command use error

Usage

```
// terminate the open channel
channel.terminate();
```

5.1.6 Terminating the Connection

The close() method in the class EppSession terminates the connection.

Required Attributes

- None

Data Returned

- None

Exceptions

- None

Result Codes

- None

Usage

```
// close the session -- this closes the actual connection to server
session.close();
```

5.2 Data Exchange Operations

5.2.1 Request/Response exchange

Client uses send() method on a established EPP session channel to send a request to the server and receives a synchronous response

Required Attributes

- None

Data Returned

- EppResponse

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- None

Usage

```
EppResponse res = channel.send(EppCommandLogout);
```

5.2.2 Hello Operation

Client can ping the server by calling hello() method on established EppChannel. Server responds with a EppGreeting .

Required Attributes

- None

Data Returned

- EppResponse

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- None

Usage

```
EppGreeting greeting = channel.hello()
```

5.2.3 Change Password operation

Client can perform a change password along with EppCommandLogin command while setting up a EPP session along with a new password.

Required Attributes

- None

Data Returned

- EppResponse

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Success
- Authentication error – invalid credentials (userid, password , language or version)
- 2251 Invalid new password

Usage

```
EppCommandLogin login = new EppCommandLogin(greeting.getServiceMenu());
// set the login's credential information
login.setClientId(userid);
login.setPassword(password);
// if new password is not null, change it
if (newPassword != null)
{
    login.setNewPassword(newPassword);
}
// set transaction id, if any
login.setClientTransactionId("MY-LOGIN-TRANSACTION-ID");
// send the actual login command and get the server's response
EppResponse res = channel.start(login);
// check for response received from the server
if (res == null)
{
    System.out.println("failed to login : " + session.getMessage());
    // error Processing ...
}
else if ( ! res.success() )
{
    System.out.println("failed to login : " + res);
    // error processing ...
}
// login command has been executed successfully
```

5.2.4 Poll Operation

Client can poll the server to obtain any pending messages. EppCommandPoll command with “req” operation will return the first message in the server queue for the client. Client has to send an EppCommandPoll command with “ack” operation for the received message to remove the message in the server queue.

Required Attributes

- None

Data Returned

- EppResponseDataPoll object is returned on request for pending messages. The Registry will return transfer notifications for domains and contacts to the losing registrar using EppCommandTransferDomain, an EppPollable object. The Registry will also return status transition notifications when a domain object in Sunrise or Landrush changes status to pendingAllocation, allocated, or rejected.

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 2001 Command syntax error

- 2002 Command use error
- 2003 Required Parameter missing
- 2004 Parameter value range error
- 2005 Parameter value syntax error
- Command failed

Usage

```
EppCommandPoll poll = new EppCommandPoll();
// set transaction id, if needed
poll.setClientTransactionId("MY-POLL-TRANSACTION-ID");
// send the poll command and get the server's response
EppResponse res = channel.send(poll);
// check the response
if (res == null || ! res.success() )
{
    // error processing
}
// get the number of messages queued */
if (res.getMessageQueued() == 0)
{
    // no pending messages
}
// get the result message id and the response data
String msgid = res.getMessageId();
EppResponseDataTransferDomain res_data;
res_data = (EppResponseDataTransferDomain) res.getResponseData();
// processing the response data ...
// send an ack back to the received message
EppCommandPoll ack = new EppCommandPoll(EppCommandPoll.OPTYPE_ACK, msgid);
// set transaction id, if needed
ack.setClientTransactionId("MY-ACK-TRANSACTION-ID");
res = channel.send(ack);
// check the response for the poll ack command
if (res == null || ! res.success() )
{
    // error processing,  or, try sending acknowledgement again
}
// check for number of messages pending to do a poll again
if( res.getMessageQueued() > 0 )
{
    // do the poll again
}
// no pending messages ...
```

5.3 Object Management Operations

Registry Toolkit allows the user to perform the following operations on domain, host and contact objects supported.

Operations	Domain	Contact	Host
Create	X	X	X
Modify	X	X	X
Query	X	X	X
Delete	X	X	X
Check	X	X	X
Renew	X		
Transfer	X	X	

The following three sections describe these operations in detail.

5.4 Domain Management Operations

5.4.1 Create Domain Name [EppCommandCreate]

This operation enters the Domain Name into the system.

Required Attributes

- Domain Name
- Admin Contact ID
- Technical Contact ID
- Billing Contact ID
- Registrant ID
- Name Servers

Optional Attributes

- Registration Period
- Registrar name/value pairs

Implicit Attributes

- Status
- Expiration Date
- Registration Date
- Sponsoring Registrar
- Created By
- Created Date
- Updated By
- Updated Date

Data Returned

- Domain Name
- Expiration Date

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2004 Parameter value range error
- 2005 Parameter value syntax error
- 2104 Billing failure
- 2302 Object exists
- 2306 Parameter value policy error
- 2400 Command failed

Usage

```
EppDomain domain = new EppDomain("example1.EXAMPLE");
// each contact is set by giving the contact id of an existing
// contact object as well as the contact type for this domain,
// more than one contact of any given type is possible.
domain.addContact("JL34" , EppDomain.CONTACT_TYPE_ADMIN);
domain.addContact("BN156", EppDomain.CONTACT_TYPE_TECH);
domain.addContact("JL34" , EppDomain.CONTACT_TYPE_BILLING);
domain.addContact("LS2"  , EppDomain.CONTACT_TYPE_BILLING);
// name servers must already exist in the registry, meaning it is
// impossible to add a subordinate name server in a create
// domain command (i.e. ns1.example.EXAMPLE) since the name server could
// not have already been created since the domain never existed;
// this must be done in a separate create nameserver and domain
// update command
domain.addNameServer("ns1.myisp.EXAMPLE");
domain.addNameServer("ns2.myisp.EXAMPLE");
// set the registration period to one year
domain.setPeriod(new EppPeriod(1, EppPeriod.UNIT_YEAR));
// set the registrant of the domain to an existing contact
domain.setRegistrant("TK450");
// set authorization information for the domain that will be used
// for domain transfer request operations.
domain.setAuthInfo(new EppAuthInfo(EppAuthInfo.TYPE_PW, "SecretPW"));
// generate an actual create domain command using the domain
// object we just generated.
EppCommandCreate cmd = new EppCommandCreate(domain);
// set transaction id, if needed
cmd.setClientTransactionId("MY-DOMAIN-CREATE-TRANSACTION-ID");
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if( res == null || ! res.success() )
{
    // error processing ...
}
// get the response data for the domain
EppResponseDataCreateDomain res_data;
res_data = (EppResponseDataCreateDomain) res.getResponseData();
if (res_data == null)
{
    // this should not happen, but need error processing any way
}
 – move this to the sunrise guide// display response data
System.out.println("Domain Name   : " + res_data.getName());
System.out.println("Creation Date  : " + res_data.getDateCreated().getTime());
System.out.println("Expiration Date: " + res_data.getDateExpired().getTime());
```

5.4.2 Modify Domain Name [EppCommandUpdateDomain]

This command modifies all or a subset of the attributes of a domain. The domain must be in a status to allow for this operation to execute. The following set of attributes can be modified through this operation:

- Admin Contact ID
- Billing Contact ID
- Technical Contact ID
- Registrant ID

- Status
- Domain Name's Name Servers
- Registrar name/value pairs
- Required Attributes
- Domain name

Data Returned

- None

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2004 Parameter value range error
- 2005 Parameter value syntax error
- 2303 Object does not exist
- 2304 Object status prohibits operation
- 2306 Parameter value policy error
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a command to update the domain name
EppCommandUpdateDomain cmd = new EppCommandUpdateDomain("example1.EXAMPLE");
// set up transaction id, if needed
cmd.setClientTransactionId("MY-DOMAIN-UPDATE-TRANSACTION-ID");
// if authInfo needs to be changed, do it here
EppAuthInfo authinfo = new EppAuthInfo(EppAuthInfo.TYPE_PW, "SecretPW4");
cmd.setNewAuthInfo(authinfo);
// set a new registrant for the domain name; this overwrites the old
// value since there is only a single domain registrant at a time
cmd.setNewRegistrant("JD334");
// remove one of the nameservers we've already set for the domain
cmd.removeNameServer("ns2.myisp.EXAMPLE");
// add a new existing nameserver we created for the domain
cmd.addNameServer("ns1.example1.EXAMPLE");
// remove one of the existing billing contacts and add a new one;
// also add a new admin contact (this will NOT replace the existing one)
cmd.removeContact(new EppContactType("JL34",  EppDomain.CONTACT_TYPE_BILLING));
cmd.addContact   (new EppContactType("CU340", EppDomain.CONTACT_TYPE_BILLING));
cmd.addContact   (new EppContactType("TR191", EppDomain.CONTACT_TYPE_ADMIN));
// add/remove status values if needed
cmd.addStatus(EppDomain.STATUS_CLIENT_TRANSFER_PROHIBITED);
cmd.removeStatus(EppDomain.STATUS_CLIENT_HOLD);
 – move to MULTI Sunrise
// send the command to the server and receive a response object.
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
else
{
    // domain name has been updated successfully
}
```

5.4.3 Query Domain Name [EppCommandInfoDomain]

Given a domain name, the InfoDomain command returns the information about the domain. The InfoDomain command returns different sets of data depending upon the ID of the requesting registrar. If the requesting registrar is the sponsoring registrar, then the server returns the complete set of data about the domain name. If the requesting registrar is not the current sponsoring registrar, the data returned is the same as a Whois query on the domain.

Required Attributes

- Domain Name

Data Returned

- Domain Name
- Sponsoring Registrar
- Registrant Contact ID
- Admin Contact’s ID
- Billing Contact’s ID
- Technical Contact’s ID

- Domain Name's Nameservers
- Expiration Date
- List of Registrar name/value pairs
- Updated Date
- (Sponsoring Registrar Only:)
- List of Domain Statuses
- Registration Date
- Registration Period
- Authorization Information
- Last Transfer Date
- List of Domain Name's Child Hosts
- Created By
- Created Date

Updated By

- ROID

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2005 Parameter value syntax error
- 2303 Object does not exist
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a command and set the domain name to query
EppCommandInfoDomain cmd = new EppCommandInfoDomain("example1.COM.EXAMPLE");
// set up transaction id, if needed
cmd.setClientTransactionId("MY-DOMAIN-QUERY-TRANSACTION-ID");
 – move to Sunrise MULTI// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
// the response data should contain a fully populated
// object as part of the response
EppResponseDataInfo data;
data = (EppResponseDataInfo) res.getResponseData();
if( data == null )
{
    // this should not happen, error processing ...
}
EppDomain obj = (EppDomain) data.getObject();
// use the EppDomain access methods to obtain domain name information ...
```

5.4.4 Delete Domain Name [EppCommandDeleteDomain]

Given a particular domain name, this operation deletes the domain from the Registry.

Required Attributes

- Domain Name

Data Returned

- None

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2005 Parameter value syntax error
- 2303 Object does not exist
- 2304 Object status prohibits operation
- 2305 Object association prohibits operation
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a command to delete the domain name
EppCommandDeleteDomain cmd = new EppCommandDeleteDomain("example1.EXAMPLE");
// set up transaction id, if needed
cmd.setClientTransactionId("MY-DOMAIN-DELETE-TRANSACTION-ID");
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
else
{
    // domain name has been deleted successfully
}
```

5.4.5 Check Domain Name [EppCommandCheckDomain]

Given a domain name, this operation checks whether the domain is already registered in the registry or not. The CheckDomainName command can be used to check for the existence of up to 1000 domain names per command.

Required Attributes

- Domain Name

Data Returned

- A positive or negative value based upon whether the Domain Name is available or not.

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2005 Parameter value syntax error
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a command to add domain names to be checked
EppCommandCheckDomain cmd = new EppCommandCheckDomain();
// set up transaction id, if needed
cmd.setClientTransactionId("MY-DOMAIN-CHECK-TRANSACTION-ID");
// add domain names to be checked for availability
cmd.add("example1.EXAMPLE");
cmd.add("example2.EXAMPLE");
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
// get response data
EppResponseDataCheck data;
data = (EppResponseDataCheck) res.getResponseData();
if (data == null)
{
    // this should not happen, error processing ...
}
// check the availability of the domain names
if( data.isAvailable("example1.EXAMPLE") )
{
    // example1.EXAMPLE is available
}
else
{
    // example1.EXAMPLE is not available
    if( data.getReason("example1.EXAMPLE") != null )
    {
        // display reason text, if any
    }
}
```

5.4.6 Renew Domain Name [EppCommandRenewDomain]

This class renews the Registration Period of a Domain Name by changing the Expiration Date and the Registration Period. This class is also used to restore registered domain names that have been inadvertently deleted but are within the 30-day Redemption Grace Period.

Required Attributes

- Domain Name
- Expiration Date – Current expiration date of the domain
- Registration Renewal Period (optional). Number of years the domain name's Registration Period will be renewed for. The total maximum number of years a domain can be registered for is 5 years. If period is not passed, by default domain is renewed for 1 year.
- Expiration year - The expiration year before renewal of the domain name.

Data Returned

- None

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2005 Parameter value syntax error
- 2104 Billing failure
- 2105 Object is not eligible for renewal
- 2303 Object does not exist
- 2304 Object status prohibits operation
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// set current expiration date to prevent the domain name from
// being renewed twice
SimpleDateFormat datefmt = new SimpleDateFormat("yyyy-MM-dd");
datefmt.setTimeZone(TimeZone.getTimeZone("GMT"));
Date expdate = datefmt.parse("2004-08-01", new ParsePosition(0));
Calendar cal = Calendar.getInstance();
cal.setTime(expdate);
// create a domain renew command
EppCommandRenewDomain cmd = new EppCommandRenewDomain("example.EXAMPLE", cal);
// set up transaction id, if needed
cmd.setClientTransactionId("MY-DOMAIN-RENEW-TRANSACTION-ID");
// renew the domain for another year
cmd.setPeriod(new EppPeriod(1, EppPeriod.UNIT_YEAR));
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
// get response data
EppResponseDataRenewDomain data;
data = (EppResponseDataRenewDomain) res.getResponseData();
if (data == null)
{
    // this should not happen, error processing ...
}
// domain name has been successfully renew, check new expiration date ...
```

5.4.7 Transfer (Request / Query) Domain Name [EppCommandTransferDomain]

This operation is used to do one of several functions: request the transfer a domain to a different registrar, query the status of an already submitted transfer request, or to approve/disapprove a transfer request. If a domain is transferred successfully, one additional year is added to the Registration Period and Expiration Date of the transferred domain.

Required Attributes

- Domain Name
- Operation Type (REQUEST, REJECT, QUERY, APPROVE, CANCEL)

Optional Attributes

- Registration Period (only for a transfer request)

Data Returned

- Domain name
- transfer Status
- Requested Registrar id
- Request date
- Sponsoring Registrar id
- Action Requested date from Sponsoring Registrar (action not necessary for Neustar SRS)
- Domain expiration date

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2005 Parameter value syntax error
- 2104 Billing failure
- 2106 Object is not eligible for transfer
- 2300 Object pending transfer
- 2301 Object not pending transfer
- 2303 Object does not exist
- 2304 Object status prohibits operation
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
EppCommandTransferDomain cmd = new EppCommandTransferDomain("example.EXAMPLE");
// set up transaction id, if needed
cmd.setClientTransactionId("MY-DOMAIN-TRANSFER-TRANSACTION-ID");
// renew the domain for another year after the transfer
cmd.setPeriod(new EppPeriod(1, EppPeriod.UNIT_YEAR));
// set the operation as "request"
cmd.setOperation(EppCommandTransfer.OPTYPE_REQUEST);
// set the authInfo related to the domain name for a
// transfer request operation. Other transfer operations do
// not need authInfo
cmd.setAuthInfo(new EppAuthInfo("pw", "2fooBar"));
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
// get response data
EppResponseDataTransferDomain data;
data = (EppResponseDataTransferDomain) res.getResponseData();
if (data == null)
{
    // this should not happen, error processing ...
}
// domain transfer request has been submitted successfully
// use access methods to get details of the transfer information
System.out.println("Domain Name    : " + data.getName());
System.out.println("Transfer status    : " + data.getTransferStatus());
System.out.println("New expiration date : " + data.getDateExpired().getTime());
System.out.println("Gaining registrar Id: " + data.getRequestClientId());
System.out.println("Request made at    : " + data.getRequestDate().getTime());
System.out.println("Losing registrar Id : " + data.getActionClientId());
System.out.println("Must act before    : " + data.getActionDate().getTime());
// ...
```

5.5 Contact Management Operations

5.5.1 Create Contact [EppCommandCreate]

This operation enters Contact Information into the system.

Required Attributes

- Name
- Organization
- Address1
- City
- Country
- Email
- Phone

Optional Attributes

- Address2
- Address3
- State/Province
- Postal Code
- Fax
- Registrar name/value pairs

Implicit Attributes

- Status
- Sponsoring Registrar
- Created By
- Created Date
- Updated By
- Updated Date

Data Returned

- Contact ID

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2004 Parameter value range error
- 2005 Parameter value syntax error
- 2302 Object exists
- 2306 Parameter value policy error
- 2308 Data management policy violation
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a contact object
EppContact contact = new EppContact();
// set contact id
contact.setId("JL34");
// set voice/fax/email
contact.setVoice("+1.1234567890", "9999");
contact.setFax("+1.1234567891");
contact.setEmail("user@myisp.EXAMPLE");
// set authInfo related to the contact object
contact.setAuthInfo(new EppAuthInfo(EppAuthInfo.TYPE_PW, "SecretPW"));
// create the internationalized (ASCII) version of the address
EppAddress int_addr = new EppAddress();
// up to 3 street address lines may be provided; NeuLevel requires
// at least one street address, although EPP-1.0 makes it optional
int_addr.setStreet(0, "1000 Test street");
int_addr.setStreet(1, "Suite 8");
int_addr.setCity("Ascii City");
int_addr.setState("Any Province");
int_addr.setPostalCode("12345");
int_addr.setCountryCode("HR");
// create the set of contact data in the internationalized (ASCII) form
EppContactData int_data = new EppContactData("Leonidas Williams", "Asciii Inc.", int_addr);
contact.setContactDataInt(int_data);
// create the localized (utf-8) version of the address (optional)
EppAddress loc_addr = new EppAddress();
loc_addr.setStreet(0, "1000 Non-Ascii Street");
loc_addr.setStreet(1, "Suite 8");
loc_addr.setCity("UTF-8 City");
loc_addr.setState("Any Province");
loc_addr.setPostalCode("12345");
loc_addr.setCountryCode("HR");
// create the set of contact data in the localized (utf-8) form (optional)
EppContactData loc_data = new EppContactData("Le\u00C3\u00B3nidas Williams", "UTF-8 Inc.",
loc_addr);
contact.setContactDataLoc(loc_data);
// create the create command object with contact information
EppCommandCreate cmd = new EppCommandCreate(contact);
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success() )
{
    // error processing ...
}
// get the response data
EppResponseDataCreateContact res_data;
res_data = (EppResponseDataCreateContact) res.getResponseData();
if (res_data == null)
{
    // this should not happen, but need error processing any way
}
// display response data
System.out.println("Contact Id   : " + res_data.getId());
System.out.println("Creation Date: " + res_data.getDateCreated().getTime());
```

5.5.2 Modify Contact [EppCommandUpdateContact]

Modifies all or a subset of the attributes of a contact. The contact must have a status that allows this operation to be executed. The following set of attributes can be modified through this operation:

- Name
- Organization
- Address 1
- Address 2
- Address 3
- City
- State/Province
- Postal Code
- Country
- Email
- Phone
- Fax
- Status
- Registrar name/value pairs

Required Attributes

- Contact ID

5.5.2.1 Data Returned

- None

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2004 Parameter value range error
- 2005 Parameter value syntax error
- 2303 Object does not exist
- 2304 Object status prohibits operation
- 2306 Parameter value policy error
- 2308 Data management policy violation
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a command to update the contact object
EppCommandUpdateContact cmd;
cmd = new EppCommandUpdateContact("CONTACT-12345");
// set transaction id, if needed
cmd.setClientTransactionId("MY-CONTACT-UPDATE-TRANSACTION-ID");
// add/remove status values, if needed
cmd.addStatus(EppContact.STATUS_CLIENT_TRANSFER_PROHIBITED);
cmd.addStatus(EppContact.STATUS_CLIENT_DELETE_PROHIBITED);
// change the authInfo, if needed
cmd.setNewAuthInfo(new EppAuthInfo("pw", "2fooBar", "NEUSTAR-12345"));
// update voce/fax/email, if needed
cmd.setNewVoice("+1.7035551234", "1234");
cmd.setNewFax("+1.7035551234", "4321");
cmd.setNewEmail("foo@bar.EXAMPLE");
// if only the name of the internationalized form needs to be changed
EppContactData int_data = new EppContactData();
int_data.setName("My New Name");
cmd.setNewContactDataInt(int_data);
// if the address part of the localized form needs to be changed
EppContactData loc_data = new EppContactData();
loc_data.setAddress(new EppAddress("100 Park Street", "Sterling", "VA", "20166", "US"));
cmd.setNewContactDataInt(loc_data);
// send the command to the server and receive a response object.
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
else
{
    // domain name has been updated successfully
}
```

5.5.3 Query Contact [EppCommandInfoContact]

Given a contact's ID, the EppCommandInfoContact command returns the information about the contact. The EppCommandInfoContact command will return different sets of data depending on the ID of the requesting registrar. If the requesting registrar is the Sponsoring Registrar, then the server returns the complete set of data about the contact. If the requesting registrar is not the current Sponsoring Registrar, the data returned is equivalent to the Whois data of the contact .

Required Attributes

- Contact ID

Data Returned

- Contact ID
- Name
- Organization
- Sponsoring Registrar
- List of Registrar name/value pairs
- Updated Date

- ROID
- (Sponsoring Registrar Only:)
- Address1
- Address2
- Address3
- City
- State/Province
- Post Code
- Country
- I15d Address1
- I15d Address2
- I15d Address3
- I15d City
- I15d State/Province
- I15d Post Code
- I15d Country
- Email
- Phone
- Fax
- List of statuses
- Authorization information
- Last Transfer Date
- Created By
- Created Date
- Updated By

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2005 Parameter value syntax error
- 2303 Object does not exist
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a command and set the contact id to query
EppCommandInfoContact cmd = new EppCommandInfoContact("JL34");
// set up transaction id, if needed
cmd.setClientTransactionId("MY-CONTACT-QUERY-TRANSACTION-ID");
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
// the response data should contain a fully populated
// object as part of the response
EppResponseDataInfo data;
data = (EppResponseDataInfo) res.getResponseData();
if( data == null )
{
    // this should not happen, error processing ...
}
EppContact obj = (EppContact) data.getObject();
// use the EppContact access methods to obtain contact id information ...
```

5.5.4 Delete Contact [EppCommandDeleteContact]

Given a contact ID, this operation deletes the contact from the Registry. The contact object must have a status that permits this operation. A contact can only be deleted if it is not associated or used by any domain name.

5.5.4.1 Required Attributes

- Contact ID

Data Returned

- None

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2005 Parameter value syntax error
- 2303 Object does not exist
- 2304 Object status prohibits operation
- 2305 Object association prohibits operation
- 2400 Command failed
- 2500 Command failed; server ending session

- 2501 Authentication error; server closing connection

Usage

```
// create a command to delete the contact object
EppCommandDeleteContact cmd = new EppCommandDeleteContact("JL34");
// set up transaction id, if needed
cmd.setClientTransactionId("MY-CONTACT-DELETE-TRANSACTION-ID");
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
else
{
    // domain name has been deleted successfully
}
```

5.5.5 Check Contact [EppCommandCheckContact]

Given a contact ID, this operation checks whether a contact has already been entered in the Registry's System or not. The EppCommandCheckContact command can be used to verify the existence of up to 1000 contacts at one time.

Required Attributes

- One or more Contact's ID (max of 1000)

Data Returned

- A positive or negative value based on whether the contact's ID is available or not.

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2005 Parameter value syntax error
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a command to add contact ids to be checked
EppCommandCheckContact cmd = new EppCommandCheckContact();
// set up transaction id, if needed
cmd.setClientTransactionId("MY-CONTACT-CHECK-TRANSACTION-ID");
// add contact ids to be checked for availability
cmd.add("contact-1");
cmd.add("contact-2");
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
// get response data
EppResponseDataCheck data;
data = (EppResponseDataCheck) res.getResponseData();
if (data == null)
{
    // this should not happen, error processing ...
}
// check the availability of the contact ids
if( data.isAvailable("contact-1") )
{
    // contact-1 is available
}
else
{
    // contact-1 is not available
    if( data.getReason("contact-1") != null )
    {
        // display reason text, if any
    }
}
```

5.5.6 Transfer (Request / Query) Contact [EppCommandTransferContact]

This operation is used to request a transfer a contact, query the status of an already submitted transfer request, or to approve/disapprove a transfer request.

Required Attributes

- Contact ID
- Operation Type (REQUEST, REJECT, QUERY, APPROVE, CANCEL)

Data Returned

- Contact ID
- Transfer status
- Requested Registrar Id
- Request date
- Sponsoring Registrar Id
- Action Requested date from Sponsoring Registrar (action not necessary for Neustar SRS)

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2005 Parameter value syntax error
- 2106 Object is not eligible for transfer
- 2300 Object pending transfer
- 2301 Object not pending transfer
- 2303 Object does not exist
- 2304 Object status prohibits operation
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
EppCommandTransferContact cmd = new EppCommandTransferContact("mycontact");
// set up transaction id, if needed
cmd.setClientTransactionId("MY-CONTACT-TRANSFER-TRANSACTION-ID");
// set the operation as "request"
cmd.setOperation(EppCommandTransfer.OPTYPE_REQUEST);
// set the authInfo related to the contact object for a
// transfer request operation. Other transfer operations do
// not need authInfo
cmd.setAuthInfo(new EppAuthInfo("pw", "2fooBar"));
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
// get response data
EppResponseDataTransferContact data;
data = (EppResponseDataTransferContact) res.getResponseData();
if (data == null)
{
    // this should not happen, error processing ...
}
// domain transfer request has been submitted successfully
// use access methods to get details of the transfer information
System.out.println("Contact Id      : " + data.getId());
System.out.println("Transfer status    : " + data.getTransferStatus());
System.out.println("Gaining registrar Id: " + data.getRequestClientId());
System.out.println("Request made at    : " + data.getRequestDate().getTime());
System.out.println("Losing registrar Id : " + data.getActionClientId());
System.out.println("Must act before    : " + data.getActionDate().getTime());
// ...
```

5.6 Host Management Operations

5.6.1 Create Host [EppCommandCreate]

Create a host object in the registry.

Required Attributes

- Host Name
- IP Address

Optional Attributes

- Registrar name/value pairs

Implicit Attributes

- Status
- Sponsoring Registrar
- Created By
- Created Date
- Updated By
- Updated Date

Data Returned

- Host Name

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2004 Parameter value range error
- 2005 Parameter value syntax error
- 2004 Parameter value range error
- 2302 Object exists
- 2308 Data management policy violation
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a new host object
EppHost host = new EppHost("ns1.example1.CO");
// associate an IPv4 address to the host object, if needed
host.addAddress(new EppIpAddress("134.5.78.92"));
// associate an IPv6 address to the host object, if needed
host.addAddress(new EppIpAddress("FFFF::134.5.78.92", EppIpAddress.TYPE_V6));
// generate an actual create host command using the host
// object we just generated
EppCommandCreate cmd = new EppCommandCreate(host);
// set transaction id, if needed
cmd.setClientTransactionId("MY-HOST-CREATE-TRANSACTION-ID");
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if( res == null || ! res.success() )
{
    // error processing ...
}
// get the response data for the domain
EppResponseDataCreateHost res_data;
res_data = (EppResponseDataCreateHost) res.getResponseData();
if (res_data == null)
{
    // this should not happen, but need error processing any way
}
// display response data
System.out.println("Host Name : " + res_data.getName());
System.out.println("Creation Date: " + res_data.getDateCreated().getTime());
```

5.6.2 Modify Host [EppCommandUpdateHost]

Modifies all or a subset of the attributes of a host. The host must have a status that permits this operation to be executed. The following set of attributes can be modified through this operation:

- Host name
- IP Addresses
- Status
- Registrar name/value pairs

Required Attributes

- Host Name

Data Returned

- None

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session

- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2004 Parameter value range error
- 2005 Parameter value syntax error
- 2303 Object does not exist
- 2304 Object status prohibits operation
- 2308 Data management policy violation
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a command to update the host object
EppCommandUpdateHost cmd = new EppCommandUpdateHost("ns1.example.EXAMPLE");
// set transaction id, if needed
cmd.setClientTransactionId("MY-HOST-UPDATE-TRANSACTION-ID");

// add/remove status values, if needed
cmd.addStatus(EppHost.STATUS_CLIENT_UPDATE_PROHIBITED);
cmd.removeStatus(EppHost.STATUS_CLIENT_DELETE_PROHIBITED);
// add/remove ip addresses, if needed
cmd.addAddress(new EppIpAddress("192.168.1.1"));
cmd.addAddress(new EppIpAddress("FFFF::192.168.1.2", EppIpAddress.TYPE_V6));
cmd.removeAddress(new EppIpAddress("192.168.1.3"));
cmd.removeAddress(new EppIpAddress("FFFF::192.168.1.4", EppIpAddress.TYPE_V6));
// rename the host object, if needed
cmd.setNewName("ns2.example.EXAMPLE");
// send the command to the server and receive a response object.
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
else
{
    // host object has been updated successfully
}
```

5.6.3 Query Host [EppCommandInfoHost]

Given a Host Name, the EppCommandInfoHost command returns information about the Host. The EppCommandInfoHost command returns different sets of data depending on the ID of the requesting registrar. If the requesting registrar is the Sponsoring Registrar, then the server returns the complete set of data about the Host. If the requesting registrar is not the current Sponsoring Registrar, the data returned is equivalent to the Whois data of the Host.

Required Attributes

- Host Name

Data Returned

- Host Name

- IP Address
- Sponsoring Registrar
- List of Registrar name/value pairs
- Updated Date
- ROID
- Sponsoring Registrar Only
- List of Statuses
- Created By
- Created Date
- Updated By

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2005 Parameter value syntax error
- 2303 Object does not exist
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a command and set the host name to query
EppCommandInfoHost cmd = new EppCommandInfoHost("ns1.example.EXAMPLE");
// set up transaction id, if needed
cmd.setClientTransactionId("MY-HOST-QUERY-TRANSACTION-ID");
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
// the response data should contain a fully populated
// object as part of the response
EppResponseDataInfo data;
data = (EppResponseDataInfo) res.getResponseData();
if( data == null )
{
    // this should not happen, error processing ...
}
EppHost obj = (EppHost) data.getObject();
// use the EppHost access methods to obtain host object information ...
```

5.6.4 Delete Host [EppCommandDeleteHost]

Given a Host Name, this operation deletes the host from the registry. The host object must have a status that permits this operation. A Host can only be deleted if it is not associated or used by any domain name.

Required Attributes

- Host Name

Data Returned

- None

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.
- Result Codes
- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2303 Object does not exist
- 2304 Object status prohibits operation
- 2305 Object association prohibits operation
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a command to delete the host object
EppCommandDeleteHost cmd = new EppCommandDeleteHost("ns1.example.EXAMPLE");
// set up transaction id, if needed
cmd.setClientTransactionId("MY-HOST-DELETE-TRANSACTION-ID");
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
else
{
    // host object has been deleted successfully
}
```

5.6.5 Check Host [EppCommandCheckHost]

Given a Host Name, this operation checks whether a host has been entered in the registry or not. The CheckHost command can be used to verify the existence of up to 1000 hosts at one time.

Required Attributes

- One or more Host Names (a max of 1000)

Data Returned

- A positive or negative value depending upon whether the Host Name is available or not.

Exceptions

- On null response, check exceptions and error messages using EppChannel getException() and getMessage() methods.

Result Codes

- 1000 Command completed successfully
- 1500 Command completed successfully; ending session
- 2001 Command syntax error
- 2002 Command use error
- 2003 Required Parameter missing
- 2005 Parameter value syntax error
- 2400 Command failed
- 2500 Command failed; server ending session
- 2501 Authentication error; server closing connection

Usage

```
// create a command to add host names to be checked
EppCommandCheckHost cmd = new EppCommandCheckHost();
// set up transaction id, if needed
cmd.setClientTransactionId("MY-HOST-CHECK-TRANSACTION-ID");
for
cmd.add("ns1.example.EXAMPLE");
cmd.add("ns1.example.EXAMPLE");
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res == null || ! res.success())
{
    // error processing ...
}
// get response data
EppResponseDataCheck data;
data = (EppResponseDataCheck) res.getResponseData();
if (data == null)
{
    // this should not happen, error processing ...
}
// check the availability of the host objects
if( data.isAvailable("ns1.example.EXAMPLE") )
{
    // ns1.example.EXAMPLE is available
}
else
{
    // ns1.example.EXAMPLE is not available
    if( data.getReason("ns1.example.EXAMPLE") != null )
    {
        // display reason text, if any
    }
}
```

5.7 Response Processing Operations

5.7.1 Result Check

Success of a request can be checked by using the success() method of EppResponse object.

Required Attributes

- None

Data Returned

- None

Exceptions

- None

Result Codes

- None

Usage

```
EppResponse res = channel.send(eppcmd);
if (res == null || ! res.success())
{
    // error processing
}
else
{
    // receive a success response from the server
    // perform additional process, if needed
}
```

5.7.2 Accessing Response Data

Data received from the server in a response can be accessed using getResponseData() method of EppResponse.

Required Attributes

- None

Data Returned

- None

Exceptions

- None

Result Codes

- None

Usage

```
EppResponse res = channel.send(eppcmd);
if (res == null || ! res.success())
{
    // error processing
}
else
{
    // receive a success response from the server
    // perform additional process, if needed
    // responses to login/logout/update/delete and some
    // poll commands will not contain any response data
    EppResponseData data = res.getResponseData();
    if( data != null )
    {
        // display response data details
    }
}
```

5.8 Error Processing

Error conditions occurred during transmission of request/response are obtained by getting exceptions on EppSession and EppChannel class.

Required Attributes

- None

Data Returned

- null (EppResponse)

Exceptions

- None

Result Codes

- None

Usage

```
EppResponse res = channel.send(eppcmd);
if (res == null || ! res.success() )
{
    // a null response indicates a connection failure condition
    // and the channel and the session should be checked for
    // exceptions
    //
    // A failure here may mean the client cannot locate the EPP XSD
    // (XML Schema) files. make sure all the XSD files are located
    // in right directory (usually local directory)
    if( res == null )
    {
        Exception exp;
        String  err;
        exp = channel.getException();
        if (exp == null )
        {
            exp = session.getException();
        }
        if (exp != null)
        {
            System.out.println("Failing with exception: ");
                exp.printStackTrace();
        }
        err = channel.getMessage();
        if (err == null )
        {
            err = session.getMessage();
        }
        if (err != null)
        {
            System.out.println("Failing with message: " + err);
        }
    }
    else
    {
        // display result messages
        Vector list = res.getResult();
        for (int i = 0; (list != null) && (i < list.size()); i++)
        {
            Object obj = list.get(i);
            if ((obj == null) || ! (obj instanceof EppResult))
            {
                // this should not happen, ignore
            }
            EppResult result = (EppResult) obj;
            // display EPP result code, message texts, reasons ...
        }
    }
}
```

5.9 Logging Operations

5.9.1 Displaying a Request or Response

A request sent or response received can be converted in to a XML element for logging or debugging purposes using toString() method on the entities.

Required Attributes

- None

Data Returned

- None

Exceptions

- None

Result Codes

- None

Usage

```
// create a command to add host names to be checked
EppCommandCheckHost cmd = new EppCommandCheckHost();
// set up transaction id, if needed
cmd.setClientTransactionId("MY-HOST-CHECK-TRANSACTION-ID");
// add host names to be checked for availability
cmd.add("ns1.example.EXAMPLE");
cmd.add("ns1.example.EXAMPLE");
// display the XML message to be sent out
System.out.println(cmd.toString());
// send the command to the server and receive a response object
EppResponse res = channel.send(cmd);
if (res != null)
{
    // display the raw message received
    System.out.println("Message Received: " + channel.getMessage());
    // display the XML message received after decoding
    System.out.println("Message Decoded : " + res.toString());
}
// ...
```

6 Support

6.1 EPP / OT&E Certification / Domain Transaction Related

For EPP support, Operational Test and Evaluation Certification (OT&E), or specific domain transaction questions, please contact your registrar relations representative or email reg-support@neustar.biz.

6.2 Third-Party Application Support

Please contact the respective third-party for support of their application.

7 Test client and Dummy servers

Neustar's EPP toolkit also includes sample test clients and dummy EPP servers in Java and C++ for testing purpose. The source and binary files are located in the following directory:

```
java/test
c++/test
```

For instructions on building and running the test clients and dummy servers, please refer to files

```
java/test/RunTestTcp
c++/test/RunTestTcp
```

8 Third-Party Required Software: Java

This section describes the packages required, programming language and protocol familiarity recommended in order to use the EPP Java APIs. For more details please consult the README document provided in the Registrar Toolkit (RTK).

8.1 JAVA 1.6 Software Development Kit

Java 2 Platform, Standard Edition (J2SE) Version 1.6

The Java 1.6 Software Development Kit is required to compile the source code for the API and the test programs as well as to compile code written for use with the API. There are several implementations of the Java 1.6 standard available for most platforms. Sun's reference implementation is available from:

http://java.sun.com/j2se/

8.2 Apache Xalan-Java 2.7.1

This is a Java API to parse and process XML documents. It is an open source library provided as part of the Apache project and includes the xercesImpl.jar which is also required by the project. Xalan and Xerces used by the EPP Java API to parse and generate XML representing EPP commands. The API is available from:

http://xml.apache.org/xalan-j/

9 Third-Party Required Software: C++

This section describes the packages required, programming language and protocol familiarity recommended to use the EPP C++ API. For more details please consult the README document provided in the RTK.

9.1 XERCES-C++ Version 1.5+

This is a C++ API to parse and process XML documents. It is an open source library provided as part of the Apache project. It is used by the EPP C++API to parse and generate XML representing EPP commands. The API (in source and binary form) is available from:

http://xml.apache.org/xerces-c/index.html

9.2 OPENSSL Version 1.0.1e

This is an open-source C library providing the full suite of SSL facilities. It is required to communicate with a registry that uses TLS or SSL over TCP as a transport layer. It can be freely downloaded (in source and binary form) from:

http://www.openssl.org/source/

9.3 DOC++ Version 3.4.8

This is an open-source documentation system that supports generation of HTML and TeX formatted C++ documentation. It is freely available from:

http://docpp.sourceforge.net

9.4 Supported C++ Platforms

LINUX	G++ 2.96
HPUX	11.0 : aCC: HP ANSI C++ B3910B A.03.13
SOLARIS	2.7/2.8: G++ 2.95.2 or SC 4.2
WINDOWS	MS VC++ 6.0 (CL 12.00.8168).

Note: you must build the executable with the /MD or /MDd options for multi-threading.

10 EPP and Extensions Information

The Extensible Provisioning Protocol (EPP) was a work product of the IETF Provisioning Registry Protocol (provreg) working group. EPP was published as a Proposed Standard (RFCs 3730, 3731, 3732, 3733, and 3734) in March 2004. It became a Draft Standard (RFCs 4930, 4931, 4932, 4933, and 4934) in May 2007, and a Standard (Standard 69; RFCs 5730, 5731, 5732, 5733, and 5734) in August 2009. It is the standard domain name provisioning protocol for generic top-level domain name registries that operate under the auspices of the Internet Corporation for Assigned Names and Numbers (ICANN). It is also used by a number of country code top-level domain registries.

The following documents can be found on the IETF Web Site at the following link:

http://www.ietf.org/rfc.html

10.1 STD 69

This document includes the following RFC 5730 through RFC 5734. The document specifies an Internet standards track protocol for the Internet community.

10.2 Extensible Provisioning Protocol (EPP) / RFC 5730

This document describes an application layer client-server protocol for the provisioning and management of objects stored in a shared central repository. Specified in XML, the protocol defines generic object management operations and an extensible framework that maps protocol operations to objects. This document includes a protocol specification, an object mapping template, and an XML media type registration.

10.3 EPP Domain Name Mapping / RFC 5731

This document describes an EPP mapping for the provisioning and management of Internet domain names stored in a shared central repository. Specified in XML, the mapping defines EPP command syntax and semantics as applied to domain names.

10.4 EPP Host Mapping / RFC 5732

This document describes an EPP mapping for the provisioning and management of Internet host names stored in a shared central repository. Specified in XML, the mapping defines EPP command syntax and semantics as applied to host names.

10.5 EPP Contact Mapping / RFC 5733

This document describes an EPP mapping for the provisioning and management of individual or organizational social information identifiers (known as "contacts") stored in a shared central

repository. Specified in Extensible Markup Language (XML), the mapping defines EPP command syntax and semantics as applied to contacts.

10.6 EPP Transport Over TCP / RFC 5734

This document describes how an EPP session is mapped onto a single Transmission Control Protocol (TCP) connection. This mapping requires use of the Transport Layer Security (TLS) protocol to protect information exchanged between an EPP client and an EPP server.

10.7 Guidelines for Extending the EPP / RFC 5735

EPP is an application layer client-server protocol for the provisioning and management of objects stored in a shared central repository. Specified in XML, the protocol defines generic object management operations and an extensible framework that maps protocol operations to objects. This document presents guidelines for use of EPP's extension mechanisms to define new features and object management capabilities.

11 Appendix A. Glossary

Domain Name. A domain name in order to be valid must be from 3 to 63 characters plus the TLD extension. It can contain only the characters a-z, 0-9, and "_". Any other characters will make the domain name invalid. For example, **abc.biz** is a valid domain name; **ab.biz** is not.

Extensible Provisioning Protocol (EPP). A protocol designed to facilitate communication between registrars and registries.

Name Server. A server that is responsible for resolution of all children of the second-level domain. Two nameservers are required for a domain to be registered with the .BIZ TLD.

Registrant. A customer who is registering a domain name via a Registrar.

Registrar. An organization responsible for the registration of domain. A registrar may provide registration services for one or more top-level domains (TLD). There can be any number of registrars for any given TLD.

Registry. Has the exclusive responsibly for maintaining and updating of Root Name Servers for its particular TLD. For.

Second-Level Domain (SLD). Refers to the next level below a top-level domain (TLD), for example **example.example**.

Shared Registry System (SRS). A Shared Registry System enables multiple registrars to register domain information on behalf of a registrant with the NeuLevel Registry.

Top Level Domain (TLD). The right-most label in a domain name, e.g., .EXAMPLE.

12 Appendix B. Status List

This section contains a list of attribute values that certain attributes can carry. Registrars can only modify Client based statuses. The Registry can modify the Client and Server based statuses. Statuses are updated using the appropriate EppCommandModify command for the object.

12.1 Domain Name Status List

A domain object can have multiple status values, each status having its own particular semantic. Statuses are either Client or Server statuses. These status values and semantics include:

1. ClientDeleteProhibited [EppDomain.STATUS_CLIENT_DELETE_PROHIBITED]: A Registrar request to delete the object must be rejected. May only be set by the sponsoring Registrar.

2. ServerDeleteProhibited [EppDomain.STATUS_SERVER_DELETE_PROHIBITED]: Any request to delete the object must be rejected. May only be set by Registry

3. ClientHold [EppDomain.STATUS_CLIENT_HOLD]: The domain must not appear in the TLD zone file. May only be set by the sponsoring Registrar.

4. ServerHold [EppDomain.STATUS_SERVER_HOLD]: The domain must not appear in the TLD zone file. May only be set by Registry.

5. ClientRenewProhibited [EppDomain.STATUS_CLIENT_RENEW_PROHIBITED]: A Registrar request to renew the object must be rejected. May only be set by the sponsoring Registrar.

6. ServerRenewProhibited EppDomain.STATUS_SERVER_RENEW_PROHIBITED]: Any request to renew the object must be rejected. May only be set by Registry.

7. ClientTransferProhibited [EppDomain.STATUS_CLIENT_TRANSFER_PROHIBITED]: A Registrar request to transfer the object must be rejected. May only be set by the sponsoring Registrar.

8. ServerTransferProhibited [EppDomain.STATUS_SERVER_TRANSFER_PROHIBITED]: Any request to transfer the object must be rejected. May only be set by Registry.

9. ClientUpdateProhibited [EppDomain.STATUS_CLIENT_UPDATE_PROHIBITED]: A Registrar request to update the object must be rejected. May only be set by the sponsoring Registrar. The only update allowed is to remove the ClientUpdateProhibited status.

10. ServerUpdateProhibited [EppDomain.STATUS_SERVER_UPDATE_PROHIBITED]: Any request to update the object must be rejected. May only be set by Registry. The only update allowed is to remove the ServerUpdateProhibited status.

11. Inactive [EppDomain.STATUS_INACTIVE]: The domain must not appear in the zone because it lacks proper delegation information (lacks nameserver information). May only be set by Registry.

12. OK [EppDomain.STATUS_OK]: No other statuses set. May only be set by Registry.

13. PendingCreate [EppDomain.STATUS_PENDING_CREATE]: A create request is successful, but the formal create is pending in the system. May only be set by Registry.

14. PendingUpdate [EppDomain.STATUS_PENDING_UPDATE]: An update request is successful, but the formal update is pending in the system. May only be set by Registry.

15. PendingDelete [EppDomain.STATUS_PENDING_DELETE]: A delete request is successful, but the formal delete is pending in the system. The domain must not be published in the zone. All requests must be rejected. May only be set by Registry.

16. PendingTransfer [EppDomain.STATUS_PENDING_TRANSFER]: A transfer request has been received for the object, and completion of the request is pending. The delete, update, and renew commands must be rejected. May only be set by Registry.

Prohibited status combinations are:

1. The OK status must not be combined with any other status.

2. The PendingDelete status must not be combined with either ClientDeleteProhibited or ServerDeleteProhibited.

3. The PendingTransfer status must not be combined with either ClientTransferProhibited or ServerTransferProhibited.

12.2 Name Server Status List

A Host object can have multiple status values, each of which has its own particular semantic. These status values and semantics include:

1. ClientDeleteProhibited [EppHost.STATUS_CLIENT_DELETE_PROHIBITED]: A Registrar request to delete the object must be rejected. May only be set by the sponsoring Registrar.

2. ServerDeleteProhibited [EppHost.STATUS_SERVER_DELETE_PROHIBITED]: Any request to delete the object must be rejected. May only be set by Registry

3. ClientUpdateProhibited [EppHost.STATUS_CLIENT_UPDATE_PROHIBITED]: A Registrar request to update the object must be rejected. May only be set by the sponsoring Registrar. The only update allowed is to remove the ClientUpdateProhibited status.

4. ServerUpdateProhibited [EppHost.STATUS_SERVER_UPDATE_PROHIBITED]: Any request to update the object must be rejected. May only be set by Registry. The only update allowed is to remove the ServerUpdateProhibited status.

5. Linked [EppHost.STATUS_LINKED]: The Object has at least one active association with another object, such as a Domain object. May only be set by Registry.

6. OK [EppHost.STATUS_OK]: No other statuses set. May only be set by Registry.

7. PendingCreate [EppHost.STATUS_PENDING_CREATE]: A create request is successful, but the formal create is pending in the system. May only be set by Registry.

8. PendingUpdate [EppHost.STATUS_PENDING_UPDATE]: An update request is successful, but the formal update is pending in the system. May only be set by Registry.

9. PendingDelete [EppHost.STATUS_PENDING_DELETE]: A delete request is successful, but the formal delete is pending in the system. The host must not be published in the zone. All requests must be rejected. May only be set by Registry.

10. PendingTransfer [EppHost.STATUS_PENDING_TRANSFER]: the subordinate domain object for this nameserver is in PendingTransfer status and the request transfer is pending.

Prohibited status combinations are:

1. The OK status must not be combined with any other status.

2. The PendingDelete status must not be combined with either ClientDeleteProhibited or ServerDeleteProhibited.

12.3 Contact Status List

A Contact object must have the capability to have multiple status values, each of which has particular semantics. These status values and semantics include:

1. ClientDeleteProhibited [EppContact.STATUS_CLIENT_DELETE_PROHIBITED]: A Registrar request to delete the object must be rejected. May only be set by the sponsoring Registrar.

2. ServerDeleteProhibited [EppContact.STATUS_SERVER_DELETE_PROHIBITED]: Any request to delete the object must be rejected. May only be set by Registry

3. ClientTransferProhibited [EppContact.STATUS_CLIENT_TRANSFER_PROHIBITED]

4. A Registrar request to transfer the object must be rejected. May only be set by the sponsoring Registrar.

5. ServerTransferProhibited [EppContact.STATUS_SERVER_TRANSFER_PROHIBITED]: Any request to transfer the object must be rejected. May only be set by Registry.

6. ClientUpdateProhibited [EppContact .STATUS_CLIENT_UPDATE_PROHIBITED]: A Registrar request to update the object must be rejected. May only be set by the sponsoring Registrar. The only update allowed is to remove the ClientUpdateProhibited status.

7. ServerUpdateProhibited [EppContact.STATUS_SERVER_UPDATE_PROHIBITED]: Any request to update the object must be rejected. May only be set by Registry. The only update allowed is to remove the ServerUpdateProhibited status.

8. Linked [EppContact.STATUS_LINKED]: The Object has at least one active association with another object, such as a Domain object. May only be set by Registry

9. OK [EppContact.STATUS_OK]: No other statuses set. May only be set by Registry.

10. PendingCreate [EppContact.STATUS_PENDING_CREATE]: A create request is successful, but the formal create is pending in the system. May only be set by Registry.

11. PendingUpdate [EppContact.STATUS_PENDING_UPDATE]: An update request is successful, but the formal update is pending in the system. May only be set by Registry.

12. PendingDelete [EppContact.STATUS_PENDING_DELETE]: A delete request is successful, but the formal delete is pending in the system. All requests must be rejected. May only be set by Registry.

13. PendingTransfer [EppContact.STATUS_PENDING_TRANSFER]: A transfer request has been received for the object, and completion of the request is pending. The delete, update, and renew commands must be rejected. May only be set by Registry.

Prohibited status combinations are:

1. The OK status may only be combined with the Linked status. It must not be combined with any other status.

2. The PendingDelete status must not be combined with either ClientDeleteProhibited or ServerDeleteProhibited.

3. The PendingTransfer status must not be combined with either ClientTransferProhibited or ServerTransferProhibited

13Appendix C. Redemption Grace Period (RGP)

The Redemption Grace Period enables registrars to restore registered domain names that have been inadvertently deleted but are within the Redemption Grace Period. This is done using EppCommandRenewDomain.

The following is a list of the additional data elements required to restore a domain. These are specified via the <unspec> element of a <domain:renew> command.

Note that the <domain:renew> command must NOT include a Registration Renewal Period value.

Table 3 Data Elements Required to Restore a Domain

Name	Purpose
RestoreReasonCode	The Reason for deletion. The following values are valid: **Code** **Description** 1 Registrant Error 2 Registrar Error 3 Judicial/Arbitration/Legal 0 Other
RestoreComment	A text explanation for the deletion. If no additional comment is necessary, a value of “none” is recommended. Spaces are not valid. Use underscores or hyphens to separate words.
TrueData	Registrar’s affirmation that the data is accurate. Must always = Y
ValidUse	Registrar’s affirmation that the domain is being restored in conformance to ICANN policies. Must always = Y

Example Command

```
<?xml version="1.0" encoding="UTF-8"?>
<epp xmlns="urn:iana:xml:ns:epp-1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:iana:xml:ns:epp-
1.0 epp-1.0.xsd">
 <command>
  <renew>
   <domain:renew xmlns="urn:iana:xml:ns:domain-1.0"
    xmlns:domain="urn:iana:xml:ns:domain-1.0" xsi:schemaLocation="urn:iana:xml:ns:domain-1.0
domain-1.0.xsd">
    <name>example.example</name>
    <curExpDate>2014-04-18</curExpDate>
   </domain:renew>
  </renew>
  <unspec>
     RestoreReasonCode=1 RestoreComment=deletedByMistake TrueData=Y ValidUse=Y
  </unspec>
  <clTRID>rat--27637957411123</clTRID>
 </command>
</epp>
```

Example Response

```
<XMP>
<?xml version="1.0" encoding="UTF-8"?>
<epp xmlns="urn:iana:xml:ns:epp-1.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:iana:xml:ns:epp-
1.0 epp-1.0.xsd">
 <response>
  <result code="1000">
   <msg>Command completed successfully</msg>
   <value>SRS Major Code: 2000</value>
   <value>SRS Minor Code: 20003</value>
   <value>--DOMAIN_SUCCESSFULLY_RESTORED</value>
  </result>
  <resData>
   <domain:creData xmlns="urn:iana:xml:ns:domain-1.0"
    xmlns:domain="urn:iana:xml:ns:domain-1.0" xsi:schemaLocation="urn:iana:xml:ns:domain-1.0
domain-1.0.xsd">
    <name>EXAMPLE.EXAMPLE</name>
    <exDate>2004-04-18T23:59:59.0Z</exDate>
   </domain:creData>
  </resData>
  <trID>
   <clTRID>rat--27637957411123</clTRID>
   <svTRID>rat--27637957411123--SERVER</svTRID>
  </trID>
 </response>
</epp>  </resData>
```