ARI Registry Services

DNRS EPP Toolkit Guide

Thursday, 2 August 2012

This document is provided pursuant to the disclaimer provided on the last page, and the following.

Revision History

| **Version** | **Date** | **Author(s)** | **Changes** |
| --- | --- | --- | --- |
| 1.0 | 2 Aug 2012 | PM | Initial Release |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Document Contact

Contact the following for details relating to this document:

| **Name** | ARI Registry Services |
| --- | --- |
| **Address** | Level 8, 10 Queens Road, Melbourne 3004 Victoria Australia |
| **Contact Number** | +61 3 9866 3710 |
| **Email** | info@ariservices.com |

About ARI Registry Services

In October 2011, AusRegistry International evolved to a name and brand identity in a move to support the continued expansion of the organisation and position it as a dominant force in the global TLD Registry Services marketplace.

ARI Registry Services is now used as a trading name of the AusRegistry International corporate entity.

Document Purpose

The purpose of this document is to provide guidance on the usage and implementation of the DNRS EPP Toolkit provided by ARI Registry Services.

It also provides usage examples for application developers to help build/integrate systems using the Toolkit API.

Intended Audience

This document is intended for use by the following:

Registrars

Registry Operators

Software Developers

Solution Architects

System Integrators

Required Reading

Relevant EPP RFCs (see Appendix).

Contents

[1 Introduction 5](#_Toc331683485)

[1.1 Toolkit Overview 5](#_Toc331683486)

[2 Installation and Setup 6](#_Toc331683487)

[2.1 Environment 6](#_Toc331683488)

[2.1.1 Java 6 6](#_Toc331683489)

[2.1.2 UTF-8 Encoding 6](#_Toc331683490)

[2.2 Configuration 6](#_Toc331683491)

[2.3 Registrar Responsibilities 6](#_Toc331683492)

[3 Quick Start Guide 8](#_Toc331683493)

[3.1 Sending a Command 8](#_Toc331683494)

[3.2 Performing Bulk Operations 10](#_Toc331683495)

[3.3 Using extensions with commands 11](#_Toc331683496)

[Receiving extension data 12](#_Toc331683497)

[4 Toolkit Implementation Notes 14](#_Toc331683498)

[4.1 Connection and Session Management 14](#_Toc331683499)

[4.2 XML Marshalling and Unmarshalling 14](#_Toc331683500)

[4.3 Logging 15](#_Toc331683501)

[5 Appendix 17](#_Toc331683502)

[5.1 Specifications Implemented 17](#_Toc331683503)

[5.1.1 RFCs 17](#_Toc331683504)

[5.1.2 Other Extensions 17](#_Toc331683505)

# Introduction

The Extensible Provisioning Protocol (EPP) was selected as the registry‐registrar protocol for communication between ARI Registry Services’ Domain Name Registry System (DNRS) and the registrars licensed to interact with the registry.

The core EPP specifications provide for management of domains, hosts (for DNS delegation), and contacts (for enabling communication with the entities responsible for a domain name registration). The protocol is extensible in various ways, including support for; extension to objects other than domains, hosts and contacts, extension of the commands defined on existing objects, and extension of the protocol to commands not defined in the core protocol.

To communicate with an EPP‐based DNRS, a registrar must send commands in the form of XML documents to the DNRS and receive and interpret the responses returned. The format of the commands and responses is formally specified in an XML schema forming part of the EPP specifications. Transmission of the service elements must be secured using Transport Layer Security version 1 (TLSv1). In addition, opening of an EPP session requires the provision of client credentials delivered via an out of band mechanism.

Implementation of much of the EPP specification can be independent of the specific registrar requirements, since every implementation must provide such services as discovering registry service information, opening and closing sessions, sending and receiving EPP service elements, as well as a simple means of translating between EPP service elements and their programmatic representation.

These services are best bundled in a library which each registrar can utilise to reduce the costs of implementing an EPP client application.

## Toolkit Overview

The EPP Toolkit developed and supplied by ARI provides the client‐side libraries which implement the core EPP specifications, the domain, host and contact mappings of the specifications, and mappings for extensions operated by ARI (for a list of all specifications and extensions implemented see the Appendix). These libraries are broken down into two key modules: an extensible set of EPP service element mappings to classes (object‐oriented programming paradigm), and an EPP network transport module.

The service element mapping module provides a simple means of translating between EPP service elements and their programmatic representation. The network transport module, which depends on session management service elements in the service element module, provides the following services; service information discovery, opening and closing EPP sessions, and sending and receiving EPP service elements.

# Installation and Setup

## Environment

The following environment specifics are required:

### Java 6

The Toolkit API was implemented in Java using only standard libraries (some of which are only standard with Java SE 6 or later) to minimise dependency on external resources.

### UTF-8 Encoding

The Toolkit uses the Java VM default character set for character encoding. Consequently, the default character set must be UTF‐8 to properly parse and encode UTF‐8 characters in sent and received EPP messages. For English Windows machines, the default character set is typically Cp1252, and can be changed to UTF‐8 by setting the *file.encoding* system property to UTF-8. This can be done on the command line with the syntax:

java –Dfile.encoding=UTF-8 ...

## Configuration

Configuration parameters are read from a properties file called toolkit.properties (default) on startup. The toolkit.properties file should be in the applications classpath for the Toolkit to read from it.

Configuration of the following properties is mandatory. All other values are set to intelligent defaults.

epp.client.clID =

epp.client.password=

epp.client.hostname =

The Toolkit is designed to be flexible enough to read properties from any source as long as the data source implements the SessionManagerProperties interface. For example, you could source your configuration parameters from an encrypted file by providing an implementation of SessionManagerProperties to read from the encrypted file.

## Registrar Responsibilities

It is your responsibility to protect the Toolkit parameter file which contains the client identifier and password for login, and also to implement suitable mechanisms to protect the cryptographic keys used by the Toolkits’ TLS implementations.

The default log files contain XML sent to and received from the EPP server. These log files may contain information of a sensitive nature, for example domain name auth info. Clients should take care to ensure this information is accessible only to those that require it. Applications may disable logging of commands; however this may impair ability to provide support.

# Quick Start Guide

## Sending a Command

ARI’s EPP Toolkit allows you to send commands to an EPP service and receive back responses. To send commands it is necessary to create a session, which will handle socket connection, the EPP greeting, and logging in.

**Create a session**

To create a session, an implementation of SessionManagerProperties is required. The default implementation (SessionManagerPropertiesImpl) uses a property file (defaulting to 'toolkit.properties' if not specified on construction) obtained from the classpath, to handle properties such as EPP server host, port, and user details. You can also use a custom implementation of SessionManagerProperties if this behaviour is not appropriate for your circumstances. The following code creates a new SessionManager:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9** |  | /\* Read in configuration properties from the toolkit.properties  file. This file is searched for on the classpath. Alternatively  you can create your own implementation of  sessionmanagerproperties should the default behaviour not suit. \*/  **properties = new SessionManagerPropertiesImpl("toolkit.properties");**  /\* Create a new session manager. This will use the properties loaded  above to set up parameters required to connect to an EPP server. \*/  **manager = SessionManagerFactory.newInstance(properties);** |

**Start a session**

After obtaining a SessionManager, it is necessary to start a session. This will create a connection, handle the EPP greeting and then login, allowing you to send accross EPP commands:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3**  **4** |  | /\* Start the session. This will automatically create a connection, send  a hello and a greeting and perform a login. The manager will be  ready to execute transactions after this call. \*/  **manager.startup();** |

**Create a command**

After starting a session, a command must be created. It is also necessary to create a response object to store the response details for your command. This will automatically be populated from the XML returned from the EPP service. The following code creates a Domain Check command and response, and sends it to the EPP server:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9** |  | // Create a domain check command  **DomainCheckCommand command = new DomainCheckCommand(domainName);**  // Create the required response object for the domain check  **final DomainCheckResponse response = new DomainCheckResponse();**  /\* Execute the command using the session manager, wrapping it in a  Transaction object \*/  **manager.execute(new Transaction(command, response));** |

**View response details**

After creating a command, you can view the details of the response:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3** |  | // Print out the details of the response  **System.out.println("EPP Response code: " +**  **response.getResults()[0].getResultCode());** |

**End a session**

After you have finished sending commands, it is possible to end the session, log out and close the socket connection:

|  |  |  |
| --- | --- | --- |
| **1**  **2** |  | // End the session, disconnecting the socket connection as well  **manager.shutdown();** |

## Performing Bulk Operations

It is recommended to send multiple commands in the same session, to avoid the overhead of connecting and sending login commands each time. The following example performs multiple domain checks, saving the results in an array of result objects:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9** |  | **final int loops = 5;**  **final DomainCheckResponse[] responses = new**  **DomainCheckResponse[loops];**  **for (int i = 0; i < loops; i++) {**  **DomainCheckCommand command = new DomainCheckCommand(domainName);**  **final DomainCheckResponse response = new DomainCheckResponse();**  **manager.execute(new Transaction(command, response));**  **responses[i] = response;**  **}** |

## Using extensions with commands

The following example shows how to use the SecDNS extension with a Domain Create. It is necessary to specify what extensions you will be using at login time. Therefore, the SessionManagerProperties will use a list of extension uris when logging in to the EPP server. In the default SessionManagerPropertiesImpl, these will be stored in the toolkit.properties file with the prefix 'xml.uri.ext'.

Set up a standard Domain Create:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8** |  | // Create a domain create command with the minimum required parameters  **final DomainCreateCommand command = new**  **DomainCreateCommand(domainName, password, contactName, new**  **String[] {contactName});**  // Create the domain create response.  **final DomainCreateResponse domainCreateResponse = new**  **DomainCreateResponse();** |

Initialise the extension class, SecDnsDomainCreateCommandExtension:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3** |  | // Create a SECDNS create command extension object  **final SecDnsDomainCreateCommandExtension ext = new**  **SecDnsDomainCreateCommandExtension();** |

Create a DSData object with the required DS data that you would like to add to the domain:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3**  **4** |  | /\* Create a DS data object, supplying key tag, algorithm, digest type  and digest \*/  **final DSData dsData = new DSData(1, 3, 1,**  **"49FD46E6C4B45C55D4AC49FD46E6C4B45C55D4AC");** |

Create a DSOrKeyType object, which is the container for all DS data and Key data associated with a domain:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3** |  | /\* Add the DS data to a DSOrKeyType object, which will store all DS  and Key data for a domain \*/  **final DSOrKeyType createData = new DSOrKeyType();** |

Add the DS data to the DSOrKeyType object:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3**  **4**  **5** |  | //Add the DS data to the DSOrKeyType object  **createData.addToDsData(dsData);**  // Add the DSOrKeyType object to the extension  **ext.setCreateData(createData);** |

Add the extension to the command. Multiple extensions can be added to the same command if this is required. You can then execute the command, which sends across the base command as well as all associate extensions:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6** |  | // Add the extension to the domain create command  **command.appendExtension(ext);**  /\* Tell the manager to execute the command. This command includes the  SECDNS extension object. \*/  **manager.execute(new Transaction(command, domainCreateResponse));** |

Receiving extension data

It is also possible to use the Toolkit to receive extension data. The following example will show how to obtain the DS data from a DomainInfo using the SecDNS extension. Similar to sending command data, you need to create a new object for the extension, this time a SecDnsDomainInfoResponseExtension.

Create a standard DomainInfoResponse:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3** |  | // Create the domain info response  **final DomainInfoResponse domainInfoResponse = new**  **DomainInfoResponse();** |

Create a SecDnsDomainInfoResponseExtension:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3** |  | // Create a SECDNS response extension object  **final SecDnsDomainInfoResponseExtension secDNSExt = new**  **SecDnsDomainInfoResponseExtension();** |

Register the extension to the standard DomainInfoResponse object:

|  |  |  |
| --- | --- | --- |
| **1**  **2** |  | // Register the extension response to the domain info response  **domainInfoResponse.registerExtension(secDNSExt);** |

Execute DomainInfoCommand:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3**  **4** |  | /\* Tell the manager to execute the command. The response includes the  response extension \*/  **manager.execute(new Transaction(new DomainInfoCommand(domainName,**  **password), domainInfoResponse));** |

Obtain the DS data associated with the domain. This will be returned as a list of DSData objects. First check that the extension has been initalised, because if there are no SecDNS extension elements in the return XML, the object will initalise. This only occurs if DNSSEC data is not applied to the domain:

|  |  |  |
| --- | --- | --- |
| **1**  **2**  **3**  **4** |  | **if (secDNSExt.isInitialised()) {**  **final List<DSData> dsDataList =**  **secDNSExt.getInfData().getDsDataList();**  **}** |

# Toolkit Implementation Notes

The Toolkit is comprised of two components, one for communicating with the registry, and the second to map java objects into their XML representation conforming to the EPP specifications. These two components are discussed briefly below.

## Connection and Session Management

The Toolkit facilitates connections to the registry using classes in the *com.ausregistry.jtoolkit2.session* package. Clients obtain one instance of the SessionManager to be shared amongst all client threads that interact with the registry. The SessionManager provides a pool of connections that are automatically created as required. EPP session management is transparent to users of the SessionManager with EPP login commands issued when new connections are established.

Note that the object and extension URIs provided in the EPP login command are sourced from the namespaces declared in the default properties file. Developers can comment out, or add new namespace URIs to have them sent to the registry during login.

Transactions (a command and its response) are executed by calling SessionManager.execute(). The session manager will pick the best available connection and process the transaction using blocking IO. The SessionManager does not manage threads; calling application threads will be used for IO thus executing transactions will block the calling thread.

EPP servers may be configured to close inactive connections. Applications that wish to keep connections alive may call the SessionManager.keepAlive() method to spawn a thread that will poll inactive sessions to prevent dropped connections.

The default implementation of SessionManager gathers data such as the number of commands issued by type, both recently (within command rate limit window) and since start‐up; average response time by session; and response count by result code. This information is exposed via the StatsManager interface and may be used for real-time monitoring of the application.

## XML Marshalling and Unmarshalling

The Toolkit provides Object->XML round-trip serialisation using classes in the \*se packages. All commands extend from the Command class, and all responses extend from the Response class. The implementation uses DOM to construct and serialise XML and DOM and XPath to evaluate XML responses from the server.

While construction of commands leads the caller to provide the minimal set of information, the Toolkit was designed to not pre-empt validations of parameter values such that changes to validation rules would not require new revisions of the Toolkit. It is the responsibility of the calling application to ensure the parameter values are correct and accurate for the target registry. The Toolkit provides a configuration option that turns on outbound schema validation to catch errors before a round-trip to the server. This is particularly useful when developing against the Toolkit.

Applications looking to extend the command/response framework should model their code from extensions provided in the core Toolkit. The com.ausregistry.jtoolkit2.se.secdns package provides an example command extension, and its use is documented in ‘3.3 Using extensions with commands’.

## Logging

The Toolkit supports the following:

A default logging implementation which logs to a configured set of destination files, the specific file depending on the target audience.

The option to define an alternative logging implementation which meets a specified interface. This provides great control over filtering log messages, and how and where to write log records.

There are two levels of logging configuration supported by the Toolkit:

Simple modification of logging behaviour via configuration parameters.

Custom implementation of logging handler classes.

The standard handler implementations log to files. There are four handler classes, one for each class of audience:

Debug handler for Toolkit developers

Maintenance handler for maintainers of the Toolkit

Support handler for messages targeted at the Registry support team

A user handler for messages relevant to the application developers and production support staff (users of the Toolkit).

The system property java.util.logging.config.filespecifies the location of the logging properties file from which logging properties are read. The location should be specified as a fully qualified filename for the best guarantee that the properties file will be found at runtime.

The properties relevant to the standard logging implementation are:

.level={ALL|FINEST|FINER|FINE|INFO|WARNING|SEVERE|NONE} defines the minimum severity of messages that will be logged.

Thus, if .level=FINER, then all messages of severity FINER or greater would be logged (FINER,FINE,INFO,WARNING,SEVERE) unless a more specific rule overrides this parameter (see below).

package-name.audience.level={ALL|FINEST|FINER|FINE| INFO|WARNING|SEVERE|NONE} overrides for the named package and audience any more general specification of logging level.

handler-class.\* FileHandler properties, as described in the FileHandler section of the Java Logging API. As long as the default handler implementations are used, all of the properties specified for FileHandler are effective. Notable properties are pattern and formatter, which control the destination file and format (plain text or XML) respectively.

package-name.audience.handlers should be left at the default values to use the provided handler implementations. The default values are provided in the logging.properties file distributed with the Toolkit.

Alternatively, the user may implement custom handler classes and register those classes using the package-name.audience.handlers parameters. Implementers should familiarise themselves with the Java Logging API [JLOGAPI] and Java Logging Overview [JLOGGUIDE] before deciding on this approach.

# Appendix

## Specifications Implemented

### RFCs

RFC 5730 Extensible Provisioning Protocol (EPP)

RFC 5731 Extensible Provisioning Protocol (EPP) Domain Name Mapping

RFC 5732 Extensible Provisioning Protocol (EPP) Host Mapping

RFC 5733 Extensible Provisioning Protocol (EPP) Contact Mapping

RFC 5734 Extensible Provisioning Protocol (EPP) Transport over TCP

RFC 3915 Domain Registry Grace Period Mapping for the Extensible Provisioning Protocol (EPP)

RFC 4114 E.164 Number Mapping for the Extensible Provisioning Protocol (EPP)

RFC 5910 Domain Name System (DNS) Security Extensions Mapping for the Extensible Provisioning Protocol (EPP)

### Other Extensions

AusRegistry EPP Extensions version 1.0 [AREXT1‐0]

AusRegistry EPP Extensions version 1.0 Domain Name Mapping [ARDOM1‐0]

AusRegistry .au EPP Extensions version 1.01 [AUEXT1‐0]

.ae EPP Extensions version 1.0 [AEEXT1‐0]

AusRegistry .au EPP Extensions version 1.1 [AUEXT1‐1]

AusRegistry .au Domain EPP Extensions version 1.0 [AUDOM1‐0]

Definition of AusRegistry

**AusRegistry** means any or all of the AusRegistry Group of companies, their related entities and their respective officers, employees, contractors or sub-contractors.

**Disclaimer**

This document has been produced by AusRegistry and is only for the information of the particular person to whom it is provided (the Recipient). This document is subject to copyright and may contain privileged and/or confidential information. As such, this document (or any part of it) may not be reproduced, distributed or published without the prior written consent of AusRegistry.

This document has been prepared and presented in good faith based on AusRegistry’s own information and sources which are believed to be reliable. AusRegistry assumes no responsibility for the accuracy, reliability or completeness of the information contained in this document (except to the extent that liability under statute cannot be excluded).

To the extent that AusRegistry may be liable, liability is limited at AusRegistry’s option to replacing, repairing or supplying equivalent goods or paying the cost of replacing, repairing or acquiring equivalent, or, in the case of services, re-supplying or paying the cost of having such re-supplied.

**Confidentiality Notice**

This document contains commercially sensitive information and information that is confidential to AusRegistry. This document is intended solely for the named recipient, and its authorised employees, and legal, financial and accounting representatives (collectively, Authorised Recipients).

The recipients of this document must keep confidential all of the information disclosed in this document, and may only use the information for the purpose specified by AusRegistry for its use. Under no circumstance may this document (or any part of this document) be disclosed, copied or reproduced to any person, other than the Authorised Recipients, without the prior written consent of AusRegistry.

**Trademarks Notice**

The names, trademarks, service marks, logos, and icons of AusRegistry appearing in this document may not be used in any manner by recipients of this document without the express prior written consent of AusRegistry. All rights conferred under law are reserved.

All other trademarks contained within this document remain the property of their respective owners, and are used only to directly describe the products being provided by them or on their behalf. Their use in no way indicates any relationship between AusRegistry and the owners of those other trademarks.

**Pricing Notice**

Any information or pricing provided in this document is subject to change without notice. Whilst AusRegistry has compiled this document in good faith, based on what it believes is accurate and up-to-date information, it is possible that the pricing or other information contained in this document may require amendment due to changing market or other circumstances (including product discontinuation, manufacturer price changes, errors, or insufficient or inaccurate information having been provided by the recipient of this document or others, and other external circumstances). Additional charges may also apply for work that is out of scope.

The pricing in this document is based on AusRegistry standard terms and conditions and is valid for a period of thirty (30) days from the date of this document.