ARI Registry Services

Universal RTK Toolkit—ARI Add-On Guide

Thursday, 4 April 2013

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 | 4/04/2013 | Peter Maurer | Initial Version |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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 implementation of the Universal RTK Toolkit Add-On provided by ARI Registry Services.

Intended Audience

This document is intended for use by the following:

Registrars

Registry Operators

Contents

[Introduction 6](#_Toc352856197)

[Setup 7](#_Toc352856198)

[Environment 7](#_Toc352856199)

[Java 6 7](#_Toc352856200)

[UTF-8 Encoding 7](#_Toc352856201)

[UTC Date 7](#_Toc352856202)

[Configuration 7](#_Toc352856203)

[Properties 7](#_Toc352856204)

[Coding Guide 9](#_Toc352856205)

[DNSSEC Extensions 9](#_Toc352856206)

[DNSSEC Information (Domain Create) 9](#_Toc352856207)

[DNSSEC Information (Domain Update) 11](#_Toc352856208)

[DNSSEC Information (Domain Info) 13](#_Toc352856209)

[IDN Information Extensions 14](#_Toc352856210)

[IDN Information (Domain Create) 14](#_Toc352856211)

[IDN Information (Domain Info) 15](#_Toc352856212)

[Premium Extensions 15](#_Toc352856213)

[Premium Acknowledgement (Domain Create) 16](#_Toc352856214)

[Premium Acknowledgement (Domain Transfer) 17](#_Toc352856215)

[Premium Price Check 18](#_Toc352856216)

[Premium Price Check Response 19](#_Toc352856217)

[Launch Extensions 19](#_Toc352856218)

[Application Create 20](#_Toc352856219)

[Application Update 21](#_Toc352856220)

[Application Info 22](#_Toc352856221)

[Application Delete 23](#_Toc352856222)

[Application Create Response 24](#_Toc352856223)

[Application Info Response Extension 25](#_Toc352856224)

[IDN Variant Extension 26](#_Toc352856225)

[IDN Variants (Domain Info) 26](#_Toc352856226)

[Key Value Extensions 27](#_Toc352856227)

[Key Value (Domain Create) 28](#_Toc352856228)

[Key Value (Domain Update) 29](#_Toc352856229)

[Key Value (Domain Info) 29](#_Toc352856230)

[IDN Variant Extension V1\_1 30](#_Toc352856231)

[Domain Info Response Variant Extension 31](#_Toc352856232)

[Domain Create Response Variant Extension 32](#_Toc352856233)

[Adding and Removing Variants 33](#_Toc352856234)

[Variant Extension 33](#_Toc352856235)

[Variants (Domain Update) 34](#_Toc352856236)

Introduction

ARI’s Universal RTK Toolkit Add-On extends the Universal RTK Toolkit, by providing extensions for:

DNSSEC information in Domain Create, Domain Update and Domain Info operations

IDN information in Domain Create and Domain Info operations, including IDN variants for Domain Info

Key value in Domain Create and Domain Update operations

Setup

Environment

The following environment specifics are required:

Java 6

Java SE 6 (JDK6.0) or later

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.

Confirm the version of Java installed using:

java –version

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 system property:

file.encoding to UTF-8.

This can be done on the command line with the syntax:

java –Dfile.encoding=UTF-8 ...

UTC Date

Date objects must be set to UTC time instead of local time.

Configuration

Properties

The Universal RTK Toolkit needs to be configured to access an ARI Domain Name Registry System (DNRS). This involves setting properties in two properties files provided by ARI’s RTK Toolkit Add-On:

*etc/rtk.properties*

*ssl/ssl.properties*

**rtk.properties**

The property in the *rtk.properties* file is *rtk.transport*. You will need to set this to: *ari.dnrs.rtk.addon.transport.EPPTransportTCPTLS*.

Using this setting also includes a fix to send the correct length of the XML when using Unicode characters.

**ssl.properties**

The properties in the *ssl.properties* file are as follows:

*ssl.keystore.file=  
ssl.truststore.location=  
ssl.truststore.pass=*

These properties identify the trust store to be used. Thus enabling changing the trust store independent of the JVM base trust store. The values of these properties are dependent on the Registrar. Values will be supplied by the Registry operator.

Coding Guide

DNSSEC Extensions

Via DNSSEC extensions, ARI’s Universal RTK Toolkit Add-On allows Registrars to:

Provide DNSSEC information in the Domain Create operation.

Create a Domain Update request which includes DNSSEC information for an ARI backed EPPD and Registry.

Process and identify the DNSSEC information of a Domain Info response that contains the Domain Name DNSSEC details from an ARI backed EPPD.

|  |
| --- |
| Setting DNSSEC extensions as an EPP service extension  To use DNSSEC extensions with your Domain operations you will need to set them as an EPP service extension on your EPPClient object. You must do this before login. You can achieve this with the following code:  eppClient = new EPPClient(eppHostName, eppHostPort, eppClientID, password);  eppClient.setEPPServiceExtensions(new String[] { XMLNamespaces.SEC\_DNS\_NAMESPACE }); |

**Note:** More than one EPP service extension can be set to the EPP client at the time of login.

DNSSEC Information (Domain Create)

You can provide DNSSEC information in the Domain Create operation.

|  |
| --- |
| Performing Domain Create with DNSSEC Information  To perform a Domain Create with DNSSEC information, you will need an instance of the SECDNSCreateCommandExtension class. This class takes in an instance of a DNSSecDataBean using the setCreateData() method:  SECDNSCreateCommandExtension secDNSExt = new SECDNSCreateCommandExtension();  DNSSecDataBean createData = new DNSSecDataBean();  secDNSExt.setCreateData(createData); |

|  |
| --- |
| Creating a Domain via KeyDataBean or DSDataBean classes  You can supply DNSSEC information to a Domain Create request by supplying a KeyDataBean or DSDataBean object.  **Adding KeyDataBean to DNSSecDataBean**  The following is an example of how to create a KeyDataBean object and add it to the DNSSecDataBean:  KeyDataBean keyData = new KeyDataBean();  keyData.setAlgorithm(3);  keyData.setPubKey("AQPJ////4Q==");  keyData.setProtocol(3);  keyData.setFlags(0);  createData.addToKeyData(keyData);  **Adding DSDataBean to DNSSecDataBean**  The following code is an example of how to create DSDataBean and add it to the DNSSecDataBean:  DSDataBean dsData = new DSDataBean();  dsData.setAlgorithm(1);  dsData.setDigest("49FD46E6C4B45C55D4D4D4D4D4D4D4D4D4D4D400");  dsData.setDigestType(1);  dsData.setKeyTag(1);  createData.addToDsData(dsData);  **Adding KeyDataBean to DSDataBean**  You can also add a KeyDataBean object to a DSDataBean object, to supply the Key Data that was used to generate the DS Data:  dsData.setKeyData(keyData); |
| Setting DNSSEC Information extension to the object  Prior to processing your Domain Create request you will need to set the extension to the object. The following code is a basic example of how to do this:  final epp\_DomainCreateReq domainCreateRequest = new epp\_DomainCreateReq();  /\* Complete any standard configuration for your domain create in addition to setting up your DNSSEC data here \*/  final epp\_Extension[] extensions = { secDNSExt };  domainCreateRequest.getCmd().setExtensions(extensions);  EPPDomainCreate domainCreate = new EPPDomainCreate();  domainCreate.setRequestData(domainCreateRequest); |

DNSSEC Information (Domain Update)

You can create a Domain Update request which includes DNSSEC information for an ARI backed EPPD and Registry.

|  |
| --- |
| Performing Domain Update with DNSSEC Information  To perform a Domain Update with DNSSEC information, you will need an instance of the SECDNSUpdateCommandExtension class:  SECDNSUpdateCommandExtension secDNSExt = new SECDNSUpdateCommandExtension(); |
| Adding or removing DNSSEC data  SECDNSUpdateCommandExtension lets you provide add or remove DNSSEC data (DSDataBean or KeyDataBean).  **To add DSDataBean data**  The following example shows how to add DSDataBean data to a Domain Update:  DSDataBean dsDataToAdd = new DSDataBean();  dsDataToAdd.setAlgorithm(1);  dsDataToAdd.setDigest("49FD46E6C4B45C55D4D4D4D4D4D4D4D4D4D4D411");  dsDataToAdd.setDigestType(1);  dsDataToAdd.setKeyTag(1);  DNSSecDataBean addData = new DNSSecDataBean();  addData.addToDsData(dsDataToAdd);  secDNSExt.setAddData(addData);  **To remove DSDataBean data**  The following example shows how to remove DSDataBean data from a domain using Domain Update.  First you need to create your DSDataBean object, supplying the algorithm, digest, digest type and key tag (and optionally the related Key data) of the DNSSEC data that you wish to remove:  DSDataBean dsData = new DSDataBean();  dsData.setAlgorithm(1);  dsData.setDigest("49FD46E6C4B45C55D4D4D4D4D4D4D4D4D4D4D411");  dsData.setDigestType(1);  dsData.setKeyTag(1);  Next, you need to create a RemoveElementBean, and add the DSDataBean to that object. Then add RemoveElementBean to the SECDNSUpdateCommandExtension:  RemoveElementBean removeElement = new RemoveElementBean();  removeElement.addToDsData(dsData);  secDNSExtension.setRemData(removeElement);  **Adding and removing KeyDataBean data**  Adding and removing KeyDataBean data is a very similar process to adding and removing DSDataBean data. Instead of using a DSDataBean, simply add a KeyDataBean to the DNSSecDataBean.  See ‘DNSSEC Information (Domain Create)’ above, for an example of using a KeyDataBean.  **Removing all DSDataBean and KeyDataBean data**  You can also remove all DSDataBean data and KeyDataBean data related to a domain using a Domain Update. The following is an example of this:  RemoveElementBean remElement = new RemoveElementBean();  remElement.setRemoveAll(true);  secDNSExtension.setRemData(remElement); |
| Changing DNSSEC data maxSigLife value  SECDNSUpdateCommandExtension also lets you change the maxSigLife value for your DNSSEC data.  The following example shows how to change the maxSigLife of a domain's DNSSEC data:  MaxSigLifeBean maxSigLife = new MaxSigLifeBean(10000);  ChangeElementBean chgElement = new ChangeElementBean();  chgElement.setMaxSigLife(maxSigLife);  secDNSExtension.setChgData(chgElement); |
| Setting DNSSEC Information extension to the command object  Prior to processing your Domain Update request you will need to set the extension to the object. The following code is a basic example of how to do this:  final epp\_DomainUpdateReq domainUpdateRequest = new epp\_DomainUpdateReq();  /\*Complete any standard configuration for your domain update in addition to setting up your DNSSEC data here \*/  final epp\_Extension[] extensions = { secDNSExt };  domainUpdateRequest.getCmd().setExtensions(extensions);  EPPDomainUpdate domainUpdate = new EPPDomainUpdate();  domainUpdate.setRequestData(domainUpdateRequest); |

DNSSEC Information (Domain Info)

You can process and identify the DNSSEC information of a Domain Info response that contains the Domain Name DNSSEC details from an ARI backed EPPD.

|  |
| --- |
| Retrieving DNSSEC Information from Domain Info  To retrieve the DNSSEC information from Domain Info, you will need to first perform a Domain Info operation:  epp\_DomainInfoReq domainInfoRequest = new epp\_DomainInfoReq();  epp\_Command commandData = new epp\_Command();  commandData.setClientTrid("cltrid");  domainInfoRequest.setCmd(commandData);  domainInfoRequest.setName("domain.name");  EPPDomainInfo domainInfo = new EPPDomainInfo();  domainInfo.setRequestData(domainInfoRequest);  domainInfo = (EPPDomainInfo) eppClient.processAction(domainInfo);  Having completed the Domain Info, you can get the extension XML from the response:  final epp\_DomainInfoRsp domainInfoResponse = domainInfo.getResponseData();  final epp\_Response infoResponse = domainInfoResponse.getRsp();  String[] infoExtensionStrings = infoResponse.getExtensionStrings();  Then you can create the SECDNSInfoResponseExtension object to parse the XML:  SECDNSInfoResponseExtension secDNSInfoResponse = new SECDNSInfoResponseExtension();  secDNSInfoResponse.fromXML(infoExtensionStrings[0]);  You can then access the DS data, Key data and maxSigLife (be aware that these will be 'null' if the domain has no associated data):  List<DSDataBean> dsDataList = secDNSInfoResponse.getInfData().getDsDataList();  List<KeyDataBean> keyDataList = secDNSInfoResponse.getInfData().getKeyDataList();  MaxSigLifeBean maxSigLife = secDNSInfoResponse.getInfData().getMaxSigLife(); |

IDN Information Extensions

Via IDN Information extensions, ARI’s Universal RTK Toolkit Add-On allows Registrars to:

Perform a Domain Create with IDN information

Obtain the user form, language and canonical form of an IDN when executing Domain Info

|  |
| --- |
| Setting IDN Information extensions as an EPP service extension  To use the IDN Information extensions with your Domain operations you will need to set them as an EPP service extension on your EPPClient object. You must do this before login. You can achieve this with the following code:  eppClient = new EPPClient(eppHostName, eppHostPort, eppClientID, password);  eppClient.setEPPServiceExtensions(new String[] { XMLNamespaces.IDNA\_NAMESPACE }); |

IDN Information (Domain Create)

You can perform a Domain Create with IDN information.

|  |
| --- |
| Performing Domain Create with IDN information  First you will need an instance of the DomainIdnCommandExtension class:  DomainIdnCommandExtension idnExt = new DomainIdnCommandExtension();  Then, the DomainIdnCommandExtension class requires a user form and a language to be set:  idnExt.setUserForm(userForm);  idnExt.setLanguage(language);  Prior to processing your Domain Create request you will need to set the extension to the object. The following code is a basic example of how to do this:  final epp\_DomainCreateReq domainCreateRequest = new epp\_DomainCreateReq();  /\*Complete any standard configuration for your domain create in addition to setting up your key value data here \*/  final epp\_Extension[] extensions = { idnExt };  domainCreateRequest.getCmd().setExtensions(extensions);  EPPDomainCreate domainCreate = new EPPDomainCreate();  domainCreate.setRequestData(domainCreateRequest); |

IDN Information (Domain Info)

You can also obtain IDN information with a Domain Info operation.

|  |
| --- |
| Performing Domain Info to obtain IDN information  The Domain Info operation utilises the same extension as Domain Create for obtaining the extension response information. The following is an example for obtaining the IDN response information. First you need to obtain the extension XML:  String[] extensionStrings = response.getExtensionStrings();  Then, you need to parse the XML using a DomainIdnCommandExtension object:  DomainIdnCommandExtension domainIdnCommandExtension = new DomainIdnCommandExtension();  domainIdnCommandExtension.fromXML(extensionStrings[0]);  Finally, you can obtain the user form, language and canonical form from the command extension:  domainIdnCommandExtension.getUserForm();  domainIdnCommandExtension.getLanguage();  domainIdnCommandExtension.getCanonicalForm(); |

Premium Extensions

Via premium extensions, ARI’s Universal RTK Toolkit Add-On allows Registrars to:

Perform a Domain Create with acknowledgement that the domain name is premium. Optionally specify the premium price for the domain.

Perform a Domain Transfer request with acknowledgement that the domain name is premium. Optionally specify the premium price for the domain.

Perform a Domain Check to determine if a domain name is premium and the price for premium domains.

Check if a domain is premium and get its premium price from the aforementioned Domain Check response.

|  |
| --- |
| Setting Premium extensions as an EPP service extension  To use the Premium extensions with your Domain operations you will need to set them as an EPP service extension on your EPPClient object. You must do this before login. You can achieve this with the following code:  eppClient = new EPPClient(eppHostName, eppHostPort, eppClientID, password);  eppClient.setEPPServiceExtensions(new String[] { XMLNamespaces.PREMIUM\_NAMESPACE }); |

Premium Acknowledgement (Domain Create)

You can perform a Domain Create with premium acknowledgement.

|  |
| --- |
| Performing Domain Create with Premium Acknowledgement  First you will need an instance of the DomainPremiumCommandExtension class:  DomainPremiumCommandExtension premiumExt =  new DomainPremiumCommandExtension(“create”, createPrice, renewPrice);  The DomainPremiumCommandExtension class optionally requires a create price and a renew price:  BigDecimal createPrice = BigDecimal.valueOf(100.00);  BigDecimal renewPrice = BigDecimal.valueOf(200.00);  Since these values are optional, they can be specified as ‘null’. Prior to processing your Domain Create request you will need to set the extension to the object. The following code is a basic example of how to do this:  final epp\_DomainCreateReq domainCreateRequest = new epp\_DomainCreateReq();  /\*Complete any standard configuration for your domain create in addition to setting up your premium extension here \*/  final epp\_Extension[] extensions = { premiumExt };  domainCreateRequest.getCmd().setExtensions(extensions);  EPPDomainCreate domainCreate = new EPPDomainCreate();  domainCreate.setRequestData(domainCreateRequest); |

Premium Acknowledgement (Domain Transfer)

You can perform a Domain Transfer with premium acknowledgement.

|  |
| --- |
| Performing Domain Transfer with Premium Acknowledgement  First you will need an instance of the DomainPremiumCommandExtension class:  DomainPremiumCommandExtension premiumExt =  new DomainPremiumCommandExtension(“transfer”, createPrice, renewPrice);  The DomainPremiumCommandExtension class optionally requires a create price and a renew price:  BigDecimal createPrice = BigDecimal.valueOf(100.00);  BigDecimal renewPrice = BigDecimal.valueOf(200.00);  Since these values are optional, they can be specified as ‘null’. Prior to processing your Domain Transfer request you will need to set the extension to the object. The following code is a basic example of how to do this:  final epp\_DomainTransferReq domainTransferRequest =  new epp\_DomainTransferReq();  /\*Complete any standard configuration for your domain transfer in addition to setting up your premium extension here \*/  final epp\_Extension[] extensions = { premiumExt };  domainTransferRequest.getCmd().setExtensions(extensions);  EPPDomainTransfer domainTransfer = new EPPDomainTransfer();  domainTransfer.setRequestData(domainTransferRequest); |

Premium Price Check

You can check if a domain is premium and if so, determine its create and renew price.

|  |
| --- |
| Performing Domain Check with Premium Extension  First you will need an instance of the DomainPremiumCommandExtension class:  DomainPremiumCommandExtension premiumExt =  new DomainPremiumCommandExtension(“check”);  Prior to processing your Domain Transfer request you will need to set the extension to the object. The following code is a basic example of how to do this:  final epp\_DomainCheckReq domainCheckRequest =  new epp\_DomainCheckReq();  /\*Complete any standard configuration for your domain transfer in addition to setting up your premium extension here \*/  final epp\_Extension[] extensions = { premiumExt };  domainCheckRequest.getCmd().setExtensions(extensions);  EPPDomainCheck domainCheck = new EPPDomainCheck();  domainCheck.setRequestData(domainCheckRequest); |

Premium Price Check Response

You can check if a domain is premium and get the premium price info from aforementioned Domain Check response.

|  |
| --- |
| Requesting Domain Check with Premium Extension  Described in previous section. |
| Processing Domain Check Response with Premium Extension  Process a Domain Check operation with the Domain Check request:  EPPDomainCheck domainCheck = new EPPDomainCheck();  domainCheck.setRequestData(domainCheckRequest);  domainCheck = (EPPDomainCheck) eppClient.processAction(domainCheck);  Once the Domain Info request has processed, you can retrieve response XML:  final String responseXML = domainCheck.getXML();  The response has a flag to indicate if the domain checked is premium. In addition it has create and renew prices for the domain if it is premium. These can be retrieved using DomainPremiumCheckResponse object:  DomainPremiumCheckResponse premiumCheckResponse =  new DomainPremiumCheckResponse();  premiumCheckResponse.fromXML(responseXML);  You can then retrieve the domain premium information using either the index of domain in request or the domain name:  boolean isPremium = premiumCheckResponse.isPremium(domainName);  boolean isPremium = premiumCheckResponse.isPremium(index);  BigDecimal createPrice = premiumCheckResponse.getCreatePrice(domainName);  BigDecimal renewPrice = premiumCheckResponse.getRenewPrice(domainName);  BigDecimal createPrice = premiumCheckResponse.getCreatePrice(index);  BigDecimal renewPrice = premiumCheckResponse.getRenewPrice(index); |

Launch Extensions

Via launch extensions, ARI’s Universal RTK Toolkit Add-On allows Registrars to:

Create a sunrise application within the sunrise period.

Update an already existing sunrise application within the sunrise period in which the application was created.

Perform an info to identify the progress and details of a sunrise application.

Cancel a sunrise application within the sunrise period so that the domain does not get allocated.

Allow access to create data for a domain as provided in an EPP domain create response compliant with RFC5730 and RFC5731. Such a service element is sent by a compliant EPP server in response to a valid domain create command with a domain create application extension.

Allow access to info data for a domain as provided in an EPP domain info response compliant with RFC5731. Such a service element is sent by a compliant EPP server in response to a valid domain info command with a domain info application extension.

|  |
| --- |
| Setting Launch extensions as an EPP service extension  To use the Launch extensions with your Domain operations you will need to set them as an EPP service extension on your EPPClient object. You must do this before login. You can achieve this with the following code:  eppClient = new EPPClient(eppHostName, eppHostPort, eppClientID, password);  eppClient.setEPPServiceExtensions(new String[] { XMLNamespaces.LAUNCH\_NAMESPACE }); |

Application Create

You can perform an Application create with an extension for Domain Create wherein you specify the phase.

|  |
| --- |
| Performing Domain Application Create  First you will need an instance of the DomainApplicationCommandExtension class:  DomainApplicationCommandExtension applicationExt =  new DomainApplicationCommandExtension(“create”);  The DomainPremiumCommandExtension class requires the phase type:  applicationExt.setPhaseType(“sunrise”);  Prior to processing your Domain Create request you will need to set the extension to the object. The following code is a basic example of how to do this:  final epp\_DomainCreateReq domainCreateRequest =  new epp\_DomainCreateReq();  /\*Complete any standard configuration for your domain create in addition to setting up your premium extension here \*/  final epp\_Extension[] extensions = { applicationExt };  domainCreateRequest.getCmd().setExtensions(extensions);  EPPDomainCreate domainCreate = new EPPDomainCreate();  domainCreate.setRequestData(domainCreateRequest); |

Application Update

You can perform an Application update with an extension for Domain Update wherein you specify the application-id.

|  |
| --- |
| Performing Domain Application Update  First you will need an instance of the DomainApplicationCommandExtension class:  DomainApplicationCommandExtension applicationExt =  new DomainApplicationCommandExtension(“update”);  The DomainPremiumCommandExtension class requires application id:  applicationExt.setApplicationId(“sunrise-application-id”);  Prior to processing your Domain Update request you will need to set the extension to the object. The following code is a basic example of how to do this:  final epp\_DomainUpdateReq domainUpdateRequest =  new epp\_DomainUpdateReq();  /\*Complete any standard configuration for your domain update in addition to setting up your premium extension here \*/  final epp\_Extension[] extensions = { applicationExt };  domainUpdateRequest.getCmd().setExtensions(extensions);  EPPDomainUpdate domainUpdate = new EPPDomainUpdate();  domainUpdate.setRequestData(domainUpdateRequest); |

Application Info

You can perform an Application info with an extension for Domain Info wherein you specify the application-id.

|  |
| --- |
| Performing Domain Application Info  First you will need an instance of the DomainApplicationCommandExtension class:  DomainApplicationCommandExtension applicationExt =  new DomainApplicationCommandExtension(“info”);  The DomainPremiumCommandExtension class requires application id:  applicationExt.setApplicationId(“sunrise-application-id”);  Prior to processing your Domain info request you will need to set the extension to the object. The following code is a basic example of how to do this:  final epp\_DomainInfoReq domainInfoRequest =  new epp\_DomainInfoReq();  /\*Complete any standard configuration for your domain info in addition to setting up your premium extension here \*/  final epp\_Extension[] extensions = { applicationExt };  domainInfoRequest.getCmd().setExtensions(extensions);  EPPDomainInfo domainInfo = new EPPDomainInfo();  domainInfo.setRequestData(domainInfoRequest); |

Application Delete

You can perform an Application delete with an extension for Domain Delete wherein you specify the application-id.

|  |
| --- |
| Performing Domain Application delete  First you will need an instance of the DomainApplicationCommandExtension class:  DomainApplicationCommandExtension applicationExt =  new DomainApplicationCommandExtension(“delete”);  The DomainPremiumCommandExtension class requires application id:  applicationExt.setApplicationId(“sunrise-application-id”);  Prior to processing your Domain delete request you will need to set the extension to the object. The following code is a basic example of how to do this:  final epp\_DomainDeleteReq domainDeleteRequest =  new epp\_DomainDeleteReq();  /\*Complete any standard configuration for your domain delete in addition to setting up your premium extension here \*/  final epp\_Extension[] extensions = { applicationExt };  domainDeleteRequest.getCmd().setExtensions(extensions);  EPPDomainDelete domainDelete = new EPPDomainDelete();  domainDelete.setRequestData(domainDeleteRequest); |

Application Create Response

You can determine the created date, name and application-id from a Domain Create response.

|  |
| --- |
| Requesting Domain Application Create with Launch Extension  Described earlier. |
| Processing Domain Application Create Response with Launch Extension  Process a Domain Create operation with the Domain Create request:  EPPDomainCreate domainCreate = new EPPDomainCreate();  domainCreate.setRequestData(domainCreateRequest);  domainCreate = (EPPDomainCreate) eppClient.processAction(domainCreate);  Once the Domain Create request has processed, you can retrieve response XML:  final String responseXML = domainCreate.getXML();  The response has create date, domain name and application-id. These can be retrieved using DomainCreateApplicationResponse object:  DomainCreateApplicationResponse launchCreateResponse =  new DomainCreateApplicationResponse ();  launchCreateResponse.fromXML(responseXML);  You can then retrieve the create date, domain name and application-id:  Calendar createDate = launchCreateResponse.getCreateDate();  String name = launchCreateResponse.getName();  String applicationId = launchCreateResponse.getApplicationId() |

Application Info Response Extension

You can determine the application-id, phase, statuses and if initialised from a Domain Info response extension.

|  |
| --- |
| Requesting Domain Application Info with Launch Extension  Described earlier. |
| Processing Domain Application Info Response Extension  Process a Domain Info operation with the Domain Info request:  EPPDomainInfo domainInfo = new EPPDomainInfo();  domainInfo.setRequestData(domainInfoRequest);  domainInfo = (EPPDomainInfo) eppClient.processAction(domainInfo);  Having completed the Domain Info, you can get the extension XML from the response:  final epp\_DomainInfoRsp domainInfoResponse = domainInfo.getResponseData();  final epp\_Response infoResponse = domainInfoResponse.getRsp();  String[] infoExtensionStrings = infoResponse.getExtensionStrings();  The response has application-id, phase, statuses and if initialised. These can be retrieved using DomainInfoApplicationResponseExtension object:  DomainInfoApplicationResponseExtension launchInfoResponse =  new DomainInfoApplicationResponseExtension();  launchInfoResponse.fromXML(infoExtensionStrings[0]);  You can then retrieve the application id, phase, statuses and if initialised:  List<String> statuses= launchInfoResponse.getStatuses();  String name = launchInfoResponse.getPhase();  Boolean isInitialized = launchInfoResponse.isInitialised();  String applicationId = launchInfoResponse.getApplicationId() |

IDN Variant Extension

Via the IDN Variant extension, ARI’s Universal RTK Toolkit Add-On allows Registrars to:

Retrieve variant information in a response to a Domain Info request (if the registered IDN domain has variants)

|  |
| --- |
| Setting the IDN Variant extension as an EPP service extension  To use the IDN Variant extension with your Domain operations you will need to set it as an EPP service extension on your EPPClient object. You must do this before login. You can achieve this with the following code:  eppClient = new EPPClient(eppHostName, eppHostPort, eppClientID, password);  eppClient.setEPPServiceExtensions(new String[] {XMLNamespaces.VARIANT\_NAMESPACE}); |

IDN Variants (Domain Info)

You can view the variants of a registered IDN in an ARI Registry in response to Domain Info requests.

|  |
| --- |
| Requesting Domain Info  First you need to create a normal domain info request:  epp\_DomainInfoReq domainInfoRequest = new epp\_DomainInfoReq();  Then set command data and input the domain DNS form to the request:  epp\_Command commandData = new epp\_Command();  commandData.setClientTrid("cltrid");  domainInfoRequest.setCmd(commandData);  domainInfoRequest.setName("domain.name"); |

|  |
| --- |
| Processing Domain Info  Process a Domain Info operation with the Domain Info request:  EPPDomainInfo domainInfo = new EPPDomainInfo();  domainInfo.setRequestData(domainInfoRequest);  domainInfo = (EPPDomainInfo) eppClient.processAction(domainInfo);  Once the Domain Info request has processed, you can retrieve response XML:  final epp\_DomainInfoRsp domain\_info\_response = domainInfo.getResponseData();  final epp\_Response response = domain\_info\_response.getRsp();  If the response XML contains any extension XML, you can also retrieve it:  final String[] extensionStrings = response.getExtensionStrings();  If the domain has any variants, the variant information is returned in the extension of the Domain Info response. You can parse this variant extension from the response XML, by using a DomainVariantCommandExtension object:  DomainVariantCommandExtension variantCommandExtension = new DomainVariantCommandExtension();  variantCommandExtension.fromXML(extensionStrings[0]);  You can then retrieve the variant information as a list of DomainVariantBean objects:  final List<DomainVariantBean> variantList = variantCommandExtension.getInfoVariantList();  The DomainVariantBean object holds the DNS form and user form of the variant. You can retrieve these values from the DomainVariantBean objects:  domainVariantBean.getName();  domainVariantBean.getUserForm(); |

Key Value Extensions

Via Key Value extensions, ARI’s Universal RTK Toolkit Add-On allows Registrars to:

Provide Key Value lists in Domain Create and Domain Update requests (the Domain Update XML has been extended to allow key value pairs).

Retrieve Key Value pairs of a domain using Domain Info requests.

|  |
| --- |
| Setting Key Value extensions as an EPP service extension  To use Key Value extensions with your Domain operations you will need to set them as an EPP service extension on your EPPClient object. You must do this before login. You can achieve this with the following code:  eppClient = new EPPClient(eppHostName, eppHostPort, eppClientID, password);  eppClient.setEPPServiceExtensions(new String[] { XMLNamespaces.KVLIST\_NAMESPACE }); |

Key Value (Domain Create)

You can provide Key Value lists in Domain Create requests and Domain Update requests.

|  |
| --- |
| Using Domain Create with Key Value extension  To use Key Value data with Domain Create you need an instance of the DomainKVCommandExtension class. You will need to instantiate this with the name of the operation as a parameter:  final DomainKVCommandExtension kvExtension = new DomainKVCommandExtension("create"); |
| Setting Key Value extension to the object  Prior to processing your Domain Create request you will need to set the Key Value extension to the object. The following code is a basic example of how to do this:  final epp\_DomainCreateReq domainCreateRequest = new epp\_DomainCreateReq();  /\*Complete any standard configuration for your domain create in addition to setting up your key value data here \*/  final epp\_Extension[] extensions = { kvExtension };  domainCreateRequest.getCmd().setExtensions(extensions);  EPPDomainCreate domainCreate = new EPPDomainCreate();  domainCreate.setRequestData(domainCreateRequest); |
| Setting Key Value pair data  Key Value data needs to be set on the DomainKVCommandExtension object. This is achieved through the addKeyValuePairToList method. This takes the list name and key with its value as a parameter.  The list name needs to be specified each time a key and value is added. The following code is an example of how to add the pairs {eligibility / true}, {policyType / new} and {registrantName / Tester} to a list with the name AU:  kvExtension.addKeyValuePairToList("AU", "eligibility", "true");  kvExtension.addKeyValuePairToList("AU", "policyType", "1");  kvExtension.addKeyValuePairToList("AU", "registrantName", "Tester"); |

**Note:** Multiple Key Value lists with different names can be set on the same extension using the same logic.

Key Value (Domain Update)

You can provide Key Value lists in Domain Update requests.

|  |
| --- |
| Providing Key Value lists  The following code shows how you can provide Key Value lists in Domain Update requests.  final DomainKVCommandExtension kvExtension = new DomainKVCommandExtension("update"); |

Key Value (Domain Info)

You can retrieve the key value pairs of a domain by using a Domain Info request.

|  |
| --- |
| Creating a Domain Info request  First create the Domain Info request:  epp\_DomainInfoReq domainInfoRequest = new epp\_DomainInfoReq();  Set command data and input domain name into the request:  epp\_Command commandData = new epp\_Command();  commandData.setClientTrid("cltrid");  domainInfoRequest.setCmd(commandData);  domainInfoRequest.setName("domain.name"); |

|  |
| --- |
| Processing a Domain Info operation with a Domain Info request  Process a Domain Info operation with the Domain Info request:  EPPDomainInfo domainInfo = new EPPDomainInfo();  domainInfo.setRequestData(domainInfoRequest);  domainInfo = (EPPDomainInfo) eppClient.processAction(domainInfo);  Once the domain info request has been processed, you can retrieve response XML.  final epp\_DomainInfoRsp domain\_info\_response = domainInfo.getResponseData();  final epp\_Response response = domain\_info\_response.getRsp();  If the response XML contains any extension XML, you can retrieve it:  final String[] extensionStrings = response.getExtensionStrings();  If the domain has any Key Value pairs, the Key Value list is returned in the extension of the Domain Info response. You can parse this key value extension from the response XML, using a DomainKVCommandExtension object as follows:  final DomainKVCommandExtension kvExtension = new DomainKVCommandExtension("info");  kvExtension.fromXML(extensionStrings[0]);  You can then retrieve Key Value Lists as list names mapped to a list of DomainKeyValueBean objects:  final HashMap<String, ArrayList<DomainKeyValueBean>> keyValueLists = kvExtension.getKeyValueLists();  You can then iterate through this collection to obtain the Key Value list names and the Key Value list items:  final List<DomainKeyValueBean> keyValueBeanList = keyValueLists.get("list- name");  final DomainKeyValueBean keyValueBean1 = keyValueBeanList.get(0);  The DomainKeyValueBean object holds the key and value. You can retrieve these values from the DomainKeyValueBean objects:  String key = keyValueBean1.getKey();  String value = keyValueBean1.getValue(); |

IDN Variant Extension V1\_1

Via Variant extensions, ARI’s Universal RTK Toolkit Add-On allows Registrars to:

Access Create and Info data for a domain as provided in an EPP Domain Create/Info response. Such a service element is sent by an EPP server in response to a valid Domain Create/Info command as implemented by the domain create and domain info, optionally with the variant extension.

Add and Remove variants.

|  |
| --- |
| Setting the IDN Variant extension as an EPP service extension  To use the IDN Variant extension with your Domain operations you will need to set it as an EPP service extension on your EPPClient object. You must do this before login. You can achieve this with the following code:  eppClient = new EPPClient(eppHostName, eppHostPort, eppClientID, password);  eppClient.setEPPServiceExtensions(new String[] {XMLNamespaces.VARIANT\_V1\_1\_NAMESPACE}); |

Domain Info Response Variant Extension

You can determine the variants using the Domain Info response variant extension.

|  |
| --- |
| Processing Domain Info Variant Response Extension  Process a Domain Info operation with the Domain Info request:  EPPDomainInfo domainInfo = new EPPDomainInfo();  domainInfo.setRequestData(domainInfoRequest);  domainInfo = (EPPDomainInfo) eppClient.processAction(domainInfo);  Having completed the Domain Info, you can get the extension XML from the response:  final epp\_DomainInfoRsp domainInfoResponse = domainInfo.getResponseData();  final epp\_Response infoResponse = domainInfoResponse.getRsp();  String[] infoExtensionStrings = infoResponse.getExtensionStrings();  The response has list of variants. These can be retrieved using DomainVariantResponseExtensionV1\_1 object:  DomainVariantResponseExtensionV1\_1 variantInfoResponse =  new DomainVariantResponseExtensionV1\_1(“info”);  variantInfoResponse.fromXML(infoExtensionStrings[0]);  You can then retrieve the variants:  List<IdnDomainVariant> variants = variantInfoResponse.getVariants();  The variant can be fetched from IdnDomainVariant as follows:  String variant = variants.get(0).getName(); |

Domain Create Response Variant Extension

You can determine the variants using the Domain Create response variant extension.

|  |
| --- |
| Processing Domain Create Variant Response Extension  Process a Domain Create operation with the Domain Info request:  EPPDomainCreate domainCreate = new EPPDomainCreate();  domainCreate.setRequestData(domainCreateRequest);  domainCreate = (EPPDomainCreate) eppClient.processAction(domainCreate);  Having completed the Domain create, you can get the extension XML from the response:  epp\_DomainCreateRsp domainCreateResponse = domainCreate.getResponseData();  final epp\_Response createResponse = domainCreateResponse.getRsp();  String[] createExtensionStrings = createResponse.getExtensionStrings();  The response has list of variants. These can be retrieved using DomainVariantResponseExtensionV1\_1 object:  DomainVariantResponseExtensionV1\_1 variantCreateResponse =  new DomainVariantResponseExtensionV1\_1(“create”);  variantInfoResponse.fromXML(createExtensionStrings[0]);  You can then retrieve the variants:  List<IdnDomainVariant> variants = variantCreateResponse.getVariants();  The variant can be fetched from IdnDomainVariant as follows:  String variant = variants.get(0).getName(); |

Adding and Removing Variants

You can perform a domain update with IDN variant extension to add and remove variants.

|  |
| --- |
| Performing Domain Update with Add/Rem variants  First you will need an instance of the DomainUpdateVariantCommandExtensionV1\_1 class:  DomainUpdateVariantCommandExtensionV1\_1 variantExt =  new DomainUpdateVariantCommandExtensionV1\_1();  The DomainPremiumCommandExtension class variants to add and remove:  applicationExt.addVariant(new IdnDomainVariant(“xn—4xa3.apple”), new IdnDomainVariant(“xn—4xu3.apple”), new IdnDomainVariant(“xn—4xr3.apple”));  applicationExt.removeVariant(new IdnDomainVariant(“xn—4xa1.apple”), new IdnDomainVariant(“xn—6xa3.apple”), new IdnDomainVariant(“xn—7xa3.apple”));  Prior to processing your Domain update request you will need to set the extension to the object. The following code is a basic example of how to do this:  final epp\_DomainUpdateReq domainUpdateRequest =  new epp\_DomainUpdateReq();  /\*Complete any standard configuration for your domain update in addition to setting up your variant extension here \*/  final epp\_Extension[] extensions = { applicationExt };  domainUpdateRequest.getCmd().setExtensions(extensions);  EPPDomainUpdate domainUpdate = new EPPDomainUpdate();  domainUpdate.setRequestData(domainUpdateRequest); |

Variant Extension

Via Variant extensions, ARI’s Universal RTK Toolkit Add-On allows Registrars to:

Create a Domain Update request to add or remove variants of a registered Domain Name for an ARI backed EPPD and Registry.

|  |
| --- |
| Setting the Variant extension as an EPP service extension  To use the Variant extension with your Domain operations you will need to set it as an EPP service extension on your EPPClient object. You must do this before login. You can achieve this with the following code:  eppClient = new EPPClient(eppHostName, eppHostPort, eppClientID, password);  eppClient.setEPPServiceExtensions(new String[] {XMLNamespaces.VARIANT\_NAMESPACE}); |

Variants (Domain Update)

You can perform a Domain Update request to add or remove variants of a Domain Name.

|  |
| --- |
| Using Domain Update with variant extension  To perform a Domain Update with variant extension, you will need an instance of the DomainVariantCommandExtension class:  final DomainKVCommandExtension kvExtension = new DomainKVCommandExtension("create"); |
| Adding or removing variants from a domain  DomainVariantCommandExtension lets you provide for variants to be added to or removed from a domain.  **To add variants**  You can set the DNS form and user form of a variant to be added to a domain, to a DomainVariantCommandExtension object as follows:  domainVariantExtension.addToVariantsToAddList(variantToAddDNSForm, variantToAddUserform);  **To remove variants**  You can set the DNS form and user form of a variant to be removed from a domain, to a DomainVariantCommandExtension object as follows:  domainVariantExtension.addToVariantsToRemoveList(variantToRemoveDNSForm, variantToRemoveUserform); |
| Creating a Domain Update request  You can create a Domain Update request and set the variant extension to the request as follows:  epp\_DomainUpdateReq domainUpdateRequest = new epp\_DomainUpdateReq();  domainUpdateRequest.setName(domainName);  domainUpdateRequest.setCmd(commandData);  // Set the extension to the Domain update request  final epp\_Extension[] extensions = {domainVariantExtension};  domainUpdateRequest.getCmd().setExtensions(extensions);  Then process the domain update request to send the request with the variant extension:  EPPDomainUpdate domainUpdate = new EPPDomainUpdate();  domainUpdate.setRequestData(domainUpdateRequest);  domainUpdate = (EPPDomainUpdate) eppClient.processAction(domainUpdate); |

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.