



# Shared Registry System

# Client Application Design

Version: 1.29  
Author: Ori Hecht  
Date: 30<sup>th</sup> July 2002

# SRS Implementation Project

# 1 Table of Contents

1	Table of Contents	2
2	Introduction	4
2.1	Background	4
2.2	Reference Documents	4
2.3	Scope	4
3	Overview	6
4	High Level Design	7
4.1	Modularity	7
4.2	Security	8
4.3	Logging	9
4.4	Error Handling	9
4.5	Portability	9
5	Detailed Design	10
5.1	Modules	10
5.1.1	SRS Server Interface	10
5.1.2	Logger Module	26
5.1.3	Log Message Generator Module	26
5.1.4	Client Native Data Module	26
5.1.5	Error Repository Module	32
5.1.6	Client Communications Module	34
5.1.7	Client XML Translation Module	36
5.1.8	Client Request Manager Module	38
5.1.9	Client User Interface Module	38
5.3	Operation Sequences	39
5.3.1	Success Sequences	39
5.3.2	Error Sequences	43
6	Glossary	52
7	Appendices	53
7.1	Appendix A – Request validate() Errors	53
7.2	Appendix B – Transaction validate() Errors	53
7.2.1	Basic Data Validation Errors	53
7.2.2	Common Transaction Errors	53
7.2.3	DomainCreate validation Errors	53
7.2.4	DomainUpdate validation Errors	54
7.2.5	DomainDetailsQry validation Errors	55
7.2.6	RegistrarCreate validation Errors	55
7.2.7	RegistrarUpdate validation Errors	55
7.2.8	RegistrarDetailsQry validation Errors	55
7.2.9	RegistrarAccountQry validation Errors	55
7.2.10	GetMessages validation Errors	55
7.2.11	WhoIs validation Errors	55
7.2.12	ActionDetailsQry validation Errors	55
7.3	Appendix C –Error Severity Levels	56
7.3.1	Client Unavailable (high)	56

7.3.2 Persistent Failure Affecting Multiple Transaction Types	56
7.3.3 Persistent Failure Affecting a Single Transaction Type	56
7.3.4 Resource Unavailable	56
7.3.5 Business Logic Error (low) (input dependant errors)	56
7.4 Appendix D – Coding Guidelines	57
7.4.1 Compilation directives	57
7.4.2 Naming conventions	57
7.4.3 Documentation	57

# 2 Introduction

## 2.1 Background

InternetNZ contracted Catalyst IT Ltd. to design and develop the Shared Registry System (SRS). As part of that contract, a Registrar Implementation Kit (RIK) was specified. The kit includes a technical specification that describes the SRS connection architecture and protocol to the level of detail necessary for registrars to develop their own interface software from the ground up, and a full set of open source software modules that registrars can choose to employ, either entirely or partially, as an interface to the SRS.

## 2.2 Reference Documents

### InternetNZ documents

[1]Schedule Four, Shared Registry System, Registrar Implementation Package - Stage One; V 1.0; 25 May 2002.

[2]The Framework and Business Rules for the Domain Name Shared Registry for .nz; V 2.0; 7 June 2002.

### Project documents

[3]Shared Registry Systems Architecture; V 1.2; 21 May 2002.

[4]Shared Registry System Detailed Requirements Specification; V 2.1; 20 June 2002.

[5]protocol.dtd; V 1.27; 20 June 2002.

### External documents

[6]Design Patterns Elements of Reusable Object-Oriented Software, by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides.

[7]<http://rhumba.pair.com/ben/perl/openpgp>; V1.0; 26 February 2002.

[8]<http://xml.apache.org/xerces-p/index.html>; V1.7.0-1; as of 4 June 2002

[9]<http://www.perldoc.com/perl5.6.1/lib/LWP/UserAgent.html>; as of 4 June 2002

[10]<http://search.cpan.org/doc/MROGASKI/Log-Agent-0.301/Agent.pm>; V0.301; 19 March 2002

[11]<http://www.w3.org/TR/2000/REC-xml-20001006>; as of 4 June 2002

[12][http://www.w3.org/PICS/DSig/SHA1\\_1\\_0.html](http://www.w3.org/PICS/DSig/SHA1_1_0.html); V1.0; as of 4 June 2002

[13]RFC 2045 as defined by the IETF; November 1996

[14]RFC 819 as defined by the IETF; August 1982

[15]RFC 1034 as defined by the IETF; November 1987

[16]RFC 1035 as defined by the IETF; November 1987

## 2.3 Scope

This document describes the interfaces and protocol which a developer or user of an SRS Client application can utilise to connect to the SRS server. It also describes the software modules supplied with the RIK, and how they could be used as part of an independent development of an SRS client application.

The document aims at supplying both an overview of the design and the principles that lead to it, and a very detailed definition of the interfaces supplied. It should be read in conjunction with the Detailed Requirements Specification Shared Registry System Detailed Requirements Specification; V 2.1; 20

June 2002., where the business rules for the various SRS transactions are described in detail, and which has precedence over this document.

This version of the document focuses on a complete description of Phase I of the RIK, but also gives an outline (to varying degrees of detail) as to the design of the full product.

The RIK was designed within the guidelines laid out by the architecture document Shared Registry Systems Architecture; V 1.2; 21 May 2002..

## 3 Overview

The SRS is intended to provide a business to business system supporting the registration of domains within the .nz namespace.

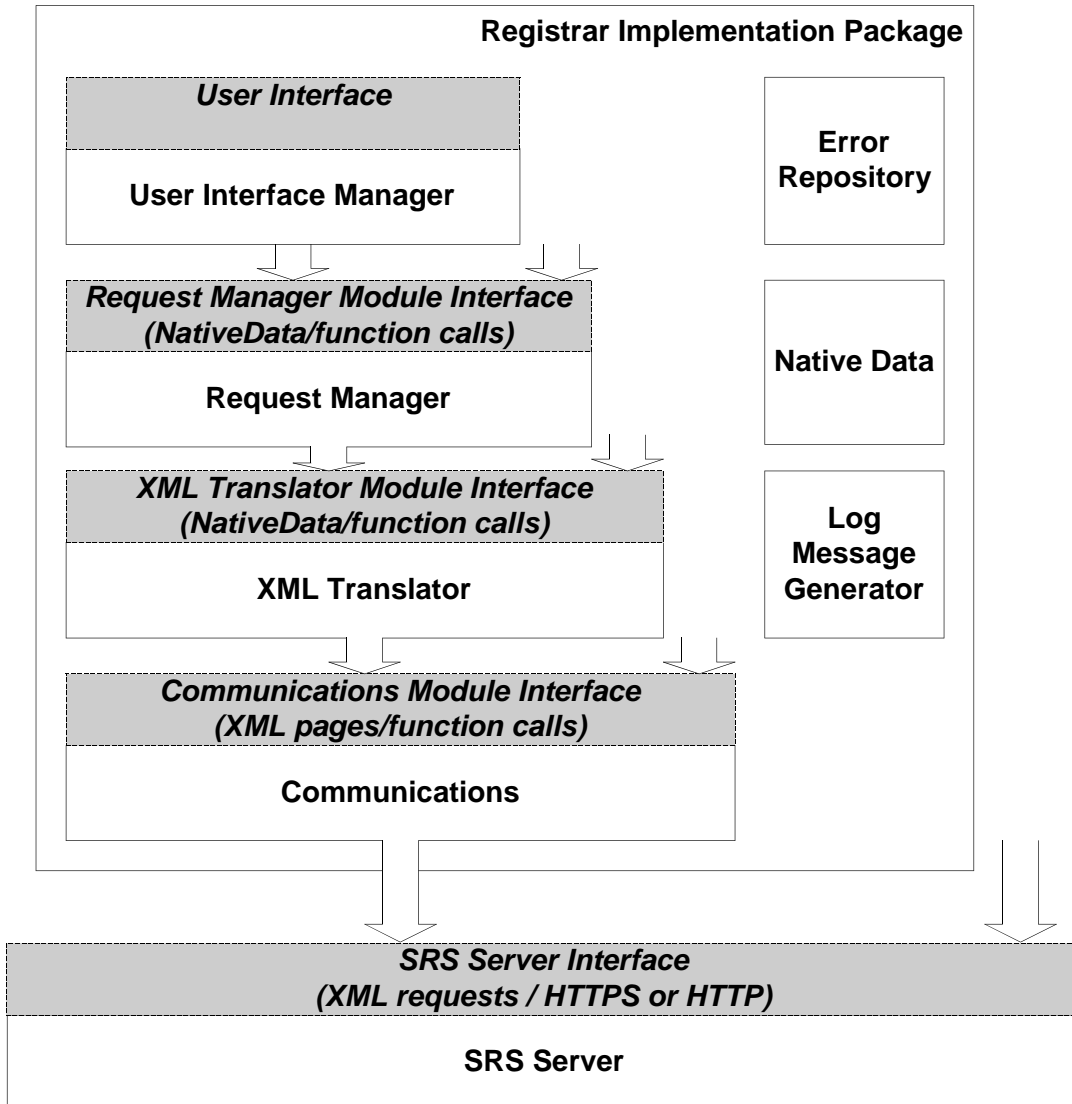
The SRS has a client – server architecture, where the registry is the server and the registrars are the clients. They in turn will have their own clients, who are the owners of the registered domain, but that is outside the scope of this document.

The RIK will provide registrars with several levels of interface, from which they can choose to interface with the server when they develop their client application. The choice ranges from interfacing directly to the server, without using any of the software supplied as part of the pack, or using the pack in its entirety, thus not developing any software at all.

# 4 High Level Design

## 4.1 Modularity

The SRS client is structured in four layers of software modules, between five published interfaces. Each layer encapsulates some aspects of the client - server protocol by using the layers below it. A developer / user may choose to access any of the interfaces directly, sidestepping the supplied modules above it.



The SRS client is structured in four layers of software modules, each exposing a well-defined interface to be used both by the layer above it and any third-party code who wishes to use it. Each layer encapsulates some aspects of the client - server protocol by using the layers below it. The lowest layer communicates with the server. A developer / user may choose to access any of the interfaces directly, sidestepping the supplied modules above it.

Four supporting software packages are used by all the layers provided with the client pack. The Client Native Data Module (CNDM) defines all dedicated data types used by the specific interfaces. The Error Repository Module (ERM) contains all information that is relevant for a client application, regarding error situations arising on the SRS server or the client. The Logger Module (LM) provides access to a

log file, and controls some basic aspects of the log messages. The Log Message Generator (LMGM) encapsulates common functionality required for logging information for the SRS.

The Client Communications Module (CCM) manages the connection to the server, and the security aspects of the communication. Its interface to the higher layers is via function calls, passing XML pages back and forth. When passed an XML page, it will add the required information and send it to the server as a request. It will return the XML page returned as the server's response, or an indication of communication failure. This module treats the data to be transferred to the server simply as buffers of bytes, without knowledge of their representing an XML data structure. A timer is implemented in this layer to recover from cases of messages getting lost between it and the SRS server.

The Client XML Translator Module (CXTM) translates a set of transactions, into an XML page complying with the client – server interface, and passes it to the CCM for transport to the server. It analyses the page returned from the server, identifying success or failure and extracting results on success. Its interface to the higher layers is via function calls with dedicated data structures (defined in the NDM) as parameters.

The Client Request Manager Module (CRMM) manages a set of transactions, expressed in minimalistic dedicated data structures (defined in the CNDM), called User Interface Data (UID). All business logic to be implemented on the client should be implemented in this layer. The interface to the higher layers shall be via function calls, with UIDs as parameters. When applicable, the full Native Data structures dedicated to a type of business request may be used as the UID to avoid redundancy.

The Client User Interface Module (CUIM), shall provide a rudimentary user interface, to enable a user to initiate a single business transaction at a time, specify the parameters for it, and view the result of the transaction.

In all modules, the interface functions are blocking and single-threaded. Any multi-threading required should be implemented in higher levels, and to allow that, all functions are thread safe and re-entrant.

All module interface functions are wrapped in an object, to avoid clashes in global namespace.

Since registrars may select to replace all modules with their own software, the client – server interface is also defined in this document. The selected interface method to the server is XML over HTTP or HTTPS (as security requires).

## 4.2 Security

The data security in this system has two orthogonal dimensions – that of operational permissions, and that of data ownership.

Permissions control what operations are allowed. They are defined in terms of read and write operations on types of records stored in the register, or specific fields within those records. Permissions are assigned to the entities that are allowed to generate requests, that is the registry itself and the registrars, and are validated on the server prior to the implementation of a request. A request may require more than one permission to be deemed permissible, depending on the type of the request and possibly on certain parameters of the request. The server shall reject a transaction if the initiating entity does not have ALL the permissions required for its completion.

Data ownership is a relationship between instances of records in the register, and the registrars responsible for their maintenance. Registrars shall be restricted in their access to data owned by them, with the one well-defined exception of transfer of ownership (see Domain). Some data has no ownership, called public data, and is accessible to all users of the system. The registry itself is normally exempt from restrictions of ownership, in that it may access all data regardless.

Each transaction sent to the server will define the registrar on behalf of which it was made, called the effective registrar. Only the registry shall be allowed to specify an effective registrar other than it. Any request arriving at the server from another registrar, where the effective registrar is not the communicating registrar, will be rejected.

The communication security is described as part of the SRS Server Interface (see Security Requirements).

## 4.3 Logging

The Log Message Generator Module (LMGM) provides the functionality to build log messages from combinations of the following elements – fixed string data, run time data, and the run time stack – and prints them to a log stream via the Logger Module (LM) which is responsible for filtering log messages at run time according to a log level value attached to each log request.

The selected log stream is syslog.

## 4.4 Error Handling

All error situations are reported by a common Error data structure (defined in CNDM). If an error occurs in one of the client layers, it will generate an Error structure and pass it back to the caller. If an error is returned from the server, one of the layers will identify it (according to the level of the error in the communication protocol with the server) and capture the information returned from the server in an Error structure.

The Error structure contains a unique identifier of the type of error (EID), an indication of severity, a description of the error, and a hint as to how to overcome the error.

The client application may choose to fill the Error structure from information returned by the SRS server, or from what is stored in the ERM on the client, and accessible by EID. This may be used as a basis for localisation of error reporting (i.e. NLS) and error-induced behaviour.

## 4.5 Portability

The software modules and the interfaces are defined and implemented in a fashion that will allow portability between operating systems. For this aim, Perl has been selected as the programming language, and XML as the interface language between the client and the server.

The requirements however, specify that the client software modules that are to be supplied together with the RIK are required to work within a single environment only, and they shall only be tested in that environment.

The developed software packages were developed and tested under the Debian GNU/Linux 3.0 operating system, using Perl version 5.6.1 and the following Perl modules:

- Crypt::OpenPGP <http://rhumba.pair.com/ben/perl/openpgp>; V1.0; 26 February 2002.
- XML::Xerces <http://xml.apache.org/xerces-p/index.html>; V1.7.0-1; as of 4 June 2002
- LWP::UserAgent <http://www.perldoc.com/perl5.6.1/lib/LWP/UserAgent.html>; as of 4 June 2002

# 5 Detailed Design

## 5.1 Modules

### 5.1.1 SRS Server Interface

#### 5.1.1.1 Overview

This interface is structured as XML transactions sent over the Internet via the HTTP or HTTPS protocol, in the form of a "POST" request. The permissible XML pages are defined in a DTD (see protocol.dtd; V 1.27; 20 June 2002.). The semantics of the data fields in the interface are described in this document. In addition, all messages sent through this layer are required to have sufficient data to identify the sending registrar, and validate the contents and source of the message.

A single XML page sent from a client to the server is called a request. The request will contain a set of SRS transactions (referred to simply as transactions throughout this document), each describing a stand-alone complete function that the server is required to implement. Each transaction is regarded by the server in the order that it appears within the request, and will completely succeed or completely fail regardless of the fate of the other transactions in the request (other than dependencies due to the order of the transactions). Each request also specifies certain parameters that are to be shared by all the transactions that it contains.

In response to each request received, the server replies with an XML page containing an overall response to the request. In some situations, the entire request will fail (i.e. an error in the request parameters, or a general failure situation on the server). The failure of a request implies the failure of all transactions included in it.

In case of a successful request, the individual responses to each transaction contained in the incoming XML page will be attached. These may each be either successful or error responses, depending on the success of each individual transaction. In fact, a successful response to a request may even contain error responses to all the transactions contained in it.

In case of a suspected attempt to breach the server's security, or the server failing to generate an XML response for whatever reason, no response will be sent.

In each error response, the server will return an EID, an indication of severity, a description of the error, and a hint as to how to overcome the error. All but the EID may be regarded as recommended values only, and the client may choose to override the values supplied by the server to implement NLS, or to fully control the behavior of the client in response to specific errors.

Rarely, a local failure on the server may cause it not to be able to verify whether a transaction succeeded or failed. This will be flagged as a failure, and a specific EID is used to identify this case.

#### 5.1.1.2 Security Requirements

All transactions attempting to modify the data in the SRS server, or to retrieve data that is not deemed public, should be transmitted via HTTPS. Such transactions that will be transmitted via HTTP will be rejected.

Authentication of the transaction source, and non repudiation, are achieved by signing a checksum of the XML message with the sender's private key.

#### 5.1.1.3 XML Requests

Each transmission from a client to the SRS server shall include the following information:

- The server assigned registrar identifier

- The XML request, encoded using UTF-8  
<http://www.perldoc.com/perl5.6.1/lib/LWP/UserAgent.html>; as of 4 June 2002
- A SHA1 checksum [http://www.w3.org/PICS/DSig/SHA1\\_1\\_0.html](http://www.w3.org/PICS/DSig/SHA1_1_0.html); V1.0; as of 4 June 2002 signed with the registrar's private key (standard PGP in ASCII armoured format).

The entire transmission shall then be application/x-www-form-urlencoded MIME encoded RFC 2045 as defined by the IETF; November 1996.

Each transmission from the SRS server to a client will include the following information:

- The XML request, encoded using UTF-8
- An SHA1 checksum signed with the Registry's private key

The entire transmission will then be application/x-www-form-urlencoded MIME encoded.

There is an upper limit on the overall size of the XML response to a request. Any request causing an excess of that shall receive an error response.

The request, the transactions, and the responses that make up the protocol between the SRS server and a client are described below at an information level – what information is included in each and what are the semantics of that information. The syntax is defined in detail in the DTD protocol.dtd; V 1.27; 20 June 2002., and subject to the XML standard <http://www.perldoc.com/perl5.6.1/lib/LWP/UserAgent.html>; as of 4 June 2002.

All parameters are mandatory unless explicitly noted as optional.

#### 5.1.1.3.1 Request / Response Parameters

A request shall contain the parameters listed below, and a list of at least one transaction.

A request response will contain the same parameters listed below, and either a transaction response for each transaction in the original request, or a single error element if the request as a whole failed.

The request parameters consist of the following:

##### [p1]RegistrarId

The unique identifier of the registrar on behalf of which the request is being made (called the effective registrar). This will be the registrar by which ownership restricted access shall be enforced.

The permissions for any transaction included will always be verified according to the registrar from whom the request was received.

##### [p2]VerMajor

The major version of the SRS server interface. The SRS server will reject any request for which this value is not equal to the major version it is supporting.

##### [p3]VerMinor

The minor version of the SRS server interface. Minor version changes will be backward compatible, but discrepancies in the minor version between the SRS server and the client may affect specific semantics of the interface.

#### 5.1.1.4 Common Response Structures

A single transaction response will be returned for each successful transaction. A special "response" is defined by the DomainTransfer structure (see DomainTransfer).

Every transaction response will include the following fields:

##### [r1]Action

The name of the action that caused this response. There are two special cases –

- "DomainTransfer" for Response like messages that notify a registrar that a domain that was registered through it has been transferred to another registrar.
- "UnknownTransaction" in the case of an Error Response to such a Transaction.

**[r2]ActionId**

The identifier of the action that caused this response.

**[r3]More**

A boolean value used for types of transactions where multiple return values are possible, but a limit to the number of returned records is imposed (see “MaxResults” and “SkipFirst” below), to indicate if there are more records matching the transaction criteria would have been returned if the limit were not imposed. This is true at the moment that the response is calculated, and may change before a continuation search is requested, due to data being removed from the SRS server. This is an optional value, which will be omitted for types of transactions where only a single return value is possible.

**5.1.1.4.1 Error**

An Error response will always contain all the fields specified, which are the following:

**[r4]ErrorId**

A unique identifier assigned to the error.

**[r5]Severity**

The standard level of severity that the SRS server assigns to this error.

**[r6]Description**

The standard description of the error that the SRS server assigns to this error.

**[r7]Hint**

The standard hint from the SRS server as to how to overcome this error.

**5.1.1.4.2 Contact**

A Contact structure will contain the following fields:

**[r8]Name****[r9]PostalAddress****[r10]Phone**

A phone number for voice contact, including country code and area code.

**[r11]Fax**

A phone number for voice contact, including country code and area code. This is optional, the default being an empty.

**[r12]EmailAddress**

A valid email address. This is optional, the default being no email address.

**5.1.1.4.3 Domain**

A Domain response will contain some of the fields specified, according to the nature of the transaction and the parameters to it. The fields are as following:

**[r13]DomainName**

The domain name.

**[r14]Status**

The domain status, being Available, Active, or PendingCancel.

**[r15]Delegate**

Whether the domain is to be delegated, providing all prerequisites for delegating a domain are satisfied.

**[r16]Term**

The number of months to bill when the domain is registered or renewed.

**[r17]EncryptedUDAI**

The UDAI that is assigned to the domain. This will be returned encrypted by the initiating registrar's public key.

**[r18]NameServers**

The list of server hosts for the domain. Each server host specification will include the fully qualified domain name (RFC 1034 as defined by the IETF; November 1987, RFC 1035 as defined by the IETF; November 1987 and others), and an ipv4 address (which may be blank). Registrars are strongly discouraged from supplying IP addresses for delegations other than those that contain their name servers themselves. The number of servers listed for a domain is currently limited to 10, but may change from time to time as deemed by InternetNZ

**[r19]Registrant**

The details of the owner of the domain, including a name and contact details.

**[r20]RegistrantRef**

The customer ID assigned to the registrant by the registrar.

**[r21]RegistrarId**

The identifier of the registrar through which the domain is registered.

**[r22]RegistrarName**

The name of the registrar through which the domain is registered.

**[r23]RegistrarContact**

The contact details for the registrar through which the domain is registered.

**[r24]AdminContact**

The contact details for administrative purposes related to the domain.

**[r25]TechnicalContact**

The contact details for technical purposes related to the domain.

**[r26]BilledUntil**

The date and time up to which the domain has been billed.

**[r27]RegisteredDate**

The timestamp of the last time the domain was registered.

**[r28]CancelledDate**

If pending cancel, the timestamp of the cancel transaction. Otherwise this will be omitted.

**[r29]LockedDate**

If locked, the timestamp of the lock transaction. Otherwise this will be omitted.

**[r30]AuditText**

The comment that was attached to the last transaction that modified this domain.

**[r31]LastChangeDate**

The timestamp of the last modification made to the domain.

**[r32]LastActionId**

The identifier of the last transaction to change the domain (coupled with "ChangedByRegistrarId").

**[r33]ChangedByRegistrarId**

The identifier of the last registrar to modify this domain.

**5.1.1.4.4 Registrar**

A Registrar response will contain some of the fields specified, according to the nature of the transaction and the parameters to it. The fields are as following:

**[r34]RegistrarId**

The identifier of the registrar.

**[r35]Name**

The name of the registrar.

**[r36]EncryptKey**

The public encryption key of the registrar.

**[r37]AccRef**

The identifier of the registrar in the registry accounting system.

**[r38]URL**

The web address of the registrar.

**[r39]RegistrarPublicContact**

The public contact details for the registrar.

**[r40]RegistrarSrsContact**

The contact details for the registrar, to be used by the registry.

**[r41]DefaultTechnicalContact**

The contact details for use as the default technical contact details for domains registered through the registrar.

**[r42]Allowed2LDs**

A list of the moderated high level domains for which the registrar is entitled to maintain domains.

**[r43]Roles**

The roles that are assigned to the registrar.

**[r44]AuditText**

The comment that was attached to the last transaction that modified this registrar.

**[r45]LastChangeDate**

The timestamp of the last modification made to the registrar.

**[r46]LastActionId**

The identifier of the last transaction to change the domain (coupled with "LastChangedBy").

**[r47]LastChangedBy**

The identifier of the last registrar to modify this registrar.

**5.1.1.4.5 DomainTransfer**

A registrar will be notified of a successful transaction transferring a domain from them to another registrar, even though they are not party to the actual transaction. This shall be done by generating a response like structure called DomainTransfer, on the SRS server, which can be initially retrieved by the registrar using a GetMessages transaction (see GetMessages) with a date range. Every registrar is expected to poll for such messages regularly.

**[r48]DomainName**

The name of the transferred domain.

**[r49]RegistrarName**

The name of the registrar to which the domain has been transferred.

**[r50]TimeStamp**

The timestamp of the transfer transaction.

**5.1.1.5 XML Transactions**

Each transaction shall be identified by a unique tag in the XML page, and have its own set of parameters.

One common type of input is an AlphanumericFilter, which is a string of characters that have to be exactly matched except for the special character '?' which may be matched by any single character, and the special character '\*' which may be matched by any number of any characters.

In the case of an AlphanumericFilter for a domain name, the special characters will not match the '.' character, except in the case of a leading '\*'. For example:

- **a\*.d** will be matched by a.b.d, but not by a.b.c.d
- **\*.c.d** will be matched by b.c.d as well as by a.b.c.d, but not by c.d
- **\*** will be match all domains

All alphanumeric filters in queries will be applied as case insensitive.

All DateRange input parameters are specified by a start date & time, and an end date & time, both being optional. If the start is omitted, it defaults to "from the beginning". If the end is omitted, it defaults to now.

The following transactions are allowed:

### 5.1.1.5.1 DomainCreate

Registers a new domain, if available, under the effective registrar.

#### Transaction parameters

**[p4]DomainName**

The requested new domain name. Must conform to the valid syntax of a standard Internet domain name RFC 819 as defined by the IETF; August 1982.

**[p5]Delegate**

Whether the domain is to be delegated, providing all prerequisites for delegating a domain are satisfied. This is an optional parameter, defaulting to true.

**[p6>ActionId**

The identifier assigned to the transaction. Each value must be unique within all ActionIds assigned to transactions made by the initiating registrar.

**[p7]RegistrantContact**

Contact details of the owner of the domain.

**[p8]RegistrantRef**

A customer identifier assigned to the registrant by the registrar. This is an optional parameter. If omitted, it defaults to blank.

**[p9]AdminContact**

Contact details for administrative purposes related to the domain. This is an optional parameter, defaulting to the registrant's contact details.

**[p10]TechnicalContact**

Contact details for technical purposes related to the domain. This is an optional parameter, defaulting to the effective registrar's technical contact.

**[p11]Term**

The number of months to bill when the domain is registered or renewed.

**[p12]NameServers**

A list of different server hosts for the domain (see Domain.NameServers). This is an optional parameter. If omitted, it shall default to none.

**[p13]AuditText**

Registrar comments or data. This is an optional parameter. If omitted, it shall default to blank.

#### Response parameters

Either a full Error response, or a full Domain response (see Domain).

### 5.1.1.5.2 DomainUpdate

Updates information of existing domains, including the option to cancel them. This transaction may only be issued on behalf of the registrar through whom the domain is registered. An exception to this is when the update transfers the domain to a new registrar, in which case it may only be issued on behalf of the intended registrar, and the UDAI of the domain must be provided.

To be acceptable, the registrar issuing the request must have all permissions (see Security) attached to the attributes that are to be modified by the transaction.

#### Transaction parameters

All the parameters of the DomainCreate transaction are included in this transaction as described there, with the following differences –

- The **DomainName** parameter is replaced by a **DomainNameFilter**, which may contain a list of AlphanumericFilters for the name of the domains to be updated.

- If the **AuditText** parameter is omitted, the corresponding field in the domain shall be modified to be blank.
- The **RegistrantContact** parameter is an optional parameter.
- If any optional parameter other than the **DomainName** and the **AuditText** is omitted, the current value of the domain field associated with that parameter will remain unchanged, unless another default behaviour is described below.

The additional parameters are –

**[p14]UDAI**

The unique identifier assigned to the owner of the domain. It should be encrypted by the registrar's private key. This is a mandatory parameter if the transfer parameter is set (unless the transaction is initiated by the registry), but otherwise it is ignored and may be omitted.

**[p15]New UDAI**

A boolean value specifying a request for a new UDAI to be issued to the domain. This is an optional parameter, defaulting to false. The server may elect to issue a new UDAI regardless of the value of this parameter.

**[p16]Renew**

A boolean value specifying a request for renewing the domain registration for a further billing period now, rather than waiting for it to expire. This is an optional parameter, defaulting to false.

**[p17]Release**

A boolean value specifying a request for making a domain available after the cancellation grace period expires. This is an optional parameter, defaulting to false.

**[p18]Lock**

A boolean value specifying the lock status of the domain. A locked domain is essentially frozen in its current status until it is unlocked, and may only be updated by the registry. This operation is only available for the registry. This is an optional parameter. If it is omitted, the lock status shall remain unchanged.

**[p19]Cancel**

A boolean value specifying a request for changing the status of a domain. If true, active domains will change status to pendingCancel. If false, domains in pendingCancel state shall revert to an active state. This is an optional parameter. If it is omitted, the domain status shall remain unchanged.

**Response parameters**

Either a full Error response, or a full Domain (see Domain) response, except for the UDAI that will only be returned if it has changed.

**5.1.1.5.3 DomainDetailsQry**

Retrieves information of multiple domains, sorted by domain name.

Registrars shall retrieve only domains that are currently registered through them. When historical information is retrieved (see below) for a period that the domain was not registered through the effective registrar, only public information as defined by the response to the WhoIs transaction (see WhoIs) shall be included.

It is important to note that the ResultsDateRange parameter strongly affects the results of the transaction. Without it, only the current status of the currently registered domains are inspected. When a ResultsDateRange parameter is given, it is applied first to give snapshots of all the domains that were registered during the specified date range, and all the changes that they went through during that date range, and then the rest of the filter parameters are applied on that. Only snapshots matching the filters will be returned, so that the snapshots belonging to a single domain may not give its full history (dropping intermediate snapshots that do not match the criteria). When a ResultsDateRange parameter is given, read the word "current" in the description of the other parameters as current to the time of the

snapshot. History for a domain will not precede the current registration date, even if it had previously been registered.

The presence of the ResultsDateRange parameter also affects the interpretation of the MaxResults and the SkipResults parameter, which will control the number of distinct domains skipped and returned by the query – each with its full history set as matches the ResultsDateRange. This means that the actual number of Domain records returned will most likely exceed the MaxResults limit.

### **Transaction parameters**

#### **[p20]RegistrarId**

A filter for the domains retrieved based on their registrar. This is an optional parameter. The default behaviour will impose no such filter. This parameter is meaningful only for the registry.

#### **[p21]DomainNameFilter**

A list of AlphanumericFilters for the domains retrieved based on their name. All domains matching any of the filters shall be returned. This is an optional parameter. The default behaviour will impose no such filter.

#### **[p22]Status**

A filter for the domains retrieved based on whether they are active or pending release (if the “Available” status is used here, the response will always return empty). This is an optional parameter. The default behaviour will impose no such filter.

#### **[p23]Delegate**

A filter for the domains retrieved based on whether they should be delegated or not. This is an optional parameter. The default behaviour will impose no such filter.

#### **[p24]Term**

A filter for the domains retrieved based on their billing term. This is an optional parameter. The default behaviour will impose no such filter.

#### **[p25]SearchDateRange**

This parameter applies the all other filter criteria to a date range, retrieving domains that matched the criteria at any time during the specified date range. The data returned will be the current data for those domains. This is an optional parameter. The default behaviour will be to apply the filter criteria only to the domain’s current status.

#### **[p26]ResultsDateRange**

This parameter causes the server to retrieve historical snapshots of domains as they were within the specified date range (see above for discussion). Records returned for the same domain will be sorted by timestamp. This is an optional parameter. Omitting it will cause only the latest data for each registered domain that matches the filters to be returned.

#### **[p27]RegisteredDateRange**

A filter for the domains retrieved based on their date of registration. This is an optional parameter. The default behaviour will impose no filter by date of registration.

#### **[p28]LockedDateRange**

Filter the domains retrieved to those that are currently locked, and the lock request that caused this fell within the date range. This is an optional parameter. The default behaviour will impose no such filter.

#### **[p29]CancelledDateRange**

Filter the domains retrieved to those that are currently cancelled (or pending cancel), and the cancel request that caused this fell within the date range. This is an optional parameter. The default behaviour will impose no such filter.

#### **[p30]BilledUntilDateRange**

A filter for the domains retrieved based on the date up to which they have been billed. This is an optional parameter. The default behaviour shall impose no such filter.

**[p31]NameServers**

A list of AlphanumericFilters for the domains retrieved based on the domain name servers. A domain will be retrieved if one of its domain name servers matches one of those listed. This is an optional parameter. The default behaviour will impose no such filter.

**[p32]RegistrantRef**

An AlphanumericFilter for the domains retrieved based on the customer reference given to it by the registrant. This is an optional parameter. The default behaviour will impose no such filter.

**[p33]RegistrantContact**

A filter based on details of the registrant contact details attached to the domain. Partial details may be provided, in which case those omitted will be disregarded when matching the filter. This is an optional parameter. The default behaviour will impose no such filter.

**[p34]AdminContact**

A filter based on details of the administrative contact details attached to the domain. Partial details may be provided, in which case those omitted will be disregarded when matching the filter. This is an optional parameter. The default behaviour will impose no such filter.

**[p35]TechnicalContact**

A filter based on details of the technical contact details attached to the domain. Partial details may be provided, in which case those omitted will be disregarded when matching the filter. This is an optional parameter. The default behaviour will impose no such filter.

**[p36]MaxResults**

The maximum domain records to be retrieved by this transaction. This is an optional parameter. The default shall be 100.

**[p37]SkipResults**

The number of domains to be skipped by this transaction. Together with the maxResults parameters, this creates a way to break down large numbers of domains being queried into packages of manageable size. This is an optional parameter. The default will be 0.

Note that the domains skipped are calculated locally for each transaction. This may cause missing and/or repeating records in the combined result of a few such requests. (i.e. if between requests domains are added, modified, transferred or removed).

**[p38]FieldList**

A list of field names, defining what fields to return for each domain. The possible fields are:

**DomainName DomainName**  
**Status Status**  
**Delegate Delegate**  
**Term Term**  
**NameServers NameServers**  
**Registrant RegistrantContact**  
**RegistrantRef RegistrantRef**  
**RegistrarId RegistrarId**  
**RegistrarName RegistrarName**  
**AdminContact AdminContact**  
**TechnicalContact TechnicalContact**  
**BilledUntil BilledUntil**  
**RegisteredDate RegisteredDate**  
**CancelledDate CancelledDate**  
**LockedDate LockedDate**  
**AuditText AuditText**  
**LastChangeDate LastChangeDate**  
**LastActionId LastActionId**  
**LastChangedBy**

This is an optional parameter. The default value shall specify DomainName and Status.

**Response parameters**

Either a full Error response or a set of Domain responses (see Domain) including only those fields that have been requested, and are not restricted by ownership.

**5.1.1.5.4 RegistrarCreate**

Creates a new registrar.

**Transaction parameters****[p39]ActionId**

The identifier assigned to the transaction. Each value must be unique within all ActionIds assigned to transactions made by the initiating registrar.

**[p40]Name**

The legal name of the Registrar.

**[p41]AccRef**

The registrar's identifier in the registry accounting system.

**[p42]URL**

The registrar's web address. This is an optional parameter. The default shall be no address.

**[p43]RegistrarPublicContact**

The public registrar contact details.

**[p44]RegistrarSrsContact**

The public registrar contact details.

**[p45]AdminContact**

The registrar contact details for administrative purposes.

**[p46]DefaultTechnicalContact**

The contact details for used as the default technical contact details for domains registered through the registrar.

**[p47]Allowed2LDs**

A list of second level domains that the registrar is allowed to maintain. This is an optional parameter. The default shall be an empty list.

**[p48]Roles**

The list of permissions assigned to the registrar, controlling the operations that they shall be allowed to request in the SRS. This is an optional parameter. The default shall be an empty list.

**[p49]EncryptKey**

The registrar's public encryption key.

**[p50]AuditText**

This is an optional parameter, intended for comments related to the registrar. If omitted, it will default to blank.

**Response parameters**

Either a full Error response, or a Registrar response (see Registrar).

**5.1.1.5.5 RegistrarUpdate**

Updates information of an existing registrar.

All the parameters of the RegistrarCreate transaction are included in this transaction as described there, but they are all optional. When omitted, the current value of the registrar field associated with that parameter will remain unchanged.

**Transaction parameters****[p51]RegistrarId**

The unique identifier assigned to the registrar.

**Response parameters**

Either a full Error response, or a Registrar response (see Registrar).

**5.1.1.5.6 RegistrarDetailsQry**

Retrieves information of existing registrars. A registrar will only be allowed to query for his own data

**Transaction parameters****[p52]RegistrarId**

The identifier of the registrar to be retrieved. This is an optional parameter. If omitted, it will retrieve all registrars.

**[p53]RegistrarName**

An AlphanumericFilter for the registrars retrieved based on their name. This is an optional parameter. The default behaviour will impose no such filter.

**[p54]ResultsDateRange**

This parameter will cause the server to return multiple records for each registrar, describing all the changes that the domain went through during the effective period. No information on Allowed2LDs or

Roles are returned when this parameter is in effect. This is an optional parameter. Omitting it will cause only the latest data for each registrar that matches the filters to be returned.

### **Response parameters**

Either a full Error response, or a set of Registrar responses (see Registrar).

#### **5.1.1.5.7 RegistrarAccountQry**

Retrieves transactions in the account of a registrar.

### **Transaction parameters**

#### **[p55]RegistrarId**

The unique identifier of the registrar for which billing transactions will be retrieved.

#### **[p56]CustomerId**

An AlphanumericFilter on the billing transactions based on the identifier given by the registrar to the customer that they are related to. This is an optional parameter. The default behaviour will impose no such filter.

#### **[p57]DomainName**

An AlphanumericFilter on the billing transactions based on the domain name that they are related to. This is an optional parameter. The default behaviour will impose no such filter.

#### **[p58]Invoice**

Filter the billing transactions retrieved to those that are related to a specific invoice. This is an optional parameter. The default behaviour will impose no such filter.

#### **[p59]InvoiceDateRange**

Filter the billing transactions retrieved to those for which an invoice was generated within the date range. This is an optional parameter. The default behaviour will impose no such filter.

#### **[p60]TransDateRange**

Filter the billing transactions retrieved to those that occurred within the date range. This is an optional parameter. The default behaviour will impose no such filter.

#### **[p61]MaxResults**

The maximal account records to be retrieved by this transaction. This is an optional parameter. The default is 100.

#### **[p62]SkipResults**

The number of account records to be skipped by this transaction. Together with the maxResults parameters, this creates a way to break down large numbers of account records being queried into packages of manageable size. This is an optional parameter. The default is 0.

Note that the account records skipped are calculated locally for each transaction. This may cause missing and/or repeating records in the combined result of a few such requests. (i.e. if between requests registrars are added or modified).

### **Response parameters**

Either a full Error response, or a set of Billing Details records, each including the following:

#### **[r51]Type**

The type of the billing transaction.

#### **[r52]Status**

Whether the billing transaction is confirmed or pendingConfirmation.

#### **[r53]RegistrarId**

The identifier of the registrar to which the billing transaction belongs.

**[r54]DomainName**

The domain to which the billing transaction is related.

**[r55]Registration / Renewal Term**

Data belonging to the accounting system, stored with the transaction record in the SRS.

**[r56]Invoice**

The identifier of the invoice to which the billing transaction belongs.

**[r57]InvoiceDate**

The timestamp of the invoice to which the billing transaction belongs.

**[r58]CustomerId**

The registrant's identifier of the customer to which the billing transaction is related, as provided by the registrar to the SRS server.

**[r59]BillingTerm**

The number of months for which the billing transaction was issued.

**[r60]Amount**

The sum of money attached to the billing transaction.

**[r61]TransactionDate**

Data belonging to the accounting system, stored with the transaction record in the SRS.

**5.1.1.5.8 GetMessages**

Retrieves responses generated by the SRS server for a registrar. Specifying no filter will result in an error. The full transaction identifier is the combination of the registrar that initiated it (the originating registrar) and the ActionId that was given to it.

A GetMessages transaction must be the only transaction in its request.

**Transaction parameters****[p63]OriginatingRegistrarId**

Identifies the registrar that assigned the ActionId to the action for which the response message is requested. This is an optional parameter. If omitted, all responses matching other criteria will be returned.

A special OTHER\_REGISTRAR value is available for requesting messages for all actions not originated by the registrar that is making this request.

**[p64]ActionId**

Identifies the action for which the response message is requested. This is an optional parameter. If omitted, all responses matching other criteria will be returned.

**[p65]EffectiveRegistrarId**

Identifies the registrar for which the response messages were intended. This is an optional parameter. If omitted, messages for all registrars will be returned (relevant only for the registry).

**[p66]TransactionRange**

The date time range for which all messages are requested. The defining time for this purpose is the transaction timestamp, as applied by the SRS server (and relates to the time that the transaction was accepted by the SRS server). This is an optional parameter. If omitted, no such filter will be applied.

**Response parameters**

Either a full Error response, or a list of responses to all the transactions that match the criteria.

**5.1.1.5.9 WhoIs**

Retrieves public information of a single existing domain.

**Transaction parameters****[p67]DomainName**

The domain for which the details are requested.

**Response parameters**

Either a full Error response, or a Domain response (see Domain) including only the following fields:

**DomainName** DomainName

**Status** Status

**Delegate** Delegate

**NameServers** NameServers

**Registrant** RegistrantContact

**RegistrarContact** RegistrarContact

**AdminContact** AdminContact

**TechnicalContact** TechnicalContact

**BilledUntil** BilledUntil

**RegisteredDate** RegisteredDate

**CancelledDate** CancelledDate

**LockedDate** LockedDate

**LastChangeDate** LastChangeDate

If the domain is available, only the domain name and the status will be included.

**5.1.1.5.10 ActionDetailsQry**

Retrieves the original XML of a previous request including the CreateDomain and/or UpdateDomain transaction sent on behalf of a registrar to the server, and the response to it.

**Transaction parameters****p1. RegistrarId**

Identifies the transaction for which the XML is requested. Note that the value OTHER\_REGISTRAR (see GetMessages.OriginatingRegistrarId) is not valid here.

**p2. ActionId**

Identifies the transaction for which the XML is requested.

**Response parameters**

Either a full Error response, or two raw XML strings – the original request and the response to it.

**5.1.1.5.11 UDAIValidQry**

Validates that a given UDAI belongs to a registered domain.

**Transaction parameters****p1. DomainName**

The domain for which the UDAI is to be validated.

**p2. UDAI**

The UDAI for the domain.

### **Response parameters**

Either a full Error response, or a boolean value noting whether the given UDAI is the valid one for the domain.

#### **5.1.1.6 Possible Errors Generated on the Server**

In addition to input validation errors, as described in the appendix (see Appendix B – Transaction validate() Errors), the following errors may be returned by the SRS server:

##### **5.1.1.6.1 Communication Errors**

Errors related to the underlying communication layer and the security protocol.

###### **[e1] SERVER\_COMMUNICATION\_TIME\_OUT**

##### **5.1.1.6.2 Malformed XML Errors**

Errors in the XML-related protocol and XML page structure or syntax.

###### **[e2] INVALID\_REQUEST\_ROOT**

###### **[e3] MISSING\_SIGNATURE**

Could not find the required encrypted signature (see Security) in the request.

###### **[e4] INVALID\_REQUEST\_TIMESTAMP**

###### **[e5] INVALID\_XML\_ROOT**

###### **[e6] ERROR\_PARSING\_XML\_REQUEST**

An unknown error has occurred when parsing the incoming XML page.

##### **5.1.1.6.3 Malformed Request Errors**

Invalid parameters of a request.

###### **[e7] INVALID\_SRS\_VERSION**

The SRS interface version used for the request (see Request / Response Parameters) is invalid or could not be found

###### **[e8] INCOMPATIBLE\_SRS\_VERSION**

The SRS interface version used for the request is incompatible with the version used by the SRS server (see Request / Response Parameters).

###### **[e9] MISSING\_EFFECTIVE\_REGISTRAR**

The required effective registrar identifier field for the request could not be found (see Request / Response Parameters).

###### **[e10] INVALID\_REGISTRAR\_ID**

The required registrar identifier is invalid or could not be found. See definition of the specific transaction type in XML Transactions.

##### **5.1.1.6.4 Invalid Request Errors**

The request was rejected because of business-related restrictions, or limits imposed on the system.

###### **[e11] RESPONSE\_TOO\_LARGE**

The response to the request would have exceeded an operational limit on the server. This will most likely be caused by too many transactions in a single request, or the filters in a query transaction being too broad.

##### **5.1.1.6.5 Malformed Transaction Errors**

Invalid parameters of a transaction.

**[e12] DUPLICATE\_TRANSACTION\_ID**

The ActionId specified for the transaction has already been used for another transaction originating from the same registrar.

**[e13] UNKNOWN\_TRANSACTION\_TYPE**

The identifier of the transaction type specifies an unknown transaction.

**5.1.1.6.6 Invalid Transaction Errors**

The transaction was rejected because of business related restrictions. This will include transactions rejected for lack of permission.

**[e14] TRANSACTION\_PERMISSION\_DENIED**

The registrar issuing this transaction does not have all the roles required to complete it.

**[e15] TRANSACTION\_TRANSMITTED\_VIA\_INSECURE\_MEDIUM**

The transaction requires a secure communication protocol, but it was received via an insecure one (see Security Requirements).

**[e16] DOMAIN\_NOT\_FOUND**

The domain related to by the transaction could not be found in the registry. Perhaps it is not registered.

**[e17] DOMAIN\_ALREADY\_EXISTS**

The domain related to by the transaction is already registered in the registry.

**[e18] DOMAIN\_OWNED\_BY\_ANOTHER\_REGISTRAR**

The domain related to by the transaction is inaccessible because it is registered via another registrar.

**[e19] INVALID\_DUPLICATION\_OF\_DOMAIN\_NAME\_SERVER**

The NameServers field in the transaction may not contain duplicate name servers.

**[e20] MESSAGE\_NOT\_FOUND**

The transaction requested by the GetMessages transaction could not be found.

**[e21] REGISTRY\_MUST\_NOT\_BE\_EFFECTIVE\_REGISTRAR**

The registry must not be the effective registrar for this transaction.

**5.1.1.6.7 Processing Errors**

Problems within the SRS server prevented the full application or response to the transaction, or to the request.

**[e22] SERVER\_IN\_READ\_ONLY\_MODE**

The SRS server is currently in read only mode, therefore all modifying requests have been rejected.

**[e23] SERVER\_COULD\_NOT\_VERIFY\_SUCCESS**

Due to a temporary failure within the SRS server, the transaction was accepted but its success or failure could not be verified. This will usually occur only when the SRS server failure causes it to change into read only mode after the transaction was received and before a response has been generated.

**[e24] TEMPORARY\_SERVER\_ERROR**

An error occurring on the server, where a failure was caused due to lack of resources, congestion, or connectivity problems within the server.

**[e25] INTERNAL\_SERVER\_ERROR**

An unexpected situation has occurred within the SRS server.

**[e26] UNKNOWN\_ERROR**

An error that can not be identified or described as any of the above has occurred within the SRS server.

## 5.1.2 Logger Module

The Logger Module (LM) is implemented by using the standard module `Log::Agent` <http://search.cpan.org/doc/MROGASKI/Log-Agent-0.301/Agent.pm>; V0.301; 19 March 2002 with `Log::Agent::Driver::Syslog`.

## 5.1.3 Log Message Generator Module

The Log Message Generator Module (LMGM) shall provide a single interface to format and log message to the log stream. It shall provide an option to include the stack trace in the log message, and allow for runtime data as well as fixed elements in the log message. The LMGM shall also define logging levels and implement a filtering mechanism based on them.

For the log stream, a standard Perl package wrapping the access to the syslog shall be selected.

A singleton Design Patterns Elements of Reusable Object-Oriented Software, by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. `LogMessageGenerator` class implemented in this module shall provide all the functionality described. All its functions shall be thread safe and re-entrant.

This module requires no information from an external source to operate.

### 5.1.3.1 LogMessageGenerator

#### 5.1.3.1.1 initialize()

This function initialises a `LogMessageGenerator`, which in turn initialises its own logging stream.

**Input**

None

**Output**

None

**Possible Errors Generated**

None.

#### 5.1.3.1.2 logMessage()

This function formats a log message, taking the options specified into account, and prints it to the log file. If the format fails, a fixed error message is printed in its place.

**Input**

- A log priority level
- A boolean value defining whether to write the runtime stack into the log as an appendix to the log message
- A format string
- Optional values to be used in conjunction with the format string

**Output**

- None.

**Possible Errors Generated**

None.

## 5.1.4 Client Native Data Module

This module defines all the data structures and the constants required for the interfaces that are not specific to the communication mechanism employed between the client and the server.

All data structures contain “knowledge” of syntactical requirements and default values for each field, and whether it is a mandatory field or an optional one, and where applicable, hide internal representation of complex data through access functions. They contain no “knowledge” of business logic.

#### 5.1.4.1 Simple types and constants

##### 5.1.4.1.1 RegistrarId

A value uniquely identifying the registrar in the SRS system. Two special values are defined:

- **NO\_REGISTRAR\_ID** - serving as an unassigned or invalid value
- **COMMUNICATING\_REGISTRAR** - where the registrar is taken to be the same one that initiated the communication. This is treated as NO\_REGISTRAR\_ID in any context where the phrase “communication initiator” is meaningless.

##### 5.1.4.1.2 ErrorId (EID)

A value uniquely identifying the type of an error in the SRS system. A special UNKNOWN\_ERROR value is defined, and used to describe an error situation where no further details are known.

#### 5.1.4.2 SrsSimpleData

The SrsSimpleData is a base class for NativeData classes, providing common functionality for initialisation, validation, print & clear. It holds the actual data fields specific to the actual derived type, as well as metadata describing it. Special access functions are available for the data itself. Such a data field contains either a simple value, in which case it is a scalar, an object - usually derived from SrsSimpleData, in which case it is a reference to the object, or a multiple value, in which case it is a reference to a list of values, all of the same type.

The metadata is defined as constants of the derived class, and shared by all instances of that class. The metadata fields are:

- **ATTRIBUTES** - an associative array of SrsSimpleAttributes (see below), keyed by the data field name, describing the different data fields of the derived SrsSimpleData class. Field names not included in this array are regarded as unknown and invalid.
- **UNKNOWN\_FIELD\_EID** - the error ID to use when an undeclared data field is encountered.
- **DEFAULT\_DATA** - a reference to an associative array of the default values for the different data fields of the derived SrsSimpleData class. This need not be a full array of all data fields. Omitted data fields shall by default have an undefined value.

An SrsSimpleAttribute is a class describing the characteristics of a data field within the context of the SrsSimpleData. The attributes included in the SrsSimpleAttribute are:

- **ifMissing** - defines the data field to be mandatory. Its value is the ErrorId that is used during validation if this field is missing. It is omitted if field not mandatory.
- **validityRegEx** - a regularExpression that is used for validating the value of the data field. It is omitted if no such validation required.
- **invalidErrorId** - the ErrorId that is used when the value of this attribute fails to match validityRegEx. Assigned only when a validityRegEx is assigned.
- **hasValidate** - a boolean value noting whether this data field contains a class with a validate() function, which will be used in validation of the SrsSimpleData object. When true, this attribute overrides the presence of validityRegEx. If omitted, defaults to false.
- **hasPrint** - a boolean value noting whether this attribute contains a class with a print() function, which will be used when printing the SrsSimpleData object. That function is expected to behave similar to the SrsSimpleData::print() function (see below), and is passed the same parameters. If omitted, defaults to false.

- **ifMultiple** – defines whether the data field allows only single values or a list of multiple values of identical type. Its value is the ErrorId to be used during validation if this field hold a list of values. It is omitted if the data field may contain multiple values.

All Errors mentioned in the ATTRIBUTES are registered in the Error::Repository (see ErrorRepository) by the derived class.

The SrsSimpleData base class provides the following functions –

- `new(ATTRIBUTES, UNKNOWN_FIELD_EID, DEFAULT_DATA, [hash of fieldName => value])` - the base class constructor, used only by the derived class constructors. The last parameter is for initialising data fields, and is optional. The data fields are set to the DEFAULT\_DATA, and then all fields for which initialising values were passed in are overwritten.
- `clear()` - sets the data fields to the DEFAULT\_DATA, leaving any fields that are not included in the default to be undefined
- `setFields([hash of fieldName => value])` – assigns the supplied values to the requested data fields, as identified by the hash keys. All fields not mentioned in the input hash table will remain unchanged.
- `hasField(fieldName)` – returns a boolean value, noting if the requested data field has a defined value.
- `getField(fieldName)` – returns the value of the requested data field.
- `getFields()` - returns the contents of the data fields in a hash.
- `validate(checkMandatoryAttributes, validateAttributes)` - checks that the data fields comply with the validation requirements described in ATTRIBUTES. The validation can be controlled by the boolean input parameters. The validation includes:
  - Check for unknown data fields. If a field name that is not included in ATTRIBUTES is found, returns UNKNOWN\_FIELD\_EID. This is always validated.
  - Check for multiple values. If a data field contains multiple values when its attribute defines an ifMultiple ErrorId, that error is returned. This is always validated.
  - Validate all data fields for which the attribute defines a method of validation (either validityRegEx or hasValidate). This is only executed if the validateAttributes parameter is true.
  - Validate that all data fields for which the attribute defines an ifMissing ErrorId have a defined value. This is only executed if the checkMandatoryAttributes parameter is true.
- `print(output, prefix)` – prints the data fields to the output stream parameter, each field on a line. If the attribute for a data field specifies hasPrint, its print function is called. Multiple values in a data field are each printed in a separate line, slightly indented. Each printed line is headed by the prefix input.

#### 5.1.4.3 Error

A common structure is used to report all error situations that return from the server, or arise in the client.

The Error structure has the following fields:

- **id** - An EID. The default value is UNKNOWN\_ERROR.
- **description** - a string describing the error in terms intended for the user. The default value is the empty string.

- **severity** – an enumerated value, classifying the severity of the error, which may be used to initiate an automatic response on the client application (i.e. paging an administrator or shutdown). The default value is the most severe. The values expected for the severity are the following –
  - **emergency** (high)
  - **alert**
  - **critical**
  - **error** (low)
- **recoveryHint** – an enumerated value, classifying the recommended method of overcoming the error, which will be one of the following –
  - **MALFORMED\_REQUEST\_ERROR** - validate the format of the request / transaction. Refer to the error message for possible details.
  - **INVALID\_REQUEST\_ERROR** - validate the content of the request / transaction. Refer to the error message for possible details.
  - **TEMPORARY\_SERVER\_ERROR** - probably a resource problem, or a server failure situation that will soon be remedied.
  - **UNKNOWN\_RESULT** - the request / transaction was accepted by the server, and could have been successfully applied on the server, but its success could not be validated. Wait a while and query the expected result to check if it succeeded.
  - **UNKNOWN\_ERROR\_HINT** - contact the registry for support.

The default value is UNKNOWN\_ERROR\_HINT.

- **isServerError** – a boolean value, true if the Error originated from the SRS server, and false if from the client application.

A special NO\_ERROR value is used to report success where an error structure is expected.

The Error structure shall have the following functions:

- **clear()** - assigns the default values to all the fields in the structure.
- **setError(anId, aDescription, aHint, aSeverity)** – assigns the supplied values to the related fields.
- **getError()** – returns a hash including all field names as keys to their values.
- **getId()**.
- **new()** - returns a new Error structure with all fields assigned to their default values.
- **new(anId, aDescription, aHint, aSeverity)** - shorthand to calling new() and then setError(anId, aDescription, aHint, aSeverity).
- **validate()** – currently always returns NO\_ERROR.

- `log()` – generate a log message from the data enclosed in the structure.

#### 5.1.4.4 Transaction

Each transaction type has a dedicated Transaction structure (for example `CreateDomainTransaction`) for holding all the parameters of that transaction.

All Transaction structures have the following common fields:

- **transactionType** - one value out of an enumeration of all transaction types, and a special `NO_TRANSACTION` value. The default value is `NO_TRANSACTION`.
- **fields** - a hash of `fieldName => value`, containing all the parameters describing the transaction. The contents of the hash are specific to the actual transaction type.
- **response** - a reference to the Response structure received in response to the transaction. The default value represents an unknown error response.
- **requiresSecurity** - a boolean value defining whether this transaction requires a secure protocol for communication with the SRS server.

All Transaction structures have the following common functions:

- `clear()` - assigns the default values to all the fields in the structure.
- `setFields([hash of fieldName => value])` – assigns the supplied values to the requested fields, as identified by the hash keys. All fields not mentioned in the input hash table will remain unchanged.
- `getFields()` - returns the contents of the fields hash.
- `getType()` - returns the contents of the `transactionType` field.
- `new()` - returns a new Transaction structure with all fields assigned to their default values.
- `new([hash of fieldName => value])` - shorthand to calling `new()` and then `setRequest` ([hash of `fieldName => value`]).
- `setResponse(Response r)` - sets the response field to the Response structure supplied as a parameter.
- `getResponse()` - returns the Response structure stored in the response field.
- `setError(Error e)` - sets the error field to the Error structure supplied as a parameter.
- `getError()` - returns the Error structure stored in the error field.
- `validate()` - checks that all the field values are syntactically correct (i.e. an integer field contains an integer, etc.), not in violation of rules concerning internal representation, and that all mandatory fields contain a meaningful value. Returns `NO_ERROR` if valid, or an Error describing the error if invalid. For list of possible Errors returned, see `TM_GETMSG_NOT_ALONE`
- `isSuccessful()` - returns false if an error is associated with the Transaction.
- `requiresSecurity()` - returns true if this Transaction type requires a secure communication protocol for the server.

#### 5.1.4.5 Response

Each transaction type has a dedicated Response structure (for example `CreateDomainResponse`) for containing all the data returned by the server in response to that transaction.

All Response structures have the following common fields:

- **transactionType** - one value out of an enumeration of all transaction types, and a special `NO_TRANSACTION` value. The default value is `NO_TRANSACTION`.

- **fields** - a hash of fieldName => value, containing all the parameters describing the response. The contents of the hash will be specific to the actual transaction type.

All Response structures shall have the following common functions:

- clear() - assigns the default values to all the fields in the structure.
- setFields([hash of fieldName => value]) - assigns the supplied values to the requested fields, as identified by the hash keys. All fields not mentioned in the input hash table will remain unchanged.
- getFields() - returns the contents of the fields hash.
- new() - returns a new Response structure with all fields assigned to their default values.
- new([hash of fieldName => value]) - shorthand to calling new() and then assign([hash of fieldName => value]).
- validate() - checks all the field values are syntactically correct (i.e. an integer field contains an integer, etc.), not in violation of rules concerning internal representation, and that all mandatory fields contain a meaningful value. Returns NO\_ERROR if valid, or an Error describing the error if invalid.
- isError() - returns false.

#### 5.1.4.6 ErrorResponse

A specialized Response (derived from the Response class), representing an error response to a transaction. Its “fields” data are disregarded.

To separate it from all successful Responses, it overrides the isError() function to return true.

The ErrorResponse has the following additional functions:

- setTransactionType(TransactionType type).
- getError() - returns the enclosed Error object.
- setError(Error e) - sets the enclosed Error object.
- setErrorFields([hash of fieldName => value]) - assigns the supplied values to the requested fields of the enclosed Error object, as identified by the hash keys. All fields not mentioned in the input hash table will remain unchanged.

#### 5.1.4.7 Request

A Request encapsulates a list of transactions to be executed in order by the server as part of a single communication, on behalf of a single registrar.

All Request structures have the following common fields:

- **effectiveRegistrar** - the registrar on behalf of which the transactions are made. The default value is COMMUNICATING\_REGISTRAR.
- **transactions** - an array of Transactions. By default the list will be empty.
- **error** - a reference to the Error structure related to the request as a whole. The default value is UNKNOWN\_ERROR.
- **securityRequired** - an enumeration (RequestSecurity), with the following values -
  - **FORCE\_SECURE** - to be sent over a secure protocol regardless of transactions included.
  - **FORCE\_INSECURE** - to be sent over an insecure protocol regardless of transactions included.
  - **DEFAULT\_SECURITY** - security of protocol determined according to transactions included.

The Request has the following functions:

- `clear()` - assigns the default values to all the fields in the structure.
- `setFields([hash of fieldName => value])` - assigns the supplied values to the requested fields, as identified by the hash keys. All fields not mentioned in the input hash table will remain unchanged.
- `getFields()` - returns the contents of the fields hash.
- `new()` - returns a new Transaction structure with all fields assigned to their default values.
- `new([hash of fieldName => value])` - shorthand to calling `new()` and then `setRequest ([hash of fieldName => value])`.
- `addTransactions([list of Transactions] t)` - appends `t` to transactions.
- `setError(Error e)`.
- `getError()`.
- `setEffectiveRegistrar(RegistrarId rid)`.
- `getEffectiveRegistrar()`.
- `setSecurityRequired(RequestSecurity rs)`.
- `getSecurityRequired()`.
- `validate()` - checks that the effectiveRegistrar is valid, calls `validate()` on all transactions included, and validates compliance with all restrictions relating to multiple transactions (i.e. a GetMessages transaction must be alone in the Request). Returns `NO_ERROR` if valid, or an Error describing the error if invalid.
- `isSuccessful()` - returns false if an error is associated with the Request.

### 5.1.5 Error Repository Module

The Error Repository Module (ERM) contains the information required to fill in an Error structure, identified by its EID. It may be thought of as a database for error data.

In the implementation supplied, the error data is hard coded into the software module. In the future, it may allow for configuration or management functions for adapting the error information by client users. In any case, a few Error structures will always remain hard coded within the module, to be used for internal failure of the ERM.

In the implementation supplied, the EIDs are strings containing the full name of the class registering them, to avoid EID clashes. It is recommended that third-party code employ the same strategy.

A singleton Design Patterns Elements of Reusable Object-Oriented Software, by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. ErrorRepository (ER) class implemented in this module provides all the functionality described.

This module requires no information from an external source to operate.

### 5.1.5.1 ErrorRepository

#### 5.1.5.1.1 initialize()

This function initialise the singleton ErrorRepository, if not already initialised. It ensures that the up to date error information is available for reading.

##### Input

None

##### Output

- A hard coded Error indicating failure to initialise the ER, or the NO\_ERROR value in case of success.

##### Possible Errors Generated

The following Errors may be returned:

[e27] COULD\_NOT\_INITIALIZE\_ER

#### 5.1.5.1.2 registerErrors()

This function adds a list of Errors to the ErrorRepository, by key of their EID. It validates that each EID is added only once.

##### Input

- A list of Errors to be added to the repository.

##### Output

- An Error indicating failure to register one of the errors, or the NO\_ERROR value in case of success.

##### Possible Errors Generated

The following Errors may be returned:

[e28] ERROR\_ALREADY\_EXISTS

This error indicates that two modules are attempting to use the same EID. This should be resolved at development time.

#### 5.1.5.1.3 overwriteError()

This function writes the error information that it has associated with a provided EID into the provided Error. Undefined values found in the repository will not be overwritten into the provided Error (leaving the original value of that field unchanged). If the EID is not found in the repository, the provided Error is left unchanged, but this is not regarded as an error situation. If the data in the ER is inaccessible, the provided Error is left unchanged, and the failure is logged.

##### Input

- An EID
- An Error to be filled

##### Output

- None.

##### Possible Errors Generated

None.

#### 5.1.5.1.4 getError()

This function returns a new Error, with the error information that it has associated with a provided EID. If the EID is not found in the repository, or if the data in the ER is inaccessible, the event is logged and the information associated with the UNKNOWN\_ERROR\_ID is provided in its place.

##### Input

- An EID

- Details for logging an error situation

**Output**

- An Error with the error information that it has associated with a provided EID.

**Possible Errors Generated**

None.

### 5.1.6 Client Communications Module

The Client Communications Module (CCM) manages the connection to the server, and the security aspects of the communication. Communication errors should be identified in this layer.

No automatic recovery strategies are implemented at this level, to reduce the tendency of the client to additionally load an overloaded server (which should be the cause of most recoverable errors at this layer).

The CCM also provides a service for encrypting and decrypting data buffers using the registrar's private key.

A singleton Design Patterns Elements of Reusable Object-Oriented Software, by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Communicator class implemented in this module provides all the functionality described in the interface.

This module requires that the following information be available to it from an external source:

- The url of the SRS server – passed in as a constructor parameter.
- The identifier of the registrar running the client application (not to be mixed with the registrar on behalf of which the requests are made) – passed in as a constructor parameter.
- The registrar's key identifier – passed in as a constructor parameter.
- The registrar's private key – installed together with Crypt::OpenPGP <http://rumba.pair.com/ben/perl/openpgp; V1.0; 26 February 2002..>

#### 5.1.6.1 Communicator

##### 5.1.6.1.1 new()

This function initialises the Communicator, and sets up the communication to the server

**Input**

- The url of the SRS server
- The identifier of the registrar running the client application
- The registrar's key identifier

**Output**

- An Error indicating failure to initialise the Communicator, or the NO\_ERROR value in case of success.

**Possible Errors Generated**

The following Errors may be returned:

**[e29] CM\_MISSING\_SRS\_SERVER\_URL**

**[e30] MISSING\_REGISTRAR\_ID**

**[e31] MISSING\_REGISTRAR\_KEY\_ID**

#### 5.1.6.1.2 send()

This function receives information to transmit to the SRS server. It adds the necessary identification of the registrar, all security information as required by the SRS server protocol (see Security Requirements), and any network communication specific data to the incoming information.

It relies on a timeout to recover from transmissions getting lost on the underlying communication.

Communication errors should be identified in this layer, described in an Error structure, and returned to the caller as a result of the communication.

For this purpose, if the CCM does not know if the communication succeeded (i.e. the message was successfully sent to the server without response), the request is deemed to have failed, and the CMM will provide an error message noting the uncertainty of the result.

#### Input

- A buffer of bytes to be sent to the server. To be accepted by the SRS server, the buffer should contain an XML page as defined in the SRS server interface (see XML Requests and XML Requests), but this is not validated in this layer.
- A parameter defining if this transmission requires a secure protocol or not. The Communicator uses HTTPS or HTTP accordingly.

#### Output

- An Error in case of failure of the communication, or NO\_ERROR value in case of success.
- A buffer of bytes received as a response from the server. In case of an error, the contents of the buffer will be undefined.

#### Possible Errors Generated

In addition to the possible Errors returned from the SRS server, the following Errors may be returned:

**[e32] CM\_PUB\_KEY\_NOT\_FOUND**

The client application's private key could not be found in the system users key ring.

**[e33] CM\_SIGNING\_ERROR**

Failed to sign the data buffer with the client application's private key.

**[e34] CM\_NO\_RESP\_FROM\_SERVER**

No response has been received from the SRS server before time out. This includes the case where the transmission failed, and was never received by the SRS server.

**[e35] CM\_INVALID\_SERVER\_RESP**

Could not separate the SRS server's signature from the data buffer that was received from it.

**[e36] CM\_INVALID\_SERVER\_SIG**

The response received from the server had an invalid or incorrect signature.

#### 5.1.6.1.3 encrypt()

This function shall encrypt a character buffer with the registrar's private key

#### Input

- A data buffer

#### Output

- An encrypted data buffer

- An Error indicating failure to encrypt the input buffer, or the NO\_ERROR value in case of success.

**Possible Errors Generated**

The following Errors may be returned:

[e37] MISSING\_INPUT\_FOR\_ENCRYPTION

[e38] ENCRPTION\_FAILED

**5.1.6.1.4 decrypt()**

This function shall decrypt a character buffer with the registrar's private key

**Input**

- An encrypted data buffer

**Output**

- A decrypted data buffer
- An Error indicating failure to decrypt the input buffer, or the NO\_ERROR value in case of success.

**Possible Errors Generated**

The following Errors may be returned:

[e39] MISSING\_INPUT\_FOR\_DECRYPTION

[e40] DECRPTION\_FAILED

**5.1.7 Client XML Translation Module**

The Client XML Translator Module (CXTM) is responsible for translating transactions to XML in the format required by the SRS server interface (see XML Requests & XML Requests), and passing them on to the CCM for transport to the SRS server. It also validates the structure of the returned XML, as defined by the DTD protocol.dtd; V 1.27; 20 June 2002., translates all responses from within the XML page to client native types, and connects them to the original transaction. During the translation, the CXTM detects successful or error responses for the request as a whole, and for each transaction.

The XTM also provides a service for validating any externally created XML page against the structure required by the SRS server, as defined by the DTD. This is intended for users and developers that do not wish to utilise the CXTM's translation capabilities.

A singleton Design Patterns Elements of Reusable Object-Oriented Software, by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. XmlTranslator class implemented in this module provides all the functionality described in the interface.

The version identifier of the SRS protocol that is supported, and the DTD describing the XML structure of the protocol is incorporated in the code. All changes to the protocol shall be treated as upgrades to this module.

This module requires that the following information be available to it from an external source:

- The url of the SRS server – passed in as a constructor parameter.
- The identifier of the registrar running the client application (not to be mixed with the registrar on behalf of which the requests are made) – passed in as a constructor parameter.
- The registrar's key identifier – passed in as a constructor parameter.

### 5.1.7.1 XmlTranslator

#### 5.1.7.1.1 new()

This function initialises the XmlTranslator, including a Communicator object.

##### Input

- The url of the SRS server - to be passed to the Communicator constructor.
- The identifier of the registrar running the client application - to be passed to the Communicator constructor.
- The registrar's key identifier - to be passed to the Communicator constructor.

##### Output

- An Error indicating failure to initialise the XmlTranslator (including the Communicator object if so instructed), or the NO\_ERROR value in case of success.

##### Possible Errors Generated

In addition to the possible EIDs returned from the CCM, the following Errors may be returned:

#### 5.1.7.1.2 execute()

This function receives a set of transactions to be packaged together in one request to the server. The transactions will be executed on the server in the order that they are placed in the request. If the request as a whole succeeds, each transaction can succeed or fail on its own merit (other than dependencies arising from the internal order between the transactions), and will receive its own data in a response attached to the original Transaction.

The Client XML Translator Module (CXTM) writes the request wrapper and all the transactions into an XML page that complies with the SRS server interface, and passes them to the CCM for transport to the server. If one of the transactions requires a secure communication with the SRS server, the CCM will be so instructed. On success of the communication, it will analyse the XML pages returned from the CCM, identifying success or failure of the overall requests. If a request succeeds, the CXTM will extract the transaction results from the XML format, translate them into either a Response or an Error, and match each one with the original Transaction. A failure returned from the CCM will be returned to the caller as is.

For this purpose, if the CXTM does not know if the request as a whole, or a single transaction, succeeded (i.e. the request was successfully sent to the server without response), the request or transaction is deemed to have failed, and the CXTM will provide an error message noting the uncertainty of the result.

##### Input

- A Request to be sent to the server.
- A flag instructing whether to validate the response from the server with the DTD. If TRUE - an ErrorResponse is returned in place of the original Response from the SRS server. If FALSE - the original response is passed on as is. The latter case may be useful if the client wishes to glean some information from the Response, regardless of the error. The default value is TRUE, and this option is not recommended as it actually hides a version discrepancy between the client and the SRS server, which may cause the information in the response to be interpreted incorrectly.

##### Output

- An Error indicating failure to execute the request as a whole, or the NO\_ERROR value in case of success.
- The individual Responses or Errors relating to each Transaction in the incoming list are assigned to the original Transaction.

**Possible Errors Generated**

If validate() fails on the request that it receives as input, the Error from that will be returned.

In addition to the possible Errors returned from the CCM, the following Errors may also be returned:

**[e41] XT\_EXEC\_MISSING\_ARG**

An expected parameter was missing in the call to execute().

**[e42] XT\_INVALID\_TRANSACTION**

One of the transactions passed in to execute() was invalid in a manner that was not detectable by its validate() function.

**5.1.7.1.3 validateXml()**

This function validates that the input complies with the XML DTD. As a by-product, it may check XML related syntax and structure, but the focus of this function is to verify that the SRS server shall accept the XML page as syntactically valid.

This is a class method.

**Input**

- A string representing an XML page

**Output**

- An Error indicating failure of the input to comply with the required SMS server interface XML structure, or the NO\_ERROR value in case of full compliance.

**Possible EIDs Generated**

The following EIDs may be returned:

**[e43] XT\_VALXML\_MISSING\_ARG**

An expected parameter was missing in the call to validateXml().

**5.1.8 Client Request Manager Module**

TBD.

**5.1.9 Client User Interface Module**

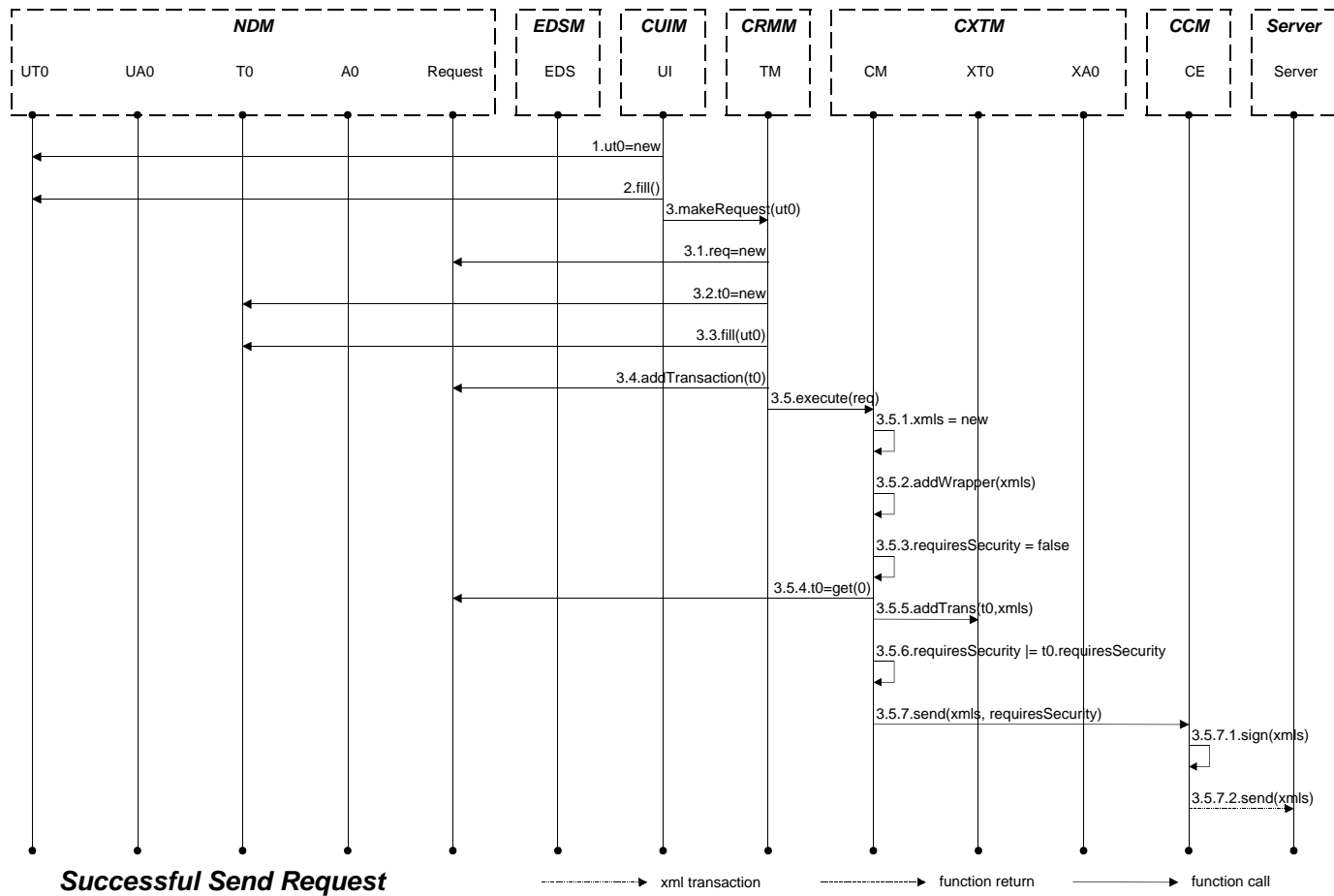
TBD.

**5.2**

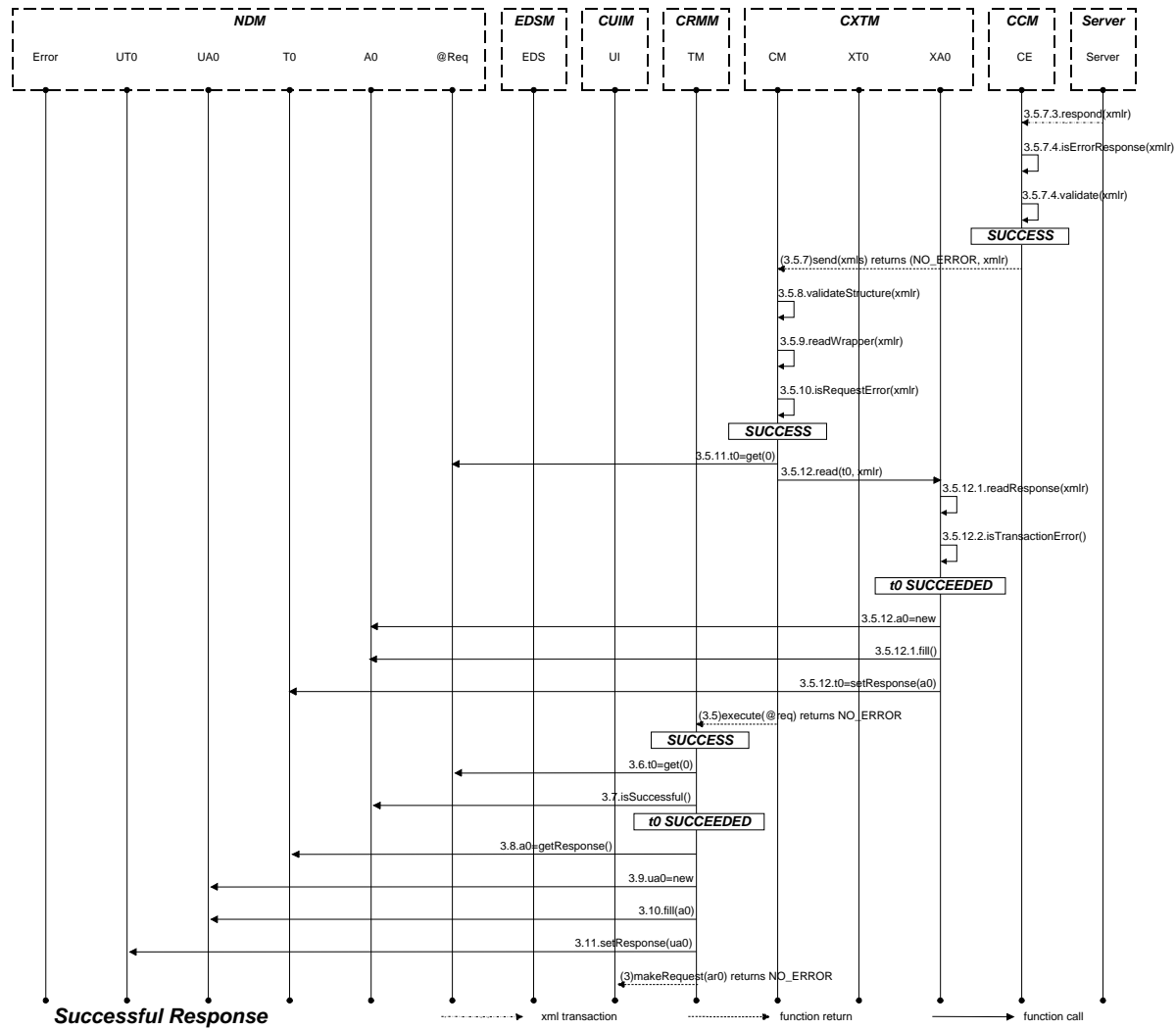
## **5.3 Operation Sequences**

### **5.3.1 Success Sequences**

#### **5.3.1.1 Successful sending of a request**

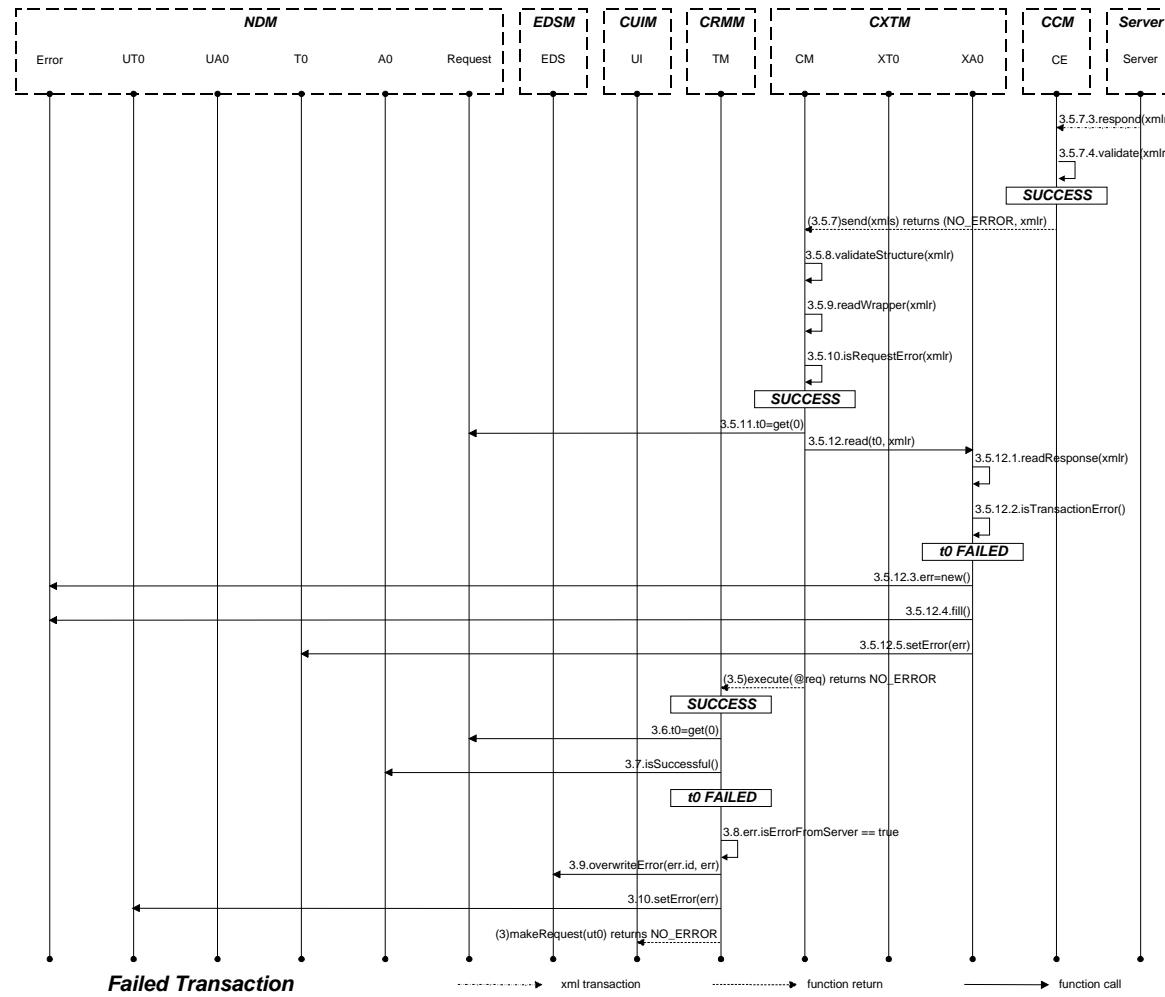


**5.3.1.2 Successful reception of a result**

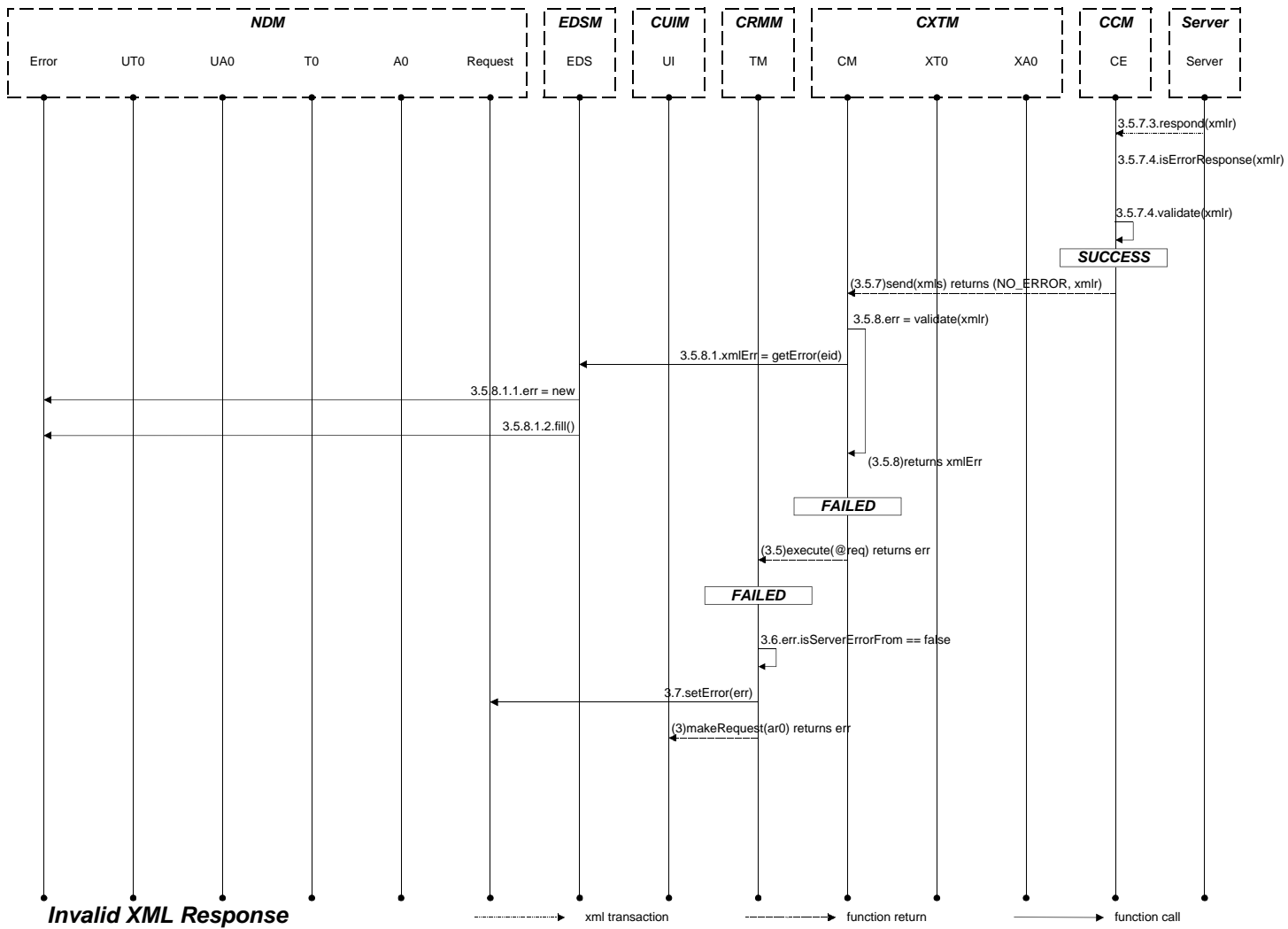


## **5.3.2 Error Sequences**

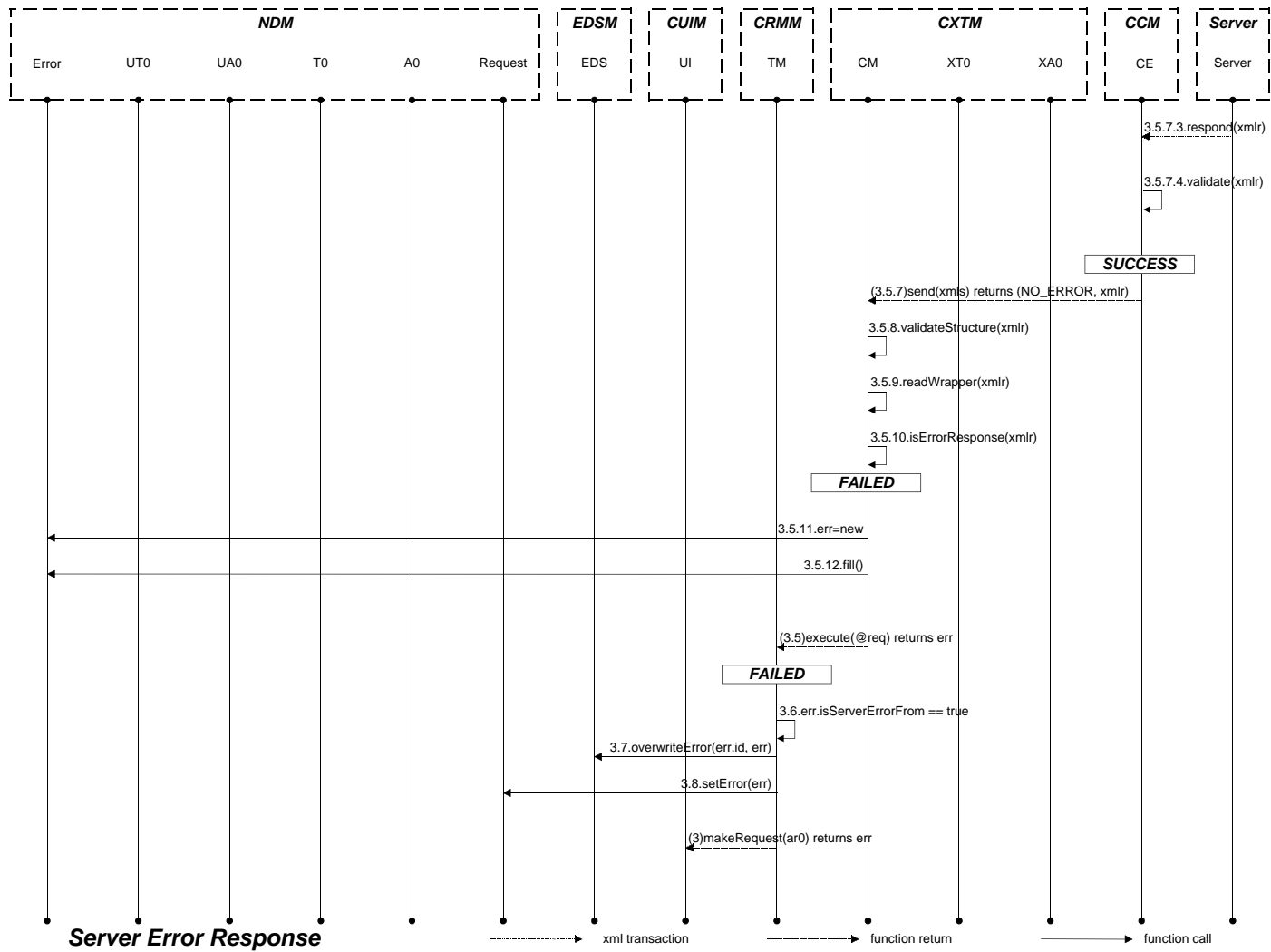
### **5.3.2.1 Failed Transaction**



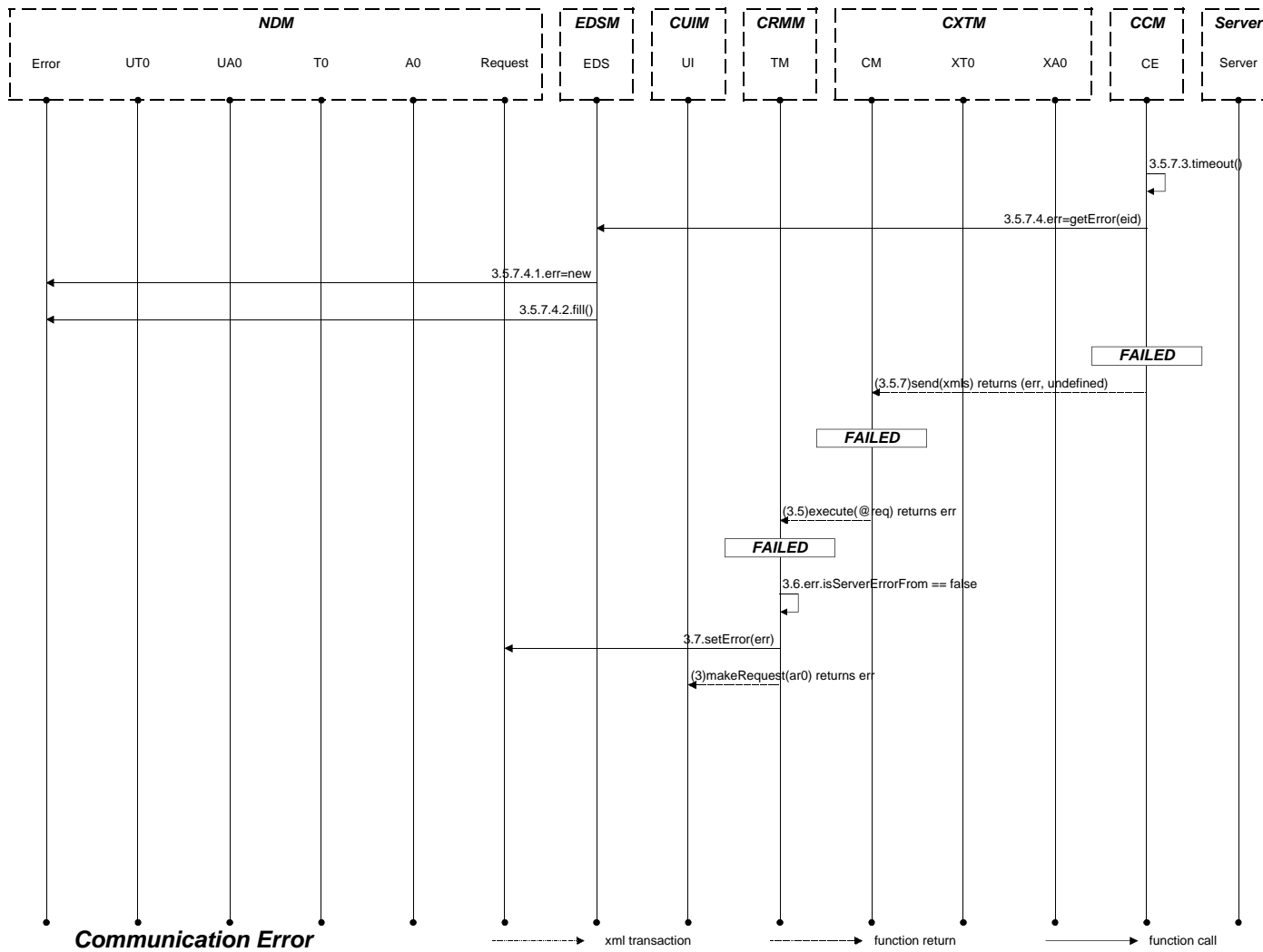
### 5.3.2.2 Invalid XML Response



### 5.3.2.3 Server Error Response



#### 5.3.2.4 Communication Error





## 6 Glossary

<b><i>Term</i></b>	<b><i>Description</i></b>
<b>CCM</b>	Client Communications Module
<b>Class Method</b>	A method of a class that does not require an instance to be called.
<b>CNDM</b>	Client Native Data Module
<b>CRMM</b>	Client Request Manager Module
<b>CUIM</b>	Client User Interface Module
<b>CXTM</b>	Client XML Translation Module
<b>DNS</b>	Domain Name Server
<b>DTD</b>	XML Document Type Definition
<b>ER</b>	Error Repository
<b>ERM</b>	Error Repository Module
<b>Effective Registrar</b>	The registrar on behalf of whom an SRS request is made
<b>EID</b>	Error IDentifier
<b>LM</b>	Logger Module
<b>LMGM</b>	Log Message Generator Module
<b>NLS</b>	National Language Support
<b>RIK</b>	Registrar Implementation Kit
<b>SRS</b>	Shared Registry System
<b>UDAI</b>	Unique Domain Authentication Identifier
<b>UID</b>	User Interface Data

# 7 Appendices

## 7.1 Appendix A – Request validate() Errors

### [e44] INVALID\_EFFECTIVE\_REGISTRAR

The effectiveRegistrar field is invalid. It should contain a none empty string.

### [e45] TM\_GETMSG\_NOT\_ALONE

A GetMessages transaction must be the only transaction in its request.

## 7.2 Appendix B – Transaction validate() Errors

### 7.2.1 Basic Data Validation Errors

#### 7.2.1.1 DateTime Validation Errors

##### [e46] UNEXPECTED\_FIELD

An unexpected data field was encountered as part of the transaction.

##### [e47] MISSING\_YEAR

##### [e48] INVALID\_YEAR

The value of the Year field should be an integer in the range.

### 7.2.2 Common Transaction Errors

The following EIDs may be returned from any Transaction:

##### [e49] INVALID\_TRANSACTION\_TYPE

The transactionType field is NO\_TRANSACTION or undefined or unknown.

##### [e50] UNEXPECTED\_FIELD

An unexpected data field was encountered as part of the transaction.

##### [e51] MISSING\_ACTION\_ID

The ActionId field is undefined. This will only be returned from Transactions that have an ActionId field.

##### [e52] MULTIPLE\_ACTION\_ID

Multiple ActionId values found where only one is allowed. This will only be returned from Transactions that have an ActionId field.

##### [e53] MALFORMED\_TRANSACTION\_RESPONSE

The response attached to the transaction is malformed in an unspecified manner.

### 7.2.3 DomainCreate validation Errors

##### [e54] MISSING\_REGISTRAR\_ID

The RegistrarId field is undefined.

##### [e55] MULTIPLE\_REGISTRAR\_ID

Multiple RegistrarId values found where only one is allowed.

[e56] **MISSING\_DOMAIN\_NAME**

[e57] **MULTIPLE\_DOMAIN\_NAME**

[e58] **INVALID\_DELEGATE**

The value of the Delegate field should be either 0 or 1.

[e59] **MULTIPLE\_DELEGATE**

[e60] **MISSING\_REGISTRANT\_CONTACT**

[e61] **MULTIPLE\_REGISTRANT\_CONTACT**

[e62] **MULTIPLE\_REGISTRANT\_REF**

[e63] **MULTIPLE\_ADMIN\_CONTACT**

[e64] **MULTIPLE\_TECH\_CONTACT**

[e65] **INVALID\_TERM**

The value of the Term field should be a positive integer.

[e66] **MULTIPLE\_TERM**

[e67] **MULTIPLE\_AUDIT\_TEXT**

## 7.2.4 DomainUpdate validation Errors

[e68] **MISSING\_REGISTRAR\_ID**

The RegistrarId field is undefined.

[e69] **MULTIPLE\_REGISTRAR\_ID**

Multiple RegistrarId values found where only one is allowed.

[e70] **INVALID\_STATUS**

The value of the Status field should be either "Available" or "PendingCancel".

[e71] **MULTIPLE\_STATUS**

[e72] **INVALID\_DELEGATE**

The value of the Delegate field should be either 0 or 1.

[e73] **MULTIPLE\_DELEGATE**

[e74] **INVALID\_TERM**

The value of the Term field should be a positive integer.

[e75] **MULTIPLE\_TERM**

[e76] **MULTIPLE\_REGISTRANT\_CONTACT**

[e77] **MULTIPLE\_REGISTRANT\_REF**

[e78] **MULTIPLE\_ADMIN\_CONTACT**

[e79] **MULTIPLE\_TECH\_CONTACT**

[e80] **MULTIPLE\_UDAI**

[e81] **INVALID\_NEW\_UDAI**

The value of the NewUDAI field should be either 0 or 1.

[e82] **INVALID\_RENEW**

The value of the Renew field should be either 0 or 1.

[e83] **INVALID\_LOCK**

The value of the Lock field should be either 0 or 1.

[e84] **INVALID\_CANCEL**

The value of the Cancel field should be either 0 or 1.

[e85] **MULTIPLE\_AUDIT\_TEXT**

## 7.2.5 DomainDetailsQry validation Errors

[e86] **MISSING\_REGISTRAR\_ID**

The RegistrarId field is undefined.

[e87] **MULTIPLE\_REGISTRAR\_ID**

Multiple RegistrarId values found where only one is allowed.

[e88] **INVALID\_STATUS**

The value of the Status field should be either "Available" or "PendingCancel".

[e89] **MULTIPLE\_STATUS**

[e90] **INVALID\_DELEGATE**

The value of the Delegate field should be either 0 or 1.

[e91] **MULTIPLE\_DELEGATE**

[e92] **INVALID\_TERM**

The value of the Term field should be a positive integer.

[e93] **MULTIPLE\_TERM**

[e94] **MULTIPLE\_SEARCH\_DATE\_RANGE**

[e95] **MULTIPLE\_RESULT\_DATE\_RANGE**

[e96] **MULTIPLE\_REG\_DATE\_RANGE**

[e97] **MULTIPLE\_LOCK\_DATE\_RANGE**

[e98] **MULTIPLE\_CANCEL\_DATE\_RANGE**

[e99] **MULTIPLE\_BILLED\_UNTIL\_DATE\_RANGE**

[e100] **MULTIPLE\_REGISTRANT\_CONTACT**

[e101] **MULTIPLE\_REGISTRANT\_REF**

[e102] **MULTIPLE\_ADMIN\_CONTACT**

[e103] **MULTIPLE\_TECH\_CONTACT**

[e104] **INVALID\_FIELD\_LIST**

The FieldList field contains an invalid value. The only values allows are "DomainName", "Status", "Delegate", "Term", "NameServers", "RegistrantContact", "RegistrantRef", "RegistrarId", "RegistrarName", "AdminContact", "LastActionId", "RegisteredDate", "LastChangeDate", "AuditText", "LockedDate", "BilledUntil", "CancelledDate", "LastChangedBy and "TechnicalContact".

## 7.2.6 RegistrarCreate validation Errors

### 7.2.7 RegistrarUpdate validation Errors

### 7.2.8 RegistrarDetailsQry validation Errors

### 7.2.9 RegistrarAccountQry validation Errors

### 7.2.10 GetMessages validation Errors

### 7.2.11 Whols validation Errors

### 7.2.12 ActionDetailsQry validation Errors

## 7.3 Appendix C –Error Severity Levels

The error severity levels are classified according to the possible affect on the service to the client user, and what actions he can take to remedy that. For example, if the SRS server is unavailable, the client will provide no service but the registrar can do nothing about it, so it is treated as a resource unavailability error. Where the cause of the error can not be pinpointed, the severity is assigned by the worst case possible cause.

### 7.3.1 Client Unavailable (high)

- Unknown server error
- Suspected communication error
- Client initialization error
- Client / server protocol mismatch
- Unknown client error
- Suspected bug in client code
- Suspected bug in server code

### 7.3.2 Persistent Failure Affecting Multiple Transaction Types

- Invalid XML
- XML input fails to match DTD
- Transaction transmitted via insecure medium
- Missing mandatory input for transaction / request
- Missing mandatory output from transaction / request
- Suspected bug in client code
- Suspected bug in server code

### 7.3.3 Persistent Failure Affecting a Single Transaction Type

- Missing mandatory input for transaction
- Attempt to execute action without permission
- Missing mandatory output from transaction

### 7.3.4 Resource Unavailable

- Attempt to write while in read only mode
- Server resource error
- Server initialization error
- Server timeout
- Client resource error

### 7.3.5 Business Logic Error (low) (input dependant errors)

- Operation on a domain not owned by registrar
- Single Transaction business logic error
- Response too large

## 7.4 Appendix D –Coding Guidelines

The following coding practices were followed when developing the Perl code:

### 7.4.1 Compilation directives

- [1]All files should include the “use strict” & “use warnings” directives.
- [2]All functions should declare expected parameters in a way that shall maximise “compile” time checking, without restricting their functionality.

### 7.4.2 Naming conventions

- [3]All function names should use mixed case word separation, beginning with lowercase.
- [4]All constants names shall be in underline separated uppercase.
- [5]All package names should use mixed case word separation, beginning with uppercase.
- [6]Element names should be meaningful, conveying what they represent (for variables) and what they do (for functions)

### 7.4.3 Documentation

- [7]All files should have a header, including a cvs version macro, and the author name.
- [8]All classes should have a header with a brief description of their purpose and data structure
- [9]All Functions should have a header intended primarily for users of the function, including a description of expected input and output, and a brief description of what they do (not how!). They may include a brief description of the implementation principles, if required, at the bottom of the header.
- [10]The unobvious pieces of code should be documented inline.
- [11]Document all that is required, but only what is required – no blabbering.