Developer's Guide
Table of Contents

Developer's Guide

1. Audience

This guide provides the next level of technical detail beyond the Technical Overview and is targeted at the software development professional. It assumes that you have already read the Technical Overview document to familiarize yourself with the basics of the SRS.

2. Goals

The goals of this document are to help you determine which API to use, familiarize you with all the SRS commands, suggest development strategies and provide assistance with using GnuPG, which is used by all SRS clients to handle digital signatures. The concepts discussed in this document are common to all SRS client implementations, regardless of API choice.

3. Choosing an API

Your choice of API may be entirely dependent on your infrastructure (i.e., the operating system of your web server). If you need to choose between APIs, though, this section is designed to help by describing how each API is implemented. You may want to skip this section if you are already committed to a particular API. Note that each API has a corresponding Reference Guide that details installation and configuration procedures specific to it.

3.1 Common Elements

As discussed in the Technical Overview, an SRS client must perform the following three tasks:

1. Create outgoing command packets and parse incoming response packets according to the SRS protocol specification

2. Digitally sign outgoing packets and verify the signature of incoming packets

3. Provide for SSL encrypted HTTP communication

Beyond platform and language considerations, how each API accomplishes these tasks is the main differentiating factor.

One component common to all APIs is the use of GnuPG for the handling of digital signatures. Regardless of API, a GnuPG installation is required on your target platform. A later section in this document presents an overview of GnuPG usage as it relates to your SRS client.

3.2 Perl API for Linux

This API is the best choice for those developing a web-based solution on Linux platforms (e.g., if you are selling names on a website hosted by a Linux server such as Apache).

The entire API is implemented in a single module called DotSRS_Client. It takes care of the overall process of sending commands to the SRS server and receiving responses; it handles the authentication and transmission of packets automatically through the use of support modules. These support modules are required and readily available from CPAN, the Comprehensive Perl Archive Network. The support modules provide interfaces to GnuPG (for digital signatures) and to native libraries that implement SSL encrypted HTTP (OpenSSL, Crypto and standard sockets libraries).

To process a command, a script calls an appropriate DotSRS_Client subroutine with the required parameters (usually a hash reference) and handles the results (usually returned in a hash reference).
Summary:

- Best choice for web-based development
- Dependent on freely available Perl modules (from CPAN)
- Dependent on GnuPG
- API is implemented in a single source file

3.3 C API for Linux

Use the C API on Linux if you are unable to use Perl on Linux, if you are building a custom GUI interface for registering domain names, or in any number of scenarios where you need to create an executable application. The C code is written to be as portable as possible and can be used on other Unix-like operating systems, though we only support the Linux 'flavor' of the various Unix clones. Modifying the API for use on unsupported platforms is entirely your responsibility if you should so endeavor.

The C API for Linux consists of source code files that should be compiled and linked to your SRS client application, along with the required OpenSSL and Crypto libraries. The main `DotSRS_Client.c` file provides functions for all SRS commands. All cryptographic processes and secure HTTP communications are performed automatically by the API routines. Standard socket routines and the OpenSSL library are used for communicating with the SRS server via HTTPS. Interaction with GnuPG is achieved by forking and piping data through the GnuPG executable (`gpg.exe`). In addition, support routines for handling lists of name-value pairs are provided.

From a client development perspective, all the client code must do is call a `DotSRS_Client` function with the appropriate parameters (usually a list of name-value pairs) and process the results (usually returned in a handle to a list of name-value pairs).
Summary:

- Best choice for developing an executable client on Linux
- Dependent on OpenSSL and Crypto libraries (freely available)
- Dependent on GnuPG

3.4 SRS Component for Windows

The SRS Component for Windows is a dual-interface COM component, which makes it ideal for use in a multitude of Windows development environments, including ASP. In fact, it was developed with ASP in mind as its target environment. The component can be accessed easily from scripting languages, Visual Basic, Visual C++ and any other environment that supports Automation or raw COM interfaces.

The entire component is housed in a single DLL, `SRSplus.dll`. It exposes a single interface, `ISrs`, which provides access to all of the SRS commands. All configuration information for the component is stored in the registry. As with the other APIs, the SRS component uses GnuPG for its digital signature handling. In addition, it requires WinSock 2.0 and the CryptoAPI with SecureChannel for performing secure HTTP. It also requires the Windows Scripting Runtime for access to the `Scripting.Dictionary` object, which the component uses to exchange data with its clients. Except for GnuPG, all of these technologies are typically installed automatically by later versions of windows (e.g., Windows 2000 and above) and are readily available from Microsoft for most Windows versions.

From a development perspective, all the client code must do is instantiate and initialize an `ISrs` object, invoke `ISrs` methods with appropriate parameters, process the results (usually returned in a `Scripting.Dictionary` object), then uninitialize and destroy the object.
Summary:

- Use for all Windows development, especially ASP pages.
- Dependent on GnuPG
- Dependent on WinSock 2.0, CryptoAPI with SecureChannel, and the Windows Scripting Runtime

4. SRS Command Reference

This section details each SRS command and its usage. Though each API has specific requirements for parameter types passed into SRS functions, they all conceptually use the same two parameter types: ***string*** and ***name-value pair*** lists. In the Perl API, these correspond to scalar and associative array types. In the C APIs, they correspond to `char*` and `NameValueList` types. The equivalent type used by the SRS Component for Windows is the `IDictionary` interface. Specific function signatures, return values and code samples are provided in the API Reference documentation for each of the APIs. In the rest of this section, we will make use of the conceptual terms ***string*** and ***name-value pair*** in order to remain API neutral. You may want to have the specific Reference for the API you will be using on hand as you read this section.

4.1 Important Considerations

Understanding the information in this section is essential to achieving a successful implementation. Keep the following in mind as you read about the SRS commands:

NAME-VALUE PAIRS The values in name-value pairs are always strings, even if the information is numeric in nature. It is the programmer's responsibility to perform type conversions on returned data, if necessary. This primarily affects C programmers.

The SRS server does not return name-value pairs in any particular order. Your code should be able to handle any arbitrary ordering of the returned data.

DATES All date values are returned from the SRS server in Unix (Linux) epoch time format, which is a count of the number of seconds since 1970/01/01 00:00 UTC.

MULTIPLE PARAMETER VALUES Some commands accept or return multiple entries of the same named value. For example, server information returned from a **Whois** command may return more than one name server if multiple name servers are defined for the domain. In this case, the name part of the name-value pairs will have a numerical index appended to the name part, separated by a space. For example, if the domain 'mydomain' has two name servers defined, the returned name-value pairs could look like this:

Name	Value
DNS SERVER NAME 1	"ns1.mydomain.net"
DNS SERVER NAME 2	"ns2.mydomain.net"

In the documentation, all parameters that exhibit this behavior have the string "1…n" appended to the parameter name. For example, in the above example, the documentation would list the parameter as DNS SERVER NAME 1…n.

UNIQUE IDs
Commands that modify data in the SRS database allow the client to pass an optional transaction ID to the SRS server. These IDs are persisted as strings, so you may use any combination of letters and numbers for the string. If supplied, the ID string will be added to the SRS server log alongside the transaction. **Do not pass a NULL for the transaction ID**, but instead pass "0", "none" or some other meaningless string.

Commands that take a transaction ID parameter also return a request ID. It is generated by the server and is also logged alongside every transaction. These IDs are useful for offline inquiries if a historical log search must be performed (e.g., in the case of a disputed transaction). Think of them as receipts or confirmation codes. If you decide *not* to send transaction IDs, it is suggested that you log the request IDs you get back as a safeguard.

ERRORS If a command fails, descriptive error strings will be returned in one or more namevalue pairs named ERROR 1…n. For example, if required parameters are missing from the **CreateContact** input, the command will fail and return multiple error strings, one for each missing piece of information:

Name	Value
ERROR 1	"Please enter a phone number."
ERROR 2	"The email address you entered is invalid."
ERROR 3	"Please select a country."

Note that if there is only one error string, that the appended error number is optional and might not appear. The following is a legal response

Name	Value
ERROR	"Please enter a phone number."

SPECIFYING DOMAIN AND TLD PARAMETERS When specifying DOMAIN and TLD parameters, periods are allowed only when the domain or tld has multiple levels. For example, do not submit ".com" as the TLD parameter for a .com tld. Instead, submit "com" as the tld. On the other hand, it is legal to submit "co.uk" since it is a multiple level tld. Examples:

TLD String	Legal?
"co.uk"	YES
".com"	NO
"com"	YES
"com.mx"	YES

For DOMAIN parameters, currently the only legal multiple level domains are in the .name tld. For domains in the .name tld it is legal to include a period in the DOMAIN parameter to separate the two domain levels. For example, "firstname.lastname" would be a legal value for the DOMAIN parameter when registering the domain "firstname.lastname.name".

REGISTRATION LENGTH Some tlds have a minimum term of registration greater than one year. As of this writing, the following tlds have a minimum 2 year registration term: com.mx, co.uk, me.uk, and org.uk. The controlling registries for each respective tld set forth the minimum registration term policies; you are urged to verify the current policies with each registry.

IP ADDRESSES Do not hardcode "dotted" IP addresses for the SRSplus servers in any code or hostfiles. Always prefer the symbolic addresses (i.e., srs.srsplus.com & testsrs.srsplus.com) since the actual IP addresses are subject to change.

4.2 The Query Commands

These commands can be thought of as read-only commands since they make no modifications to any data in the SRS system. Of these, **DomainInfo** is perhaps the most important, since the data returned by the command will be needed to register the queried domain. In alphabetical order, the query-only commands are:

<table>
<tr><th colspan="2">AccountBalance</th></tr>
<tr><td>Description</td><td>Returns balance information from your partner account</td></tr>
<tr><td>Perl Subroutine:</td><td>($status, \%balance_ref) = account_balance ($tld)</td></tr>
<tr><td>C Function:</td><td>int SrsAccountBalance
(
char* [in] TLD,
NameValueList** [in,out] ppResponse
);</td></tr>
<tr><td>ISrd Method:</td><td>HRESULT AccountBalance
(
[in]BSTR bstrTLD,
[out,retval]IDictionary** ppResult
);</td></tr>
<tr><td>Input:</td><td>Optional: A string specifying the TLD account to check. Omitting the TLD parameter will result in "tv" being used by default.</td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>BUYING POWER</td><td>Dollar amount with 2 decimal point precision</td></tr>
<tr><td>STORED VALUE</td><td>Dollar amount with 2 decimal point precision</td></tr>
<tr><td>UNPAID CHARGES</td><td>Dollar amount with 2 decimal point precision</td></tr>
</table></td></tr>
</table>

Notes:	STORED VALUE is the current balance in your deposit account. This is the account that gets debited when you register a domain. Normally, when your account balance dips below a certain threshold (currently US$100 & $50), you will automatically be sent an email notifying you that your balance is low. If your balance should fall to zero, your buying privileges may be suspended until you deposit funds into your account. BUYING POWER and UNPAID CHARGES are legacy features from a time when partner accounts had a line of credit with SRSplus. In this case, STORED VALUE is the amount of funds available in your registration fund. The input TLD parameter is a legacy feature from a time when partner accounts were tied to specific TLDs. This is no longer the case, so any TLD value will achieve the same results. It is best to simply specify an empty string for the value.

DomainInfo	
Description:	Returns availability for a specific domain name.
Perl Subroutine:	`(\%info_ref) = domain_info ($domain, $tld)`

C Function:	`int SrsDomainInfo`
	`(`
	`char* [in] domain,`
	`char* [in] TLD,`
	`NameValueList** [in,out] ppResponse`
	`);`

ISrd Method:	`HRESULT DomainInfo`
	`(`
	`[in]BSTR bstrDomain,`
	`[in] BSTR bstrTLD,`
	`[out,retval]IDictionary** ppResult`
	`);`

Input:	Required: A string specifying the second level domain to check. Note that the “domain” parameter is the plain second level domain name. Do not include the TLD extension (i.e., pass in “mydomain”, not “mydomain.tv”) Optional: A string specifying the TLD. Omitting the TLD parameter will result in “tv” being used by default.
Output:	A name-value list with the following name-value pairs: **Name** / **Value** DOMAIN STATUS — “FIXED” or “UNAVAILABLE” PRICE PRICE — Whole dollar amount (i.e., “50”, not “50.00”) of the wholesale price. EFFECTIVE PRICE — The price that the partner will be charged.
Notes:	This command should be used to determine if a domain is available and the price for which it is being offered. A status of UNAVAILABLE means the domain has either already been sold, or is for any other reason not available. A status of FIXED means that the domain is available. IMPORTANT: Always base your business logic on the PRICE and EFFECTIVE PRICE values. You will need the pricing information for use in the **RegisterDomain** command.

<table>
<tr><th colspan="2">MultiDomainInfo</th></tr>
<tr><td>Description:</td><td>Returns availability and pricing information for many domain names at once.</td></tr>
<tr><td>Perl Subroutine:</td><td>(\%info_ref) = multidomain_info (\%query_ref)
C Function: int SrsMultiDomainInfo
(
NameValueList* [in] pQuery,
NameValueList** [in, out] ppResponse
);</td></tr>
<tr><td>ISrd Method:</td><td>HRESULT MultiDomainInfo
(
[in]IDictionary* pQuery,
[out,retval]IDictionary** ppResponse
);</td></tr>
<tr><td>Input:</td><td>A name-value list with the following format:
<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN 1…n</td><td>Required: Second level domain name to check</td></tr><tr><td>TLD 1…n</td><td>Optional: Corresponding TLD for domain. If omitted, “tv” will be used.</td></tr></table>
Note that the “domain” parameters are the plain second level domain names. Do not include the TLD extension (i.e., pass in “mydomain”, not “mydomain.tv”)

EXAMPLE:

<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN 1</td><td>“thisdomain”</td></tr><tr><td>TLD 1</td><td>“tv”</td></tr><tr><td>DOMAIN 2</td><td>“thatdomain”</td></tr><tr><td>TLD 2</td><td>“com”</td></tr></table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following possible name-value pairs:

<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN STATUS 1…n</td><td>“FIXED” or “UNAVAILABLE”</td></tr><tr><td>PRICE 1…n</td><td>Whole dollar amount (i.e., “50”, not “50.00”) of the wholesale price.</td></tr><tr><td>EFFECTIVE PRICE 1…n</td><td>The price that the partner will be charged.</td></tr></table>
EXAMPLE:

<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN STATUS 1</td><td>“FIXED”</td></tr><tr><td>PRICE 1</td><td>“50”</td></tr><tr><td>EFFECTIVE PRICE 1</td><td>“35”</td></tr><tr><td>DOMAIN STATUS 2</td><td>“FIXED”</td></tr><tr><td>PRICE 2</td><td>“2000”</td></tr><tr><td>EFFECTIVE PRICE 2</td><td>“1750”</td></tr></table></td></tr>
<tr><td>Notes:</td><td>This command retrieves the same information as DomainInfo, except that it allows you to query more than one domain at a time (i.e., batch queries). Refer to the notes above for DomainInfo, since they also apply to MultiDomainInfo.</td></tr>
</table>

<table>
<tr><th colspan="2">Whois</th></tr>
<tr><td>Description:</td><td>Obtains domain name server and contact information for the given domain and TLD combination.</td></tr>
<tr><td>Perl Subroutine:</td><td>(\%info_ref) = whois ($domain, $tld)</td></tr>
<tr><td>C Function:</td><td>int SrsWhois
(
char* [in] domain,
char* [in] TLD,
NameValueList** [in, out] ppResponse
);</td></tr>
<tr><td>ISrd Method:</td><td>HRESULT Whois
(
[in]BSTR bstrDomain
[in] BSTR bstrTLD,
[out,retval]IDictionary** ppResponse
);</td></tr>
<tr><td>Input:</td><td>Required: A string specifying the second level domain (SLD) for which to obtain Whois information. Note that the “domain” parameter is the plain SLD name. Do not include the TLD extension (i.e., pass in “mydomain”, not “mydomain.tv”)

Optional: A string specifying the TLD. Omitting the TLD parameter will result in “tv” being used by default.</td></tr>
<tr><td>Output:</td><td>A name-value list with the following possible name-value pairs:

NORMAL SET:

<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>REGISTRATION DATE</td><td>Date registered (in epoch time)</td></tr>
<tr><td>EXPIRATION DATE</td><td>Date this registration will expire (in epoch time)</td></tr>
<tr><td>DNS SERVER NAME 1…n</td><td>Name server(s) specified for this domain</td></tr>
<tr><td>FNAME RESPONSIBLE PERSON</td><td>First name</td></tr>
<tr><td>LNAME RESPONSIBLE PERSON</td><td>Last name</td></tr>
<tr><td>ORGANIZATION RESPONSIBLE</td><td>PERSON Business or Organization</td></tr>
<tr><td>FNAME TECHNICAL CONTACT</td><td>First name</td></tr>
<tr><td>LNAME TECHNICAL CONTACT</td><td>Last name</td></tr>
<tr><td>ORGANIZATION TECHNICAL CONTACT</td><td>Business or Organization</td></tr>
<tr><td>FNAME BILLING CONTACT</td><td>First name</td></tr>
<tr><td>LNAME BILLING CONTACT</td><td>Last name</td></tr>
<tr><td>ORGANIZATION BILLING CONTACT</td><td>Business or Organization</td></tr>
<tr><td>FNAME ADMIN CONTACT</td><td>First name</td></tr>
<tr><td>LNAME ADMIN CONTACT</td><td>Last name</td></tr>
<tr><td>ORGANIZATION ADMIN CONTACT</td><td>Business or Organization</td></tr>
</table>

EXTENDED SET: (everything in the normal set plus these)
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>WHOIS SERVER</td><td>The whois server for this domain if other than whois.srsplus.com</td></tr>
<tr><td>PRICE</td><td>The one-year wholesale registration fee</td></tr>
<tr><td>EFFECTIVE PRICE</td><td>The actual price charged to the partner, taking all discounts and/or commissions into account.</td></tr>
<tr><td>AUTOCHARGE</td><td>Current automatic renewal setting. Returns 1 or 0 (1=on, 0=off)</td></tr>
<tr><td>RESPONSIBLE PERSON</td><td>Current contact ID</td></tr>
<tr><td>TECHNICAL CONTACT</td><td>Current contact ID</td></tr>
<tr><td>BILLING CONTACT</td><td>Current contact ID</td></tr>
<tr><td>ADMIN CONTACT</td><td>Current contact ID</td></tr>
<tr><td>ADDRESS1 RESPONSIBLE</td><td>1st line of address information</td></tr>
</table>
</td></tr>
</table>

PERSON	
ADDRESS2 RESPONSIBLE PERSON	2nd line of address information
CITY RESPONSIBLE PERSON	City
PROVINCE RESPONSIBLE PERSON	Province (State)
POSTAL CODE RESPONSIBLE PERSON	Postal Code (Zip code)
COUNTRY RESPONSIBLE PERSON	Two letter country code
PHONE RESPONSIBLE PERSON	Phone number
EMAIL RESPONSIBLE PERSON	Email address
ADDRESS1 TECHNICAL CONTACT	1st line of address information
ADDRESS2 TECHNICAL CONTACT	2nd line of address information
CITY TECHNICAL CONTACT	City
PROVINCE TECHNICAL CONTACT	Province (State)
POSTAL CODE TECHNICAL CONTACT	Postal code (Zip code)
COUNTRY TECHNICAL CONTACT	Two letter country code
PHONE TECHNICAL CONTACT	Phone number
EMAIL TECHNICAL CONTACT	Email address
ADDRESS1 BILLING CONTACT	1st line of address information
ADDRESS2 BILLING CONTACT	2nd line of address information
CITY BILLING CONTACT	City
PROVINCE BILLING CONTACT	Province (State)
POSTAL CODE BILLING CONTACT	Postal code (Zip code)
COUNTRY BILLING CONTACT	Two letter country code
PHONE BILLING CONTACT	Phone number
EMAIL BILLING CONTACT	Email address
ADDRESS1 ADMIN CONTACT	1st line of address information
ADDRESS2 ADMIN CONTACT	2nd line of address information
CITY ADMIN CONTACT	City
PROVINCE ADMIN CONTACT	Province (State)
POSTAL CODE ADMIN CONTACT	Postal code (Zip code)
COUNTRY ADMIN CONTACT	Two letter country code
PHONE ADMIN CONTACT	Phone number
EMAIL ADMIN CONTACT	Email address
DNS TYPE	DNS setting type

Notes:

Notes: The potential data returned by the **Whois** command depends on whether you are the partner-of-record for the domain specified in the query (i.e., whether you registered it). If so, you will receive all available data from the normal *and* extended sets. If not, you will only receive data available from the normal set.

Only the available data for the domain is returned. For example, if no DNS server information has ever been set, there will be no DNS SERVER 1…n name-value pairs returned.

Depending on the DNS type the domain is set for, different information will be returned.

- PARKED has no additional parameters
- YOUSERVE is our traditional hosting
- WESERVE we will serve the A CNAME and MX records for a domain

If the domain is not registered, this command will return an error. Note that this is not an appropriate command to use for determining domain availability (see **DomainInfo**).

IMPORTANT: The **Whois** command only returns information for domains that were registered through SRSplus. It does ***not*** retrieve its results from the general referral whois system and cannot be used to retrieve information about an arbitrary domain.

4.3 The Contact Management Commands

These commands are used to create, modify and retrieve information about contact records for your customers. When you register a domain name, you must provide the contact IDs of both a `RESPONSIBLE PERSON` and a `TECHNICAL CONTACT`. Using a contact ID of 0 (zero) in one of these fields is a special case that will use the contact information in your partner account profile as the contact information for this field. This is perfectly appropriate for use in the `TECHNICAL CONTACT` field, but it should only be used in the `RESPONSIBLE PERSON` field if you, the partner, are the actual customer buying the domain name. The `RESPONSIBLE PERSON` contact is always required to be the actual customer purchasing the domain.

NOTE: The `RESPONSIBLE PERSON` contact will appear as the Registrant in inquiries made through the whois server. The `TECHNICAL CONTACT` will appear as the Technical Contact.

There are two approaches to contact management you need to choose between:

- The first approach is to keep track of customer contact IDs as you create them and use an internal address book to store the IDs. In the event that the same customer returns to buy another domain, you can then use the same contact ID every time. This way, changes made to the singular contact record are reflected in the **Whois** information for all domains bought by that customer. For example, if your customers log in to an account on your site and you keep customer specific data, you might add a contact ID field to this data the first time a domain is registered and reuse the contact ID for subsequent registrations.

- The second approach is to treat all contact records as “one-offs.” In this approach, you create a new contact record for the customer every time he makes a purchase. This approach is wasteful since it is possible for many copies of essentially the same contact data to exist at one time on the server. It does save you from having to keep track of contact IDs, though. Note that if you do not keep track of contact IDs that it is impossible to edit an existing contact. To change a `TECHNICAL CONTACT` or `RESPONSIBLE PERSON` contact in this situation, you will have to create an entirely new contact with the updated information and then alter the appropriate contact field in the domain record via the **ChangeDomain** command (described below).

The first approach is considered a “Best Practice” and is the preferred method. It is understood, though, that some implementations may not allow for internal tracking of contact IDs, in which case the second approach may be the only option.

<table>
<tr><th colspan="2">CreateContact</th></tr>
<tr><td>Description:</td><td>Creates a new contact record and returns its unique ID.</td></tr>
<tr><td>Perl Subroutine:</td><td><pre>($contact_id, $request_id) =
create_contact($transaction_id, \%contact_ref)</pre></td></tr>
<tr><td>C Function:</td><td><pre>int SrsCreateContact
(
const char* [in] transaction_id,
NameValueList* [in] ContactData,
NameValueList**[in,out] ppResponse
);</pre></td></tr>
<tr><td>ISrd Method:</td><td><pre>HRESULT CreateContact
(
[in]BSTR bstrTransID,
[in]IDictionary* pData,
[out,retval]IDictionary** ppResponse
);</pre></td></tr>
<tr><td>Input:</td><td>A name-value list with the following format:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>FNAME</td><td>Required: Contact’s first name</td></tr>
<tr><td>LNAME</td><td>Required: Contact’s last name</td></tr>
<tr><td>ORGANIZATION</td><td>Optional: Business or Organization name</td></tr>
<tr><td>EMAIL</td><td>Required: Contact’s email address</td></tr>
<tr><td>ADDRESS1</td><td>Required: 1st line of contact’s address</td></tr>
<tr><td>ADDRESS2</td><td>Required: 2nd line of address</td></tr>
<tr><td>CITY</td><td>Required: City of contact’s address</td></tr>
<tr><td>PROVINCE</td><td>Required: Province/State of contact’s address</td></tr>
<tr><td>POSTAL CODE</td><td>Required: ZIP code or postal code of contact’s address</td></tr>
<tr><td>COUNTRY</td><td>Required: Two-letter country abbreviation of contact’s address in UPPERCASE (i.e., ccTLD country codes)</td></tr>
<tr><td>PHONE</td><td>Required: Phone number of contact</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following possible name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>CONTACTID</td><td>The unique ID of the newly created contact record. Used subsequently by the RegisterDomain, ChangeDomain, EditContact and GetContactInfo commands.</td></tr>
<tr><td>REQUESTID</td><td>SRS server generated processing ID</td></tr>
</table></td></tr>
<tr><td>Notes:</td><td>All input values are stored in the SRS system as human-readable strings to accommodate various phone and address formats. Note that only ASCII characters are allowed.

The country codes required for the COUNTRY parameter are the same as the ccTLD designations as assigned by IANA, which are roughly equivalent to the ISO3166-1 country codes. There are a few deviations, though. See Appendix A for a complete list of the accepted country codes.</td></tr>
</table>

<table>
<tr><th colspan="2">EditContact</th></tr>
<tr><td>Description:</td><td>Modifies fields in an existing contact record.</td></tr>
<tr><td>Perl Subroutine:</td><td><pre>($contact_id, $request_id) =
edit_contact($transaction_id, \%contact_ref)</pre></td></tr>
<tr><td>C Function:</td><td><pre>int SrsEditContact
(
const char* [in] transaction_id,
NameValueList* [in] pContactData,
NameValueList**[in,out] ppResponse
);</pre></td></tr>
<tr><td>ISrd Method:</td><td><pre>HRESULT EditContact
(
[in]BSTR bstrTransID,
[in]IDictionary* pData,
[out,retval]IDictionary** ppResponse
);</pre></td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>CONTACTID</td><td>Required: Unique ID from a previous CreateContact command</td></tr>
<tr><td>ORGANIZATION</td><td>Optional: New value for the business/organization name</td></tr>
<tr><td>EMAIL</td><td>Optional: New value for the customer’s email address</td></tr>
<tr><td>ADDRESS1</td><td>Optional: New value for the 1st address line</td></tr>
<tr><td>ADDRESS2</td><td>Optional: New value for the 2nd address line</td></tr>
<tr><td>CITY</td><td>Optional: New value for the city</td></tr>
<tr><td>PROVINCE</td><td>Optional: New value for the province/state</td></tr>
<tr><td>POSTAL CODE</td><td>Optional: New value for the postal/zip code</td></tr>
<tr><td>COUNTRY</td><td>Optional: Two-letter country abbreviation of contact’s address in UPPERCASE (i.e., ccTLD country code)</td></tr>
<tr><td>PHONE</td><td>Optional: Contact’s phone number</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>CONTACTID</td><td>The unique ID of the contact that was updated. It should always match the input CONTACTID</td></tr>
<tr><td>REQUESTID</td><td>SRS server generated processing ID</td></tr>
</table></td></tr>
<tr><td>Notes:</td><td>You cannot modify a contact record unless you originally created it.

The FNAME and LNAME fields are not updateable. This helps prevent unauthorized transfers of ownership.

Only include parameters you want to update in the input list.

The country codes required for the COUNTRY parameter are the same as the ccTLD designations as assigned by IANA, which are roughly equivalent to the ISO3166-1 country codes. There are a few deviations, though. See Appendix A for a complete list of the accepted country codes.</td></tr>
</table>

<table>
<tr><th colspan="2">GetContactInfo</th></tr>
<tr><td>Description:</td><td>Retrieves fields of an existing contact record.</td></tr>
<tr><td>Perl Subroutine:</td><td>(\%info_ref) = get_contact_info($contact_id)</td></tr>
<tr><td>C Function:</td><td>int SrsGetContactInfo
(
const char* [in] contact_id,
NameValueList* [in, out] ppResponse
);</td></tr>
<tr><td>ISrd Method:</td><td>HRESULT GetContactInfo
(
[in]BSTR bstrContactID,
[in]IDictionary* pData,
[out,retval]IDictionary** ppResponse
);</td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:

<table><tr><th>Name</th><th>Value</th></tr><tr><td>CONTACTID</td><td>Required: Unique ID from a previous CreateContact command</td></tr></table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:

<table><tr><th>Name</th><th>Value</th></tr><tr><td>FNAME</td><td>Contact’s first name</td></tr><tr><td>LNAME</td><td>Contact’s last name</td></tr><tr><td>ORGANIZATION</td><td>Optional: Business or Organization name</td></tr><tr><td>EMAIL</td><td>Contact’s email address</td></tr><tr><td>ADDRESS1</td><td>1st line of contact’s address</td></tr><tr><td>ADDRESS2</td><td>Optional: 2nd line of address</td></tr><tr><td>CITY</td><td>City of contact’s address</td></tr><tr><td>PROVINCE</td><td>Province/State of contact’s address</td></tr><tr><td>POSTAL CODE</td><td>ZIP code or postal code of contact’s address</td></tr><tr><td>COUNTRY</td><td>Two-letter country abbreviation</td></tr><tr><td>PHONE</td><td>Contact’s phone number</td></tr></table></td></tr>
<tr><td>Notes:</td><td>Optional fields might not be returned if left empty.</td></tr>
</table>

4.4 The Domain Registration and Manipulation Commands

These commands are used to register, release (delete), renew and modify domain names, including name server information.

<table>
<tr><th colspan="2">RegisterDomain</th></tr>
<tr><td>Description:</td><td>Register (buy) a domain.</td></tr>
<tr><td>Perl Subroutine:</td><td>($request_id, \%response_ref) =
register_domain ($transaction_id, \%domain_info_ref)</td></tr>
<tr><td>C Function:</td><td>int SrsRegisterDomain
(
const char* [in] transaction_id,
NameValueList* [in] pDomainInfo,
NameValueList** [in, out] ppResponse
);</td></tr>
<tr><td>ISrd Method:</td><td>HRESULT RegisterDomain
(
[in]BSTR bstrTransID,
[in]IDictionary* pData,
[out,retval]IDictionary** ppResponse
);</td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:

<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN</td><td>Required: domain name to register</td></tr><tr><td>TLD</td><td>Optional: Defaults to ‘tv’ if omitted.</td></tr><tr><td>RESPONSIBLE PERSON</td><td>Required: Contact ID of the purchasing customer</td></tr><tr><td>TECHNICAL CONTACT</td><td>Required: Contact ID of the technical contact for the domain</td></tr><tr><td>BILLING CONTACT</td><td>Optional: Contact ID of the billing contact for the domain</td></tr><tr><td>ADMIN CONTACT</td><td>Optional: Contact of the administrative contact for the domain</td></tr><tr><td>DNS SERVER NAME 1…n</td><td>Optional: Up to 13 name servers may be specified for a domain</td></tr><tr><td>TERM YEARS</td><td>Number of years to register name (maximum of 10)</td></tr><tr><td>PRICE</td><td>The first year wholesale registration price for the domain as returned by a previous DomainInfo command. (Do not multiply by the TERM YEARS).</td></tr><tr><td>ADDITIONAL DATA</td><td>Required for certain domains. See Notes.</td></tr></table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs

<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN</td><td>Echo of the input value.</td></tr><tr><td>TLD</td><td>Echo of the input value.</td></tr><tr><td>REQUESTEDID</td><td>SRS server generated processing ID</td></tr><tr><td>PRICE</td><td>The amount charged to your partner account.</td></tr><tr><td>EXPIRATION DATE</td><td>Date this registration will expire (in epoch time).</td></tr></table></td></tr>
</table>

Notes:

• The contact related parameters expect the contact ID returned by a previous **CreateContact** command. A value of 0 (zero) can be used to default the contact information to that in your partner account profile. This feature should only be used for the `RESPONSIBLE PERSON` field if you, the partner, are the actual customer buying the domain name. In short, the `RESPONSIBLE PERSON` contact is always required to be the actual customer purchasing the domain name.

• The input `PRICE` should match the wholesale price that was returned by a previous **DomainInfo** command for this domain name. Note that this parameter is the one-year price as returned by **DomainInfo** and *should not* be multiplied by the `TERM YEARS` value. Also, do not accidentally use the `EFFECTIVE PRICE` value.

• There is no grace period for registrations. If you believe a domain has been registered erroneously, please contact partners@srsplus.com.

• If you do not specify any DNS servers, a default server may be set for you. This default server simply serves up a generic "Coming Soon" web page.

• The `ADDITIONAL DATA` parameter must be set for domains where the controlling registry has additional requirements for registration. Currently, the `.us` and `.de` ccTLDs have additional requirements, which are detailed in the chart below. The `ADDITIONAL DATA` parameter is used to send a secondary set of name-value pairs that specify the additional required information. That is, the value part of the `ADDITIONAL DATA` name-value pair is another name-value pair in a text format where the name and value are separated by a colon followed by a space.

The following chart shows the current requirements:

ccTLD	ADDITIONAL DATA value
`.de`	`UseDefaultRegistrantAddress: <0|1>` Domains in the .de ccTLD require that the registrant's address be within the country (Germany). A value of 0 indicates that the address specified in the contact ID specified for RESPONSIBLE PERSON should be used as the address to validate. A value of 1 indicates that an automatically provided default address within the country should be used as the address to validate.

`.us`	`CustomerType:<data1>`
	`AnticipatedUsage:<data2>`
	
	Where `<data1>` **is one of:**
	`PersonUSCitizen`, `PersonPermanentUSResident`, `OrganizationIncorporatedInUS`, **or** `OrganizationWithActiveUSPresence`
	
	And **where** `<data2>` **is one of:**
	`ForProfit`, `NotForProfit`, `Personal`, `Educational` **or** `Governmental`
	
	IMPORTANT: The two name-value pairs should be separated only by white space (i.e, tab or space). Do *not* separate with a linefeed or carriage return.

Example:

Name	Value
ADDITIONAL DATA	CustomerType: PersonUSCitizen AnticipatedUsage: Personal

IMPORTANT:
• Some registries do not guarantee real-time registration. There may be a delay (up to five days) before the domain is actually registered in the target registry. For detailed information, refer to the **ccTLD FAQ** on the SRSplus website at:
http://www.srsplus.com/en/srsplus/support/cctld.shtml.
(The remainder of this page is intentionally blank)

<table>
<tr><th colspan="2">ChangeDomain</th></tr>
<tr><td>Description:</td><td>Modify contact and domain name server information for a domain.</td></tr>
<tr><td>Perl Subroutine:</td><td><pre>($request_id) =
change_domain ($transaction_id, \%domain_info_ref)</pre></td></tr>
<tr><td>C Function:</td><td><pre>int SrsChangeDomain
(
const char* [in] transaction_id,
NameValueList* [in] pDomainInfo,
NameValueList** [in,out] ppResponse
);</pre></td></tr>
<tr><td>ISrd Method:</td><td><pre>HRESULT ChangeDomain
(
[in]BSTR bstrTransID,
[in]IDictionary* pData,
[out,retval]IDictionary** ppResponse
);</pre></td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>DOMAIN</td><td>Required: Domain name to modify</td></tr>
<tr><td>TLD</td><td>Optional: Defaults to ‘tv’ if omitted</td></tr>
<tr><td>RESPONSIBLE PERSON</td><td>Optional: Contact ID of the purchasing customer. Use only if contact has changed.</td></tr>
<tr><td>TECHNICAL CONTACT</td><td>Optional: Contact ID of the technical contact for the domain. Use only if contact has changed.</td></tr>
<tr><td>BILLING CONTACT</td><td>Optional: Contact ID of the billing contact for the domain. Use only if contact has changed.</td></tr>
<tr><td>ADMIN CONTACT</td><td>Optional: Contact ID of the administrative contact for the domain. Use only if contact has changed.</td></tr>
<tr><td>AUTOCHARGE</td><td>Optional: Turns automatic renewal of domain registrations on or off. A value of “1” indicates ON; a value of “0” indicates OFF.</td></tr>
<tr><td>DNS SERVER NAME 1…n</td><td>Optional: Up to 13 name servers may be specified for the domain.</td></tr>
<tr><td>DNS TYPE</td><td>Optional: will default to YOUSERVE

DNS setting type
PARKED
YOUSERVE
WESERVE</td></tr>
<tr><td>DNS RECORD TYPE 1..N</td><td>Optional: The A, CNAME, MX record</td></tr>
<tr><td>DNS RECORD LEFT 1..N</td><td>Optional: Left side of the DNS record</td></tr>
<tr><td>DNS RECORD RIGHT 1..N</td><td>Optional: Right side of DNS record</td></tr>
<tr><td>DNS RECORD PRIORITY</td><td>Optional: For MX records what priority</td></tr>
<tr><td>DOMAIN PROTECT</td><td>Optional: Turns Domain Protect on or off. A value of “1” indicates ON; a value of “0” indicates OFF.</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>REQUESTED</td><td>SRS server generated processing ID</td></tr>
</table></td></tr>
</table>

Notes:	You must be the partner-of-record for the domain in order to update its information. The contact related parameters expect the contact ID returned by a previous **CreateContact** command. A value of 0 (zero) can be used to default the contact information to that in your partner account profile. This is perfectly appropriate for use in the `TECHNICAL CONTACT` field, but it should only be used in the `RESPONSIBLE PERSON` field if you, the partner, are the actual customer buying the domain name. In short, the `RESPONSIBLE PERSON` contact is always required to be the actual customer purchasing the domain.

RenewDomain	
Description:	Renews a domain registration for a given number of years
Perl Subroutine:	`($request_id, \%response_ref) = renew_domain ($transaction_id, \%domain_info_ref)`
C Function:	`int RenewDomain ( const char* [in] transaction_id, NameValueList* [in] pDomainInfo, NameValueList** [in,out] ppResponse );`
ISrd Method:	`HRESULT RenewDomain ( [in]BSTR bstrTransID, [in]IDictionary* pData, [out,retval]IDictionary** ppResponse );`
Input:	A name-value list with the following name-value pairs: **Name** / **Value** `DOMAIN` — Required: Domain name to modify `TLD` — Optional: Defaults to 'tv' if omitted `TERM YEARS` — Required: Number of years to register name (maximum of 10). `EFFECTIVE PRICE` — The one-year registration price for the domain as returned by a previous **Whois** command. (Do not multiply by the `TERM YEARS`).
Output:	A name-value list with the following name-value pairs: **Name** / **Value** `REQUESTED` — SRS server generated processing ID `PRICE` — The amount charged to your partner account. `EXPIRATION DATE` — Date the registration will expire (in epoch time).
Notes:	You must be the partner-of-record for the domain in order to renew it. The total term years for a domain cannot exceed 10 years at any one time. That is, if the current registration does not expire for 3 more years and an attempt is made to renew the domain for 8 years, the command will fail.

4.5 Nameserver Commands

These commands are used to add, remove and get information about nameservers. Before specifying a nameserver for a domain (via the **RegisterDomain** or **ChangeDomain** commands) you should make sure the nameserver has been registered.

<table>
<tr><th colspan="2">RegisterNameServer</th></tr>
<tr><td>Description:</td><td>Add a nameserver.</td></tr>
<tr><td>Perl Subroutine:</td><td><pre>($request_id) =
register_nameserver ($transaction_id, \%ns_info_ref)</pre></td></tr>
<tr><td>C Function:</td><td><pre>int SrsRegisternameServer
(
const char* [in] transaction_id,
NameValueList* [in] pNSInfo,
NameValueList** [in,out] ppResponse
);</pre></td></tr>
<tr><td>ISrd Method:</td><td><pre>HRESULT RegisterNameServer
(
[in]BSTR bstrTransID,
[in]IDictionary* pData,
[out,retval]IDictionary** ppResponse
);</pre></td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>DNS SERVER NAME</td><td>Required: Full name of the nameserver to add (e.g.,s1.mydomain.tv)</td></tr>
<tr><td>DNS SERVER IP</td><td>Required: IP address of the nameserver to add</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>REQUESTEDID</td><td>SRS server generated processing ID</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="2">ReleaseNameServer</th></tr>
<tr><td>Description:</td><td>Delete a nameserver.</td></tr>
<tr><td>Perl Subroutine:</td><td><pre>($request_id) =
release_nameserver ($transaction_id, \%ns_info_ref)</pre></td></tr>
<tr><td>C Function:</td><td><pre>int SrsReleaseNameServer
(
const char* [in] transaction_id,
NameValueList* [in] pNSInfo,
NameValueList** [in,out] ppResponse
);</pre></td></tr>
<tr><td>ISrd Method:</td><td><pre>HRESULT ReleaseNameServer
(
[in]BSTR bstrTransID,
[in]IDictionary* pData,
[out,retval]IDictionary** ppResponse
);</pre></td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>DNS SERVER NAME</td><td>Required: Full name of the nameserver to remove (e.g./ ns1.mydomain.tv)</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>REQUESTED</td><td>SRS server generated processing ID</td></tr>
</table></td></tr>
<tr><td>Notes:</td><td>You can only release nameservers for which you are the partner-of-record (i.e., nameservers that you registered).</td></tr>
</table>

<table>
<tr><th colspan="2">NameServerInfo</th></tr>
<tr><td>Description:</td><td>Retrieve information about a nameserver.</td></tr>
<tr><td>Perl Subroutine:</td><td><pre>($request_id) =
nameserver_info ($transaction_id, \%ns_info_ref)</pre></td></tr>
<tr><td>C Function:</td><td><pre>int SrsNameServerInfo
(
const char* [in] transaction_id,
NameValueList* [in] pNSInfo,
NameValueList** [in,out] ppResponse
);</pre></td></tr>
<tr><td>ISrd Method:</td><td><pre>HRESULT NameServerInfo
(
[in]BSTR bstrTransID,
[in]IDictionary* pData,
[out,retval]IDictionary** ppResponse
);</pre></td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>DNS SERVER NAME</td><td>Required: Full name of the nameserver to remove (e.g., ns1.mydomain.tv)</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>DNS SERVER NAME</td><td>Full name of the nameserver</td></tr>
<tr><td>DNS SERVER IP</td><td>IP address of the nameserver</td></tr>
<tr><td>DNS REGISGTRATION DATE</td><td>Date the nameserver was registered (in epoch time)</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="2">ModifyNameserver</th></tr>
<tr><td>Description:</td><td>Retrieve information about a nameserver.</td></tr>
<tr><td>Perl Subroutine:</td><td><pre>($request_id) =
nameserver_info ($transaction_id, \%ns_info_ref)</pre></td></tr>
<tr><td>C Function:</td><td><pre>int SrsModifyNameServer (const char* transaction_id,
NameValueList* pNSInfo, NameValueList** ppResponse);</pre></td></tr>
<tr><td>ISrd Method:</td><td><pre>HRESULT ModifyNameServer(
[in]BSTR bstrTransID,
[in]IDictionary* pData,
[out,retval]IDictionary** ppResponse
);</pre></td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>DNS SERVER NAME</td><td>Required: Full name of the nameserver to remove (e.g., ns1.mydomain.tv)</td></tr>
<tr><td>DNS SERVER IP</td><td>The IP address (IPv4)</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>DNS SERVER NAME</td><td>Full name of the nameserver</td></tr>
<tr><td>DNS SERVER IP</td><td>IP address of the nameserver</td></tr>
<tr><td>DNS REGISTRATION DATE</td><td>Date the nameserver was registered (in epoch time)</td></tr>
</table></td></tr>
</table>

4.6 Domain Transfer Commands

These commands are used to request, accept, deny and obtain information about domain transfers. **IMPORTANT: Currently, the transfer commands support only com, net, org, biz, info, cc and tv TLDs.**

<table>
<tr><th colspan="2">RequestTransfer</th></tr>
<tr><td>Description:</td><td>Request an inbound transfer of a domain to SRSplus from another registrar.</td></tr>
<tr><td>Perl Subroutine:</td><td><pre>($requestID, \%response_ref) =
request_transfer ($transaction_id, \%ns_info_ref)</pre></td></tr>
<tr><td>C Function:</td><td><pre>int SrsRequestTransfer
(
const char* [in] transaction_id,
NameValueList* [in] pDomainInfo,
NameValueList** [in,out] ppResponse
);</pre></td></tr>
<tr><td>ISrd Method:</td><td><pre>HRESULT RequestTransfer
(
[in]BSTR bstrTransID,
[in]IDictionary* pData,
[out,retval]IDictionary** ppResponse
);</pre></td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>DOMAIN</td><td>Required: Domain name for which you are requesting the transfer.</td></tr>
<tr><td>TLD</td><td>Required: TLD of the domain name for which you are requesting the transfer</td></tr>
<tr><td>CURRENT ADMIN EMAIL</td><td>Required: The email address of the Administrative contact as listed by the current owner in the WHOIS system.</td></tr>
<tr><td>AUTH_CODE</td><td>Required for all supported domain names: Your authorization code from the respective registry.</td></tr>
<tr><td>RESPONSIBLE PERSON</td><td>Optional: Contact ID of a contact record to associate with this domain once the transfer is completed.</td></tr>
<tr><td>TECHNICAL CONTACT</td><td>Optional: Contact ID of a contact record to associate with this domain once the transfer is completed.</td></tr>
<tr><td>BILLING CONTACT</td><td>Optional: Contact ID of a contact record to associate with this domain once the transfer is completed.</td></tr>
<tr><td>ADMIN CONTACT</td><td>Optional: Contact ID of a contact record to associate with this domain once the transfer is completed.</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>REQUESTID</td><td>SRS server generated processing ID</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="2">OutboundTransferResponse</th></tr>
<tr><td>Description:</td><td>Respond to a transfer request where you are the losing partner. You may ACCEPT or DENY the pending request.</td></tr>
<tr><td>Perl Subroutine:</td><td>(\%response_ref) = outbound_transfer_response
($transaction_id, $domain, $tld,response_string)</td></tr>
<tr><td>C Function:</td><td>int SrsOutboundTransferResponse
(
const char* [in] transaction_id,
const char* [in] domain,
const char* [in] tld,
const char* [in] response_string,
NameValueList** [in, out] ppResponse
);</td></tr>
<tr><td>ISrs Method:</td><td>HRESULT OutboundTransferResponse
(
[in]BSTR bstrTransID,
[in]BSTR bstrDomain,
[in]BSTR bstrTLD,
[in]BSTR bstrResponseString,
[out,retval]IDictionary** ppResponse
);</td></tr>
<tr><td>Input:</td><td>PARAMETERS:
Optional: A TRANSACTION ID string.

Required: DOMAIN name for which you are responding.

Required: TLD of the domain name for which you are responding.

Required: TRANSFER RESPONSE to send. Either “ACCEPT” or “DENY”.</td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:<table><tr><th>Name</th><th>Value</th></tr><tr><td>REQUESTID</td><td>SRS server generated processing ID</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="2">ViewPendingTransfer</th></tr>
<tr><td>Description:</td><td>Returns information about pending transfers, either INBOUND or OUTBOUND.</td></tr>
<tr><td>Perl Subroutine:</td><td>(\%response_ref) =
view_pending_transfers ($transaction_id, transfer_type)</td></tr>
<tr><td>C Function:</td><td>int SrsViewPendingTransfers
(
const char* [in] transaction_id,
const char* [in] transfer_type,
NameValueList** [in, out] ppResponse
);</td></tr>
<tr><td>ISrs Method:</td><td>HRESULT ViewPendingTransfers
(
[in]BSTR bstrTransID,
[in]BSTR bstrTransferType,
[out,retval]IDictionary** ppResponse
);</td></tr>
<tr><td>Input:</td><td>PARAMETERS:
Optional: A TRANSACTION ID string.

Required: String specifying type of pending transfers to view. Must be either “INBOUND” or “OUTBOUND”.

NOTE: INBOUND refers to domains that are awaiting approval by the losing registrar to be transferred to SRSplus. OUTBOUND refers to domains that are awaiting your approval to be transferred away from SRSplus.</td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table><tr><th>Name</th><th>Value</th></tr><tr><td>REQUESTID</td><td>SRS server generated processing ID</td></tr><tr><td>DOMAIN 1…n</td><td>Domain that is pending approval.</td></tr><tr><td>TLD 1…n</td><td>Corresponding TLD</td></tr><tr><td>DATE LOGGED 1…n</td><td>Date the transfer request was recorded for the corresponding domain and tld. Given in Linux/Unix epoch time (i.e., count of seconds since midnight Jan 1, 1970.)</td></tr></table>
EXAMPLE:

<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN 1</td><td>“testdomain”</td></tr><tr><td>TLD 1</td><td>“net”</td></tr><tr><td>DATE LOGGED 1</td><td>“1007497172”</td></tr><tr><td>DOMAIN 2</td><td>“anotherdomain”</td></tr><tr><td>TLD 2</td><td>“com”</td></tr><tr><td>DATE LOGGED 2</td><td>“1001961773”</td></tr></table>
NOTE: Once a transfer is complete, it no longer is included in the results. To determine if a transfer was successful or not, obtain the WHOIS information for the domain and compare to the expected results.</td></tr>
</table>

<table>
<tr><th colspan="2">QueryTransferID</th></tr>
<tr><td>Description:</td><td>Returns available information regarding a transfer.</td></tr>
<tr><td>Perl Subroutine:</td><td>(\%response_ref) = $srs_client->query_transfer_id($transaction_id, $transfer_id)</td></tr>
<tr><td>C Function:</td><td><pre>int SrsQueryTransfer
(
const char* [in] transaction_id,
char* [in] transfer_id,
NameValueList** [in, out] ppResponse
);</pre></td></tr>
<tr><td>ISrd Method:</td><td><pre>HRESULT QueryTransfer(
[in]BSTR bstrTransID,
[in]BSTR bstrTransferID,
[out,retval]IDictionary** ppResponse
);</pre></td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>TRANSACTION ID</td><td>Optional: Transaction ID string.</td></tr>
<tr><td>TRANSFER ID</td><td>Required: The ID of the transfer you wish to query</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>REQUESTID</td><td>SRS server generated processing ID</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="2">QueryRejectedTransfer</th></tr>
<tr><td>Description:</td><td>Returns available information regarding why a transfer was rejected.</td></tr>
<tr><td>Perl Subroutine:</td><td><pre>(\%response_ref) = $srs_client-
>query_rejected_transfer($transaction_id, $transfer_id);</pre></td></tr>
<tr><td>C Function:</td><td><pre>int SrsQueryRejectedTransfer
(
const char* [in] transaction_id,
char* [in] transfer_id,
NameValueList** [in, out] ppResponse
);</pre></td></tr>
<tr><td>ISrd Method:</td><td><pre>HRESULT QueryTransfer(
[in]BSTR bstrTransID,
[in]BSTR bstrTransferID,
[out,retval]IDictionary** ppResponse
);</pre></td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>TRANSACTION ID</td><td>Optional: Transaction ID string.</td></tr>
<tr><td>TRANSFER ID Required:</td><td>The ID of the transfer you wish to query</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>REQUESTID</td><td>SRS server generated processing ID</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="2">ResendTransferAuthorization</th></tr>
<tr><td>Description:</td><td>Resends a transfer authorization email</td></tr>
<tr><td>Perl Subroutine:</td><td><pre>($response_ref) =
resend_transfer_authorization($transaction_id,
$transfer_id);</pre></td></tr>
<tr><td>C Function:</td><td>NOT YET AVAILABLE</td></tr>
<tr><td>ISrd Method:</td><td>NOT YET AVAILABLE</td></tr>
<tr><td>Input:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>TRANSACTION ID</td><td>Optional: Transaction ID string.</td></tr>
<tr><td>TRANSFER ID</td><td>Required: The ID of the transfer you wish to query</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>REQUESTID</td><td>SRS server generated processing ID</td></tr>
</table></td></tr>
<tr><td>Notes:</td><td>You can resend an authorization e-mail up to 2 times per transfer ID.</td></tr>
</table>

4.7 VAS Commands

These commands are used to purchase, renew, cancel, and query VAS services.
NOTE: Only available on certain TLD's.

*To sync the expiration date of private registration with the expiration date of the domain, send a value of "-1" as the term.

<table>
<tr><th colspan="2">BuyVas</th></tr>
<tr><td>Description:</td><td>This allows purchase of a Value Added Service (VAS).</td></tr>
<tr><td>Perl Subroutine:</td><td>$vas_id = buy_vas ($transaction_id, $vas_action_ref)</td></tr>
<tr><td>C Function:</td><td><pre>int SrsBuyVas(const char* transaction_id,
NameValueList*
pArgs, NameValueList** ppResponse);</pre></td></tr>
<tr><td>ISrs Method:</td><td><pre>HRESULT BuyVas(
[in]BSTR bstrTransID,
[in]IDictionary* pArgs,
[out,retval]IDictionary** ppResponse
);</pre></td></tr>
<tr><td>Input:</td><td>PARAMETERS:

Required:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>DOMAIN</td><td>The domain name</td></tr>
<tr><td>TLD</td><td>Corresponding TLD</td></tr>
<tr><td>VAS TYPE</td><td>'PRIVATE REG' or other</td></tr>
<tr><td>TERM</td><td>Term of VAS</td></tr>
<tr><td>OTHER INFO</td><td>Other information in Name Value Pair as required by the TYPE of VAS</td></tr>
</table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>PRICE</td><td>The price of the VAS</td></tr>
<tr><td>EXPIRATION DATE</td><td>The expiry date</td></tr>
<tr><td>REQUEST ID</td><td>ID of the request</td></tr>
<tr><td>VAS ID</td><td>VAS ID</td></tr>
</table></td></tr>
</table>

<table>
<tr><th colspan="2">QueryVas</th></tr>
<tr><td>Description:</td><td>List the VAS on a domain.</td></tr>
<tr><td>Perl Subroutine:</td><td>query_vas($transaction_id, $vas_action_ref)</td></tr>
<tr><td>C Function:</td><td>int SrsQueryVas(const char* transaction_id,
NameValueList*
pArgs, NameValueList** ppResponse);</td></tr>
<tr><td>ISrs Method:</td><td>HRESULT QueryVas(
[in]BSTR bstrTransID,
[in]IDictionary* pArgs,
[out,retval]IDictionary** ppResponse
);</td></tr>
<tr><td>Input:</td><td>PARAMETERS:

Required:<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN</td><td>Domain Name</td></tr><tr><td>TLD</td><td>Corresponding TLD</td></tr><tr><td>VAS ID</td><td>‘PRIVATE REG’ or other</td></tr></table></td></tr>
<tr><td>Output:</td><td>Number can go from 1 through however many VAS are on the domains.<table><tr><th>Name</th><th>Value</th></tr><tr><td>EXPIRATION DATE</td><td>The expiry date of the VAS</td></tr><tr><td>VAS TYPE</td><td>Type of VAS</td></tr><tr><td>CREATION DATE</td><td>Create date of the VAS</td></tr><tr><td>VAS ID</td><td>VAS ID</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="2">RenewVas</th></tr>
<tr><td>Description:</td><td>Renew a VAS (Can renew for greater term than the domain name)</td></tr>
<tr><td>Perl Subroutine:</td><td>$request_id = renew_vas($transaction_id,
$vas_action_ref);</td></tr>
<tr><td>C Function:</td><td>int SrsRenewVas(const char* transaction_id,
NameValueList*
pArgs, NameValueList** ppResponse);</td></tr>
<tr><td>ISrs Method:</td><td>HRESULT RenewVas(
[in]BSTR bstrTransID,
[in]IDictionary* pArgs,
[out,retval]IDictionary** ppResponse
);</td></tr>
<tr><td>Input:</td><td>PARAMETERS:
Optional:<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN</td><td>Domain Name</td></tr><tr><td>TLD</td><td>Corresponding TLD</td></tr></table>
Required:<table><tr><th>Name</th><th>Value</th></tr><tr><td>VAS ID</td><td>‘PRIVATE REG’ or other</td></tr><tr><td>TERM</td><td>Term Length</td></tr></table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:

<table><tr><th>Name</th><th>Value</th></tr><tr><td>PRICE</td><td>The price of the VAS</td></tr><tr><td>EXPIRATION DATE</td><td>The expiry date</td></tr><tr><td>EFFECTIVE PRICE</td><td>Effective price of VAS</td></tr><tr><td>REQUEST ID</td><td>ID of the request</td></tr></table></td></tr>
</table>

<table>
<tr><th colspan="2">ModifyVas</th></tr>
<tr><td>Description:</td><td>Modify a VAS; change options on a VAS.</td></tr>
<tr><td>Perl Subroutine:</td><td>$request_id = modify_vas($transaction_id, $vas_action_ref);</td></tr>
<tr><td>C Function:</td><td>int SrsModifyVas(const char* transaction_id, NameValueList* pArgs, NameValueList** ppResponse);</td></tr>
<tr><td>ISrs Method:</td><td>HRESULT ModifyVas(
[in]BSTR bstrTransID,
[in]IDictionary* pArgs,
[out,retval]IDictionary** ppResponse
);</td></tr>
<tr><td>Input:</td><td>PARAMETERS:

Optional:

<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN</td><td>Domain</td></tr><tr><td>TLD</td><td>Corresponding TLD</td></tr></table>
Required:

<table><tr><th>Name</th><th>Value</th></tr><tr><td>VAS_ID</td><td>‘PRIVATE REG’ or other</td></tr><tr><td>MODIFY_OPTIONS</td><td>What you want to happen</td></tr></table></td></tr>
<tr><td>Output:</td><td>*No functionality at this time.</td></tr>
</table>

<table>
<tr><th colspan="2">CancelVas</th></tr>
<tr><td>Description:</td><td>Cancels a VAS.</td></tr>
<tr><td>Perl Subroutine:</td><td>$request_id = cancel_vas($transaction_id, $vas_action_ref);</td></tr>
<tr><td>C Function:</td><td>int SrsBuyVas(const char* transaction_id, NameValueList* pArgs, NameValueList** ppResponse);</td></tr>
<tr><td>ISrs Method:</td><td>HRESULT CancelVas(
[in]BSTR bstrTransID,
[in]IDictionary* pArgs,
[out,retval]IDictionary** ppResponse
);</td></tr>
<tr><td>Input:</td><td>PARAMETERS:

Optional:

<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN</td><td>Domain</td></tr><tr><td>TLD</td><td>Corresponding TLD</td></tr></table>
Required:
<table><tr><th>Name</th><th>Value</th></tr><tr><td>VAS_ID</td><td>‘PRIVATE REG’ or other</td></tr></table></td></tr>
<tr><td>Output:</td><td><table><tr><th>Name</th><th>Value</th></tr><tr><td>REQUEST_ID</td><td>The request id</td></tr></table>A name-value list with the following name-value pairs:

Note: VAS is removed</td></tr>
<tr><td>Notes:</td><td>No credits will be honored for VAS transactions made in error.</td></tr>
</table>

<table>
<tr><th colspan="2">PriceVas</th></tr>
<tr><td>Description:</td><td>Obtain the price of a VAS</td></tr>
<tr><td>Perl Subroutine:</td><td>$price = price_vas($transaction_id, $vas_action_ref);</td></tr>
<tr><td>C Function:</td><td>int SrsPriceVas(const char* transaction_id,
NameValueList* pArgs, NameValueList** ppResponse);</td></tr>
<tr><td>ISrs Method:</td><td>HRESULT PriceVas(
[in]BSTR bstrTransID,
[in]IDictionary* pData,
[out,retval]IDictionary** ppResponse
);</td></tr>
<tr><td>Input:</td><td>PARAMETERS:
Required:<table><tr><th>Name</th><th>Value</th></tr><tr><td>DOMAIN</td><td>Domain Name</td></tr><tr><td>TLD</td><td>Corresponding TLD</td></tr><tr><td>VAS_TYPE</td><td>‘PRIVATE REG’ or other</td></tr></table></td></tr>
<tr><td>Output:</td><td>A name-value list with the following name-value pairs:<table><tr><th>Name</th><th>Value</th></tr><tr><td>PRICE</td><td>Amount to be charged</td></tr></table></td></tr>
</table>

5.GnuPGEssentials

As mentioned previously, GnuPG, the Gnu Privacy Guard, is used by all API versions to digitally sign outgoing packets and to verify the signature of incoming packets. GnuPG is a robust program with dozens of commands. There are only a handful of these commands, though, you need to be familiar with in order to get “up and running.” This section focuses on these commands and the processes you must perform with GnuPG before you can communicate with the SRS server. First, you will need to install GnuPG.

5.1 Installation for Linux

GnuPG for Linux is available from the Download section of www.gnupg.org as a gzipped tarball of source code that you must compile into an executable (It is assumed that you have an appropriate C compiler and make utility installed on your machine). Here are the quick start instructions as listed in the readme file:

1) Unpack the TAR. With GNU tar you can do it this way:
2) >tar -xzvf gnupg-x.y.z.tar.gz
3) >cd gnupg-x.y.z
4) >./configure
5) >make
6) >make install
7) You’ll end up with a gpg binary in /usr/local/bin.

5.1.1 Path Settings

On Linux systems, the key and trust databases of GnuPG are expected by default to be located in the directory `/home/username/.gnupg`. This directory can be overridden by setting the environment variable `GNUPGHOME` to an alternate path ***before*** running the executable `gpg.exe`for the first time. Accomplish this by using the `export` command:

- Example: `export GNUPGHOME /home/username/.gnupg`
  The order is important! If you create your key pair before setting `GNUPGHOME`, the key pair will not be located in the custom `GNUPGHOME` directory and you will likely get an error message indicating a missing default secret key when you attempt to use the API.

5.2 Installation for Windows

The Gnu Privacy Guard is distributed for Windows as a pre-compiled executable, the latest version of which is available at `http://www.gnupg.org/download.html`. Unzip the distribution to a directory of your choosing and add the directory to the system `PATH` environment variable so that other programs and components can find `gpg.exe`. Create the directory `C:\GNUPG`, which is where GnuPG will locate its databases by default.

5.3 GnuPG Configuration

The following six steps will guide you through the process of creating your keypair, as well as importing and signing the SRS keys. You should create your keys and send the public key to `registry@srsplus.com` as soon as practical. After your key submission is processed, your partner ID, a vital piece of information, will be returned to you. You will need your partner ID for API configuration. You can still move forward and develop your client application, but until you receive your partner ID, you will not be able to connect to the SRS servers.

Step 1: Keypair Creation

Now that you have GnuPG installed, you can create your default key pair. (For Linux systems, you should be logged in as the user whose account your SRS client process will run as).

- Run the command “`gpg --gen-key`”
- Follow the on screen instructions, choosing the default choices for all options
- Use the email name you used when signing up for your SRSplus Partner account when prompted for the email name of the key
- Provide the full name of your organization as the description for the key

Step 2: Export the Public Key

Now that you have created your keys, you will need to export your public key and submit it to SRS PLUS in order for the SRS server to verify your identity.

- Run the command “`gpg --export -a > publickey.txt`”
- This creates a plain ASCII text file of your public key.
- Send the file to `registry@srsplus.com` (Once the key is received and processed, you will be sent back your partner ID and allowed access to the test SRS server).

- Keep a copy of the file somewhere safe as a backup

Step 3: Export the Private Key

At this point, it is a good idea to also export your private key to a file and save both of your keys somewhere safe. If you suffer a catastrophic failure that ruins your keys, you may resubmit a new public key. It may take up to 24-hours for it to become valid in our system, though. If you have backup copies of your key pair, then you can simply re-import them and suffer less downtime.

- Run the command "`gpg --export-secret-key –a > privatekey.txt`" This creates a plain ASCII text file of your private key.
- Keep a copy of the file somewhere safe as a backup. Protect this file from theft and altering.

Step 4: Import the SRS Public Keys

A keyblock file named `SRS.KEY` is provided in the `KEYS` directory of the Toolkit distribution. Importing this keyblock will add two keys to your keyring, one for the test SRS server and one for the production SRS server. Accomplish this with the following command:

```
gpg --import SRS.KEY
```

You should verify the "fingerprints" of the keys. Run the following command and verify that the output is similar:

```
gpg –-list-keys

pub 1024D/83254FDB 2000-07-27 Test registry <testregistry@www.tv>
uid                           SRSPlus test account
<testregistry@srsplus.com>
sub 1024g/DCFB3D2E 2000-07-27

pub 1024D/415202CC 2000-08-01 DotTV Registry <registry@www.tv>
uid                           SRS Plus <registry@srsplus.com>
sub 2048g/778C3D8A 2000-08-01
```

Step 5: Sign the SRS Keys

Once the keys have been imported, you need to locally sign the keys in order for them to be added to your list of "trusted" keys. Locally sign the keys with the following commands:

```
gpg –-lsign-key 83254FDB
gpg –-lsign-key 415202CC
```

Note that referring to a key by its ID instead of its email name prevents any ambiguity.

Step 6: Ensure the Default Key

All versions of the SRS APIs expect the public key of the default key pair to be the one that was submitted to `registry@srsplus.com`. You must ensure that only one private key exists in your keyring and that it corresponds to the public key you submitted. If necessary, run the `gpg --delete-secret-key` command for each extraneous private key.

That's all there is to it! GnuPG should now be configured correctly for use with the SRS APIs.

6. Putting It Together

Now that you are familiar with the SRS commands and have set up GnuPG, you can begin planning the development of your client. This section discusses some common issues you should consider before beginning your development effort.

6.1 Minimal Client Features

You should make sure your client can perform the following tasks, since they are the most common:

- Create contacts for your customers
- Check the availability of a domain
- Register a domain
- Change DNS information for a domain
- Release a domain
- Renew a domain

6.2 Command Dependencies

Recall that some commands require information gathered by other commands. You should consider the following dependencies:

- Registering a domain requires contact information obtained from a **CreateContact** command and pricing information obtained from a **DomainInfo** command.
- The **EditContact** command requires the contact ID from a contact record previously created with the **CreateContact** command. • The **RenewDomain** command requires pricing information obtained via the **Whois** command.
- A nameserver must be registered with the **RegisterNameServer** command *before* adding it to a domain via the **RegisterDomain** or **ChangeDomain** commands.

6.3 Contact Management

You should consider the method for managing contact IDs you will use. Refer back to section “4.3 The Contact Management Commands” for a description of the two most common methods.

6.4 Logging Transaction IDs and Request IDs

You should also decide if you want to send meaningful `TRANSACTIONID` parameters. If so, you will likely need to devise a system for creating unique IDs, such as using a sequence table in a database, or you may simply wish to send along a customer ID or order number with the transaction. Note that transaction IDs may be alphanumeric. Please keep the length of the ID string as small as possible. **Always specify a non-empty `TRANSACTIONID`, even if it is simply “0”.**

You should consider logging the `REQUESTID` values returned by the SRS server, which should be sufficient to resolve any disputes. Think of the `REQUESTID`s as receipts, or confirmation numbers.

7. The Certification Process

All SRSplus Partners must pass a suite of tests in order to achieve certification. Once certified, you are given access to the live SRS server and your account is made "Active." The test suite performs all the SRS commands and prints the client/server interactions to a log file. You submit this log file to `registry@srsplus.com` and await a response. Once the log file is processed, you will be sent an email either confirming your certification, or informing you that there was a problem with your submission and that you must re-submit another log. In the case of failure, you will be given suggestions for correcting your implementation.

For your convenience, we provide appropriate code to perform the certification tests in the API Toolkits. For Perl, a script called `certify.pl` is provided. For the C APIs, the sample `certify.c`, in conjunction with the sample `Makefile`, will create an executable called `certify.exe`. For the COM object, a Visual Basic based executable called `Certify.exe` is provided. Detailed instructions for running the tests are provided in the Reference document for each API.

APPENDIX A:

Country Codes for Contact Commands

AC	Ascension Island	HR	Croatia (local name: Hrvatska)
AD	Andorra	HT	Haiti
AE	United Arab Emirates	HU	Hungary
AF	Afghanistan	ID	Indonesia
AG	Antigua And Barbuda	IE	Ireland
AI	Anguilla	IL	Israel
AL	Albania	IM	Isle of Man
AM	Armenia	IN	India
AO	Angola	IO	British Indian Ocean Territory
AQ	Antarctica	IQ	Iraq
AR	Argentina	IR	Iran (Islamic Republic Of
AS	American Samoa	IS	Iceland
AT	Austria	IT	Italy
AU	Australia	JE	Jersey
AW	Aruba	JM	Jamaica
AZ	Azerbaijan	JO	Jordan
BA	Bosnia and Herzegovina	JP	Japan
BB	Barbados	KE	Kenya
BD	Bangladesh	KG	Kyrgyzstan
BE	Belgium	KH	Cambodia
BF	Burkina Faso	KI	Kiribati
BG	Bulgaria	KM	Comoros
BH	Bahrain	KP	Korea, Democratic People's Republic Of
BI	Burundi	KR	Korea, Republic Of
BJ	Benin	KW	Kuwait
BM	Bermuda	KY	Cayman Islands
BN	Brunei Darussalam	KZ	Kazakhstan
BO	Bolivia	LA	Lao People's Democratic Republic
BR	Brazil	LB	Lebanon
BS	Bahamas	LI	Liechtenstein
BT	Bhutan	LK	Sri Lanka
BV	Bouvet Island	LR	Liberia
BW	Botswana	LS	Lesotho
BY	Belarus	LT	Lithuania
BZ	Belize	LU	Luxembourg
CA	Canada	LV	Latvia
CC	Cocos (Keeling) Islands	LY	Libyan Arab Jamahiriya
CD	Congo, Democratic People's Republic	MK	Macedonia, The Former Yugoslav
CF	Central African Republic	MO	Macau
CG	Congo	PF	French Polynesia
CH	Switzerland	PM	St. Pierre And Miquelon
CI	Cote d'Ivoire	SB	Solomon Islands
CK	Cook Islands	SD	Sudan
CL	Chile	SE	Sweden
CM	Cameroon	SG	Singapore

CN	China	SH	St. Helena
CO	Colombia	SI	Slovenia
CR	Costa Rica	SJ	Svalbard and Jan Mayen Islands
CU	Cuba	SK	Slovakia (Slovak Republic)
CV	Cape Verde	SL	Sierra Leone
CX	Christmas Island	SO	Somalia
CY	Cyprus	SR	Suriname
CZ	Czech Republic	SV	El Salvador
DE	Germany	SY	Syrian Arab Republic
DJ	Djibouti	SZ	Swaziland
DK	Denmark	TC	Turks and Caicos Islands
DM	Dominica	TD	Chad
DO	Dominican Republic	TF	French Southern Territories
DZ	Algeria	TG	Togo
EC	Ecuador	TH	Thailand
EE	Estonia	TJ	Tajikistan
EG	Egypt	TK	Tokelau
EH	Western Sahara	TM	Turkmenistan
ER	Eritrea	TN	Tunisia
ES	Spain	TO	Tonga
ET	Ethiopia	TP	East Timor
FI	Finland	TR	Turkey
FJ	Fiji	TT	Trinidad and Tobago
FK	Falkland Islands (Malvinas)	TV	Tuvalu
FO	Faroe Islands	TW	Taiwan, Province Of China
FR	France	TZ	Tanzania, United Republic Of
GA	Gabon	UA	Ukraine
GD	Grenada	UG	Uganda
GE	Georgia	UK	United Kingdom
GF	French Guiana	UM	United States Minor Outlying Islands
GG	Guernsey	US	United States
GH	Ghana	UY	Uruguay
GI	Gibraltar	UZ	Uzbekistan
GL	Greenland	VA	Vatican City State (Holy See)
GM	Gambia	VE	Venezuela
GN	Guinea	VG	Virgin Islands (British)
GP	Guadeloupe	VI	Virgin Islands (U.S.)
GQ	Equatorial Guinea	VN	Viet Nam
GR	Greece	VU	Vanuatu
GS	South Georgia and The South Sandwich Islands	WF	Wallis and Futuna Islands
GT	Guatemala	YE	Yemen
GU	Guam	YU	Yugoslavia
GW	Guinea-Bissau	ZA	South Africa
GY	Guyana	ZM	Zambia
HK	Hong Kong	ZR	Zaire
HM	Heard and Mc Donald Islands	ZW	Zimbabwe
HN	Honduras		