

C API Reference for Linux

Table of Contents

Section	Page #
1. Introduction	1
2. Development Tasks	1
3. The API Toolkit	1
3.1 Organization	1
3.2 Dependencies	2
3.2.1 Tools	2
3.2.2 OpenSSL	2
3.2.3 GnuPG	3
4. Configuration and Compilation	3
4.1 Partner Configuration	3
4.2 Server Configuration	3
4.3 Logging Configuration	3
5. Function Reference	4
5.1 NameValueLists	4
5.1.1 Deallocation Responsibilities	4
5.1.2 NameValueList Helper Function Listing	4
5.2 C API for Linux Specific Functions	5
5.3 Query Functions	6
SrsAccountBalance	6
SrsDomainInfo	7
SrsMultiDomainInfo	8
SrsWhois	9
5.4 Contact Management Functions	10
SrsCreateContact	10
SrsEditContact	11
SrsGetContactInfo	12
5.5 Domain Registration and Manipulation Functions	13
SrsRegisterDomain	13
SrsChangeDomain	15
SrsReleaseDomain	16

SrsRenewDomain	17
5.6 Nameserver Functions	19
SrsRegisterNameServer	19
SrsReleaseNameServer	20
SrsNameServerInfo.....	21
5.7 Domain Transfer Functions	22
SrsRequestTransfer.....	22
SrsOutboundTransferResponse	23
SrsViewPendingTransfers	24
6. Testing and Certification	25
6.1 The Test Environment vs. the Production Environment ..	25
6.2 Running the Certification Test.....	25
6.2.1 Logging Client Activity	25
6.2.2 Running the Text Executable.....	26
6.3 Switching Environments.....	26
7. The Entire Integration Pathway	26
8. Next Steps.....	26

C API for Linux Reference

1. Introduction

This document details specific information you will need to develop an SRS client for the Linux operating system using the C programming language. It is assumed at this point that you have read the Technical Overview and Developer's Guide documents. As you develop your SRS client, you will want to have a copy of the Developer's Guide at hand since it details all the legal input and output values for each SRS command. We will not duplicate that information here since it is common to all API versions. Rather, we will focus on issues and features specific to the C API for Linux.

2. Development Tasks

The development of your SRS client will at a minimum involve the following steps:

- Download and uncompress the C API Toolkit for Linux
- Download, install and configure GnuPG, the Gnu Privacy Guard (as documented in the Developer's Guide, section 5.1)
- Download and install required support libraries (i.e., OpenSSL).
- Customize the header files in the API with your configuration information (e.g., your partner ID, your cryptographic key name, preferred log file name, etc.)
- Compile the API source
- Create your custom client application and link it to the API object files
- Test your client
- Run the certification tests.
- After passing certification, reconfigure and recompile the API sources to use the SRS production (live) server.

This Reference, along with supplemental material from the Developer's Guide, provides detailed instructions for completing all the above steps, except for the creation of your custom client. We do provide many code examples that you can use as a base for your own development efforts.

3. The API Toolkit

If you have not done so already, download the C API Toolkit for Linux from the Developer's Library section of our website:

http://www.srsplus.com/en/srsplus/partners_dev_library.shtml

Uncompress the archive contents to a directory of your choosing.

3.1 Organization

The Toolkit is organized into the following directory structure:

Toolkit	Contains <code>readme</code> and/or <code>releasenotes</code> files. Read these files with a text viewer for the latest information about the API.
Toolkit/docs	Contains a copy of this document and the Developer's Guide.

Toolkit/keys	Contains a keyblock file called <code>SRS.KEY</code> containing the SRS public keys for importing into your GnuPG database with the “ <code>gpg --import SRS.KEY</code> ” command.																														
Toolkit/source	<p>Contains the <code>.h</code> and <code>.c</code> files that make up the API, as well as a sample <code>Makefile</code> and <code>main.c</code> file that together will create an executable suitable for running the certification test.</p> <table border="1"> <thead> <tr> <th colspan="2">Filelist</th> </tr> <tr> <th>Filename</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>SrsDefs.h</code></td> <td>Declares common symbols and strings for use throughout the API. You will need to edit this file to specify certain parameters, such as your partner ID and to which SRS server you are connecting (test or production).</td> </tr> <tr> <td><code>SrsClient.h</code></td> <td>Function prototypes for all public SRS functions. This is the file you will <code>#include</code> in your client implementation.</td> </tr> <tr> <td><code>SrsClient.c</code></td> <td>Implementation of all SRS functions.</td> </tr> <tr> <td><code>NameValueList.h</code></td> <td>typedef and prototypes for the <code>NameValueList</code> helper functions.</td> </tr> <tr> <td><code>NameValueList.c</code></td> <td>Implementation of the <code>NameValueList</code> helper functions.</td> </tr> <tr> <td><code>main.c</code></td> <td>Example program that runs the entire suite of certification test commands.</td> </tr> <tr> <td><code>Makefile</code></td> <td>Sample makefile for creating the certification suite executable.</td> </tr> <tr> <td><code>SrsCrypto.h</code></td> <td>Function prototypes for the cryptographic support functions.</td> </tr> <tr> <td><code>SrsCrypto.c</code></td> <td>Implementation of the cryptographic support functions. Pipes data through the GnuPG executable, <code>gpg.exe</code>.</td> </tr> <tr> <td><code>SrsInet.h</code></td> <td>Function prototypes for the Internet support functions.</td> </tr> <tr> <td><code>SrsInet.c</code></td> <td>Implementation of the Internet support functions. Uses sockets and OpenSSL.</td> </tr> <tr> <td><code>SocketHelper.h</code></td> <td>Function prototypes for socket helper functions.</td> </tr> <tr> <td><code>SocketHelper.c</code></td> <td>Implementation of the socket helper functions. Basically, a socket connection helper routine.</td> </tr> </tbody> </table>	Filelist		Filename	Description	<code>SrsDefs.h</code>	Declares common symbols and strings for use throughout the API. You will need to edit this file to specify certain parameters, such as your partner ID and to which SRS server you are connecting (test or production).	<code>SrsClient.h</code>	Function prototypes for all public SRS functions. This is the file you will <code>#include</code> in your client implementation.	<code>SrsClient.c</code>	Implementation of all SRS functions.	<code>NameValueList.h</code>	typedef and prototypes for the <code>NameValueList</code> helper functions.	<code>NameValueList.c</code>	Implementation of the <code>NameValueList</code> helper functions.	<code>main.c</code>	Example program that runs the entire suite of certification test commands.	<code>Makefile</code>	Sample makefile for creating the certification suite executable.	<code>SrsCrypto.h</code>	Function prototypes for the cryptographic support functions.	<code>SrsCrypto.c</code>	Implementation of the cryptographic support functions. Pipes data through the GnuPG executable, <code>gpg.exe</code> .	<code>SrsInet.h</code>	Function prototypes for the Internet support functions.	<code>SrsInet.c</code>	Implementation of the Internet support functions. Uses sockets and OpenSSL.	<code>SocketHelper.h</code>	Function prototypes for socket helper functions.	<code>SocketHelper.c</code>	Implementation of the socket helper functions. Basically, a socket connection helper routine.
Filelist																															
Filename	Description																														
<code>SrsDefs.h</code>	Declares common symbols and strings for use throughout the API. You will need to edit this file to specify certain parameters, such as your partner ID and to which SRS server you are connecting (test or production).																														
<code>SrsClient.h</code>	Function prototypes for all public SRS functions. This is the file you will <code>#include</code> in your client implementation.																														
<code>SrsClient.c</code>	Implementation of all SRS functions.																														
<code>NameValueList.h</code>	typedef and prototypes for the <code>NameValueList</code> helper functions.																														
<code>NameValueList.c</code>	Implementation of the <code>NameValueList</code> helper functions.																														
<code>main.c</code>	Example program that runs the entire suite of certification test commands.																														
<code>Makefile</code>	Sample makefile for creating the certification suite executable.																														
<code>SrsCrypto.h</code>	Function prototypes for the cryptographic support functions.																														
<code>SrsCrypto.c</code>	Implementation of the cryptographic support functions. Pipes data through the GnuPG executable, <code>gpg.exe</code> .																														
<code>SrsInet.h</code>	Function prototypes for the Internet support functions.																														
<code>SrsInet.c</code>	Implementation of the Internet support functions. Uses sockets and OpenSSL.																														
<code>SocketHelper.h</code>	Function prototypes for socket helper functions.																														
<code>SocketHelper.c</code>	Implementation of the socket helper functions. Basically, a socket connection helper routine.																														

3.2 Dependencies

The C API for Linux requires a few additional components that are not part of the Toolkit download. These dependencies are detailed in the sections below.

3.2.1 Tools

It is assumed that you are using a distribution of Linux that includes the Gnu C Compiler or its equivalent. The C API is written to be as portable as possible. If you are using a compiler other than `gcc`, though, you may need to modify the source files for compiler-specific requirements. Similarly, it is assumed you have a standard `make` utility installed. If not, you may need to create your own `makefile` or modify the sample one.

3.2.2 OpenSSL

The C API for Linux uses the OpenSSL library to perform its SSL operations. OpenSSL is available for download at <http://www.openssl.org/>. The download contains instructions for configuring and installing the OpenSSL library. We also provide a HOW-TO document on our website at http://www.srsplus.com/en/srsplus/partners_faqs_tech.shtml.

3.2.3 GnuPG

If you have not already followed the instructions in the Developer's Guide for installing and configuring GnuPG, you should do so now. In particular, you need to create your public/private key pair and submit your public key to registry@srsplus.com as soon as practicable. 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 configuring the API. You can still compile and link the API with your client code, but without the partner ID, you will not be able to communicate with the SRS servers.

4. Configuration and Compilation

The API requires information about your public/private key pair and your partner ID for API calls to succeed. In addition, there are some preprocessor symbols (i.e., #defines) you need to understand.

4.1 Partner Configuration

Before you compile the API source, you will need to customize the partner information in the header file `SrsDefs.h`. Specifically, you will need these three pieces of information:

- The email name used when creating your public/private key pair in GnuPG.
- The passphrase (i.e., password) for your private key.
- Your partner ID, as given to you by SRSplus.

Edit the file `SrsDefs.h` and make the following changes:

- Enter the email name for your key pair as the value for `#define REGISTRAR_EMAIL`
- Enter the password for your private key as the value for `#define PASSPHRASE`
- Enter your partner ID as the value for `#define REGISTRAR_ID`

Make sure that the values for `REGISTRAR_EMAIL` and `PASSPHRASE` are quoted since they are strings. Conversely, ensure that the value for `REGISTRAR_ID` is *not* quoted since it is a numeric value.

4.2 Server Configuration

The preprocessor symbol `TEST_MODE` determines which server environment the API will use. When defined, `TEST_MODE` routes all traffic to the test SRS server. If not defined, all traffic will be sent to the production SRS server. Section 6.3 below details the use of this symbol as it relates to switching from the test environment to the production environment *after* passing the certification tests. For now, you should leave the symbol defined.

4.3 Logging Configuration

The only other symbols you may need to change in the header file are related to the built-in logging functions. The symbol `LOGGING_ENABLED` will, if defined, cause all outgoing and incoming raw command packet data to be collected and written to the file specified by the symbol `LOGFILE`. The default `LOGFILE` value is `"SRS.LOG"`. You may change it to suit your particular needs or preferences. While building and testing your client, you should leave `LOGGING_ENABLED` defined (the default configuration) to help with debugging your client, should the need arise. Note that you will need this logging functionality when running the certification tests, as detailed in section 6.2.1 below.

5. Function Reference

Each SRS command has a corresponding API function prefaced with the string "Srs." In addition, there are a couple of C API-specific functions with which to familiarize yourself. You will need to consult the SRS Command Reference in the Developer's Guide to determine legal input and output parameters for the functions. In the Developer's Guide, all parameters are referred to as **strings** or lists of **name-value pairs**. In the C API, these two types correspond directly to the types `char*` and `NameValueList`. As a C programmer, you should be quite familiar with the `char*` type. Let us now acquaint you with the `NameValueList` type.

5.1 NameValueLists

The `NameValueList` type is used to exchange data with various API functions. It is implemented as a very simple linked list of dynamically allocated name-value pair strings, called `NameValuePair`. This type is very similar to the associative array (hash) type in Perl, or a string-to-string map class in C++.

The example code for the SRS Functions (below) shows the `NameValueList` helper functions in action. You may want to look ahead at the example for **SrsCreateContact** right now. It demonstrates most of the helper functions in a practical application.

5.1.1 Deallocation Responsibilities

Many functions take a pointer to a `NameValueList` as an input parameter and a handle (i.e., pointer to a pointer) to a `NameValueList` as an output parameter. It is the responsibility of the caller to free any `NameValueList` objects received via a handle. `NameValueList` objects sent into a method are likewise the responsibility of the caller.

The basic rule: Free all `NameValueLists`.

5.1.2 NameValueList Helper Function Listing

The following table lists the helper functions implemented in `NameValueList.c`:

NameValueList Helper Functions	
<code>NameValueList* NameValueList_Alloc(void);</code>	Allocates an empty <code>NameValueList</code> header (i.e., no child nodes are allocated)
<code>void NameValueList_Free(NameValueList* pList);</code>	Frees a <code>NameValueList</code> , all its child <code>NameValuePair</code> nodes and all the strings in the nodes.
<code>int NameValueList_AddPair</code> (<code>NameValueList* pList,</code> <code>char* name,</code> <code>char* value</code>) <code>;</code>	Allocates a <code>NameValuePair</code> and memory for the input strings, copies the strings into the structure and links the node to the given list. If the key name already exists in the list, then the old value string will be released replaced with the new value.
<code>char* NameValueList_Lookup(char* name);</code>	Returns the value string for the given input name or NULL if not found.
<code>int NameValueList_Size(NameValueList*);</code>	Returns the number of nodes in a list

5.2 C API for Linux Specific Functions

There are only two functions in the C API for Linux that do not correspond exactly to SRS commands. They are:

SrsStartup
Description: Initializes API and libraries
Function Prototype: <code>int SrsStartup(void);</code>
Notes: This function should be called exactly once at the beginning of your client code before calling any other API function.

SrsShutdown
Description: Shuts down the API and releases any internally allocated resources.
Function Prototype: <code>int SrsShutdown(void);</code>
Notes: This function should be called exactly once in your client code <i>after</i> you are finished using all API functions. Do not call any API functions after calling this routine.

(The remainder of this page is intentionally blank.)

5.3 Query Functions

These functions correspond directly to the SRS Query Commands. Refer to section 4.2 of the Developer's Guide for a detailed listing of legal input and output parameters.

SrsAccountBalance

Description:

Returns balance information from your partner account.

Function Prototype:

```
int SrsAccountBalance(char* TLD, NameValueList** ppResponse);
```

Example Code:

```
#include "SrsClient.h"

int main(int argc, char* argv[])
{
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;
    char* szValue = NULL;

    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }

    if( SUCCESS != SrsAccountBalance("tv", &pResponse ) )
    {
        printf( "Error: SrsAccountBalance failed\n");
        SrsShutdown();
        exit ( 1 );
    }
    else
    {
        //
        // Find the STORED VALUE, which is the actual balance
        //
        if ( pResponse )
        {
            szValue = (char*)NameValueList_Lookup( pResponse, "STORED VALUE");
            if( szValue )
            {
                printf( "Your available balance is $%s\n", szValue );
            }
            else
            {
                printf( "Couldn't find STORED VALUE\n");
            }
            //
            // always free returned NameValueLists
            //
            NameValueList_Free( pResponse );
        }
    }
    SrsShutdown();
    return 0;
}
```

SrsDomainInfo

Description:

Returns availability information for a specific domain name.

Function Prototype:

```
int SrsDomainInfo(char* domain, char* TLD, NameValueList** ppResponse);
```

Example Code:

```
#include "SrsClient.h"

int main(int argc, char* argv[])
{
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;

    if( argc != 2)
    {
        printf ( "Usage: domaininfo <domain>\n");
        printf ( "Example: domaininfo mydomain\n\n");
        exit( 1 );
    }

    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }

    if( SUCCESS != SrsDomainInfo( argv[1], "tv", &pResponse ) )
    {
        printf( "Error: SrsDomainInfo failed\n");
        SrsShutdown();
        exit ( 1 );
    }
    else
    {
        //
        // print out the NameValueList response
        //
        if ( pResponse )
        {
            pNode = pResponse->head;
            while( pNode )
            {
                printf( "%s: %s\n", pNode->name, pNode->value );
                pNode = pNode->next;
            }
            //
            // always free returned NameValueLists
            //
            NameValueList_Free( pResponse );
        }

    }

    SrsShutdown();
    return 0;
}
```

SrsMultiDomainInfo

Description:

Returns availability and pricing information for many domain names at once.

Function Prototype:

```
int SrsMultiDomainInfo( NameValueList*   pQuery,
                        NameValueList**  ppResponse );
```

Example code:

```
#include "SrsClient.h"
int main(int argc, char* argv[])
{
    NameValueList* pList;
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;
    int i;
    char cbName[32];

    if( argc < 2)
    {
        printf ( "Usage: multidomaininfo <domain 1...n>\n");
        printf ( "Example: multidomaininfo domain1 domain2\n\n");
        exit( 1 );
    }
    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }
    pList = NameValueList_Alloc();
    if( pList )
    {
        for( i=1; i < argc; i++)
        {
            sprintf( cbName, "DOMAIN %d", i );
            NameValueList_AddPair( pList, cbName, argv[i]);
            sprintf( cbName, "TLD %d", i );
            NameValueList_AddPair( pList, cbName, "tv");
        }
        if (SUCCESS == SrsMultiDomainInfo( pList, &pResponse) )
        {
            // print out the NameValueList response
            if ( pResponse )
            {
                pNode = pResponse->head;
                while( pNode )
                {
                    printf( "%s: %s\n", pNode->name, pNode->value );
                    pNode = pNode->next;
                }
                NameValueList_Free( pResponse );
            }
        }
        NameValueList_Free( pList );
    }

    SrsShutdown();
    return 0;
}
```

SrsWhois

Description:

Obtains domain name server and contact information for the given domain and TLD combination.

Function Prototype:

```
int SrsWhois(char* domain, char* TLD, NameValueList** ppResponse);
```

Example Code:

```
#include "SrsClient.h"

int main(int argc, char* argv[])
{
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;

    if( argc != 2)
    {
        printf ( "Usage: whois <domain> (do not include extension)\n");
        printf ( "Example: whois mydomain\n\n");
        exit( 1 );
    }

    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }

    if( SUCCESS != SrsWhois( argv[1], "tv", &pResponse ) )
    {
        printf( "Error: SrsWhois failed\n");
        SrsShutdown();
        exit ( 1 );
    }
    else
    {
        //
        // print out the NameValueList response
        //
        if ( pResponse )
        {
            pNode = pResponse->head;
            while( pNode )
            {
                printf( "%s: %s\n", pNode->name, pNode->value );
                pNode = pNode->next;
            }
            //
            // always free returned NameValueLists
            //
            NameValueList_Free( pResponse );
        }
        SrsShutdown();
        return 0;
    }
}
```

5.4 Contact Management Functions

These functions correspond directly to the SRS Contact Management Commands. Refer to section 4.3 of the Developer's Guide for a detailed listing of legal input and output parameters.

SrsCreateContact

Description:

Creates a new contact record and returns its unique ID.

Function Prototype:

```
int SrsCreateContact( const char*      transaction_id,
                    NameValueList*  pContactData,
                    NameValueList** ppResponse );
```

Example code:

```
#include "SrsClient.h"
int main(int argc, char* argv[])
{
    NameValueList* pList;
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;
    char* szTransID = "1";

    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }
    pList = NameValueList_Alloc();
    if( pList )
    {
        NameValueList_AddPair( pList, "FNAME", "John");
        NameValueList_AddPair( pList, "LNAME", "Public");
        NameValueList_AddPair( pList, "ORGANIZATION", "DotTV");
        NameValueList_AddPair( pList, "EMAIL", "jqpublic@www.tv");
        NameValueList_AddPair( pList, "ADDRESS1", "100 Main Street");
        NameValueList_AddPair( pList, "ADDRESS2", "Suite 200");
        NameValueList_AddPair( pList, "CITY", "Los Angeles");
        NameValueList_AddPair( pList, "PROVINCE", "CA");
        NameValueList_AddPair( pList, "POSTAL CODE", "90028");
        NameValueList_AddPair( pList, "COUNTRY", "US");
        NameValueList_AddPair( pList, "PHONE", "555-1212");

        if (SUCCESS == SrsCreateContact( szTransID, pList, &pResponse) )
        {
            // print out the NameValueList response
            if ( pResponse )
            {
                pNode = pResponse->head;
                while( pNode )
                {
                    printf( "%s: %s\n", pNode->name, pNode->value );
                    pNode = pNode->next;
                }
                NameValueList_Free( pResponse );
            }
        }
        NameValueList_Free( pList );
    }
    SrsShutdown();
    return 0;
}
```

SrsEditContact

Description:

Modifies fields in an existing contact record.

Function Prototype:

```
int SrsEditContact( const char*    transaction_id,
                   NameValueList* pContactData,
                   NameValueList** ppResponse );
```

Example Code: In this code fragment, we will change the email address and phone number for an existing contact. We use the fictional contact ID of 1000 in the example. In real code, the ID must be for an existing contact previously created with **SrsCreateContact**.

```
//
//call SrsStartup before we get here
//
NameValueList* pList;
NameValueList* pResponse = NULL;
NameValuePair* pNode = NULL;
char* szTransID = "1";

pList = NameValueList_Alloc();
if( pList )
{
    // update the email and phone fields
    NameValueList_AddPair( pList, "CONTACTID", "1000");
    NameValueList_AddPair( pList, "EMAIL", "somebody@www.tv");
    NameValueList_AddPair( pList, "PHONE", "(310)555-1212");

    if (SUCCESS == SrsEditContact( szTransID, pList, &pResponse) )
    {
        // print out the NameValueList response
        if ( pResponse )
        {
            pNode = pResponse->head;
            while( pNode )
            {
                printf( "%s: %s\n", pNode->name, pNode->value );
                pNode = pNode->next;
            }
            NameValueList_Free( pResponse );
        }
    }
    NameValueList_Free( pList );
}
//
// clean up code would go here (i.e., SrsShutdown)
//
```

SrsGetContactInfo

Description:

Retrieves fields of an existing contact record.

Function Prototype:

```
int SrsGetContactInfo( const char*      [in] contact_id,
                      NameValueList** [in, out] ppResponse );
```

Example code:

```
#include "SrsClient.h"
int main(int argc, char* argv[])
{
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;

    if( argc != 2 )
    {
        printf("Usage: GetContactInfo <contactID>\n");
        exit( 1 );
    }

    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }

    if (SUCCESS == SrsGetContactInfo( argv[1], &pResponse) )
    {
        // print out the NameValueList response
        if ( pResponse )
        {
            pNode = pResponse->head;
            while( pNode )
            {
                printf( "%s: %s\n", pNode->name, pNode->value );
                pNode = pNode->next;
            }
            NameValueList_Free( pResponse );
        }
    }
    else
    {
        printf("GetContactInfo failed.\n");
    }

    SrsShutdown();
    return 0;
}
```

5.5 Domain Registration and Manipulation Functions

These functions correspond directly to the SRS Domain Registration and Manipulation Commands. Refer to section 4.4 of the Developer's Guide for a detailed listing of legal input and output parameters.

SrsRegisterDomain
Description: Register (buy) a domain.
Function Prototype: <pre>int SrsRegisterDomain(const char* transaction_id, NameValueList* pDomainInfo, NameValueList** ppResponse);</pre>

Example Code:

```
#include "SrsClient.h"

int main(int argc, char* argv[])
{
    NameValueList* pList;
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;
    char* szTransID = "1";
    char szPrice[16];
    int price = 0;

    if ( argc != 4 )
    {
        printf ("Usage: registerdomain <domain> <responsible person ID>
<tech contact ID>\n");
        exit ( 1 );
    }

    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }
    //
    // first, make sure domain is available and get the PRICE
    //
    if ( SUCCESS != SrsDomainInfo( argv[1], "tv", &pResponse ) )
    {
        printf( "SrsDomainInfo failed!\n");
        SrsShutdown();
        exit ( 1 );
    }
    if( 0 == strcmp( NameValueList_Lookup(pResponse,"DOMAIN STATUS"),
"UNAVAILABLE" ) )
    {
        printf( "%s.tv is not available.\n", argv[1] );
        NameValueList_Free( pResponse );
        SrsShutdown();
        return 0;
    }
    //
    // it was available, so get the price
    //
    strcpy( szPrice, NameValueList_Lookup( pResponse, "PRICE" ) );
```

```

if( 0 == strcmp(szPrice,"" ))
{
    printf( "No price returned.  Cannot register domain.\n");
    NameValueList_Free( pResponse );
    SrsShutdown();
    exit ( 1 );
}
//
// remember to deallocate the response
//
NameValueList_Free( pResponse );
//
// attempt to register the name for 1 year,
// using the contact IDs from the command line
//
pList = NameValueList_Alloc();
if( pList )
{
    NameValueList_AddPair( pList, "DOMAIN", argv[1]);
    NameValueList_AddPair( pList, "TLD", "TV");
    NameValueList_AddPair( pList, "RESPONSIBLE PERSON", argv[2] );
    NameValueList_AddPair( pList, "TECHNICAL CONTACT", argv[3] );
    NameValueList_AddPair( pList, "TERM YEARS", "1");
    NameValueList_AddPair( pList, "PRICE", szPrice );

    if (SUCCESS == SrsRegisterDomain( szTransID, pList, &pResponse) )
    {
        //
        // print out the NameValueList response
        //
        if ( pResponse )
        {
            pNode = pResponse->head;
            while( pNode )
            {
                printf( "%s: %s\n", pNode->name, pNode->value);
                pNode = pNode->next;
            }
            NameValueList_Free( pResponse );
        }
    }
    NameValueList_Free( pList );
}

SrsShutdown();
return 0;
}

```

SrsChangeDomain

Description:

Modify contact and domain name server information for a domain.

Function Prototype:

```
int SrsChangeDomain( const char*    transaction_id,
                    NameValueList* pDomainInfo,
                    NameValueList** ppResponse );
```

Example Code: This code fragment shows how you might set DNS information for a domain after it has been registered.

```
//
//call SrsStartup before we get here
//
NameValueList* pList;
NameValueList* pResponse = NULL;
NameValuePair* pNode = NULL;
char* szTransID = "1";

pList = NameValueList_Alloc();
if( pList )
{
    // update the DNS server fields
    NameValueList_AddPair( pList, "DNS SERVER NAME 1", "ns1.somedomain.com");
    NameValueList_AddPair( pList, "DNS SERVER NAME 2", "ns2.somedomain.com");

    if (SUCCESS == SrsChangeDomain( szTransID, pList, &pResponse) )
    {
        // print out the NameValueList response
        if ( pResponse )
        {
            pNode = pResponse->head;
            while( pNode )
            {
                printf( "%s: %s\n", pNode->name, pNode->value );
                pNode = pNode->next;
            }
            NameValueList_Free( pResponse );
        }
    }
    NameValueList_Free( pList );
}
//
// clean up code would go here (i.e., SrsShutdown)
//
```

SrsReleaseDomain

Description:

Release a domain back into the pool of available domain names.

Function Prototype:

```
int SrsReleaseDomain( const char*    transaction_id,
                    NameValueList*  pDomainInfo,
                    NameValueList** ppResponse );
```

Example Code:

```
#include "SrsClient.h"
int main(int argc, char* argv[])
{
    NameValueList* pList;
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;
    char* szTransID = "1";

    if( argc != 2)
    {
        printf ( "Usage: releasedomain <domain>\n");
        printf ( "Example: releasedomain mydomain\n\n");
        exit( 1 );
    }

    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }

    pList = NameValueList_Alloc();
    if( pList )
    {
        NameValueList_AddPair( pList, "DOMAIN", argv[1]);
        NameValueList_AddPair( pList, "TLD", "TV");

        if (SUCCESS == SrsReleaseDomain( szTransID, pList, &pResponse) )
        {
            //
            // print out the NameValueList response
            //
            if ( pResponse )
            {
                pNode = pResponse->head;
                while( pNode )
                {
                    printf( "%s: %s\n", pNode->name, pNode->value );
                    pNode = pNode->next;
                }
                NameValueList_Free( pResponse );
            }
        }
        NameValueList_Free( pList );
    }
    SrsShutdown();
    return 0;
}
```

SrsRenewDomain

Description:

Renews a domain registration for a given number of years

Function Prototype:

```
int SrsRenewDomain( const char*    transaction_id,
                   NameValueList* pDomainInfo,
                   NameValueList** ppResponse );
```

Example Code:

```
#include "SrsClient.h"
int main(int argc, char* argv[])
{
    NameValueList* pList;
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;
    char* szTransID = "1";
    char* szPrice = NULL;
    char cbPrice[16];
    int price = 0;

    if ( argc != 3 )
    {
        printf ( "Usage: renewdomain <domain> <# of years>\n" );
        exit ( 1 );
    }

    if( SUCCESS != SrsStartup() )
    {
        printf ( "Fatal error: Could not start up SRS.\n" );
        exit ( 1 );
    }
    //
    // get the PRICE via Whois
    //
    if ( SUCCESS != SrsWhois( argv[1], "tv", &pResponse ) )
    {
        printf( "SrsDomainInfo failed!\n" );
        SrsShutdown();
        exit ( 1 );
    }
    szPrice = (char*)NameValueList_Lookup( pResponse, "PRICE" );
    if( szPrice )
    {
        strcpy( cbPrice, szPrice );
    }
    else
    {
        printf( "No price returned.  Cannot register domain.\n" );
        NameValueList_Free( pResponse );
        SrsShutdown();
        exit ( 1 );
    }
    //
    // remember to deallocate the response
    //
    NameValueList_Free( pResponse );
    //
    // attempt to renew the domain,
```

```

// using the number of years from the command line
//
pList = NameValueList_Alloc();
if( pList )
{
    NameValueList_AddPair( pList, "DOMAIN", argv[1]);
    NameValueList_AddPair( pList, "TLD", "TV");
    NameValueList_AddPair( pList, "TERM YEARS ", argv[2] );
    NameValueList_AddPair( pList, "PRICE", cbPrice );

    if (SUCCESS == SrsRenewDomain( szTransID, pList, &pResponse) )
    {
        //
        // print out the NameValueList response
        //
        if ( pResponse )
        {
            pNode = pResponse->head;
            while( pNode )
            {
                printf( "%s: %s\n", pNode->name, pNode->value );
                pNode = pNode->next;
            }
            NameValueList_Free( pResponse );
        }
        NameValueList_Free( pList );
    }
}

SrsShutdown();
return 0;
}

```

(The remainder of this page is intentionally blank.)

5.6 Nameserver Functions

These functions correspond directly to the SRS Nameserver Commands. Refer to section 4.5 of the Developer's Guide for a detailed listing of legal input and output parameters.

SrsRegisterNameServer

Description:

Add a nameserver.

Function Prototype:

```
int SrsRegisterNameServer
(
  const char* transaction_id,
  NameValueList* pNSInfo,
  NameValueList** ppResponse
);
```

Example Code:

```
#include "SrsClient.h"
int main(int argc, char* argv[])
{
  NameValueList* pList;
  NameValueList* pResponse = NULL;
  NameValuePair* pNode = NULL;
  char* szTransID = "1";
  if( argc != 3)
  {
    printf ( "Usage: registernameserver <nameserver> <ip addr>\n");
    printf ( "Example: registernameserver nsl.server.tv 123.0.0.34\n\n");
    exit( 1 );
  }
  if( SUCCESS != SrsStartup() )
  {
    printf ("Fatal error: Could not start up SRS.\n");
    exit ( 1 );
  }
  pList = NameValueList_Alloc();
  if( pList )
  {
    NameValueList_AddPair( pList, "DNS SERVER NAME", argv[1]);
    NameValueList_AddPair( pList, "DNS SERVER IP", argv[2]);
    if (SUCCESS==SrsRegisterNameServer( szTransID, pList, &pResponse) )
    {
      //
      // print out the NameValueList response
      //
      if ( pResponse )
      {
        pNode = pResponse->head;
        while( pNode )
        {
          printf( "%s: %s\n", pNode->name, pNode->value );
          pNode = pNode->next;
        }
        NameValueList_Free( pResponse );
      }
    }
    NameValueList_Free( pList );
  }
  SrsShutdown();
  return 0;
}
```

SrsReleaseNameServer

Description:

Remove a previously registered nameserver.

Function Prototype:

```
int SrsRegisterNameServer
(
    const char* transaction_id,
    NameValueList* pNSInfo,
    NameValueList** ppResponse
);
```

Example Code:

```
#include "SrsClient.h"
int main(int argc, char* argv[])
{
    NameValueList* pList;
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;
    char* szTransID = "1";
    if( argc != 2)
    {
        printf ( "Usage: releasenameserver <domain>\n");
        printf ( "Example: releasenameserver ns1.server.tv\n\n");
        exit( 1 );
    }
    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }
    pList = NameValueList_Alloc();
    if( pList )
    {
        NameValueList_AddPair( pList, "DNS SERVER NAME", argv[1]);

        if (SUCCESS==SrsReleaseNameServer( szTransID, pList, &pResponse) )
        {
            //
            // print out the NameValueList response
            //
            if ( pResponse )
            {
                pNode = pResponse->head;
                while( pNode )
                {
                    printf( "%s: %s\n", pNode->name, pNode->value );
                    pNode = pNode->next;
                }
                NameValueList_Free( pResponse );
            }
        }
        NameValueList_Free( pList );
    }
    SrsShutdown();
    return 0;
}
```

SrsNameServerInfo

Description:

Obtain information about a previously registered nameserver.

Function Prototype:

```
int SrsNameServerInfo
(
const char* transaction_id,
NameValueList* pNSInfo,
NameValueList** ppResponse
);
```

Example Code:

```
#include "SrsClient.h"
int main(int argc, char* argv[])
{
    NameValueList* pList;
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;
    char* szTransID = "1";

    if( argc != 2)
    {
        printf ( "Usage: nameserverinfo <nameserver>\n");
        printf ( "Example: nameserverinfo ns1.mydomain.tv\n\n");
        exit( 1 );
    }

    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }

    pList = NameValueList_Alloc();
    if( pList )
    {
        NameValueList_AddPair( pList, "DNS SERVER NAME", argv[1]);

        if (SUCCESS==SrsNameServerInfo( szTransID, pList, &pResponse) )
        {
            //
            // print out the NameValueList response
            //
            if ( pResponse )
            {
                pNode = pResponse->head;
                while( pNode )
                {
                    printf( "%s: %s\n", pNode->name, pNode->value );
                    pNode = pNode->next;
                }
                NameValueList_Free( pResponse );
            }
        }
        NameValueList_Free( pList );
    }
    SrsShutdown();
    return 0;
}
```

5.7 Domain Transfer Functions

These functions correspond directly to the SRS Domain Registration and Manipulation Commands. Refer to section 4.6 of the Developer's Guide for a detailed listing of legal input and output parameters.

SrsRequestTransfer

Description:

Request a transfer of a domain to SRSplus from another registrar.

Function Prototype:

```
int SrsRequestTransfer
(
  const char*      [in] transaction_id,
  NameValueList*  [in] pTransferInfo,
  NameValueList** [in, out] ppResponse
);
```

Example Code:

```
#include "SrsClient.h"
int main(int argc, char* argv[])
{
    NameValueList* pResponse = NULL;
    NameValueList* pList = NULL;
    NameValuePair* pNode = NULL;
    char* szTransID = "1";

    if( argc != 4)
    {
        printf ( "Usage: transfer <domain> <tld> <email>\n");
        printf ( "Example: transfer testdomain com john@public.net\n\n");
        exit( 1 );
    }

    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }
    pList = NameValueList_Alloc();
    if( pList == NULL)
    {
        printf("Fatal error: Out of memory.\n");
        exit( 1 );
    }

    NameValueList_AddPair( pList, "DOMAIN", argv[1]);
    NameValueList_AddPair( pList, "TLD", argv[2]);
    NameValueList_AddPair( pList, "CURRENT ADMIN EMAIL", argv[3] );

    if (SUCCESS==SrsRequestTransfer( szTransID, pList, &pResponse) )
    {
        //
        // print out the NameValueList response
        //
        if ( pResponse )
        {
            pNode = pResponse->head;
            while( pNode )
            {
```

```

        printf( "%s: %s\n", pNode->name, pNode->value );
        pNode = pNode->next;
    }
    NameValueList_Free( pResponse );
}
}
NameValueList_Free( pList );
SrsShutdown();
return 0;
}

```

SrsOutboundTransferResponse

Description:

Respond to a transfer request where you are the losing partner. You may ACCEPT or DENY the pending request.

Function Prototype:

```

int SrsOutboundTransferResponse
(
    const char*      [in] transaction_id,
    const char*      [in] domain,
    const char*      [in] tld,
    const char*      [in] response_string,
    NameValueList**  [in, out] ppResponse
);

```

Example Code:

```

#include "SrsClient.h"
int main(int argc, char* argv[])
{
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;
    char* szTransID = "1";

    if( argc != 4)
    {
        printf ( "Usage: trans_response <domain> <tld> <ACCEPT|DENY>\n");
        printf ( "Example: trans_response testdomain com DENY\n\n");
        exit( 1 );
    }

    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }

    if (SUCCESS ==
        SrsOutboundTransferResponse ( szTransID, argv[1], argv[2], argv[3]) )
    {
        //
        // print out the NameValueList response
        //
        if ( pResponse )
        {
            pNode = pResponse->head;
            while( pNode )
            {
                printf( "%s: %s\n", pNode->name, pNode->value );
                pNode = pNode->next;
            }
        }
    }
}

```

```

        NameValueList_Free( pResponse );
    }
}
SrsShutdown();
return 0;
}

```

SrsViewPendingTransfers

Description:

Returns information about pending transfers, either INBOUND or OUTBOUND.

Function Prototype:

```

int SrsViewPendingTransfers
(
const char*          [in] transaction_id,
const char*          [in] transfer_type,
NameValueList**     [in, out] ppResponse
);

```

Example Code:

```

#include "SrsClient.h"
int main(int argc, char* argv[])
{
    NameValueList* pResponse = NULL;
    NameValuePair* pNode = NULL;
    char* szTransID = "1";

    if( argc != 2)
    {
        printf ( "Usage: view_transfers <INBOUND|OUTBOUND> <tld>\n");
        printf ( "Example: view_transfers OUTBOUND\n\n");
        exit( 1 );
    }

    if( SUCCESS != SrsStartup() )
    {
        printf ("Fatal error: Could not start up SRS.\n");
        exit ( 1 );
    }

    if (SUCCESS== SrsViewPendingTransfers ( szTransID, argv[1] ) )
    {
        //
        // print out the NameValueList response
        //
        if ( pResponse )
        {
            pNode = pResponse->head;
            while( pNode )
            {
                printf( "%s: %s\n", pNode->name, pNode->value );
                pNode = pNode->next;
            }
            NameValueList_Free( pResponse );
        }
    }
    SrsShutdown();
    return 0;
}

```

6. Testing and Certification

Once you send in your public key and receive back your partner ID, you will have access to the test SRS server. While developing your client, you can perform SRS commands in the test environment without fear of incurring charges or destroying registry data. This server is functionally equivalent to the production server, but it is separate and isolated. Other differences are explored next.

6.1 The Test Environment vs. the Production Environment

The main difference between the two is that the database used by the test server is isolated and separate from the production server database. This test database does not necessarily reflect real-time conditions. It gets updated periodically with data from the production database, but you should never make assumptions about the state of the test database at any time. A name you register in it one day may not be there the next.

The test server does not actually charge against your account when you perform **RegisterDomain** commands. In fact, while in the testing environment, your account will have a zero balance, but you will have nearly unlimited buying power. When you perform the **AccountBalance** command you will see a response like the following:

Name	Value
BUYING POWER	10000000.00
STORED VALUE	0.00
UNPAID CHARGES	100.00

Since the test server is updated from the live server occasionally, your `STORED VALUE` may reset to the current live value from time to time. Once the `STORED VALUE` reaches zero, the `UNPAID CHARGES` will increase until it reaches the limit specified in `BUYING POWER`. `BUYING POWER` will be sufficiently large to accommodate about 200,000 test registrations before your test `BUYING POWER` needs to be reset.

6.2 Running the Certification Tests

Once compiled into an executable, the sample code provided in the API Toolkit is used to perform the certification tests. You must successfully complete the certification tests and submit a log file of your client's activity during this test in order to gain access to the live production server. Once the log file is reviewed and approved, you will gain "Active" status and may begin selling domain names to your customers. You will be notified of this status via email.

6.2.1 Logging Client Activity

You will need to enable the API's built in logging feature while running the certification tests. The log file will show a record of the raw command messages being sent back and forth between your SRS client and the SRS server. This log file must be submitted to registry@srsplus.com in order to achieve certification and receive access to the live production SRS server.

Enable the logging feature by defining the symbol `LOGGING_ENABLED` in `SrsDefs.h`. The default is for the symbol to be defined, but if you have changed the setting, you must re-enable it. The preprocessor symbol `LOGFILE` specifies the name of the file that will contain the log entries. The default name is `SRS.LOG`. You may change this filename if you desire.

IMPORTANT: Whenever changing information in the header file `SrsDefs.h`, you must recompile the API source files so that the changes are propagated throughout the code.

6.2.2 Running the Test Executable

The sample `main.c` file will perform all the SRS commands required for the certification test. The supplied `makefile` is used to compile and link the executable. The steps required to run the test are:

- Ensure that the symbols `TEST_MODE` and `LOGGING_ENABLED` are defined in `SrsDefs.h`.
- Clean out existing object files and use the given `makefile` to compile the `certify.exe` executable (i.e., delete all object files and run `make`)
- Rename or delete any existing log file
- Run the `certify.exe` executable

After the test has finished, email the resulting log file (as an attachment) to registry@srsplus.com with the subject line "Certification Log." Be sure to specify your organization's name in the body of the message. Once the log is reviewed you will be sent a reply email with your results. You will either pass or fail the test. In the case of failure, you will be given specific reasons why your test was rejected and you may try again after making corrective changes. Once you pass the test, you are given access to the live production server and your partner account is made "Active" in our system. Assuming that you have met all other financial and contractual obligations, you may then modify the API configuration to use the live server and begin selling domain names immediately.

6.3 Switching Environments

As discussed in section 4.2, the server environment is determined by the presence or absence of the symbol `TEST_MODE`, which is defined in the `SrsDefs.h` file. When `TEST_MODE` is defined, you will be working against the test SRS server. When it is not defined (accomplished by commenting out the `#define TEST_MODE` line in `SrsDefs.h`), you will be working against the live production server. Attempting to use the production server before completing the certification tests will result in error responses, typically "Client not known to server." After you do pass the certification tests, you may still work in the test server. In fact you may switch back and forth whenever necessary, or even run concurrently in both environments from two separately configured systems.

IMPORTANT: Whenever you switch modes, you must recompile the API source files so that the change is propagated throughout the code.

7. The Entire Integration Pathway

For your easy reference, the complete process from download to certification is presented in figure 7.1. You may want to make a separate copy of this flowchart and use it to keep track of your progress.

8. Next Steps

You should now feel prepared to develop your custom SRS client. This is the last formal document in the progression specified in the documentation roadmap. For further detailed help, take a look at the various FAQs and HOW-TO documents on our website at http://www.srsplus.com/en/srsplus/partners_faq_tech.shtml. We have based the topics for these supplemental documents on feedback from developers who have already used the APIs to create SRS clients. If you are having a problem with a specific process, or with a certain step in the integration pathway, it is likely that others have had the same problem and that we have an available FAQ or HOW-TO document addressing the topic.

Figure 7.1: Pathway from Signup to Certification

