

SRS Component for Windows Reference Table of Contents

Section	Page #
1. Introduction	1
2. Development Tasks	1
3. The SRS Component Toolkit	1
3.1 Organization	1
3.2 Dependencies.....	2
4. Installation.....	3
4.1 GnuPG Installation for Windows	3
4.2 SRS Component Registration.....	3
5. Configuration	3
5.1 GnuPG Configuration.....	3
5.2 SRS Component Configuration	3
6. ISrs Method Reference	4
6.1 Programming Notes.....	4
6.2 Component Specific Methods	5
6.3 Query Functions	6
AccountBalance.....	6
DomainInfo	6
MultiDomainInfo	7
Whois.....	8
6.4 Contact Management Functions	9
CreateContact	9
Edit Contact.....	11
GetContactInfo.....	11
6.5 Domain Registration and Manipulation Functions	12
RegisterDomain.....	12
ChangeDomain	14
ReleaseDomain	15
RenewDomain.....	16

6.6 Nameserver Functions	17
RegisterNameServer	17
ReleaseNameServer	18
NameServerInfo.....	18
6.7 Domain Transfer Functions	19
RequestTransfer.....	19
OutboundTransferResponse	19
ViewPendingTransfers	20
7. Testing and Certification	20
7.1 The Testing vs. Production Environment.....	20
7.2 Running the Certification Tests.....	21
7.3 Switching Environments	22
8. The Entire Integration Pathway	22
9. Next Steps.....	22
Appendix A: ASP Tips.....	23
Appendix B: C++ Tips	24

SRS Component for Windows Reference

1. Introduction

This document details specific information you will need to develop an SRS client for the Windows operating system using the SRSplus COM component. It is assumed at this point that you have read the Technical Overview and Developer's Guide documents. As you develop your SRS client application, 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. Rather, we will focus on issues and features specific to the SRS Component for Windows.

2. Development Tasks

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

- Download and uncompress the SRS Component Toolkit for Windows
- Register the SRS Component DLL
- Download, install, and configure GnuPG, the Gnu Privacy Guard
- Create system registry keys and values for your configuration information (e.g., your partner ID, your cryptographic key name, preferred log file name, etc.)
- Create your custom client application
- Test your application
- Run the certification tests.
- After passing certification, reconfigure settings 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 SRS Component Toolkit

If you have not done so already, download the SRS Component Toolkit for Windows from the Developer's Library section of our website:

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

Unzip the archive contents to a directory of your choosing and make note of it. You will be working from this directory later to register the COM component and to set configuration information.

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 Toolkit.
Toolkit/docs	Contains a copy of this document and the Developer's Guide.

Toolkit/keys	Contains a keyblock file called SRS.KEY containing the SRS public keys for importing into your GnuPG database with the "gpg --import SRS.KEY" command.										
Toolkit/com	Contains the SRS COM component, as well as a sample Visual Basic application suitable for running the certification test suite. <table border="1" data-bbox="537 363 1385 617"> <thead> <tr> <th colspan="2">Filelist</th> </tr> <tr> <th>Filename</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>SRSplus.dll</td> <td>The SRS COM component housing</td> </tr> <tr> <td>Certify.exe</td> <td>Visual Basic executable that performs the certification test suite. NOTE: Requires Visual Basic 6.0 Runtime.</td> </tr> <tr> <td>Config.reg</td> <td>Used to create a skeleton structure of registry keys with default settings.</td> </tr> </tbody> </table>	Filelist		Filename	Description	SRSplus.dll	The SRS COM component housing	Certify.exe	Visual Basic executable that performs the certification test suite. NOTE: Requires Visual Basic 6.0 Runtime.	Config.reg	Used to create a skeleton structure of registry keys with default settings.
Filelist											
Filename	Description										
SRSplus.dll	The SRS COM component housing										
Certify.exe	Visual Basic executable that performs the certification test suite. NOTE: Requires Visual Basic 6.0 Runtime.										
Config.reg	Used to create a skeleton structure of registry keys with default settings.										

3.2 Dependencies

The SRS Component for Window requires a few additional components that are not part of the Toolkit download. These dependencies are detailed in the sections below.

3.2.1 GnuPG

The GNU Privacy Guard is used by the SRSplus Shared Registry System to perform the creation and verification of digital signatures. Instructions for installing GnuPG for Windows are given in section 4 of this document. Instructions for configuring GnuPG are presented in section 5.

3.2.2 WinSock 2.0 & CryptoAPI

These components provide secure TCP/IP communication with the SRS servers. WinSock 2.0 implements the TCP/IP communication layer; CryptoAPI provides access to Secure Channel, Microsoft's security provider for TLS, SSL and PCT. Both are available for most versions of Windows.

3.2.3 Windows Scripting Runtime

The SRS Component uses the "Scripting.Dictionary" object provided by the Scripting Runtime component. Windows Script technologies are integral to many Internet applications on Windows. The Scripting Runtime is installed by many applications and is available from Microsoft at:

<http://www.microsoft.com/msdownload/vbscript/scripting.asp>.

3.2.4 Internet Explorer High Encryption Pack

The SRS servers require 128-bit encryption for secure communications. Install the High Encryption Pack if your system does not already have 128-bit encryption. You can determine your encryption level via the Help->About Internet Explorer menu item in Internet Explorer, which reports the current encryption level as "Cipher Strength." If the cipher strength is not 128-bits, click on the "Update Information" link and you will be redirected to a download site.

3.2.5 Windows Versions

The SRS Component will work only on Windows versions for which all of the above required components are available.

4. Installation

4.1 GnuPG 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. Now, create your keypair and import and sign the SRSplus server keys as described in section 5.3 of the Developer's Guide.

4.2 SRS Component Registration

The SRS Component, like all COM components, must be registered on the target machine before its services are made available. A single console command accomplishes this task. Open a console window and perform the command `"regsvr32 SRSplus.dll"` in the directory to which `SRSplus.dll` was decompressed. This will register the component in the registry and should cause a dialog box confirming the registration to appear.

5. Configuration

5.1 GnuPG Configuration

Refer to section 5.3 of the Developer's Guide for detailed instructions.

5.2 SRS Component Configuration

Before you can use the SRS Component in an application, you will need to set some configuration values in the registry. A `.reg` file is provided in the COM directory of the Toolkit distribution. This file is used to import a skeleton structure of all the necessary registry keys and settings. Simply double-click on the `CONFIG.REG` file to import this placeholder configuration information into the registry. Once these keys and settings have been imported, you will need to use the `REGEDIT` program to edit some settings specific to your configuration. These are detailed below.

5.2.1 Partner Configuration

There are three partner related pieces of information the SRS Component requires:

- 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.

Run `REGEDIT` and navigate to `HKEY_LOCAL_MACHINE\Software\SRSplus`:

- Enter the email name for your key pair as the value for `RegistrarEmail`
- Enter the password for your private key as the value for `Passphrase`
- Enter your partner ID as the value for `RegistrarID`

5.2.2 Server Configuration

The setting `TestMode` determines which server environment the SRS Component will use. When `TestMode` is set to a non-zero value, all traffic will be routed to the test SRS server. If zero, all traffic will be sent to the production SRS server. Section 7.3 below details the use of this setting as it relates to switching from the test environment to the production environment *after* passing the certification tests. For now, you should leave the default value of 1 in place.

5.2.1 Logging Configuration

The only other settings you may need to change are related to the built-in logging functions. The setting `EnableLogging` will, if set to a non-zero value, cause all outgoing and incoming raw command packet data to be collected and written to the file specified by the setting `Logfile`. The default `Logfile` value is "SRS.LOG". You should change it to suit your particular needs or preferences. While building and testing your client, you should leave `EnableLogging` equal to 1 (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 7.2.1 below.

6. ISrs Method Reference

Each SRS command has a corresponding method available through the `ISrs` interface. In addition, there are a few component-specific methods you should become familiar with. You will need to consult the SRS Command Reference in the Developer's Guide to determine legal input and output parameters for the SRS methods. In the Developer's Guide, all parameters are referred to as **strings** or lists of **name-value pairs**. These two types correspond directly to the types `BSTR` and `IDictionary`. When using Visual Basic or one of the scripting languages, the conversion to `BSTR` from strings is performed automatically via the use of `VARIANT`s.

Most examples in this section are written as ASP pages using VBScript, since it is expected that the SRS Component will most likely be used for creating web applications. For those interested in using the SRS Component in a C++ application, notes and sample code are provided in Appendix B.

6.1 Programming Notes

6.1.1 ISrs

The programmatic identifier for `ISrs` is `SRSplus.Srs`. This is the interface you will use to perform all SRS commands.

6.1.2 IDictionary

Objects of the type `IDictionary` are used to exchange data with various methods on the `ISrs` interface. It is implemented by the Windows Scripting Runtime and has the programmatic identifier `Scripting.Dictionary`. `IDictionary` objects are basically equivalent to the associative array (hash) type in Perl, or a string-to-string map class in C++.

6.1.3 Errors

The SRS Component supports the `IErrorInfo` interface. This is useful in scripting environments such as VBScript since the `Err` variable will automatically have a descriptive error string set in the `Description` property whenever an error occurs. See the sample ASP pages below for examples of using `Err.Description`.

6.1.4 Deallocation Responsibilities

Normal COM allocation and deallocation rules apply to the SRS Component. (This primarily affects C++ programmers)

6.2 Component Specific Methods

There are only three methods on the `ISrs` interface that do not correspond exactly to SRS commands. They are:

Startup
Description: Initializes object
Function Prototype: <code>HRESULT Startup(void);</code>
Notes: This function should be called exactly once at the beginning of your client code before calling any other method.

Shutdown
Description: Uninitializes the object and releases any internally allocated resources.
Function Prototype: <code>HRESULT Shutdown(void);</code>
Notes: This method should be called exactly once in your client code <i>after</i> you are finished using the object. Do not call any methods after calling this routine.

ConvertToDate
Description: Converts a UNIX/Linux epoch date (i.e., seconds since midnight 1/1/1970 UTC) and converts it to a VARIANT type DATE, corrected for the local time zone.
Function Prototype: <code>HRESULT ConvertToDate(long lEpochTime, //[in] DATE* pDate); //[out,retval]</code>
Notes: All date values are returned by the SRS servers in UNIX/Linux epoch time format. This method is provided as a convenience.

6.3 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.

AccountBalance

Description:

Returns balance information from your partner account.

Function Prototype:

```
HRESULT AccountBalance( BSTR bstrTLD,           //[in]
                       IDictionary** ppResult); //[out,retval]
```

Example ASP Page: Obtaining your account balance

```
<%@ Language=VBScript %>
<% Option Explicit
   Dim Hresult, Srs
   Dim Balance, Key
   Set Srs = Server.CreateObject("SRsplus.Srs")
   Hresult = Srs.Startup()
   If Hresult=0 Then
       Set Balance = Srs.AccountBalance("tv")
       Hresult = Srs.Shutdown()
   End If
%>
<HTML><BODY>
<P>Account Balance returned:</P>
<P>
   <TABLE border=1 cellPadding=1 cellSpacing=1 width="60%">
   <%For Each Key in Balance.Keys
       Response.Write("<TR><TD>" & Key & "</TD>")
       Response.Write("<TD>" & Balance.Item(Key) & "</TD></TR>")
   Next
   %>
   </TABLE>
</P>
</BODY></HTML>
```

DomainInfo

Description:

Returns availability information for a specific domain name.

Function Prototype:

```
HRESULT DomainInfo(BSTR bstrDomain, // [in]
                   BSTR bstrTLD,    // [in]
                   IDictionary** ppResponse); // [out,retval]
```

Example ASP Page: How to determine if a domain is available and, if so, what it costs

```
<%@ Language=VBScript %>
<% Dim Hresult, Srs, Results
   Dim strDomain, strStatus, strPrice
   Dim isAvailable
   Dim showResults
   showResults = False
   Set Srs = Server.CreateObject("SRsplus.Srs")
   Hresult = Srs.Startup()
   If Hresult=0 Then
       strDomain = Request.Form("domain")
       If Len(strDomain)>=1 Then
           showResults = True
           Set Results = Srs.DomainInfo( strDomain, "tv" )
           strStatus = Results.Item("DOMAIN STATUS")
```

```

        If StrComp( strStatus, "UNAVAILABLE", vbTextCompare) Then
            strPrice = Results.Item("PRICE")
            isAvailable = True
        Else
            isAvailable = False
        End If
    End If
    Hresult = Srs.Shutdown()
End If
%>
<HTML>
<BODY>
<%
If showResults Then
    If isAvailable Then
        Response.Write("<P>The domain " & strDomain & ".tv is available for $" & _
            strPrice & ".</P>")
    Else
        Response.Write("<P>The domain " & strDomain & ".tv is not available.</P>")
    End If
Else
%>
<FORM action="" method=POST id=form1 name=form1>
<P>Enter a domain to check:</P>
<P>www.<INPUT id=txt1 name=domain>.tv</P>
<P><INPUT id=submit1 name=command type=submit value="Check Now"></P>
</FORM>
<%End If%>
</BODY>
</HTML>

```

MultiDomainInfo

Description:

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

Function Prototype:

```

HRESULT MultiDomainInfo( IDictionary* pQuery, // [in]
                        IDictionary** ppResponse); // [out,retval]

```

Visual Basic Example:

(Prints results to the Immediate window)

```

Dim Hresult, Srs, Query, Results, Key
Set Srs = CreateObject("SRSplus.Srs")
Hresult = Srs.Startup()
If Hresult=0 Then
    Set Query = CreateObject("Scripting.Dictionary")
    Query.Add "DOMAIN 1", "firstdomain"
    Query.Add "TLD 1", "tv"
    Query.Add "DOMAIN 2", "seconddomain"
    Query.Add "TLD 2", "tv"
    Set Results = Srs.MultiDomainInfo( Query )
    For Each Key in Results.Keys
        Debug.Print Key & ": " & Results.Item( Key )
    Next
    Hresult = Srs.Shutdown()
End If

```

Whois

Description:

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

Function Prototype:

```
HRESULT Whois( BSTR bstrDomain, // [in]
               BSTR bstrTLD,   // [in]
               IDictionary** ppResponse); // [out,retval]
```

Example ASP Page: Displaying basic whois information

```
<%@ Language=VBScript %>
<%Option Explicit
    Dim Srs, Hresult
    Dim Results, isReserved
    Dim strDomain, strFname, strLname, strOrg, strNote
    Dim strTempDate, strRegDate, strExpDate
    Dim showResults
    showResults = False

    strDomain = Request.Form("domain")
    If Len(strDomain) <> 0 Then
        showResults = True
        Set Srs = Server.CreateObject("SRsplus.Srs")
        Hresult = Srs.Startup()
        If Hresult= 0 Then
            Set Results = Srs.Whois( strDomain, "tv" )
            If results.Exists("NOTE")Then
                isReserved = True
                strNote = "No Whois information is available for this domain."
            Else
                isReserved = False
                strFname = results.Item("FNAME RESPONSIBLE PERSON")
                strLname = results.Item("LNAME RESPONSIBLE PERSON")
                strOrg = results.Item("ORGANIZATION RESPONSIBLE PERSON")
                If Len(strOrg)=0 Then strOrg = "(None)"
                strTempDate = results.Item("REGISTRATION DATE")
                If Len(strTempDate)=0 Then
                    strRegDate="(Unspecified)"
                Else
                    strRegDate = Srs.ConvertToDate( strTempDate )
                End If
                strTempDate = results.Item("EXPIRATION DATE")
                If Len(strTempDate)=0 Then
                    strExpDate="(Unspecified)"
                Else
                    strExpDate = Srs.ConvertToDate( strTempDate )
                End If
            End If
            Hresult = Srs.Shutdown()
        End If
    End If

%>
<HTML>
<BODY>
<%If showResults=False Then %>
<FORM action="" method=post id=form1 name=form1>
<FONT size=4>Lookup Whois information for: </FONT>
<STRONG>www.<INPUT id=txt1 name=domain >.tv</STRONG>
<INPUT type="submit" value="Lookup" id=submit1 name=submit1>
</FORM><FONT size=4>
<%Else%>
<P>Whois information for "<%=strDomain%> .tv"</FONT></P>
<P>
    <%If isReserved Then%>
    <%=strNote%>
    <%Else%>
    <TABLE border=1 cellPadding=1 cellSpacing=0 width="450" style="WIDTH: 450px">
```

```

<TR>
  <TD>Registrant</TD>
  <TD><%=strFname%>&nbsp;&nbsp;&nbsp;<%=strLname%></TD></TR>
<TR>
  <TD>Organization</TD>
  <TD><%=strOrg%></TD></TR>
<TR>
  <TD>Date Registered</TD>
  <TD><%=strRegDate%></TD>
</TR>
<TR>
  <TD>Expires</TD>
  <TD><%=strExpDate%></TD>
</TR>
</TABLE>
<%
  End If
End If
%></P>
</BODY>
</HTML>

```

6.4 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.

CreateContact

Description:

Creates a new contact record and returns its unique ID.

Function Prototype:

```

HRESULT CreateContact( BSTR bstrTransID,    // [in]
                      IDictionary* pData, // [in]
                      IDictionary** ppResponse); // [out,retval]

```

Example ASP Page: Creating a new contact record

```

<%@ Language=VBScript %>
<%
  Option Explicit
  Dim Hresult, Srs
  Dim Info, Results
  Dim strErr, strTemp
  Dim Success, isRequiredMissing

  Success = False
  isRequiredMissing = False

  If StrComp(Request.Form("submit1"), "Create")= 0 Then
    Set Srs = Server.CreateObject("SRSplus.Srs")
    Hresult = Srs.Startup()
    If Hresult=0 Then
      Set Info = Server.CreateObject("Scripting.Dictionary")

      SetField Info, "fname" , "FNAME", True
      SetField Info, "lname" , "LNAME"      , True
      SetField Info, "org"   , "ORGANIZATION", False
      SetField Info, "email" , "EMAIL"      , True
      SetField Info, "phone" , "PHONE"      , True
      SetField Info, "addr1" , "ADDRESS1"   , True
      SetField Info, "addr2" , "ADDRESS2"   , False
      SetField Info, "city"  , "CITY"       , True
      SetField Info, "prov"  , "PROVINCE"   , True
      SetField Info, "postal", "POSTAL CODE", True
      SetField Info, "country", "COUNTRY"   , True

      If isRequiredMissing = True Then
        strErr = "Missing required field!"
      End If
    End If
  End If
%>

```

```

                Success = False
            Else
                On Error Resume Next
                Err.Clear
                Set Results = Srs.CreateContact( "0", Info )
                If Err <> 0 Then
                    strErr = Err.Description
                Else
                    If Results.Exists("CONTACTID") Then
                        Success = True
                    Else
                        strErr = "No contact ID was returned!"
                    End If
                End If
            End If
            Hresult = Srs.Shutdown()
        End If
    End If

    Sub SetField(ByRef rp, ByVal formID, ByVal key, ByVal required )
        Dim strTemp
        strTemp = Request.Form( formID )
        If Len(strTemp)=0 Then
            If required=True Then
                isRequiredMissing = True
            End If
        Else
            rp.Add key, strTemp
        End If
    End Sub

%>
<HTML>
<BODY>
<P>Create a new contact:</P>
<P>
<FORM action="" id=FORM1 method=post name=FORM1>
<TABLE border=1 width="400" bgColor=beige >

    <TR>
        <TD>
            <P align=center>Contact Information</P></TD>
        </TR>
    <TR>
        <TD>
            <TABLE width="100%" >
                <TR>
                    <TD><P align=right>First Name*</P></TD>
                    <TD><INPUT id=fname name=fname style="WIDTH: 220px"></TD></TR>
                <TR>
                    <TD><P align=right>Last Name*</P></TD>
                    <TD><INPUT id=lname name=lname style="WIDTH: 220px"></TD></TR>
                <TR>
                    <TD>
                        <P align=right>Organization</P></TD>
                    <TD><INPUT id=org name=org style="WIDTH: 220px"></TD></TR>
                <TR>
                    <TD>
                        <P align=right>Email Address*</P></TD>
                    <TD><INPUT id=email name=email style="WIDTH: 220px"></TD></TR>
                <TR>
                    <TD>
                        <P align=right>Phone*</P></TD>
                    <TD><INPUT id=phone name=phone
                        style="WIDTH: 220px"></FONT></TD></TR>
                <TR>
                    <TD>
                        <P align=right>Address1*</P></TD>
                    <TD><INPUT id=addr1 name=addr1 style="WIDTH: 220px"></TD></TR>
                <TR>
                    <TD>
                        <P align=right>Address2</P></TD>

```

```

        <TD><INPUT id=addr2 name=addr2 style="WIDTH: 220px"></TD></TR>
    <TR>
        <TD>
            <P align=right>City*</P></TD>
        <TD><INPUT id=city name=city style="WIDTH: 220px"></TD></TR>
    <TR>
        <TD>
            <P align=right>State(Province)*</P></TD>
        <TD><INPUT id=prov name=prov style="WIDTH: 220px"></TD></TR>
    <TR>
        <TD>
            <P align=right>Postal Code*</P> </TD>
        <TD><INPUT id=postal name=postal style="WIDTH: 220px"></TD></TR>
    <TR>
        <TD>
            <P align=right>Country(two letter code)*</P></TD>
        <TD><INPUT id=country name=country style="WIDTH: 220px"></TD></TR>
    </TABLE>
</TD>
</TR>
</TABLE></P>

<P><INPUT id=submit1 name=submit1 type=submit value="Create"></P></FORM>
<%If Success=True Then%>
    <P>Contact ID: <%=Results.Item("CONTACTID")%></P>
<%Else%>
<P><STRONG><%=strErr%></STRONG></P>
<%End If%>
</BODY>
</HTML>

```

EditContact

Description:

Modifies fields in an existing contact record.

Function Prototype:

```

HRESULT EditContact( BSTR bstrTransID,    // [in]
                    IDictionary* pData,  // [in]
                    IDictionary** ppResponse); // [out,retval]

```

Example ASP Page: Editing a new contact record

Same as for CreateContact with the following changes:

1. Use the method EditContact instead of CreateContact
2. Remove rows for editing the first and last name since they are not updateable
3. Specify the CONTACTID of the contact record to modify in the Info dictionary object (e.g., Info.Add "CONTACTID", "12345")

GetContactInfo

Description:

Retrieve all fields from an existing contact record.

Function Prototype:

```

HRESULT GetContactInfo( BSTR bstrContactID, // [in]
                       IDictionary** ppResponse); // [out,retval]

```

Example ASP Page: Examine a contact record

```

<%@ Language=VBScript %>
<%Dim Hresult, Srs, Results
    Dim strContactID
    Dim showResults
    Dim strErr
    strErr = ""
    showResults = False
    Set Srs = Server.CreateObject("SRSplus.Srs")

```

```

Hresult = Srs.Startup()
If Hresult=0 Then
    strContactID = Request.Form("contactid")
    If Len(strContactID)>=1 Then
        On Error Resume Next
        Set Results = Srs.GetContactInfo( strContactID )
        If Err<> 0 Then
            strErr = Err.description
        Else
            showResults = True
        End If
    End If
    Hresult = Srs.Shutdown()
End If
%>

<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
</HEAD>
<BODY>
<FORM action="./GetContactInfo.asp" method=POST id=form1 name=form1>
    <P>Enter ContactID:<INPUT id=contactid name=contactid>
    <INPUT id=submit1 name=submit1 type=submit value=Submit>
    </P>
</FORM>
<P>&nbsp;</P>
<%If showResults = True Then%>
    <P><FONT size=2><STRONG>RAW DATA:</STRONG></FONT> </P>
    <TABLE border=1 cellPadding=1 cellSpacing=0 width="450">
    <%
        Dim currKey
        For Each currKey In Results.Keys
            Response.Write "<TR><TD><FONT size=2>" & currKey & _
                "</FONT></TD><TD><FONT size=2>" & results.Item(currKey) & _
                "</TD></FONT></TR>"
        Next
    %>
</TABLE>
<%Else%>
<p><%=strErr%></p>
<%End If%>
</BODY>
</HTML>

```

6.5 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.

RegisterDomain

Description:

Register (buy) a domain.

Function Prototype:

```

HRESULT RegisterDomain( BSTR bstrTransID,    // [in]
                       IDictionary* pData,  // [in]
                       IDictionary** ppResponse); // [out,retval]

```

Example ASP Page: Register a domain

```

<%@ Language=VBScript %>
<%Option Explicit
    Dim Srs, Hresult
    Dim Params, Info, Results
    Dim Domain, ContactID, Years
    Dim MaxPrice
    Dim ErrString

```

```

MaxPrice = 50.00
ErrString = ""

If Request.Form("submit1") = "Register" Then
    Set Srs = Server.CreateObject("SRSplus.Srs")
    Hresult = Srs.Startup()
    If Hresult = 0 Then
        Domain = Request.Form("domain")
        ContactID = Request.Form("contactID")
        Years = Request.Form("years")
        '
        ' check domain availability and price
        '
        Set Info = Srs.DomainInfo( Domain, "tv" )
        If Info.Item("DOMAIN STATUS") <> "UNAVAILABLE" Then
            If Cdbl(Info.Item("PRICE")) <= Cdbl(MaxPrice) Then
                '
                'register and transfer to confirmation page if ok
                '
                Set Params = Server.CreateObject("Scripting.Dictionary")
                Params.Add "DOMAIN", Domain
                Params.Add "TLD", "tv"
                Params.Add "RESPONSIBLE PERSON", ContactID
                Params.Add "TERM YEARS", Years
                Params.Add "PRICE", Info.Item("PRICE")

                Set Results = Srs.RegisterDomain( "0", Params )

                If Results.Exists("REQUESTID") Then
                    Session("Confirm.RequestID") = Results.Item("REQUESTID")
                    Session("Confirm.Domain") = Domain
                    Response.Redirect( "Confirm.asp" )
                Else
                    Response.Redirect("SrsError.htm")
                End If
            Else
                '
                'too expensive
                '
                ErrString = "Price out of range (i.e., > $" & MaxPrice & ")."
            End If
        Else
            '
            'not available
            '
            ErrString = "The domain " & Domain & " is not available."
        End If
        Hresult = Srs.Shutdown()
    End If
End If

%>
<HTML>
<HEAD>
<BODY>
<P><FONT size=4><STRONG>Register Domain</STRONG></FONT></P>
<P>
<FORM action="" id=FORM1 method=post name=FORM1>
    DOMAIN:
    www.
    <INPUT id=domain name=domain > .tv
    <BR>
    CONTACT
    ID:
    <INPUT id=contactID name=contactID>
    (previously created)
    <BR>
    Registration length:
    <INPUT id=years name=years value="1" style="HEIGHT: 22px; WIDTH: 36px">
    (in years)
    <BR>

```

```

        <INPUT type="submit" value="Register" id =submit1 name=submit1 >
</FORM>
</P>
<%
If ErrString<>" " Then
    Response.Write("<P>ERROR: " & ErrString & "</P>" )
End If
%>
</BODY>
</HTML>

```

ChangeDomain

Description:

Modify contact and domain name server information for a domain.

Function Prototype:

```

HRESULT ChangeDomain( BSTR bstrTransID,    // [in]
                    IDictionary* pData,  // [in]
                    IDictionary** ppResponse); // [out,retval]

```

Example ASP Page: How to change DNS information for a domain

```

<%@ Language=VBScript %>
<% Option Explicit
Dim Hresult, Srs
Dim strDomain, strTemp, strErr
Dim Info, Results
Dim FailFlag, Success

Success = False
strDomain = Request.Form("domain")
If Len(strDomain)<>0 Then
    Set Srs = Server.CreateObject("SRSplus.Srs")
    Hresult = Srs.Startup()
    If Hresult=0 Then
        Set Info = Server.CreateObject("Scripting.Dictionary")
        Info.Add "DOMAIN", strDomain
        Info.Add "TLD","tv"
        '
        'set all the included fields
        '
        SetField Info, "dns1", "DNS SERVER NAME 1"
        SetField Info, "dns2", "DNS SERVER NAME 2"

        On Error Resume Next
        Set Results = Srs.ChangeDomain( "0", Info )
        If Err <> 0 Then
            strErr = Err.description
        Else
            If results.Exists("REQUESTID") Then
                Success = True
            Else
                strErr = "UNEXPECTED: Succeeded, but no REQUESTID returned!"
            End If
        End If
        Hresult = Srs.Shutdown()
    End If
Else
    strErr = "Please specify a domain."
End If

Sub SetField(ByRef rp, ByVal formID, ByVal key )
    Dim strTemp
    strTemp = Request.Form( formID )
    If Len(strTemp)<>0 Then
        rp.Add key, strTemp
    End If
End Sub

End Sub
%>

```

```

<HTML>
<BODY>

<P><FONT size=4><STRONG>Modify Domain Information</STRONG></P>
<P><FONT size=2>Note that any DNS changes entered here will override the
existing settings.</FONT></P>
<FORM action="" id=FORM1 method=post name=FORM1></FONT><P>
<P>
<P>
<TABLE border=1 cellPadding=2 cellSpacing=1 style="WIDTH: 400px" width=400 >

  <TR>
    <TD>
      <P align=right>Domain Name</P></TD>
      <TD><INPUT id=text1 name=domain
        style="HEIGHT: 22px; WIDTH: 200px"></TD></TR>
    <TR>
      <TD>
        <P align=right> DNS Server1</P></TD>
      <TD><INPUT id=text2 name=dns1 style="WIDTH: 200px"
        ></TD></TR>
    <TR>
      <TD>
        <P align=right> DNS Server2</P></TD>
      <TD><INPUT id=text2 name=dns2 style="WIDTH: 200px"
        ></TD></TR>

</TABLE></P>
<P><INPUT id=submit1 name=submit1 type=submit value=Submit></P>
</FORM>

<%If success=True Then%>
  <TABLE border=1 cellPadding=1 cellSpacing=0 width="450" style="WIDTH: 450px">
  <%
    Dim currKey
    For Each currKey In Results.Keys
      Response.Write "<TR><TD>" & currKey & "</TD><TD>" & _
        Results.Item(currKey) & "</TD></TR>"
    Next
  %>
</TABLE>
<%Else%>
<P><STRONG><%=strErr%></STRONG></P>
<%End If%>
</BODY>
</HTML>

```

ReleaseDomain

Description:

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

Function Prototype:

```

HRESULT ReleaseDomain( BSTR bstrTransID,    // [in]
                      IDictionary* pData,  // [in]
                      IDictionary** ppResponse); // [out,retval]

```

Example ASP Page: How to release (cancel) a domain

```

<%@ Language=VBScript %>
<%
  Option Explicit
  Dim Hresult, Srs
  Dim strDomain, strErr
  Dim Info, Results, Success
  Success = False

  strDomain = Request.Form("domain")
  If Len(strDomain)<>0 Then

```

```

Set Srs = Server.CreateObject("SRSplus.Srs")
Hresult = Srs.Startup()
If Hresult=0 Then
    Set Info = Server.CreateObject("Scripting.Dictionary")
    Info.Add "DOMAIN", strDomain
    Info.Add "TLD", "tv"
    On Error Resume Next
    Err.Clear
    Set Results = Srs.ReleaseDomain( "0", Info )
    If Err.Number <> 0 Then
        strErr = Err.Description
    Else
        If Results.Exists("REQUESTID") Then
            Success = True
        Else
            strErr = "Domain released but no REQUESTID returned."
        End If
    End If
    Hresult = Srs.Shutdown()
End If
End If
%>
<HTML>
<BODY>
<P>Release Domain</P>
<P>Enter domain to release:</FONT></P>
<FORM action="" id=FORM1 method=post name=FORM1>
<P><INPUT id=text1 name=domain>.TV</P>
<P><INPUT id=submit1 name=submit1 type=submit value="Release" style="LEFT: 246px; TOP:
103px"></P></FORM>
<%If Success=True Then%>
<P><STRONG>Released <%=strDomain%>.tv successfully.</STRONG> </P>
<%Else%>
<P><STRONG><%=strErr%></STRONG></P>
<%End If%>
</BODY>
</HTML>

```

RenewDomain

Description:

Renews a domain registration for a given number of years

Function Prototype:

```

HRESULT RenewDomain( BSTR bstrTransID,    // [in]
                    IDictionary* pData,  // [in]
                    IDictionary** ppResponse); // [out,retval]

```

Visual Basic Example:

(Prints results to the Immediate window)

```

Dim Hresult, Srs, Query, Results, Key
Dim strDomain
strDomain = "mikey"
Set Srs = CreateObject("SRSplus.Srs")
Hresult = Srs.Startup()
If Hresult = 0 Then
    Set Results = Srs.Whois(strDomain, "tv")
    If Results.Exists("PRICE") Then
        Set Query = CreateObject("Scripting.Dictionary")
        Query.Add "DOMAIN", strDomain
        Query.Add "TLD", "tv"
        Query.Add "PRICE", Results.Item("PRICE")
        Query.Add "TERM YEARS", "2"
        Set Results = Nothing
        On Error Resume Next
        Set Results = Srs.RenewDomain("0", Query)
        If Err.Number <> 0 Then

```

```

        Debug.Print Err.Description
    Else
        For Each Key In Results.Keys
            Debug.Print Key & ": " & Results.Item(Key)
        Next
    End If
Else
    Debug.Print "The domain " & strDomain & " did not return a PRICE."
End If
Hresult = Srs.Shutdown()
End If

```

6.6 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.

RegisterNameServer
<p>Description: Add a nameserver.</p>
<p>Function Prototype:</p> <pre> HRESULT RegisterNameServer(BSTR bstrTransID, // [in] IDictionary* pData, // [in] IDictionary** ppResponse); // [out,retval] </pre>

Example ASP Page: How to register a name server

```

<%@ Language=VBScript %>
<% Option Explicit
    Dim Hresult, Srs
    Dim strDomain
    Dim Info, Results, Success
    Dim strErr, strIP, strDate
    Success = False

    strDomain = Request.Form("domain")
    strIP = Request.Form("ipaddr")
    If Len(strDomain)<>0 Then
        Set Srs = Server.CreateObject("SRSplus.Srs")
        Hresult = Srs.Startup()
        If Hresult=0 Then
            Set Info = Server.CreateObject("Scripting.Dictionary")
            Info.Add "DNS SERVER NAME", strDomain
            If Len(strIP) <> 0 Then
                Info.Add "DNS SERVER IP", strIP
            End If
            On Error Resume Next
            Err.Clear
            Set Results = GlobalSrs.RegisterNameServer( "0", Info )
            If Err.Number <> 0 Then
                strErr = Err.description
            Else
                If Results.Count <>0 Then
                    Success = True
                Else
                    strErr = "Successful, but no date returned."
                End If
            End If
            Hresult = Srs.Shutdown()
        End If
    End If

%>
<HTML>
<BODY>
<P>Register Nameserver</P>
<P></P>
<FORM action="" id=FORM1 method=post name=FORM1>

```

```

<P>Enter nameserver:<INPUT id=text1 name=domain> </P>
<P>Enter IP address:<INPUT id=text2 name=ipaddr style="LEFT: 287px; TOP: 182px"> </P>
<P><INPUT id=submit1 name=cmd type=submit value=Register></P>
<P>
</FORM>
<%If success=True Then%>
  <P>RESULTS:</P>
  <TABLE border=1 cellPadding=1 cellSpacing=0 width="450" >
  <%
    Dim currKey
    For Each currKey In results.Keys
      Response.Write "<TR><TD>" & currKey & "</TD><TD>" & results.Item(currKey) &
    "</TD></TR>"
    Next
  %>
</TABLE>
<%Else%>
<P><STRONG><%=strErr%></STRONG></P>
<%End If%>
</BODY>
</HTML>

```

ReleaseNameServer

Description:

Remove a previously registered nameserver.

Function Prototype:

```

HRESULT ReleaseNameServer( BSTR bstrTransID, // [in]
                           IDictionary* pData, // [in]
                           IDictionary** ppResponse); // [out,retval]

```

Visual Basic Example:

(Prints results to the Immediate window)

```

Dim Hresult, Srs, Query, Results, Key
Set Srs = CreateObject("SRSplus.Srs")
Hresult = Srs.Startup()
If Hresult=0 Then
  Set Query = CreateObject("Scripting.Dictionary")
  Query.Add "DNS SERVER NAME", "nsl.release-me.tv"
  Set Results = Srs.ReleaseNameServer( "0", Query )
  For Each Key in Results.Keys
    Debug.Print Key & ": " & Results.Item( Key )
  Next
  Hresult = Srs.Shutdown()
End If

```

NameServerInfo

Description:

Obtain information about a previously registered nameserver.

Function Prototype:

```

HRESULT NameServerInfo( BSTR bstrTransID, // [in]
                       IDictionary* pData, // [in]
                       IDictionary** ppResponse); //[out,retval]

```

Visual Basic Example:

(Prints results to the Immediate window)

```

Dim Hresult, Srs, Query, Results, Key
Set Srs = CreateObject("SRSplus.Srs")
Hresult = Srs.Startup()
If Hresult=0 Then
  Set Query = CreateObject("Scripting.Dictionary")
  Query.Add "DNS SERVER NAME", "nsl.somedomain.tv"
  Set Results = Srs.NameServerInfo( "0", Query )

```

```

    For Each Key in Results.Keys
        Debug.Print Key & ": " & Results.Item( Key )
    Next
    Hresult = Srs.Shutdown()
End If

```

6.7 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.

RequestTransfer

Description:

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

Function Prototype:

```

HRESULT RequestTransfer( BSTR bstrTransID,    // [in]
                        IDictionary* pData,  // [in]
                        IDictionary** ppResponse); // [out,retval]

```

Visual Basic Example:

(Prints results to the Immediate window)

```

Dim Hresult, Srs, Query, Results, Key, TransInfo
Set Srs = CreateObject("SRSplus.Srs")
Hresult = Srs.Startup()
If Hresult=0 Then
    Set TransInfo = CreateObject("Scripting.Dictionary");
    TransInfo.Add "DOMAIN", "xfer0000"
    TransInfo.Add "TLD", "net"
    TransInfo.Add "CURRENT ADMIN EMAIL", "fake_person@somedomain.com"
    Set Results = Srs.RequestTransfer("0", TransInfo )
    For Each Key in Results.Keys
        Debug.Print Key & ": " & Results.Item( Key )
    Next
    Hresult = Srs.Shutdown()
End If

```

OutboundTransferResponse

Description:

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

Function Prototype:

```

HRESULT OutboundTransferResponse([in]BSTR bstrTransID,
                                [in]BSTR bstrDomain,
                                [in]BSTR bstrTLD,
                                [in]BSTR bstrResponseString,
                                [out,retval]IDictionary** ppResponse);

```

Visual Basic Example:

(Prints results to the Immediate window)

```

Dim Hresult, Srs, Query, Results, Key
Set Srs = CreateObject("SRSplus.Srs")
Hresult = Srs.Startup()
If Hresult=0 Then
    Set Results = Srs.OutboundTransferResponse( "0", "testdomain", "org", "ACCEPT" )
    For Each Key in Results.Keys
        Debug.Print Key & ": " & Results.Item( Key )
    Next
    Hresult = Srs.Shutdown()
End If

```

ViewPendingTransfers

Description:

Returns information about pending transfers, either INBOUND or OUTBOUND.

Function Prototype:

```
HRESULT ViewPendingTransfers( BSTR bstrTransID,    // [in]
                             IDictionary* pData,  // [in]
                             IDictionary** ppResponse); // [out,retval]
```

Example ASP Page: View Pending Inbound and Outbound Transfers

```
<%@ Language=VBScript %>
<%
    Option Explicit
    Dim Hresult, Srs
    Dim Inbound, Outbound, Key
    Set Srs = Server.CreateObject("SRsplus.Srs")
    Hresult = Srs.Startup()
    If Hresult=0 Then
        Set Inbound = Srs.ViewPendingTransfers("0", "INBOUND" )
        Set Outbound = Srs.ViewPendingTransfers("0", "OUTBOUND" )
        Hresult = Srs.Shutdown()
    End If
%>
<HTML><BODY>
<P>Pending INBOUND Transfers:</P>
<P>
    <TABLE border=1 cellPadding=1 cellSpacing=1 width="60%">
    <%For Each Key in Inbound.Keys
        Response.Write("<TR><TD>" & Key & "</TD>")
        Response.Write("<TD>" & Inbound.Item(Key) & "</TD></TR>")
    Next
    %>
    </TABLE>
</P>
<P>Pending OUTBOUND Transfers:</P>
<P>
    <TABLE border=1 cellPadding=1 cellSpacing=1 width="60%">
    <%For Each Key in Outbound.Keys
        Response.Write("<TR><TD>" & Key & "</TD>")
        Response.Write("<TD>" & Outbound.Item(Key) & "</TD></TR>")
    Next
    %>
    </TABLE>
</P>
</BODY></HTML>
```

7. 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 separate and isolated. Other differences are explored next.

7.1 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 may 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.

7.2 Running the Certification Tests

A sample Visual Basic application called `Certify.exe` is included in the Toolkit. This application 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. (Refer to section 7 of the Developer's Guide for detailed information about the certification test criteria).

NOTE: `Certify.exe` requires the Visual Basic 6.0 Runtime. If not already installed on your machine, the VB 6.0 Runtime is freely available from Microsoft and other various download sites. See the Development Resources section of our website at http://www.srsplus.com/en/srsplus/partners_dev_library.shtml for links.

7.2.1 Logging Client Activity

You will need to enable the SRS Component'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 setting the value of `EnableLogging` in the configuration settings to a non-zero value. Also, specify a filename for the log file in the `Logfile` setting (we suggest `SRS.LOG`). Be sure to specify a full path for the filename and ensure that the process running the component has read/write permissions for the directory.

IMPORTANT: Configuration information is only read by the component when the `Startup` method is invoked. This is important to remember if, for example, you create an Application scope SRS object in IIS. In such a case, you would need to restart the IIS service in order to reinitialize the SRS Component to use the new logfile setting (or any other).

7.2.2 Running the Test Application

The sample `Certify.exe` file will perform all the SRS commands required for the certification test. The steps required to run the test are:

- Ensure that the settings `TestMode` and `EnableLogging` are set to 1
- Configure the `Logfile` setting

- Rename or delete any existing log file
- Run the `Certify.exe` application

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 include 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 configuration settings to use the live server and begin registering domains immediately.

7.3 Switching Environments

As discussed in section 5.2.2, the server environment is determined by the value of the setting `TestMode`. When `TestMode` is non-zero, you will be working against the test SRS server. When it is equal to zero, 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: Remember that the SRS Component only recognizes changes in configuration when the `Startup` method is invoked. You must reinitialize the component after configuration changes. This may affect your application if, for example, you keep a global instance of the SRS Component.

8. The Entire Integration Pathway

For your easy reference, the complete process from download to certification is presented in figure 8.1. You may want to make a separate copy of this flowchart and use it to keep track of your progress.

9. Next Steps

You should now feel prepared to develop your custom SRS client application. 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.

Appendix A: ASP Tips

SRS Component as an Application Object

You will improve performance of your web application as well as reduce the amount of scripting necessary to use the SRS Component if you create a global instance of the component. This can be done by instantiating the SRS Component with an <OBJECT> tag in the Global.asa file and calling the Startup and Shutdown methods during the Application_OnStart and Application_OnEnd events, respectfully. Here is a sample Global.asa file:

```
<OBJECT ID="GlobalSrs" SCOPE=APPLICATION RUNAT=SERVER
  PROGID="SRSplus.Srs">
</OBJECT>

  Sub Application_OnStart
    'start up global srs object
    Dim retval
    On Error Resume Next
    Err.Clear
    retval = GlobalSrs.Startup()
  End Sub

  Sub Application_OnEnd
    'shutdown the srs object
    Dim retval
    retval = GlobalSrs.Shutdown()
  End Sub
```

The SRS Component can now be referred to throughout the web application as GlobalSrs without ever calling the Startup or Shutdown methods. For example, the AccountBalance example can now be written like this:

```
<%@ Language=VBScript %>
<%Dim Balance
  Dim StrErr
  Dim Success
  Success = True
  On Error Resume Next
  Err.Clear
  Set Balance = GlobalSrs.AccountBalance( "tv" )
  If Err.number <> 0 Then
    StrErr = Err.description
    Success = False
  End If
%>
<P>Account Balance Information</P>
<P>
<%
  Dim Key
  If Success = True Then
    Response.Write("<TABLE border=1 cellPadding=1 cellSpacing=1 width=""40%"">")
    For Each Key in Balance.Keys
      Response.Write("<TR><TD>" & Key & "</TD><TD>" & Balance.Item(Key) & "</TD></TR>")
    Next
    Response.Write("</TABLE>")
  Else
    Response.Write("Error: " & StrErr )
  End If
%>
</P>
</BODY>
</HTML>
```

IMPORTANT: Remember that the SRS Component only recognizes changes in configuration when the `Startup` method is invoked. You must reinitialize the component after configuration changes. In the case of using the SRS Component as a global Application object, you will need to either restart IIS or, in the case where you have set your Application Protection to High (Isolated), unload the web application before configuration changes will be recognized.

Directory and User Permissions Issues

In order for the SRS Component to work correctly, you must take the following into account when configuring your web application in IIS:

1. The directory that contains `SRSplus.dll` must have execute permission for the user the web application is running as.
2. If you enable logging, the directory you specify for the log file must have read/write permission for the user the web application is running as.

The user your web application is running as will depend on how you have configured the web application. For example, if you have set High (Isolated) Application Protection, then the process running the SRS Component will be `IWAM_MACHINENAME`.

Appendix B: C++ Tips

Using IDictionary in C++

The `IDictionary` interface is implemented by the file `scrrun.dll`. Use the `#import` directive to convert type library information into C++ classes. Then create objects and invoke methods as you would any other COM object. For example:

```
//  
//Import the type library  
//  
#import "scrrun.dll" named_guids raw_interfaces_only raw_native_types  
  
//  
// Create an IDictionary object  
//  
IDictionary* pDictionary = NULL;  
  
HRESULT hr = ::CoCreateInstance  
(  
    Scripting::CLSID_Dictionary,  
    NULL,  
    CLSCTX_ALL,  
    IID_IDictionary,  
    (void **)& pDictionary  
);  
  
if( FAILED(hr) )  
{  
    return hr;  
}  
  
//  
// How to add name-value pairs  
//  
CComVariant vtKey = "KEYNAME";  
CComVariant vtValue= "sample value";  
CComVariant vtTemp;  
  
hr = pDictionary->Add( &vtKey, &vtValue );  
if( FAILED(hr) )  
{  
    pDictionary->Release();  
    return hr;  
}
```

```

//
// How to retrieve a value for a key
//
hr = pDictionary->get_Item( &vtKey, &vtTemp );
if( SUCCEEDED(hr) )
{
    // the value is returned in the VARIANT
    BSTR bstrValue = vtTemp.bstrVal;
}

pDictionary->Release();

```

Using ISrs in C++

The ISrs interface is implemented by the file SRSplus.dll. Use the #import directive to convert type library information into C++ classes. Then create objects and invoke methods as you would any other COM object. For example:

```

//
// Import the type library
//
// (you will need to add the path to SRSplus.dll)
#import "SRSplus.dll" named_guids no_namespace raw_interfaces_only raw_native_types
//
// Create an ISrs object
//
ISrs* pSrs = NULL;

HRESULT hr = ::CoCreateInstance
(
    CLSID_Srs,
    NULL,
    CLSCTX_ALL,
    IID_ISrs,
    (void **)& pSrs
);

if( FAILED(hr) )
{
    return hr;
}

//
// Use the object
//
hr = pSrs->Startup();
if( SUCCEEDED(hr) )
{
    //
    // Implementation of application goes here
    //
    pSrs->Shutdown();
}

pSrs->Release();

```

Figure 8.1: Pathway from Signup to Certification

