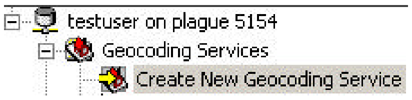
Programmatically geocoding with Locators and ArcSDE.

DRAFT #5   6/25/2001

(I use Layer in place of FeatureClass a lot. And SDE in place of ArcSDE)

What is a Locator?

A Locator (or Geocoding Service as they are known in ArcGIS 8.1) is primarily metadata; a list of parameters describing a reference layer and its columns which can be used for address matching (i.e. a STREETS line layer having attributes like TO and FROM, LEFT and RIGHT house numbers, etc.) You create Locators through ArcCatalog under a Database Connection with this:

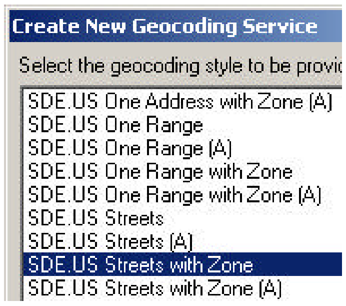


testuser on plague 5154
Geocoding Services
Create New Geocoding Service

When SDE is installed, a number of templates (or address styles) are created and stored in the SDE.LOCATORS table. You can get a list of Locators in SDE using sdelocator –o list. There is a Locator type flag which has the following values:

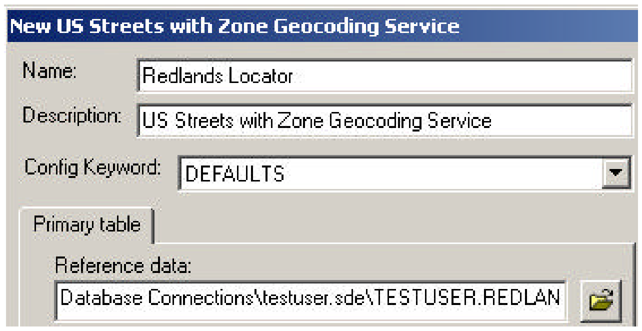
t (0) = template	An address style for geocoding stored in the database (similar to a class).
v (1) = verified	A working locator which can be used for geocoding (similar to an instance of a class).
d (2) = defined	A locator attached to a dataset, usually for purposes of re-matching.

You get a popup of different templates you can use. A typical template to use is “US Streets with Zone”.



Create New Geocoding Service
Select the geocoding style to be provic
SDE.US One Address with Zone (A)
SDE.US One Range
SDE.US One Range (A)
SDE.US One Range with Zone
SDE.US One Range with Zone (A)
SDE.US Streets
SDE.US Streets (A)
SDE.US Streets with Zone
SDE.US Streets with Zone (A)

Then you can give it a name, and navigate to your reference data (the SDE Layer for your Streets data).



New US Streets with Zone Geocoding Service
Name: Redlands Locator
Description: US Streets with Zone Geocoding Service
Config Keyword: DEFAULTS
Primary table
Reference data: Database Connections\testuser.sde\TESTUSER.REDLAN

Click OK and accept all the defaults. Et, Voila! You have a Locator. This is persisted in an RDBMS as a record in the SDE.LOCATORS table and a list of about 160 PROPERTY/VALUE's in the SDE.METADATA table. You can also persist them locally on a PC in a .LOC file.

What is an SSA?

An SSA (Server Side Application) is like a portal allowing developers to extend the SDE C API's capabilities. An SSA could be considered a service added to the SDE server. It is basically a shared object (lib*.so) library on Unix, or a DLL on WinNT, that can be invoked to run some functions on the server, communicating to the RDBMS if necessary, and passing back some result to the user. SSA's are stored in a directory called "ssa" under the SDEHOME directory.

From the Java doc: "An SSA is an SDE application that runs in the SDE server's process space. Server Side Applications are similar in concept to a DBMS stored procedure, which is a SQL application running in the DBMS kernel. A typical SeSSA operation sequence is: 1) Create an SeSSA object with appropriate parameters 2) Execute 3) If SSA is a query type application - fetch a single SeRow from the SeSSA - Retrieve column values and feature geometry from the fetched SeRow. 4) Close the SeSSA at the end of a series of fetches"

Calling an SSA.

An SSA is called using the SE_stream_invoke_ssa() function in the SDE C API, or by instantiating the SeSSA class in the SDE Java API. You specify which SSA you want to call, for example "locssa", and pass in input parameters using two arrays, one array of input descriptions (size and type), and a second array of the input parameter values.

Here is a comparison of the two (taken from the 8.1 SDEHelp.chm):

SE_stream_invoke_ssa (C API)

```
LONG SE_stream_invoke_ssa
        (SE_STREAM stream,
        BOOL query_stream,
        SHORT num_inputs,
        SE_INPUT_DESC *input_desc,
        void **data,
        CHAR *ssaName);
```

Parameters

stream	The stream handle.
query_stream	set to TRUE if SSA fetches data.
num_inputs	The number of input parameters the client will send to the SSA.
input_desc	The description of the input data in a SE_INPUT_DESC structure.
data	The pointer to an array of input data.
ssaName	The name of the server-side application.

Description

SE_stream_invoke_ssa configures a server-side application (SSA). After this function is called, SE_stream_execute , SE_stream_fetch, and other API functions will use the server-side application versions.

SeSSA (Java API)

```
public SeSSA(SeConnection conn,
          boolean queryStream,
          short numInputs,
          SeSSA.SeInputDesc[ ] inDesc,
          java.util.Vector data,
          java.lang.String ssaName)
     throws SeException
```

Parameters:

conn - SeConnection handle.

queryStream - set to TRUE if SSA fetches data.

numInputs - the number of input parameters the client will send to the SSA.

inDesc - the description of the input data in a SE_INPUT_DESC structure.

data - vector containing an array of data.

ssaName - the server-side apllication library name.

Creates an SeSSA object that initializes a server-side application inside the server process.

The LOCATOR SSA

The Geocoding team built an SSA which is now included with SDE at 8.1, called the LOCATOR SSA. The LOCATOR SSA has 5 subfunctions. These are wrapped up by ArcObjects and called mostly by ArcCatalog and ArcMap, and the command-line sdelocator utility. The main one we're interested in is probably the first, the Single Locate. Here is how they are defined:

```
#define LOCSSA_SINGLE_LOCATE  1
#define LOCSSA_LOCATE_TABLE   2
#define LOCSSA_GET_INPUTS     3
#define LOCSSA_SHAPE_ATTRS    4
#define LOCSSA_LOCATOR_MGR    5
```

Each of the subfunctions takes an array of inputs: an array of SE_INPUT_DESC (C) or SeInputDesc (Java) which specify type and size, and an array of void* (C) or a java.util.Vector for the actual input values. The number of inputs varies depending on the subfunction, but each begins with the subfunction number (integer), followed by the Locator in string format.

1.) Single Locate LOCSSA_SINGLE_LOCATE

A single Address match

The 3rd and 4th input parameters of LOCSSA_SINGLE_LOCATE are address line 1 and address line 2 (i.e. Address and Zone for "US Streets with Zone") but there can be additional address lines depending on the Locator. The full list of required inputs can be obtained using the LOCSSA_GET_INPUTS subfunction.

2.) Locate Table LOCSSA_LOCATE_TABLE

Batch geocoding of an entire table residing in the DBMS

The 3rd input parameter of LOCSSA_LOCATE_TABLE is the table name to be batch processed, and 4th input parameter is an optional where-clause.

3.) Get Inputs LOCSSA_GET_INPUTS

Returns inputs required for Locator. For example, the inputs required for a locator built using the "US Streets with Zone" template are 2 strings, STREET (an address with number, prefix, streetname, suffix, etc.) and ZONE (or ZIP code). So given this, you could probably skip calling this 99% of the time, as you know what inputs are required. But if you weren't sure, this would return them.

LOCSSA_GET_INPUTS takes no additional input parameters.

4.) Get Shape Attributes LOCSSA_SHAPE_ATTRS

Returns a single shape from the layer the locator's based on, for the purpose of getting its SE_COORDREF, etc.

LOCSSA_SHAPE_ATTRS takes no additional input parameters.

5.) Locator Manager LOCSSA_LOCATOR_MGR

This function is called to create, delete or refresh locators. It manages a locator's soundex table(s) and its indexes.

The 3rd parameter input of LOCSSA_LOCATOR_MGR is a sub-subfunction number: 0 = create locator, 1 = delete locator, and 2 = refresh locator. The 4th parameter is the Locator's name.

Single Locate in more detail

Here's where it gets more complicated. The LOCSSA_SINGLE_LOCATE function for a "US Streets with Zone" locator takes 4 inputs.

Inputs are passed in using an array SE_INPUT_DESC structures (which just say type and size), followed by an array of void pointers to the actual data for each input.

1. The first parameter is an integer representing the subfunction to be called, so in the case of LOCSSA_SINGLE_LOCATE, this would be 1.

2. The second parameter is the Locator as a string. Basically the entire Locator (approximately 150 records out the METADATA table) needs to get passed in as a single string. There is a function to get the number of properties, an array of the properties, and an array of the property values. This has to be called and each property appended to a string using the following format:

   PROPERTY=(UPPER-CASE 2-DIGIT HEX LENGTH OF PROP_VAL)"PROP_VAL";

   Example:

   RD.Val.Table1=12"SBCTRANS.STREETS53";

   The property is not quoted, followed by an equals sign, and 12, the hex representation of 18, the length of the prop_value, followed by the prop_value in double quotes and then a semi-colon.

3. The third parameter is STREET (an address with number, prefix, streetname, suffix, etc.).

4. The fourth parameter is ZONE (or zip code).

Once all this is set up, you call SE_stream_invoke_ssa:

SE_stream_invoke_ssa(stream, TRUE, num_inputs, input_desc, data, "locssa");

This will set the stream up with the results. The results are retrieved from the server with SE_stream_fetch(). It is sort of like retrieving columns from a table. The code is the same. You can describe the columns being returned before binding them or calling the SE_stream_get_* family of functions if you like. Or you can hardcode these as they rarely change.

The LOCSSA_SINGLE_LOCATE function by default returns

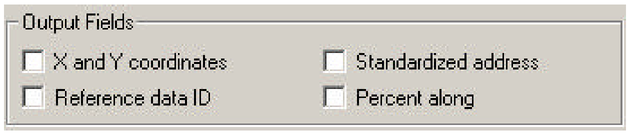
Shape	SE_SHAPE_TYPE
Status	SE_STRING_TYPE
Score	SE_SMALLINT_TYPE
Side	SE_STRING_TYPE

Status is one of:

M = Matched
U = Unmatched (not successfully matched)
T = Tie (two candidates that were valid matches had the
same score)
I = rematched Interactively (not yet implemented... right
now the interactive matcher sets the status to M)

Additional parameters that can be returned from LOCSSA_SINGLE_LOCATE

You can set any of these additional parameters on the locator and get additional props back from the address_match stream, in addition to Shape, Status, Score, Side



Output Fields

☐ X and Y coordinates	☐ Standardized address
☐ Reference data ID	☐ Percent along

In the bottom-right corner of the "New US Streets with Zone Geocoding Service" dialog, there were some optional Output Fields. You can go back to ArcCatalog and toggle these on if you like, or you can modify the properties of the Locator at run-time by setting these properties before invoking the SSA.

Some of the properties can be found in the METADATA and set to False by default, while a few of them are extra and not in the METADATA at all (such as Interpolate and CandidateMode).

Here are the property names, and the extra return columns they will add to the result, if set to True:

1. WriteXYCoordFields

X	SE_DOUBLE_TYPE
Y	SE_DOUBLE_TYPE

   Example:

X	: "-122.176354"
Y	: "48.158837"

2. WriteStandardizedAddressField

Stan_addr	SE_STRING_TYPE

   Example:

Stan_addr	: 549 | W | | AUBURN | WAY | N | 03180

3. WritePercentAlongField

Pct_along	SE_DOUBLE_TYPE

   Example:

Pct_along	: "0.000000"

4. WriteReferenceIDField (gives you the SE_ROW_ID [usually ObjectID] of the Street feature the address falls on)

Ref_ID	SE_INTEGER_TYPE

   Example:

Ref_ID	: "8555"

5. Interpolate

   (not sure what this does. It is supposed to be set to TRUE now?)

6. CandidateMode (returns multiple candidates for a single locate; fetch() gets called until stream is exhausted)

   You get all these, but not Status (or XYCoordFields, if set):

LeftFrom	SE_STRING_TYPE
LeftTo	SE_STRING_TYPE
RightFrom	SE_STRING_TYPE
RightTo	SE_STRING_TYPE
PreDir	SE_STRING_TYPE
PreType	SE_STRING_TYPE
StreetName	SE_STRING_TYPE
StreetType	SE_STRING_TYPE
SufDir	SE_STRING_TYPE
LeftZone	SE_STRING_TYPE
RightZone	SE_STRING_TYPE
PreDir1	SE_STRING_TYPE
PreType1	SE_STRING_TYPE
StreetName1	SE_STRING_TYPE
Type1	SE_STRING_TYPE
SufDir1	SE_STRING_TYPE
LeftZone1	SE_STRING_TYPE
RightZone1	SE_STRING_TYPE
PreDir2	SE_STRING_TYPE
PreType2	SE_STRING_TYPE
StreetName2	SE_STRING_TYPE
Type2	SE_STRING_TYPE
SufDir2	SE_STRING_TYPE
LeftZone2	SE_STRING_TYPE
RightZone2	SE_STRING_TYPE
Addr_type	SE_STRING_TYPE

   Example:

```
Shape          : ""
Score          : "18"
Side           : " "
LeftFrom       : "17401"
LeftTo         : "17499"
RightFrom      : "17400"
RightTo        : "17498"
PreDir         : ""
PreType        : ""
StreetName     : "117TH"
StreetType     : "PL"
SufDir         : "NE"
LeftZone       : ""
RightZone      : ""
PreDir1        : ""
PreType1       : ""
StreetName1    : ""
Type1          : ""
SufDir1        : ""
LeftZone1      : ""
RightZone1     : ""
PreDir2        : ""
PreType2       : ""
StreetName2    : ""
Type2          : ""
SufDir2        : ""
LeftZone2      : ""
RightZone2     : ""
Addr_type      : "A"
```

SAMPLE FOR GEOCODING USING THE JAVA API.

(Note: the JAVA API that shipped with SDE 8.1 Final had a bug where the SeSSA.fetch() was returning -10 "NETWORK I/O OPERATION FAILED". You'll need a jar that fixes this before this sample will work).

```
package sde_geocoding_example;

import com.esri.sde.sdk.client.*;

public class geocode {

  public static final int LOCSSA_SINGLE_LOCATE = 1;
  public static final int LOCSSA_LOCATE_TABLE  = 2;
  public static final int LOCSSA_GET_INPUTS    = 3;
  public static final int LOCSSA_SHAPE_ATTRS   = 4;
  public static final int LOCSSA_LOCATOR_MGR   = 5;

  public static final int TYPE_INTEGER = 2;
  public static final int TYPE_STRING = 5;

  public String locator_as_string(SeLocator locator) {

    String locator_as_string = "";

    try {

      String paramvalpair;
      String hexlen;

      String[] Properties = locator.getProperties();
      String[] Values = locator.getValues();

      for(int n = 0; n < locator.getNumProperties(); n++){
        hexlen = Integer.toHexString( Values[n].length() ).toUpperCase();
        if( hexlen.length() == 1 ) hexlen = "0" + hexlen;

        paramvalpair = Properties[n] + "=" + hexlen + "\"" + Values[n] + "\";";
        locator_as_string = locator_as_string + paramvalpair;
      }
    }
    catch(SeException e) {
      System.out.println(e);
    }
    return locator_as_string;
  }


  public void locator_set_property_value(SeLocator locator, String property, String prop_value) {

    try {

      String[] Properties = locator.getProperties();
      String[] Values = locator.getValues();
      int NumProperties = locator.getNumProperties();

      for(int n = 0; n < NumProperties; n++)
        if(Properties[n].equalsIgnoreCase(property))
          Values[n] = prop_value;

      locator.setPropertiesValues(NumProperties, Properties, Values);
    }
    catch(SeException e) {
      System.out.println(e);
    }

  }
```

```
public geocode() {
  System.out.println("Connecting to SDE...");

  // Connect to SDE
  try {
    SeConnection conn = new SeConnection("plague", 5154, "", "sbctrans", "sbctrans");

    boolean queryStream = true;
    short numInputs = 4;
    SeSSA.SeInputDesc[] inDesc = new SeSSA.SeInputDesc[numInputs];
    java.util.Vector data = new java.util.Vector(numInputs);
    String ssaName = "locssa";

    String locName = "Seattle May 4 2001";
    SeLocator locator = new SeLocator(conn, locName);

    locator_set_property_value(locator, "WriteXYCoordFields", "True");
    locator_set_property_value(locator, "WriteReferenceIDField", "True");
    locator_set_property_value(locator, "WriteStandardizedAddressField", "True");

    Integer ssa_function = new Integer(LOCSSA_SINGLE_LOCATE);
    String locator_as_string = locator_as_string(locator);
    String street_string = "348 E Pike St";
    String zone_string = "98122";

    inDesc[0] = new SeSSA.SeInputDesc();
    inDesc[0].setType( TYPE_INTEGER );
    inDesc[0].setSize( 0 );
    data.add( ssa_function );

    inDesc[1] = new SeSSA.SeInputDesc();
    inDesc[1].setType( TYPE_STRING );
    inDesc[1].setSize( locator_as_string.length() );
    data.add( locator_as_string );

    inDesc[2] = new SeSSA.SeInputDesc();
    inDesc[2].setType( TYPE_STRING );
    inDesc[2].setSize( street_string.length() );
    data.add(street_string);

    inDesc[3] = new SeSSA.SeInputDesc();
    inDesc[3].setType( TYPE_STRING );
    inDesc[3].setSize( zone_string.length() );
    data.add(zone_string);

    SeSSA ssa = new SeSSA(conn, queryStream, numInputs, inDesc, data, ssaName);

    ssa.execute();

    SeRow row = ssa.fetch();

    SeShape shape = row.getShape(0);
    String Status = row.getString(1);
    Short Score = row.getShort(2);
    String Side = row.getString(3);
    Double X = row.getDouble(4);
    Double Y = row.getDouble(5);
    String Stan_addr = row.getString(6);
    Integer Ref_ID = row.getInteger(7);

    System.out.println("Status:     " + Status );
    System.out.println("Score:      " + Score );
    System.out.println("Side:       " + Side );
    System.out.println("X:          " + X );
    System.out.println("Y:          " + Y );
    System.out.println("Stan_addr:  " + Stan_addr );
    System.out.println("Ref_ID:     " + Ref_ID );

    // Close the connection
    conn.close();
    System.out.println("Connection closed.");
```

```
      }
      catch(SeException e) {
        System.out.println(e);
      }
    }

    public static void main(String[] args) {
      geocode geocode1 = new geocode();
    }
  }
```

SAMPLE FOR GEOCODING USING THE C API.

```
#include <stdlib.h>
#include <string.h>
#include <sdetype.h>
#include <sdeerno.h>


#ifndef WIN32
#define stricmp strcasecmp
#endif

/* ERROR CHECKING MACRO */
#define ec(f) check_error(NULL, NULL, f, #f)

#define LOCSSA_SINGLE_LOCATE  1
#define LOCSSA_LOCATE_TABLE   2
#define LOCSSA_GET_INPUTS     3
#define LOCSSA_SHAPE_ATTRS    4
#define LOCSSA_LOCATOR_MGR    5



void  check_error(SE_CONNECTION handle, SE_STREAM stream, LONG rc, CHAR * comment)
{

  SE_ERROR error;
  CHAR  error_string[SE_MAX_MESSAGE_LENGTH];
  LONG  temp_rc;

  if ((rc != SE_SUCCESS) && (rc != SE_FINISHED)) {
    error_string[0] = '\0';
    SE_error_get_string(rc, error_string);
    printf("Error in \"%s\": %ld\n", comment, rc);
    printf("%s.\n", error_string);

    /* Print extended error info, if any */

    if ((SE_DB_IO_ERROR == rc) | (SE_INVALID_WHERE == rc)) {
      if (NULL == handle)
       /* Assume this is a stream error */
       temp_rc = SE_stream_get_ext_error(stream, &error);

      else
       /* Assume this is a connection error */
       temp_rc = SE_connection_get_ext_error(handle, &error);

      if (SE_SUCCESS == temp_rc)
       printf(
              "ExtendedError Code: %d\n"
              "ExtendedError String1:\n%s\n"
              ,error.ext_error
              ,error.err_msg1
         );
      if (error.err_msg2[0] != '\0')
       printf(
              "ExtendedError String2:\n%s\n"
              ,error.err_msg2
         );
    }
    exit(0);
  }
}
```

```
LONG  locatorinfo_as_string(
                              SE_LOCATORINFO locator,
                              CHAR ** locator_as_string
)
{
  LONG  i;
  LONG  num_props = 0;
  CHAR **properties = NULL;
  CHAR **values = NULL;
  LONG  buf_size = 0;

  unsigned int PROPERTY_WIDTH = 32;
  unsigned int PROP_VALUE_WIDTH = 255;

  char  tmp_string[300];

  ec(SE_locatorinfo_get_property_values(locator, &num_props, &properties, &values));

  buf_size = num_props * PROPERTY_WIDTH + num_props * PROP_VALUE_WIDTH;

  *locator_as_string = calloc(1, buf_size);

  for (i = 0; i < num_props; i++) {
    sprintf(tmp_string, "%s=%02X\"%s\";", properties[i], strlen(values[i]), values[i]);
    strcat(*locator_as_string, tmp_string);
  }

  if (num_props)
    SE_locatorinfo_free_property_values(num_props, properties, values);

  return SE_SUCCESS;
}



void locatorinfo_set_property_value(
  SE_LOCATORINFO locatorinfo,
  const CHAR * property,
  const CHAR * prop_value
)
{
  LONG  i, rc = SE_SUCCESS;
  LONG  NumProperties = 0, num_new_props = 0;
  CHAR **Properties;
  CHAR **Values;

  ec(SE_locatorinfo_get_property_values(locatorinfo, &NumProperties, &Properties, &Values));

  for (i = 0; i < NumProperties; i++)
    if (!stricmp(property, Properties[i]))
      strcpy(Values[i], prop_value);

  ec(SE_locatorinfo_set_property_values(locatorinfo, NumProperties, Properties, Values));

  SE_locatorinfo_free_property_values(NumProperties, Properties, Values);

}
```

```
main(int argc, char *argv[])
{

  CHAR *server = "plague";
  CHAR *instance = "5154";
  CHAR *database = 0;
  CHAR *user = "sbctrans";
  CHAR *password = "sbctrans";

  CHAR *locator = "Seattle May 4 2001";

  LONG  ssa_function = LOCSSA_SINGLE_LOCATE;
  CHAR *locator_as_string;
  CHAR *street_string = "348 E Pike St";
  CHAR *zone_string = "98122";

  const CHAR *ssaName = "locssa";

  CHAR  Status[2];
  SHORT Score;
  CHAR  Side[2];
  LFLOAT X;
  LFLOAT Y;
  CHAR  Stan_addr[64];
  LONG  Ref_ID;

  LONG  rc = 0;
  SE_CONNECTION conn = 0;
  SE_ERROR err;
  SE_INPUT_DESC *input_desc = 0;
  SE_LOCATORINFO locatorinfo = 0;
  SE_STREAM stream;
  SHORT num_inputs = 4;
  void **data = 0;

  ec(SE_connection_create(server, instance, database, user, password, &err, &conn));

  ec(SE_stream_create(conn, &stream));

  ec(SE_locatorinfo_create(&locatorinfo));

  ec(SE_locator_get_info(conn, locator, locatorinfo));

  /* add params to locator to get X,Y and OID from shape */
  locatorinfo_set_property_value(locatorinfo, "WriteXYCoordFields", "True");
  locatorinfo_set_property_value(locatorinfo, "WriteReferenceIDField", "True");
  locatorinfo_set_property_value(locatorinfo, "WriteStandardizedAddressField", "True");

  locatorinfo_as_string(locatorinfo, &locator_as_string);

  input_desc = (SE_INPUT_DESC *) realloc(input_desc, num_inputs * sizeof(SE_INPUT_DESC));
  data = (void **) realloc(data, num_inputs * sizeof(void **));

  input_desc[0].sde_type = SE_INTEGER_TYPE;
  input_desc[0].size = 0;
  data[0] = (void *) &ssa_function;

  input_desc[1].sde_type = SE_STRING_TYPE;
  input_desc[1].size = strlen(locator_as_string);
  data[1] = (void *) locator_as_string;

  input_desc[2].sde_type = SE_STRING_TYPE;
  input_desc[2].size = strlen(street_string);
  data[2] = (void *) street_string;

  input_desc[3].sde_type = SE_STRING_TYPE;
  input_desc[3].size = strlen(zone_string);
  data[3] = (void *) zone_string;

  ec(SE_stream_invoke_ssa(stream, TRUE, num_inputs, input_desc, data, ssaName));
  ec(SE_stream_execute(stream));
  rc = SE_stream_fetch(stream);
  ec(rc);
  if (rc == SE_SUCCESS) {

    ec(SE_stream_get_string(stream, 2, Status));
    ec(SE_stream_get_smallint(stream, 3, &Score));
    ec(SE_stream_get_string(stream, 4, Side));

    rc = SE_stream_get_double(stream, 5, &X);
    if (rc != SE_SUCCESS && rc != SE_NULL_VALUE)
      check_error(NULL, stream, rc, "SE_stream_get_double(stream, 5, &X)");
    if (rc == SE_NULL_VALUE)
      X = 0;

    rc = SE_stream_get_double(stream, 6, &Y);
    if (rc != SE_SUCCESS && rc != SE_NULL_VALUE)
      check_error(NULL, stream, rc, "SE_stream_get_double(stream, 6, &Y)");
    if (rc == SE_NULL_VALUE)
      Y = 0;

    ec(SE_stream_get_string(stream, 7, Stan_addr));
    ec(SE_stream_get_integer(stream, 8, &Ref_ID));

  }
  ec(SE_stream_close(stream, TRUE));


  printf("Status:    %s\n", Status);
  printf("Score:     %d\n", Score);
  printf("Side:      %s\n", Side);
  printf("X:         %f\n", X);
  printf("Y:         %f\n", Y);
  printf("Stan_addr: %s\n", Stan_addr);
  printf("ObjectID:  %d\n", Ref_ID);


  SE_stream_free(stream);

  SE_connection_free(conn);

}
```

THAT'S IT!

Have fun.

Questions and comments encouraged.