SPIRION

Spirion Extensions

Algorithmic Validation of UIDAI/Aadhaar Numbers with Spirion's Search API

Table of Contents

Algorithmic Validation of UIDAI/Aadhaar Numbers with Spirion's Search API

Introduction

Spirion's Search API creates customized algorithmic search parameters for user-defined data types. Using this feature, computational logic validates potential matches initially captured by pattern-based definitions. This extension covers the steps necessary to incorporate Verhoeff validation when configuring UIDAI/Aadhaar number searches with Spirion.

Requirements

Before working on the steps outlined in this document, please confirm the following:

- The latest SDM console is accessible.
- An integrated development environment (IDE) capable of editing and compiling Microsoft Visual C++ files is available.
    - Microsoft Visual Studio Enterprise 2019 is used in the procedure explained below.
- The Search API sample project has been downloaded from the Spirion Knowledge Base.
- UIDAI sample data is obtainable.

Important Notes

Spirion Software Versions

SDM Console version 11.8.2 and Spirion Agent version 11.8.7 were used to perform the steps described in this document.

Verhoeff Algorithm

The Verhoeff algorithm validates a check digit by referencing the following tables and procedure:

d(j,k)		k									
		0	1	2	3	4	5	6	7	8	9
j	0	0	1	2	3	4	5	6	7	8	9
	1	1	2	3	4	0	6	7	8	9	5

j	inv(j)
0	0
1	4

p(pos,num)		num									
		0	1	2	3	4	5	6	7	8	9
pos(mod 8)	0	0	1	2	3	4	5	6	7	8	9
	1	1	5	7	6	2	8	3	0	9	4

	2	2	3	4	0	1	7	8	9	5	6
	3	3	4	0	1	2	8	9	5	6	7
	4	4	0	1	2	3	9	5	6	7	8
	5	5	9	8	7	6	0	4	3	2	1
	6	6	5	9	8	7	1	0	4	3	2
	7	7	6	5	9	8	2	1	0	4	3
	8	8	7	6	5	9	3	2	1	0	4
	9	9	8	7	6	5	4	3	2	1	0

2	3
3	2
4	1
5	5
6	6
7	7
8	8
9	9

	2	5	8	0	3	7	9	6	1	4	2
	3	8	9	1	6	0	4	3	5	2	7
	4	9	4	5	3	1	2	6	8	7	0
	5	4	2	8	6	5	7	3	9	0	1
	6	2	7	9	3	8	0	6	4	1	5
	7	7	0	4	6	9	1	3	2	5	8

1. Starting with the rightmost digit, convert the number to an array.
    a. For example, `2363` becomes `[3, 6, 3, 2]`.
2. Initialize the checksum, `c`, to `0`.
3. Iterate over the array, replacing `c` with `d(c, p(i mod 8, n_i))`.
4. After the last step, the check is correct if `c` is `0`.
5. To calculate the correct value itself, set the first number in the array to "0" (to remove the reported check digit from consideration) before completing step #3, looking up `inv(c)` instead for step #4.
    a. If this calculated value is equal to the check digit, it passes the Verhoeff check.

Example

The example above – `2363` – is validated in the following steps:

i	n_i	$p(i,n_i)$	*c*
0	3	3	3
1	6	3	1
2	3	3	4
3	2	1	0

Alternatively, we can confirm that "3" is a valid check digit by following step 5 from the process in the previous section.

i	n_i	$p(i,n_i)$	*c*
0	0	0	0
1	6	3	3
2	3	3	1
3	2	1	2

Looking up `inv(2)` in the table on the previous page, "3" is confirmed as the correct check digit.

Process

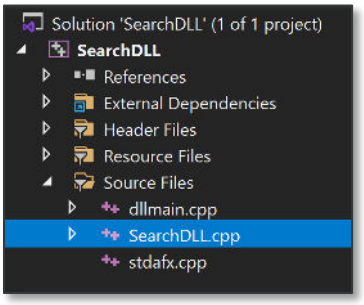
Custom data types created using Spirion's Search API require the creation of a DLL file that contains the user-defined logic articulated in C++ code. Once compiled, the DLL file is added to the SDM Console and hosted locally on any Spirion Agent engaged in a search for the custom data type.

Creating a DLL

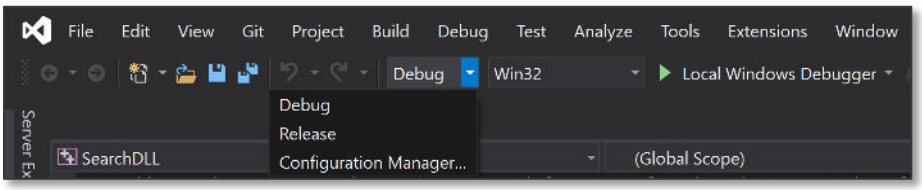
Spirion has created a sample project that should be used as the template for any custom data types created using the Search API. It is available for download within the Search API Knowledge Base article. Unpack it and open ***SearchDLL.vcxproj*** in an IDE, such as Visual Studio 2019 documented in this walkthrough.

NOTE: *The line(s) specified throughout this section references the SearchDLL file in its original state (i.e. prior to adding any new code specified below).*

1. In the project tree, navigate to ***SearchDLL.cpp***.



2. From the ***Solution Configurations*** pulldown menu, toggle from "Debug" to "Release."



3. On ***line 19***, specify a ***CUSTOM_SEARCH_NAME***.
4. On ***line 20***, specify a ***RESULT_TYPE***.
    a. The default value of "12001" may be used once, but all subsequent DLLs must feature a unique value.
5. On ***line 63***, specify a regular expression per the guidance in the comments.
    a. **NOTE:** *As emphasized in the project's comments, backslashes must be escaped.*

b. For UIDAIs, enter the following baseline regex:

```
(^|\\s)[2-9]\\d{3}((-|\\s)*\\d{4}){2}(\\s|$)
```

6. On ***line 92***, specify any keywords that should be used to validate a match.
   a. For example, consider the following keywords in a regular expression:

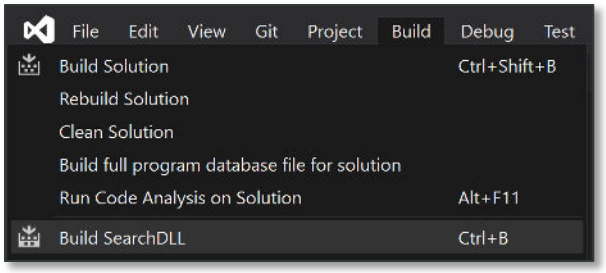
```
aadhar|uidai|adhar|adhaar|aadhaar
```

7. Change ***dataType*** on ***line 99*** to equal "1" so the above regex is used for keyword validation.
8. Replace ***lines 160 through 166*** with the following code:

```
//UIDAI LOGIC

// multiplication table

static int d[][10] = {

  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9},

  {1, 2, 3, 4, 0, 6, 7, 8, 9, 5},

  {2, 3, 4, 0, 1, 7, 8, 9, 5, 6},

  {3, 4, 0, 1, 2, 8, 9, 5, 6, 7},

  {4, 0, 1, 2, 3, 9, 5, 6, 7, 8},

  {5, 9, 8, 7, 6, 0, 4, 3, 2, 1},

  {6, 5, 9, 8, 7, 1, 0, 4, 3, 2},

  {7, 6, 5, 9, 8, 2, 1, 0, 4, 3},

  {8, 7, 6, 5, 9, 3, 2, 1, 0, 4},

  {9, 8, 7, 6, 5, 4, 3, 2, 1, 0}

};


// permutation table

static int p[][10] = {

  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9},

  {1, 5, 7, 6, 2, 8, 3, 0, 9, 4},

  {5, 8, 0, 3, 7, 9, 6, 1, 4, 2},

  {8, 9, 1, 6, 0, 4, 3, 5, 2, 7},

  {9, 4, 5, 3, 1, 2, 6, 8, 7, 0},
```

```
  {4, 2, 8, 6, 5, 7, 3, 9, 0, 1},
  {2, 7, 9, 3, 8, 0, 6, 4, 1, 5},
  {7, 0, 4, 6, 9, 1, 3, 2, 5, 8}
};

// inverse table
static int inv[] = { 0, 4, 3, 2, 1, 5, 6, 7, 8, 9 };

int len = answer.size();
int n = len - 1;
int sumScope = len - 2;

const int chkDigit = answer[n] - _T('0');
int c = 0;

for (int i = 0; i < (len / 2); i++) {
        swap(answer[i], answer[n]);
        n = n - 1;
}

for (int i = 1; i < len; i++) {
        c = d[c][p[((i) % 8)][answer[i] - _T('0')]];
}

if (inv[c] != chkDigit) { return(false); }
```

9. Repeat the previous step on ***lines 212 through 218***.
10. ***Save*** changes to "SearchDLL.cpp."
11. Navigate to ***Build*** in the main menu and select ***Build SearchDLL***.



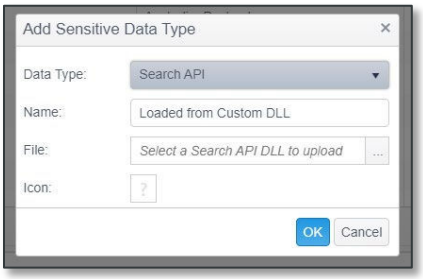
12. Note the export path indicated in the IDE's compilation logs to find where the newly built DLL file resides.
13. Rename this file to "SearchUIDAI.dll".
    a. For ongoing maintenance of this custom data type, consider renaming the "SearchDLL" project to "SearchUIDAI" for clarity.

Configuring the Spirion Console

Adding the DLL

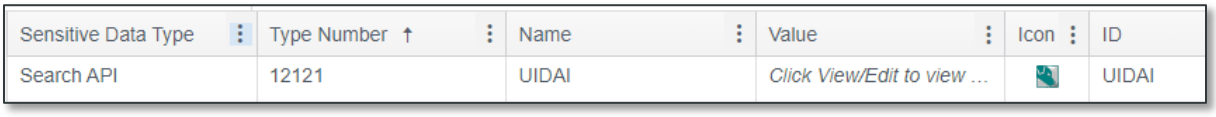
Once built, the DLL now needs to be added to the SDM Console..

1. Log into SDM.
2. Navigate to the ***Admin*** tab and select ***Sensitive Data Types*** from the menu on the left.
3. Click ***Add*** from the ribbon menu.
4. From the ***Data Type*** pulldown, select "Search API."
5. Click the **...** button next to ***File***.



6. Browse to the DLL(s) generated for this exercise in the previous section and click ***Open***.
7. Select ***Ok*** to finalize the entry.

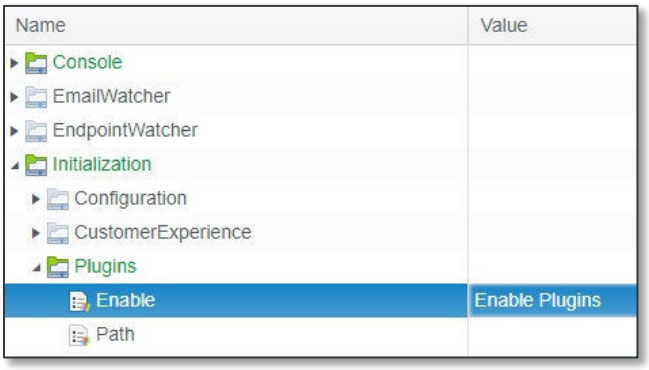
8. The custom data type associated with the DLL file will now appear as a "Search API" entry with its associated "Type Number" and "Name" as specified in the previous section.



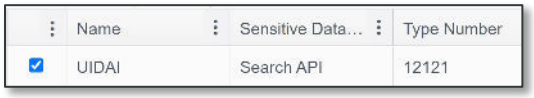
Setting the SDM Scan Policy

The Search API should be enabled via "Scheduled Task" policy (as opposed to a "System" policy).

1. Navigate to the ***Policies*** tab.
2. Modify or create a ***Scheduled Task*** policy that will be sent to the Spirion Agent(s) for discovery scans using the Search API custom data type(s).
3. Under ***Settings*** expand ***Initialization > Plugins*** and set ***Enable*** to "Enable Plugins."
4. ***Initialization > Plugins > Path*** can be left blank (its default setting) unless the Spirion Agent is installed to a nonstandard location.



5. Under ***Sensitive Data Types*** within the same policy, select UIDAI.



6. The Agent(s) assigned to this policy ***must*** be configured per the guidance in the following section before being used in a search.

Staging the Spirion Agent

The DLL file needs to be hosted locally within the Spirion Agent's install directory for Search API data types to be returned in scans initiated by either the Console or Agent UI.

1. Using Windows File Explorer, navigate to the installation path of the Spirion Agent.
    a. This is ***C:\Program Files (x86)\Spirion*** by default.
2. Create a folder called "Plugins" and place the DLL file(s) in that new directory.

Outcomes

With the custom DLL added to both the SDM Console and the Spirion Agent, scans configured to use the Search API will include matches for UIDAI numbers that are validated programmatically.

SDM Console Searches

Search API entries in the SDM Console are included in scan results so long as "Initialization > Plugins > Enable" is set to "Enable Plugins" in an actively engaged policy – selection via the policy's "Sensitive Data Types" menu is used for tracking purposes only.

Restricting Results

Individual Search API data types can be selectively included in a discovery scan by editing the Scheduled Task policy to include the following adjustments:

1. Navigate to the ***Settings*** of a Scheduled Task configured to enable the Search API.
2. Under ***Console***, edit ***matchTypesCustom*** to include the "Type Number" of the custom data type that should be included.
3. All other Search API entries will be omitted from scan results.