

findIT S2 Information Pack



Contents

Introduction.....	7
System Requirements.....	8
Installation	9
Running the Installer	9
Completing the Installation.....	14
Troubleshooting and Further Information	14
Amending the Web Service Configuration XML File.....	15
Breakdown of a 'datasource' element.....	15
Breakdown of the 'batchdatasource' section	19
Breakdown of the 'outputfields' section	20
matchKeySettings.xml.....	20
Using the Administration Program	22
Connection String Encryption.....	22
Load Configuration File	23
Generate Keys.....	24
Rebuild Indexes.....	25
Using the findIT S2 Interface.....	26
Using US address correction	28
Using UK and ROW address correction	30
International Address Verification WCF Interface	31
Address Formatting.....	31
Character Translation.....	31
Address Service Usage Logging	32
Configuring UK and ROW Addressing Formats	34
AddressFormatConfig.xml.....	34
CasingConfiguration.xml.....	34
Configuring the findIT S2 Interface	35
New Entry Settings	35
Button Shortcuts.....	35
Modules	35
Unique Fields.....	35
Contact and Address Input Fields	36
Results Grid.....	36
Output Fields.....	37
Addressing Modules	37

The findIT S2 Web Service	39
AdminBulkGenerateKeys (List of Data source IDs, Encrypted Admin Password)	39
CancelAdminBulkGenerateKeys()	40
CancelIndexKeysTable().....	40
CompareRecords().....	40
ConfigurationErrorList().....	40
DeleteAllDatabaseObjects()	40
DeleteKeys(Unique record reference, Data source ID).....	41
EnableService(Boolean status).....	41
FindMultiRecords(XML Search String).....	41
FindRecords(XML Search String).....	41
GetBulkGenerateProgress()	43
GetDatasourcesWithKeys()	43
GetIndexingProgress()	43
GetLastError()	43
GetOutputFields()	43
IndexKeysTable(Encrypted Admin Password)	43
InsertKeys(Unique record reference, Data source ID)	44
IsServiceEnabled().....	44
LoadConfiguration(File path to XML Configuration File)	44
PerformPendingKeyUpdates().....	44
ReturnKeys(Unique record reference, Data source ID)	44
UpdateKeys(Unique record reference, Data source ID).....	44
ValidateAdmin(Encrypted Admin Password).....	45
The US Addressing Web Service.....	47
The Key Updates Windows Service.....	57
CRM System Integration	58
Appendix A – Troubleshooting	59
Installation Errors	59
Web Browser Errors.....	59
Administration Program Errors.....	60
Appendix B – Quick Start Check List.....	62
Appendix C – Warning values returned from ConfirmAddressXML.....	63
Appendix D – Addressing Web Service Quick Start Guide.....	64
Appendix E – findIT S2 WCF Search Interface.....	69
findITS2SearchService	69
FindRecords(FindRecordsInput input)	69
CompareRecords(List<Record> records)	69

ParseRecords(List<Record> records).....	69
FindRecordsInput	69
List<string> SpecificDataSource.....	69
List<UniqueRefSearchInfo> UniqueRefSearches.....	69
Record SearchRecord	69
UniqueRefSearchInfo	69
string DataSourceID	69
string TableName	69
string ColumnName	70
string UniqueRefValue	70
Record	70
FindRecordsResult	71
bool Success	71
String LastError	71
List<FindRecordsResultItem> Results.....	71
int TotalTime	71
FindRecordsResultItem	71
int ResultCode.....	71
FindRecordsInput Input	71
string InputFields	71
Record NormalisedRecord	71
List<Match> Matches	71
List<string> Warnings.....	71
int Count	71
int Elapsed.....	72
Match	72
string DataSourceID	72
Record Record	72
double Score.....	72
double MinScore.....	72
double MaxScore.....	72
CompareRecordsResult	73
bool Success	73
string LastError	73
List<Comparison> Results	74
int TotalTime	74
Comparison	74
string Record1	74

string Record2	74
Scores Scores	74
Scores.....	74
ParseRecordsResult	74
List<Record> ResultRecords	74
bool Success	75
string LastError	75
Appendix F – International Address Verify Service Interface	76
InternationalAddressVerifyService.....	76
ProcessResult Process(Address address, ProcessOptions processOptions, Processes processes)	76
Address.....	76
ProcessOptions.....	77
string FIS2AddressLineFormattingConfig	77
string AddressLineSeparator.....	77
string CasingIgnoreFields.....	77
string CertifiedCountryList.....	77
int ConfidenceThreshold	77
int ContextResultCacheSize	77
int ContextCountryCacheSize	78
string CountryFields.....	78
string CustomFields	78
int CustomFieldConfidence	78
string DataDirectory	78
string DefaultCountry	78
int FieldNameWeight	78
string ForceCountry	78
string GeocodeCountryList	78
bool IgnoreUnmatched	78
string LogFileName.....	78
int MatchScoreAbsoluteThreshold	78
int MatchScoreThresholdFactor	78
int MaxResults.....	79
int MinimumMatchscore	79
int MinimumPostcode	79
int MinimumVerificationLevel.....	79
bool OutputAddressFormat	79
CasingOptions OutputCasing	79
string OutputScript	79

int QueryResultCacheSize	79
RangeDecoposeOptions RangeDecompose.....	79
int ReferenceDatasetCacheSize	79
int ReferencePageCacheSize	80
string SuppressAdditionFields.....	80
string SuppressAddressFields	80
string SuppressFields.....	80
string TransliterationIgnoreFields	80
bool ToolInfo	80
bool UseCustomLexicons.....	80
bool UseSymbolicTransliteration	80
string VerifyCountryList.....	80
string LogInput	80
string LogOutput	80
Processes	80
ProcessResult	81
string AccuracyCode	81
List<ProcessResultItem> ResultItems	81
ResultStatus Status	81
string LastError	81
ProcessResultItem	81
ProcessResultItemField	83
string Name	83
string Value	83
int Confidence.....	83
string Info.....	83
float MatchScore	83
FieldStatus Status.....	83
MatchInfo	83
FormatInfo.....	84

Introduction

findIT S2 is a SOAP based web service and web UI for real-time data cleansing which allows you to embed data quality functions into your applications or clients systems.

Delivered as an IIS .NET web application and built on helpIT systems' proven data matching engine, findIT S2 can work with any data driven application, website/web application or CRM. Target integrations are within CRM, ERP, e-Commerce, web based systems and call center applications.

findIT S2 is designed to be embedded or integrated into the frontend application and reside between the data entry process and database. The intelligence behind findIT S2's UI simplifies the data entry process by assisting the user during data entry and reporting suspect duplicates to the UI, and calls a postal reference database to ensure that addresses are entered completely, accurately and in fewer keystrokes.

The data quality engine extrapolates further reference data, standardizes or corrects inputs and post the information back to the underlying database, and can be configured to run in interactive, real-time, or batch modes.

Additionally findIT S2 can connect to multiple data sources to enhance data or to identify potential duplicates or existing accounts in other databases or suppression files- making it ideal for ecommerce and Red Flag environments to provide an immediate alert to existing or potential known fraud accounts.

It is simple to install (through a single installer) and needs minimal configuration to get up and running. The findIT S2 comes in the form of the following components.

findIT S2 Service – The findIT S2 service is a duplicate identification and data cleaning web service and has two main purposes. It is able to clean, case and match data, and also manage the methods used by the administration program such as methods to create / update / delete the necessary keys to perform the matching process. The web service can be used by multiple clients over the web to query the database it has been configured to protect and is configured using its XML configuration file (findITConfiguration.xml)

Certified US Addressing Web Service – [US Only] – The US Addressing web service is implemented using two main modes of operation, interactive reverse lookup as well as unattended mode to correct a fully keyed address. Both of these methods are seamlessly integrated into the findIT S2 User Interface.

findIT S2 User Interface – The web interface is served from a dedicated web site and comprises a single UI page featuring data capture fields, the matching results grid which displays xml results returned from the data cleansing web service and an information feedback pane. The findIT S2 UI page is customizable by the administrator to suit potential requirements using the XML configuration file (ControlConfig.xml)

An Administration Application – This Windows's application offers an interface to configure the web service by loading in an XML configuration file. This application is also used to generate the necessary keys table required in the database(s) to perform matching processes. You can also disable the web service if necessary, and tidy up any objects created by findIT S2 on the databases in question.

A Key Updates Windows Service – This is a windows service that periodically calls to the findIT S2 service to perform any updates to matching keys that are pending due to changes in the data that the web service is configured to manage. The interval of the checking can be set by the administrator.

System Requirements

The following are the hardware and software requirements for the findIT S2 -

Hardware Requirements -

- *Processor* – Minimum 1.8-GHz processor, recommended multi-core or multiple 1.8-GHz CPU or higher

- *Memory* – Minimum 1-GB RAM, recommended 2-GB RAM or higher

- *Hard Disk* – It is recommended that you have at least the size of the database that you are going to plug findIT S2 into available as free space on the hard disk to permit the database to grow when generating keys.

N.B - For address correction an additional 2.5 GB is required to store the USPS data files. These files are refreshed on a bi-monthly basis, and so this size requirement could increase in time.

Software Requirements -

- *Microsoft Windows Operating System* – XP, Vista, Server 2003, Server 2008, Windows 7

- *.NET Framework* – Minimum version 3.5 – You will be prompted during install to download this, if you do not have it installed on your web server.

- *Microsoft Internet Information Services (IIS)* – Minimum version 5.1 (XP), 6.0 (Server), 7.0 + (Vista and Windows 7)

- *RDBMS* – SQL Server 2005, 2008

Installation

Running the Installer

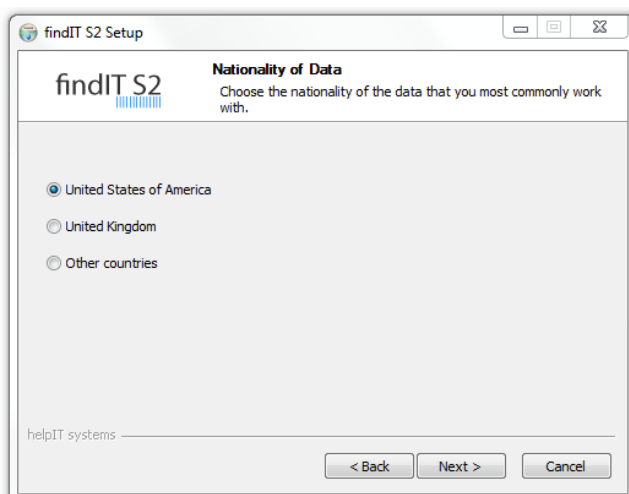
To install the software, run the installer package as system administrator and follow the steps screen by screen to configure findIT S2.



Licensed users can go to [findIT S2 download link](#) to download the latest version of findIT S2. Software updates are handled using the *check for updates* application for existing users and there is an optional reminder service that will periodically check for the availability of updates.

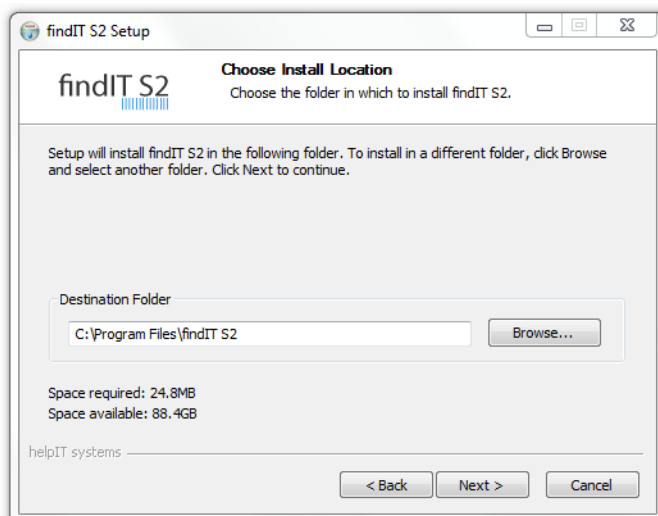
On running the installer, navigate through the wizard page by page. After accepting the software license agreement, you will then have to set up the software install page by page, as outlined below.

Nationality of Data - (Optional): This dialog will be displayed for new users only and sets the default regional settings for the matchIT API data matching engine to the most appropriate values for the nationality selected.



If you have previously installed the product, this screen will not be displayed.

Install Location - After running the installer and accepting the software license agreement, you will be asked to select an install folder for the web application, documentation and service administration programs.



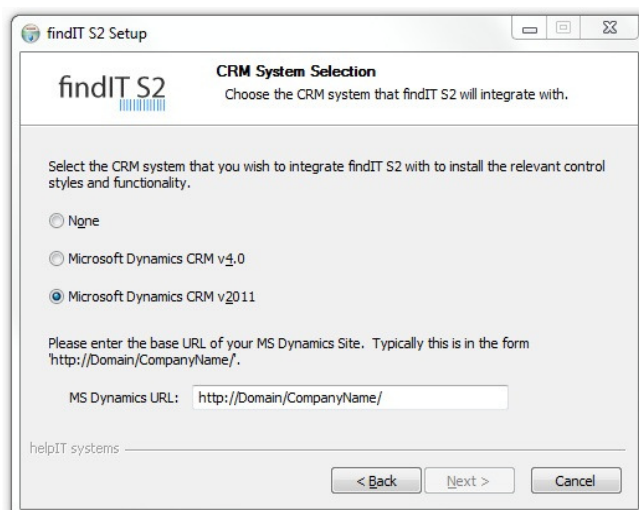
findIT S2 installs as a website (which serves the User interface) and one (two if you install the optional US addressing web service) web service on your and the installer will create the required web applications automatically and host these in IIS for you. Depending on operating system and CRM integration the outcome of this process may look a little different. On non CRM integrations the web sites will be configured as follows :

Windows 2003 / Windows Vista / Windows Server 2008/ Windows 7 - the installer will create a new web site, *findIT S2*.

Windows XP - the installer will create the 2 or 3 virtual directories under *Default Web Site*.

CRM integration web site locations are covered in the next section, as to allow integration, they need to reside under the same web site as the host CRM system.

Integration Options - After deciding the install location, you will then be prompted to select how you would like to integrate findIT S2 into your system, there are currently three options :



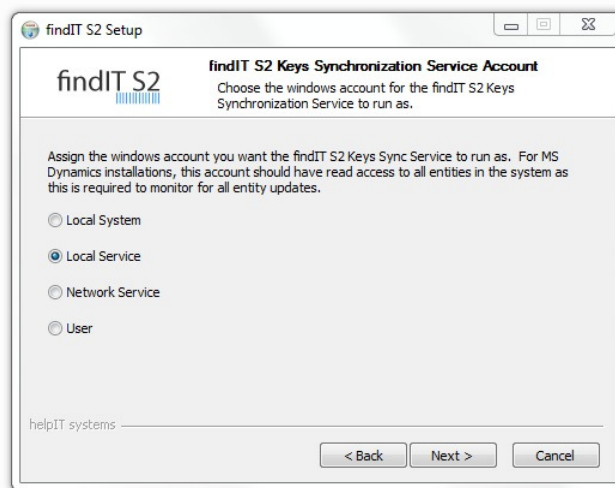
- findIT S2 for Microsoft Dynamics 4.0,
- findIT S2 for Microsoft Dynamics 2011

- The non CRM integrated version of findIT S2, which allows integration of the web service and interface into a custom implementation.

The CRM specific versions contain various assets to incorporate findIT S2 into the appropriate CRM interface and specific XML configuration options so the product can link to a default database in each case.

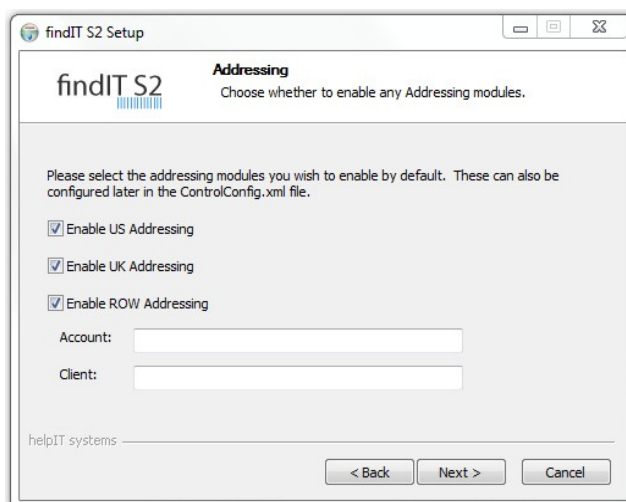
When installing for either version of Microsoft Dynamics, you will also be asked at this point to specify the base URL that points to your Dynamics deployment.

Keys Synchronization Service Account – This step enabled you to specify the windows account for the Keys Synchronization Windows Service to run as, the options are Local System, Local Service, Network Service or a custom user account.



As per the intro text in the installer, Dynamics Deployment installations should use a windows account that has read privileges on all entities within dynamics to ensure accurate match key updates and synchronization.

Addressing - (Note US addressing available to US users only) – On this screen, select which addressing modules you wish to enable in the UI. If you have account and client details for ROW addressing, you can enter them now, or otherwise do it later manually.

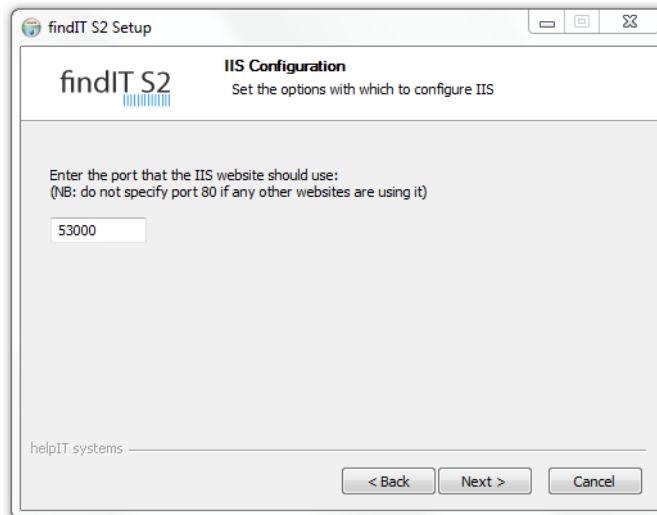


After you have completed the installation of findIT S2, you will be required to install the bi-monthly data update on the server, from a supplied DVD or via a downloaded link supplied by

the helpIT systems support team. The data is required to allow address correction during entry and also at point of record submission to the database.

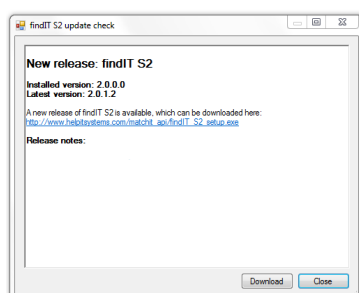
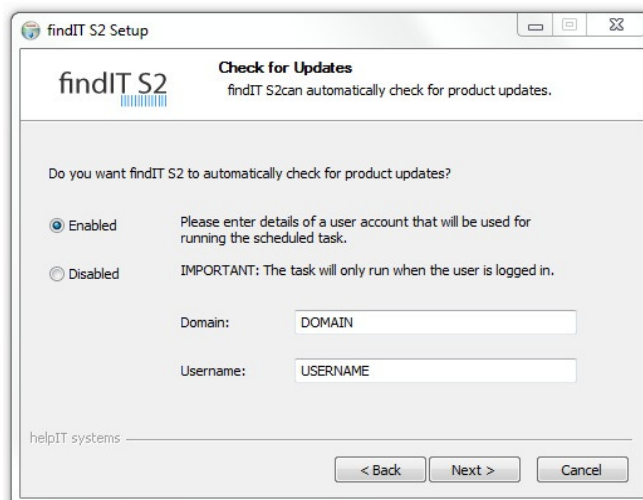
You are able to activate or deactivate US address correction after installation by editing the findIT S2 service's configuration file.

IIS Configuration - The next step is to specify a port number for findIT S2's web applications



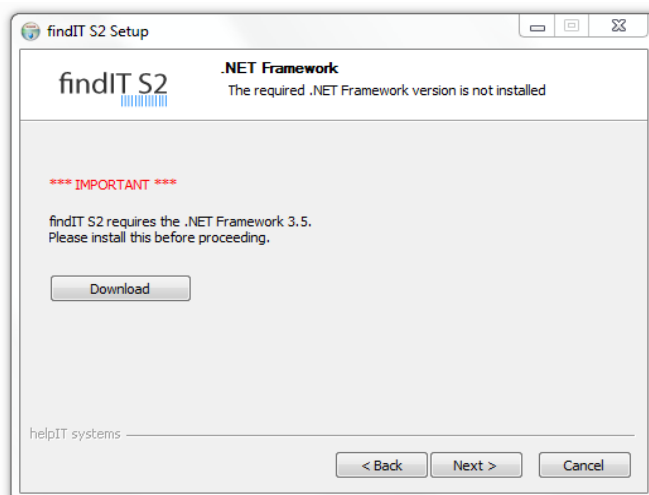
The default value is 53000 but this can be configured to suit a particular environment.

Check for Updates - findIT S2 has the option to automatically check for updates. This process contacts the helpIT systems website to determine if there is a recommended upgrade currently available.

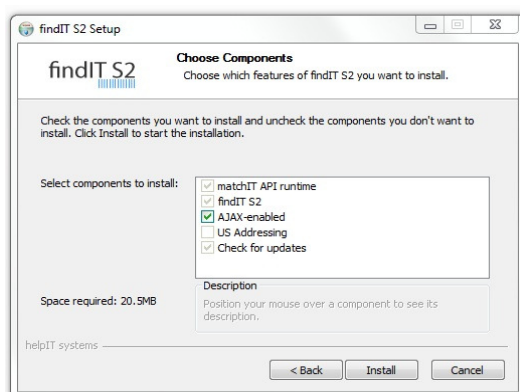


When looking to upgrade a live system you will need to take measures to ensure uninterrupted usage by scheduling this around system downtime and also run the required procedures from the findIT S2 Admin program. The check for updates service will give you the release notes, as well as a link to download the newest setup executable.

.NET Framework - If you do not have the Microsoft .NET Framework 3.5 on your machine, you will be prompted to install this before completing the installation. This is a prerequisite, so ensure that this is installed prior to completing the findIT S2 install. Clicking on the 'download' button will take you to the location in the Microsoft Download Center to acquire this update.



Final Confirmation - Once you have navigated through each screen (and have installed the .NET Framework 3.5 as required), you will be given a summary of the components you are due to install



At this point, it is possible to specify whether the Web UI of the product is AJAX Enabled or not. AJAX-enabled data entry allows you to get live feedback when entering contact data. For dedupe, this means that as you complete a field of contact information and tab to the next field, the matches grid will automatically update which additional matching information, with addressing, this feature will automatically complete fields of address information, or even prompt you to select from a list of possible candidates.

Please note, for AJAX to work effectively, you will need to review your host system to ensure the functionality is supported. There is a real benefit on having AJAX enabled on a system, so it is installed by default. You are able to deselect the feature as appropriate from the "Choose Components" screen. Please see details regarding the benefits of using an AJAX enabled interface in the usage section using the findIT S2 interface.

Completing the Installation

On completion of the install you can view the readme.txt file which contains pertinent information including the current release notes for the current version.

The findIT S2 folder is accessible from the Windows Start menu, and contains the following shortcuts :

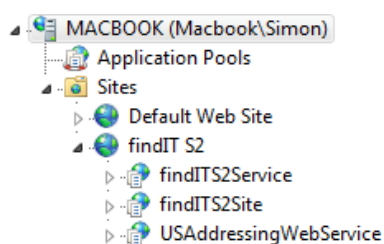
- Configuration Folder
- findIT S2 Administrative Tool
- findIT S2 Test Page
- matchIT API readme
- findIT S2 readme
- findIT S2 Uninstaller
- Documentation
- Release Notes
- Utilities

To view the newly installed web services, go to :

Control Panel → Administrative Tools → Internet Information Services (IIS) Manager

You will notice the following web applications have now been created in IIS; the installer takes care of all this.

In Windows 7/Windows Server 2008/Windows Vista/Windows 2003, this will look something like this for a non CRM installation.



Note that in Windows XP (or for versions of IIS previous to 6.0), the applications will be added under the Default Web Site.

Troubleshooting and Further Information

Note – If during the install process the virtual directories failed to be created, you will have received a message box saying so. If this is the case they will have to be set up manually. For this or any other information regarding issues during installation, please refer to the troubleshooting section at the end of this guide, **Appendix A**.

Amending the Web Service Configuration XML File

At the heart of the findITS2 Web Service is an XML Configuration File - **findITConfiguration.xml**. By Default this is located in the 'config' sub folder root of the findIT S2 directory. It is worth noting at this point that, depending on the CRM option you chose during the installation of findIT S2, your findITConfiguration.xml file will be configured differently.

For example, a Microsoft Dynamics install will come with 3 data sources already structured in the configuration file, with some default field mappings to get you started. The only parts of the file in this case that are left blank for you to fill in are the `connectionstring` and `connectionstringadmin` nodes of each data source.

Of course, you may want to map the fields a little differently depending on what fields you want to search on. The complete structure of the configuration file is described below.

Within the root node `<config>` there can be multiple `<datasource>` nodes (each of which contains the information for a single source of data, which could comprise a single table of contacts from a single database, multiple tables from a single database, or even multiple tables from multiple databases on the same server), a single `<batchdatasource>` node (containing settings relating to the CompareRecords() web method) and a single `<outputfields>` which contains the fields that will be output in the results when using the web method to find matches on a data submission. Data from separate database servers are set up as separate data sources.

Breakdown of a 'datasource' element

The opening tag for a datasource element has three attributes as follows

```
<datasource id="1" name="ds1" type="sqlserver">
```

The first two are simply a reference to distinguish between different data sources in the XML file. The `id` attribute should be an integer and `name` can be an integer, string or mixture as in the above case. The `type` attribute specifies the type of database that the datasource will be using – in this case 'sqlserver'.

The first node within a data source node specifies the connection string for the database that is going to be used when the service queries the data.

```
<connectionstring value="Data Source=localhost; Initial Catalog=example; User ID=user; Password=pass" />
```

The value attribute will need to be updated to contain the database / credentials of your data for user level access. Note that if the data you are going to be using is contained in more than 1 database (on the same server), make sure that all databases in question allow access to the same user – in the above case for example, you would need to make sure that all databases allow access to the user 'user' under the password 'pass'.

Note if you are using a SQL instance that doesn't use the default port, you can specify the new port as follows as part of the data source property.

```
<connectionstring value="Data Source=localhost,1733; Initial Catalog=example; User ID=user; Password=pass" />
```

In the above example, 1733 is the port number assigned to the SQL Server instance.

In addition to the connection string we have a related value, the connectionstringadmin node.

```
<connectionstringadmin
value="oJO0SUY70I5CHD/+lowik/iBdKTwPRtZs40I/dNKM8J+MnjYAsAK9be6UaU+HcLAqE2p+ota
yU/u8pc3LlGHaxUfCsJWtYvo5zu2UpXgvbo=" />
```

As you can see this is not like the regular connection string, it is encoded in the XML file. You can generate this string in the findIT S2 Admin application, and simply paste the encrypted value back into the XML file. This is covered in more detail in the section *Using the Administration Program*.

It is also important to note here that the main table of data for the data source in question (which in this case is the 'contacts' table, as described below), should exist in the database that you specify in the connection string. Any tables thereafter that are to be joined to the main table of data can have their database specified in the database attribute of the table node (as shown below) if they happen to exist in a different database to the one specified in the connection string.

The following node defines the name of the database that is used to create the findIT S2 specific database objects.

```
<findits2objects database="findIT_S2" />
```

By default, the name of the database is set to `findIT_S2` and it is important to note here that the database will be created if it does not exist, so the user account specified in the previous `connectionstringadmin` node will need the privileges necessary to create a database. This database will store objects such as the keys table, updates table and error record log table so that these items are not created in the source database. As well as the database, it is also possible to specify a `schema` attribute, which if not specified will default to `dbo`.

In the case of a Dynamics Installation, there is another node worth mentioning here, which is the `<keyssynchronization type="date" />` node. What this does is changes the method of record update monitoring to use a modified date column (offered as standard in Dynamics) rather than using triggers, which is the default method in a standard install when this node is not specified. If using this node, it is also necessary in the table definitions (below) to specify, for each table, the name of the column to monitor for updates, by using a `synccolumn` attribute. It is also possible to specify an `offset` attribute in the `keyssynchronization` node, to allow for any time zone differences that may cause issues. For example, to use an offset of 5 hours, you would specify `offset="-5"`. At the time of writing, it is only possible to use a column that contains a UTC format DateTime value.

The next node defines the tables that contain the data – note the `keyssynchronization` node has been included in the example below for reference.

```
<keyssynchronization type="date" />

<tables>
  <table name="Contacts" uniqueref="ID" synccolumn="ModifiedDate" />
  <table name="Addresses" uniqueref="ID" join="contacts" joincolumn="ID" />
  <table name="Keys" keystable="1" uniqueref="ID" join="Contacts"
joincolumn="ID" />
</tables>
```

As you can see, individual tables are contained within the `<tables>` node. Each individual `<table>` node can contain various attributes which are described below:

`name` : The name of the table in the database (Mandatory)

`schema` : The name of the table schema used in SQL queries, "dbo" by default (Optional)

`uniqueref` : The name of the unique reference column of the table in question (Mandatory)

`join` : The name of the table that the table in question needs to join to (Mandatory if the table in question is not classed as the main table of data)

`joincolumn` : The name of the column in the join table that the table in question will be joined on (Mandatory if a 'join' table has been specified).

`database` : The name of the database that the table in question exists in (Optional – if not specified the database set as the 'Initial Catalog' in the connection string will be used).

`keystable` : A flag to specify that the table in question is the keys table (Mandatory if the table in question is the keys table. Possible values, 0 or 1.

`synccolumn` : If using the `<keyssynchronization type="date" />` setting, you can use this attribute on each table to specify the column that contains the date field to monitor for updates. An example of this is shown in the tables snippet above.

In the code above, the main table of data would be the table called 'Contacts', which exists in the database 'example1' as no database attribute is specified so the default database in the connection string is used.

The second table that is defined, the table called 'addresses', exists in a database called 'example2', as set by the database attribute. You can see also here that the addresses table joins to the contacts table (`join="contacts"`) by its own unique reference column (`uniqueref="ID"`) to the 'ID' column in the contacts table (`joincolumn="ID"`).

The last table is defined as the keys table. This table will be generated by the administration program. It needs the attribute to specify that it is the keys table (`keystable="1"`) and also needs to join to the main table of data (`join="Contacts" joincolumn="ID"`).

Basically you need to amend the XML in the `<tables>` node to match the structure of the data in your database. In some situations it may be necessary to specify some conditional columns in the table definitions that records would have to adhere to in order to be counted as a record. The most common of these is a deleted flag column. For exactly this purpose it is possible to define sub nodes within a table node called `conditionalcolumn` nodes. The following is an example of such a structure.

```
<table name="Contacts" uniqueref="ID">
  <conditionalcolumn columnname="deleted" isequalto="false" value="True"
    isintegertype="false" />
</table>
```

In the example above, the contacts table contains a conditional column called `deleted` for which records must not be equal to (indicated by `isequalto="false"`) the value `True`. The columns data type is simply defined as either integer or non-integer, in this case being the latter by setting `isintegertype="false"`. Any number of conditional columns can be defined within any table (apart from the keys table for which there are no conditional columns).

The next node is the `<fieldmappings>` node, which contains definitions for all the mappings of your datafields in the datasource to the datafields in the matchIT® record object. Below is an example of one of the mappings

```
<fieldmapping matchITfield="FullName" columnname="CustomerName" />
```

Each `<fieldmapping>` node has 2 attributes. The `matchITfield` attribute is the name of the field in the matchIT record object – do not change this. The `columnname` attribute is the name field in your database that contains the relevant data to be mapped to the 'FullName' field of the matchIT record object. Any matchIT record field for which you don't think your database contains an equivalent should be left blank, as follows

```
<fieldmapping matchITfield="Address7" columnname="" />
```

In place of the `matchITfield` attribute, it is also possible to use a `passthroughfield` attribute. Use this attribute in place of the `matchITfield` attribute when you want to map a column in your data source and return it in queries to the web service, but it doesn't map specifically to any matchIT fields that are available in the matchIT API.

The next node in the XML is the `<settings>` node. The settings within this node are what will ultimately affect the resulting output.

The first three nodes within the `<settings>` node are very straight forward, and are explained below

```
<maxrecords value="10" />
```

The value attribute of this node contains an integer, which is the maximum number of records that can be returned for a query (for the datasource in question). For no limit, set this value to 0. If a particular query exceeds the limit, no results are returned and a warning is displayed.

```
<maxupdates value="1000" />
```

This attribute sets a limit, 1000 by default, to the number of key synchronization related updates that can occur per interval of the keys sync service check. This prevents lengthy queries relating to large numbers of updates consuming excess resource on the machine.

```
<querytimeout value="30" />
```

The value attribute of this node contains an integer, which specifies the length of time (in seconds) of a SQL query timeout. By default it is set to 30 seconds.

```
<omitblankkeysfromquery value="true" />
```

This node allows the user to force all queries to include blank values when searching for records. An example would be when submitting a first address line to search on that lacks premise information – If the premise key is specified in the match keys, but is empty, it will not be search on by default. Setting this option to false would force the same query to specifically select candidate records that have a blank premise.

```
<keyfields mkNameKey="1" mkOrganizationKey="1" />
```

This node (shortened here from the default) is used to determine what keys are output to the keys table. If you know that you are not going to make use of a particular column in searches, then you can save overhead by disabling it. Note that any changes to this setting will require a regeneration of keys to recreate the table schema, and the user will need to make sure that the keys defined in the `matchkeysettings.xml` file correspond to the columns available in the keys table, determined by this setting.

The matchIT API settings (the matching engine) are kept in a separate XML file, to promote readability of the configuration files. To access this file, you need to update the `matchitapisettingspath` node also.

```
<matchitapisettingspath value="C:\Program Files\findIT  
S2\matchITAPISettings\settings.xml" />
```

These settings allow you to alter things like the constraints on address look up, the scoring system and other settings related to record matching. Any changes to these setting should be done carefully and in consultation with the support team at helpIT systems.

It is also possible to define a `<updateableschema>` node to define a schema name to which updates tables should belong, however this shouldn't be needed very often. In the cases where it is, the node takes a 'value' attribute, the value of which should be the schema name.

The `<scoreoutputs>` node is used to turn on or off specific score components that get returned with a match when calling FindRecords. By default, only the total, min and max scores are returned. It is possible however to return the constituent values that make up the total score, such as name and address score, simply by setting the corresponding attribute in this node to a value of 1.

Another part of the functionality that is configurable in this section is the implicit LIKE style matching that is implicitly performed by the web service when search with Name or Company data. These searches are enabled by default, and are performed regardless of which key configuration is used from the match key settings. They are configurable through adding the following node to the `<settings>` node section

```
<likematching>
  <enabled value="false" />
  <minlength value="5" />
</likematching>
```

The LIKE searching can be switched on or off through the `<enabled>` node, and the threshold word length limit for performing the searches can be set through the `<minlength>` node.

In version 2.1.0 the match keys were moved to a separate configuration file to improve readability of the configuration XML document as well as encourage reusability between configurations, without having to set the same matchkeys in each file.

You can change the location of the new file, matchKeySettings.xml in this node

```
<matchkeysettingspath value="C:\Program Files\findIT S2\config\matchKeySettings.xml" />
```

See section matchKeySettings.xml for more information on the configuration of this file.

It is possible to specify an optional node in the settings section to control whether or not the data that is returned in a query to the web service is the original source data, or a cleaned and normalised version produced by the web service process. By default, this node is omitted from the configuration, and the option is set to true (so a parsed and cleansed version of the source data is returned). If you wish to override this, you can do so by adding the following node to the settings section –

```
<normalisesourceresultsdata value="false" />
```

Breakdown of the 'batchdatasource' section

This section outlines a special kind of data source that is used purely for the CompareRecords() web method, used for batch comparison. It is a highly trimmed down version of a datasource element, with a few extra settings that are unique to the batch process.

The only sub node within the `<batchdatasource>` node is the `<settings>` node, which in turn only contains a limited number of sub nodes for batch settings.

The `<matchitapisettingspath>` setting, as for the standard datasource, defines the location of the matchIT API settings file that contains the settings for the matchIT API engine. The following setting, `<maxclustersize>`, determines the maximum number of records that can be submitted to the method. By default this value is 200 – Be aware that the number of comparisons effectively squares with this number, and so processing time and return messages can become quite bloated.

The `<minimumscorethreshold>` node determines the fraction of the maximum possible score that needs to be achieved in order for the comparison to be classified as a match. For

example, a threshold of 0.7 against a max score of 100 will need to score a minimum of 70 to be reported as a match.

The final node, `<scoreoutputs>`, simply determines the score components that are returned along with the unique refs within a comparison node in the return message of a call to `CompareRecords()`. To enable a score component, simply set its attribute value to 1, and likewise set it to 0 to disable.

Breakdown of the 'outputfields' section

The `<outputfields>` node simply contains all the fields that are defined in a `<fieldmappings>` node with a data source, in the following format

```
<outputfield matchITfield="FullName" />
```

Only the fields specified in a field mappings node are available as output nodes. To remove a field from the results output of the web method (namely `FindRecords`, which returns results in XML format, described in the 'Web Service' section), simply comment the node out as follows

```
<!-- outputfield matchITfield="FullName" / -->
```

Note that there is only one `<outputfields>` node in the XML file. The output fields apply to all data sources, and cannot be defined per data source. If a mapping for a particular output field is available in one data source but not in another, the one for which a mapping does not exist will simply display as blank in the results.

As with the field mapping nodes, if you wish to specify an output field that you mapped as a pass through field rather than a matchIT field in the field mappings section, use a `passthroughfield` attribute in place of a `matchITfield` attribute in the `<outputfield>` node.

Important Note – It is a good idea once the `findITConfiguration.xml` file is in the correct state to take a copy of it and back it up somewhere in case the operational one is ever deleted. If this ever happens the copy can simply be placed into the root of the findIT S2 directory and loaded through the admin program. Note that the copied file's permissions will need to allow the 'Users' group 'Read' and 'Read & Execute' permissions in order for the web service to be able to load the file.

matchKeySettings.xml

This document contains all the combinations of match key fields for data entry and lookup.

The `<matchkeys>` node contains the definitions of the keys to match data on depending upon which fields are inputted. Looking at the example below

```
<input field1="Company" field2="Contact" minmatchscore="0">
  <key key1="mkOrgName1" key2="mkName1" />
  <key key1="mkOrgName1" key2="mkOrgName2" />
  <key key1="mkName1" key2="mkName2" />
</input>
```

If the fields that had been used as input were Company and Contact, then these are the keys that would be used to match the data, i.e. the 3 sets of keys contained within the `<input>` node. Also note here that, if a combination of inputs is used for which there is no definition in the match keys settings file, then the closest possible combination is picked. Using the example above, if the input fields used were Company, Contact and Telephone, but there was no combination in the settings file that matched this, then the above definition using just Company and Contact would be used, as it is the closest fit.

As well as being dependent on the input data, the choice that the web service makes on which keys to use also takes into account what fields have been mapped in the data source being queried. For example, if Company and Contact were submitted to the web service to query a data source that only had a company mapping, with no name data, then the above definition would not be used – there is no point as there is no 'Contact' data to query. Instead, it would treat the query as if only the Company data had been submitted, and select a set of match keys to use accordingly.

Currently, the `fieldx` attributes (where `x` is an integer) can have the following values –

- FirstNames
- LastName
- Contact
- Company
- Address1
- Town
- Region
- Postcode
- Country
- Email
- Telephone
- CustomField1-9

An important point to note here is that, whilst the fields Company through to CustomField1-9 in the list above have a one to one correspondence between data submitted to the service vs. input fields looked for in the match key settings file, there is a special case with the first 3 name related fields if submitting the 'FullName' data element to the web service, which is explained below.

If a single word is submitted as a 'FullName', the web service will always parse this as a 'LastName', and so will use 'LastName' to look for keys to use in the match keys settings file. If however multiple words are submitted as a 'FullName', these will be parsed into their first and last name components, and so the 'Contact' input term will be used to determine the keys to use. In order to make use of the FirstNames input, first name data has to be specifically submitted to the web service using the 'FirstNames' attribute. Note that calls to the web service are described in detail further on in this document.

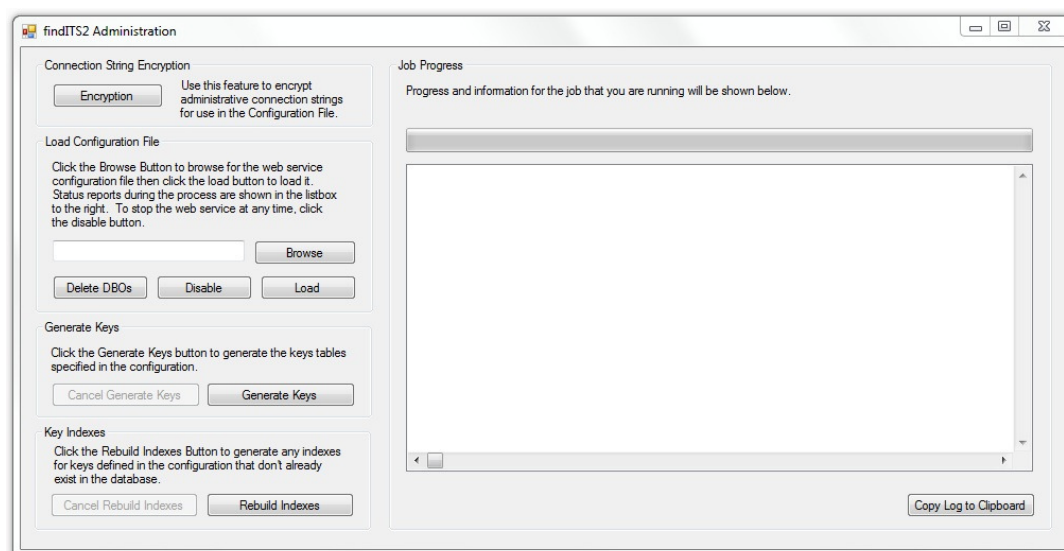
To help with determining what set of match keys were used with a query, the input fields that were parsed and used to select the keys from the match keys settings file are returned in the result of a query to the web service for each data source. For more information on the format of this, please refer to the findIT S2 Web Service section of this document, specifically the FindRecords and FindMultiRecords references.

You will also notice that each `<input>` node contains a `minmatchscore` attribute which specifies the minimum score required by a match to be returned as a result when being searched on using the input in question. This attribute MUST be specified for each key, and as a default is set to zero, which means all results will be returned.

Using the Administration Program

Once you have configured the XML configuration file to your specification, it needs to be loaded into findIT S2 using the administration tool.

This can be accessed via the shortcut in the 'Start' menu that was created during the installation of findIT S2, accessed through 'Start > Programs > findIT S2 > findIT S2 Admin'. Clicking on the shortcut will open up the admin program as follows.



The administration program is split into two sections, on the left are the functions and tools to help set up and maintain the configuration and database settings, on the right is the feedback window, which reports out the progress and status of the operations and functions as they are running.

The left-hand side has 4 main functions –

- Creating an encrypted string to permit administrative access to web service functions.
- Loading and validating the configuration file.
- Generating the matching keys table for each data source.
- Creating any indexes that don't already exist and are defined as match keys in the configuration file *

Note * that the building of all indexes will take place during the Generate Keys– The rebuilding of key indexes should be run if any new match key combinations are defined in the configuration and reloaded, as to create the relevant indexes for these new combinations.

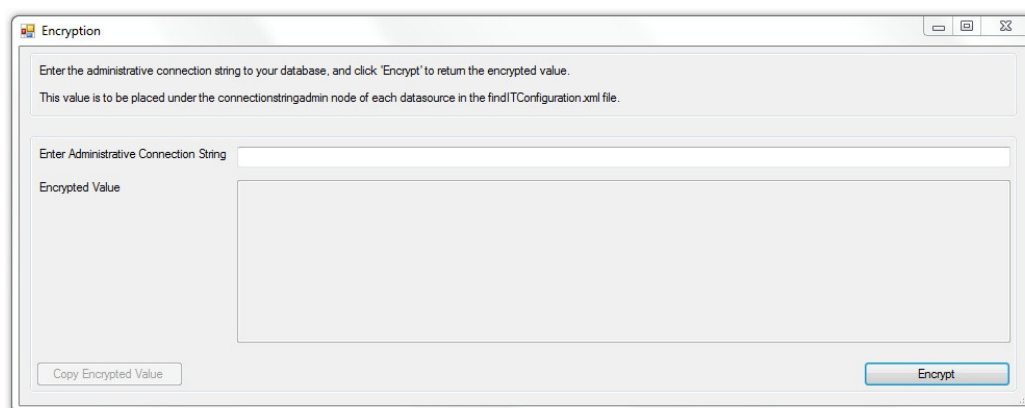
Connection String Encryption

The connection string encryption is used to create an encoded version of an administrative connection string to the SQL Server database.

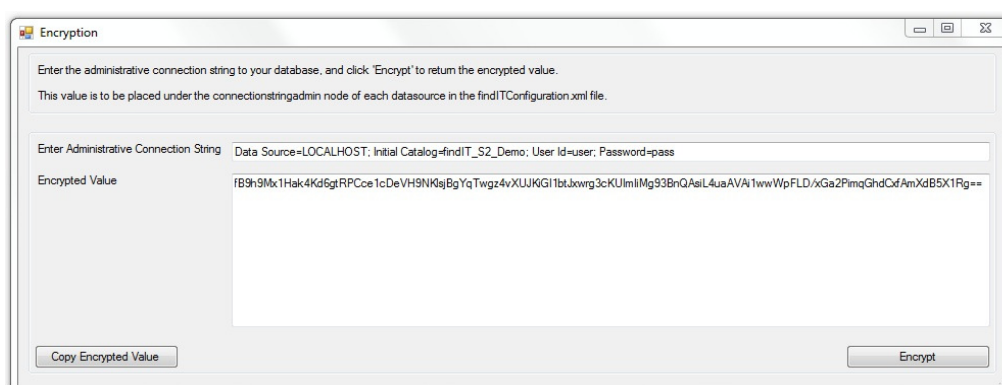
i.e. Create a string for administrative level access to the database “

```
"Data Source=localhost; Initial Catalog=contacts; User ID=admin;Password=pw;"
```

Store this to the clipboard, and click the Encryption button to bring up this dialog :



Paste the admin string in the connection string box, press Encrypt, and you will be given an encrypted string in the Encrypted box :



You then can paste this value back into the findITConfiguration.xml into the connectionstringadmin value.

```

The connection string node contains the connection string for the database server in question. The connection string
requires username and password credentials as the database will be accessed via the web service. The database specified
here in the connection string should be the database that the main table of data for this datasource exists in.
-->
<connectionstring value="" />
<connectionstringadmin value="fB9h9Mx1Hak4Kd5gtRPCce1cDeVH9NK9jBgYqT1wgz4vXUJKGI1btLxwrg3cKUImlMg93BnQAsiL4uaAVA1wwwWpFLD/xGa2PimqGhdCxfAmXdB5X1Rg==" />
<!--

```

The reason we use two different connections is for security purposes. A regular user of findIT S2 may be someone who undertakes regular data entry, wants to look up records in a system, or even look up a validated address. However, the connection string also allows access to do things like create database tables, drop indexes and create database triggers.

As you can see the two levels of permission are very different, so we have two different connections. One to allow a regular user read/write access to the underlying database, and the other for administrative power users. The administrative string is encoded, to prevent any sensitive connection information being accessible outside of the database.

Load Configuration File

This section deals with the validation / loading of the XML configuration file. The right side of the form shows information and progress for any job that is being run.

First, browse to the configuration file that you have just amended by using the 'Browse' button, and hit the load button. If all is well with the configuration, you will get a 'Configuration Loaded. Result Code: 1' message, a result code of 1 meaning that the executed method was successful.

If the load was unsuccessful, a result code of -2 will be returned (meaning 'Not configured'), along with an error message explaining what the problem is. Common errors are discussed in the 'Troubleshooting' section below.

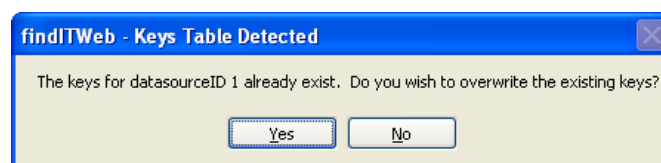
The 'Disable' button will disable the web service, and make it unavailable to users. Commonly this button will be used when a change is going to be made to the XML configuration file, which will then be reloaded to reactivate the Web Service.

The 'Delete DBOs' (Meaning Delete Database Objects) button will delete all database objects created by findIT S2 (tables and triggers) from all databases in the current configuration, as well as disabling the web service. Typically this would be used to clean a CRM database of any objects created by findIT S2 before an upgrade, as to not corrupt any upgrading process to the database.

Generate Keys

Once the configuration is loaded the next step, generate keys, needs to be carried out if the keys do not already exist (for example, if this is the first time findIT S2 has been used with the datasource in question). Simply click the 'Generate Keys' button to execute the method.

If the keys table for the datasource in question that you have specified does indeed already exist, you will be prompted with a message box like the one below



By clicking yes, the keys will be re-generated for the datasource. Clicking no will skip the key generation for the datasource in question.

During the key creation process, the progress bar above the list box indicates how far into the key generation the program is, as well as counter in the list box actually stating how many records have been processed, and how many there are to be processed in total. Once the actual key generation part of the process is complete, the progress bar and list box will give indications of the Indexing progress, which happens immediately after. Also during the process, the 'Cancel Generate Keys' button will become active, which can be used at any time during the process to cancel the process.

Just to note, the process is a bulk process, using SQL Server's bulk append functionality, because of this there will be a small delay at the end of the processing while SQL Server loads the information back into the table.

On success, the message 'Key Generation Complete. Result Code: 1' will be displayed. Otherwise, a result code of -2 will be displayed along with the accompanying error message. Common errors for this process are discussed in the 'Troubleshooting' section below and additional information will be displayed in the event log, as previously mentioned. If the generation for any data source was aborted by clicking 'No' on the dialog box displayed above, then this will also be mentioned in the summary message at the end of the process.

Note – As is covered in the AdminBulkGenerateKeys web method description in this document, the generate keys process is both multithreaded and monitored for errors. If the key generation for a data source fails after being kicked off from the Admin Program, the user will be shown a message indicating why the process failed, and indicate to them that the key generation will be completed when re-initiated automatically through a call to PerformPendingKeyUpdates() in the Keys Synchronisation Windows Service.

Rebuild Indexes

If any new search key combinations are defined in the configuration file, the file should be reloaded and the 'Rebuild Indexes' button should be clicked to build the indexes relevant to the new search key combinations. This process can also be cancelled at any time by clicking the 'Cancel Rebuild Indexes' button.

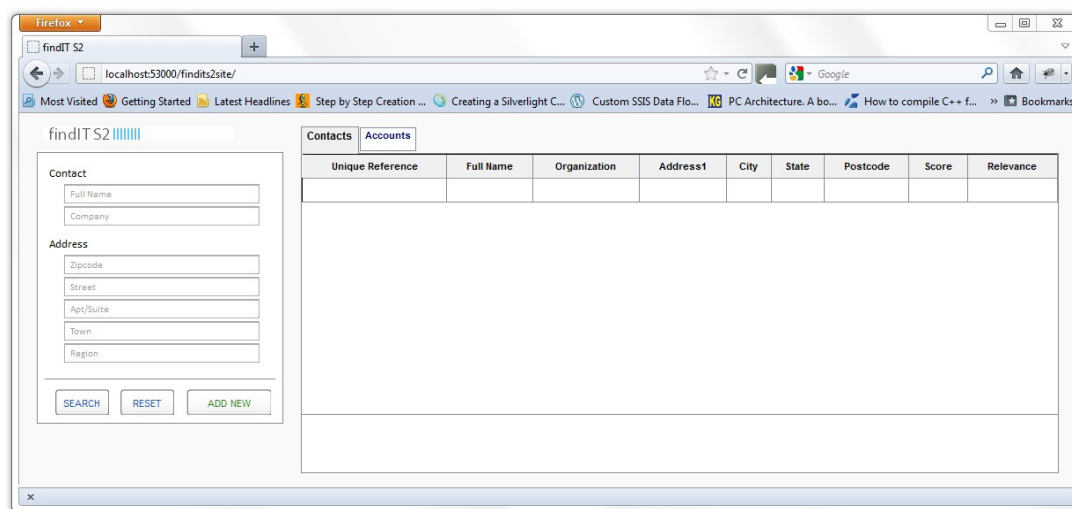
Using the findIT S2 Interface

Now that the XML configuration has been loaded, it will be possible to perform searches on the data that you have configured the XML to read. You will need to open the findIT S2 interface, you can do this in a few ways.

In your browser you can navigate to the following location to open the page :

<http://{your domain}:{port number}/findITS2Site>

There is also shortcut from the Windows Start menu to the findIT S2 Test Page, or you can browse the site from within the IIS manager.



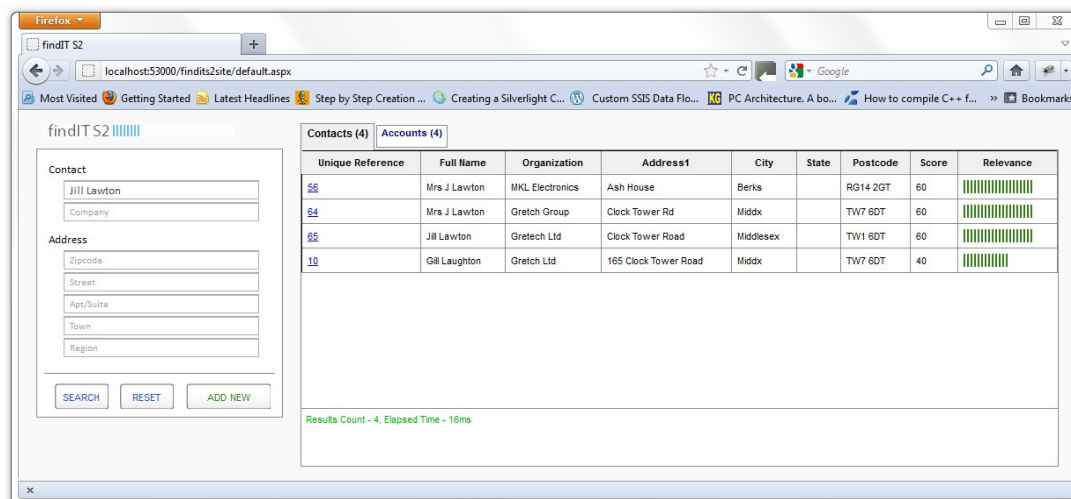
The findIT S2 interface is split in three main sections, which allows data entry, duplicate look up and data cleansing functions all in the data capture interface.

- **Search Pane** : A fully customizable set of data entry field, with buttons to Search for records, Reset the form or (in the case of CRM integration) add the keyed record to into the system.
- **Results Grid** : Again a customizable view of the matching candidates which is updated according to the fields populated in the Search Pane.
- **The Information/Warning Pane** : This returns information regarding your search, the number of matching records as well as the response time.

The operation of the findIT S2 interface can be slightly different depending on what components of findIT S2 you have installed on your server.

If you selected to AJAX enable the UI during installation, as you tab out of fields of data, the results are returned in real time, both for matching and US address correction. The AJAX functionality can be switched of if desired in the ControlConfig.xml file, which is addressed in a later section.

To test findIT S2, enter a name you know exists in the database, and if AJAX is on, simply tab out of the field, or alternatively hit the 'Submit' button. Provided the data exists, you should be presented with a set of results in a table next to the input fields, as follows



Some information on the lookup is displayed underneath the results grid, describing how many results were returned and the time it took to complete the process.

The search can be refined by typing in more information into the input fields and re-submitting the form or tabbing out of the field. Doing so will have an effect on the results that are returned and their relevance to the data entered, which can be seen in the last column's graphical relevance meter.

In the results pane it is possible to tab between different results grids that are user definable in the **ControlConfig.xml** file. As will be discussed later, multiple results grids can be configured to display results from different data sources that have been defined in the findIT S2 Service configuration file - **findITConfiguration.xml**.



Clicking on the appropriate tab will allow you to see the results for each data source. In the example, we have *contacts* currently selected (that has retrieved 4 results from the search data entered, denoted by the value in the parentheses), however if a user would like to see the results returned for the accounts data instead, they would simply click on the *Accounts* data source tab.

The columns displayed in each results grid are completely configurable in ControlConfig.xml, so a search against the contacts can have result fields displayed if compared to the Accounts tab. The configuration of these results grids and the rest of the UI is covered in the section **Configuring the findIT S2 Interface**.

Using US address correction

The US correction is seamlessly integrated into the data entry fields in the search pane by default. The operation of this functionality is designed to be as smooth as possible, so the design and order of these fields help with the process.

There are in fact, two checks for address, the as you type address correction, which will prompt you with possible values and auto-fill where it can, and also a recursive check prior to entry into the database, which will give you any last suggestions and allow the user to pick what they typed, or the possible corrected value.

To use the as you type address correction, tab into the ZIP field, this is where the and key in a five digit ZIP code then tab out of the field. The ZIP code needs to be the first field captured, because this initiates a search of the USPS database to cache candidate addresses which will help us auto-complete the rest of the address.

The address correction service will take this information and determine if there is a city and state it can return for this ZIP code.

Address

95128
Street
Apt/Suite
San Jose
CA

In some cases, the ZIP code may not be valid, or it cannot be resolved down to just one city or state.

Where this is the case, you will see a warning sign which indicate that at this point, the US address correction service can't complete the indicated field at this time. This is not a problem, just tab to the street field and continue typing.

Address

90027
Street
Apt/Suite
City 
CA

The street field features auto-suggest to aid in completing the first line of address. As you type in the street line and get to the fifth character, a drop down will appear to guide you through the recognized values.

Address

95128
400 p
400 Patch Ave
400 Patton Ave
400 Pamlar Ave
CA

You can select a value by using the arrow keys to navigate to the correct value, then press enter or tab to select the value. As a note, the first line of address check will check against the CASS database to ensure that it lies within an accepted range for the selected street. In the example above, 400 Patch Ave is valid, however 600 Patch Ave is not in an accepted range, so as you type, you will not be given any choices to select, and you will need to complete the field manually.

Once you have completed the street line, a few different things can happen, according to what has been keyed.

You will see the ZIP code updated to feature a ZIP+4 code and no further action is required.

You can still type a suite or apartment number in if you want to, however the address is deliverable without this additional information, it doesn't explicitly

You will see the ZIP code updated with a ZIP+4 code, however you will also see a warning sign against the Apt/Suite field.

What this indicates is that the current address is set to a default ZIP+4 code for that address, which is deliverable, however, you need to add in details for a sub-premise if the address is to be an explicit delivery point. The default ZIP+4 code is usually assigned to properties where there are multiple units or apartments.

By adding in the required sub premise information, you will be able to find a specific address delivery point, the ZIP+4 will be updated and the warning will disappear. To indicate the sub premise information is valid, a green tick will appear beside it.

For records where the first line of address was not recognized as deliverable, no ZIP+4 code will be assigned and there is no prompt for a sub premise. The US address correction service has noted there is already a problem rationalizing the address to a delivery range, so it can't offer any further information. The Street line of the address will feature the warning icon and the user can still go ahead and enter an apartment or suite number, however this wasn't verified.


need an apartment or suite number for this address to be a deliverable address.

Address

95128-2144
400 Patch Ave
San Jose
CA


Address

95128-2560
560 S Winchester Blvd
San Jose
CA




Address

95128-2500
560 S Winchester Blvd
500
San Jose
CA



Address

10044
500 Main Street
New York
NY



Using UK and ROW address correction

The UK and ROW address correction work in a very similar manner. For a quick start guide on installing and getting up and running with the Addressing Web Service, refer to Appendix D.

For the sake of demonstration we will use a UK address. On first load, the address fields will look as follows.

The quick entry area for Address Lookup is surrounded by a grey border. Once an address is entered in to the address lookup text box, and depending on whether or not Ajax is enabled for the region selected in the drop down, a lookup will be performed by either tabbing out of the text box, or clicking the verify lookup tick button.

In the case of UK addressing, if a search is submitted that cannot be verified exactly from the information entered, and ambiguity list will appear and a selection will need to be made. This is shown below

Once selected, the verified address will be transferred to the address input fields as per the configuration (described in the next section). A result code description is also shown in the results pane (not shown here).

As mentioned above, ROW addressing works in exactly the same way, apart from ambiguities – It is activated (assuming it has been enabled) by selecting an ROW country in the country drop down. For example, selecting France and looking up a French address would cause the UI to switch to using the ROW service. Different countries have different address formats and different casing rules, which are both configurable for each country, described in the next section.

International Address Verification WCF Interface

As well as the SOAP based addressing service for UK and ROW address correction, there is also a strongly typed WCF based interface available. Integration into certain CRM systems may be better suited to this interface. The WCF based service can be referenced on the following URL –

http://{your local domain}/findITS2AddressingService/InternationalAddressVerifyService.svc

Address Formatting

For a fuller explanation on this interface and its types / methods, please refer to Appendix F.

It is possible to specify custom formatting for the address elements that are returned as part of a Process Result Item. This is controlled through xml configuration files that, by default, are configured in the Web.config file of the service to exist in the 'Formatting' sub folder in the service installation directory. To apply a custom formatting configuration to a web service result, you need to pass the name of the configuration file (without the .xml file extension) in the 'FIS2AddressLineFormatConfig' parameter of the Process Options argument. Each Process Result Item returned from the service will contain information about the custom formatting that was applied to the result in the 'FIS2FormattingInfo' property.

The structure of an element in an address line formatting configuration file is as follows (A default example configuration is installed to the Formatting directory with S2, so you can also refer to that) –

```
<formattingConfiguration>
  <countries>United Kingdom|Default</countries>
  <mappings>
    <address1>{BUILDINGNAME} {PREMISENUMBER} {STREET}</address1>
    <address2>{LOCALITY}</address2>
    <address3>{ADMINISTRATIVEAREA}</address3>
    <address4>{POSTCODE}</address4>
  </mappings>
</formattingConfiguration>
```

Note that it is possible to specify a pipe delimited list of countries to save repetitive definitions. Also note that there is a special case 'Default', which is used if the country of the address in question is not defined in the configuration. If no 'Default' is defined, then the standard formatting provided by the addressing engine will be used, as will be the case if no custom formatting option is provided in the call to the service.

A complete list of the address fields that can be configured is included in the example formatting file. By default, if a custom formatting configuration is used to format a returned address, any elements that are not defined in the configuration are returned blank.

Character Translation

It is possible to enable character translation within address elements that are returned as part of a Process Result Item. This is controlled through an xml configuration file that, by default, is called CharacterTranslationConfiguration.xml and is located in the Translation sub folder. The name and location of this file is configured in the Web.config file, which can be changed if necessary. To activate character translation, the XML node 'active' must be changed to 'true'. To add or remove characters to translate and the characters that they must be translated to, add or remove 'translation' nodes within the translations area.

By default, the XML file contains a list of nodes that will translate certain foreign characters to their English equivalents.

The format of the XML file is as follows. Changes made to the XML file will not take effect until the findIT S2 service is restarted.

```
<config>
  <active>>false</active>
  <translations>
    <translation in="à" out="a" />
    <translation in="ê" out="e" />
    <translation in="ì" out="i" />
    <translation in="ò" out="o" />
    <translation in="ù" out="u" />
    <translation in="À" out="A" />
    <translation in="È" out="E" />
    <translation in="Ì" out="I" />
    <translation in="Ò" out="O" />
  </translations>
</config>
```

Address Service Usage Logging

With the WCF Interface, it is also possible to monitor the usage of the service through log files. By default, logging is disabled – the configuration of the logging is controlled through an xml file that is pointed to in the Web.config of the addressing service, which by default is **~/Logging/LoggingConfiguration.xml**.

The configuration file itself is commented to give a description of each setting (all of which should be self-explanatory), complete with a listing of available fields to use where appropriate, however below is a further description of each setting.

enabled – This setting simply determines whether the logging is on or off. By default logging is disabled.

logFilePath – This setting is used to specify the path to use for the log file. It is blank by default, which will cause the file to be written to the common application data folder within the sub folder structure findIT S2/Logs. You can use this setting to specify a custom path to write the file to (note the path should include the file name). In doing this, it is important to ensure that the directory in question has appropriate permissions on it to allow the user that the service is running under to write to it.

maxQueueSize – This is the max number of service call logs to queue up in memory before writing to disk. By default this is set to 1000, however this can be increased / decreased as necessary. There will be a performance overhead in processing the in-memory queue, which is processed on a call to the service that causes the queue limit to be hit, so this needs to be considered when setting the value.

The following three settings are related to the archiving of the log files. To save disk space, the text based log file will be archived to a compressed zip file according to the configuration settings below. There is a background thread that runs every 60 seconds when the service is active to verify the these settings against the current state of the log file, and take the appropriate action.

logArchiveTimeSpan – This setting is used to specify the minimum time span that should have elapsed between the first call recorded in the active log file and the current time. It is possible to set individually the number of days, hours, minutes and seconds to elapse

between archiving each log. Whether or not the file gets archived due to this setting being satisfied is also dependent on the following settings, `lowerLogFileSizeLimit`.

lowerLogFileSizeLimit – This setting is used to define the minimum size (in bytes), that the log file must be in order to be archived. If this file size is not achieved, then the file will not be archived, regardless of any other archive setting.

upperLogFileSizeLimit – This setting is used to specify the maximum size that a log file can grow to before being automatically archived. If when checked the log file has exceeded this size, the file will be archived, regardless of any other setting.

inputFields – This setting is simply used to specify the input fields to store in the log that were submitted to the service call. An exhaustive list is included in the default logging configuration file.

outputFields – This setting is simply used to specify the output fields to store in the log that were returned to the client from a service call. An exhaustive list is included in the default logging configuration file.

The structure of the log file is set out so that each call to the service is logged line by line. Each line is tab delimited, and will contain information such as the time of the call, the IP address of the calling client, the windows user context that the call was executed under and the input / output fields that were specified to be output in the logging configuration file.

Configuring UK and ROW Addressing Formats

With the range of different formatting and casing rules for different countries, it is necessary to allow things to be configurable at a country level. The following 2 files allow that to be possible.

AddressFormatConfig.xml

This file controls the position that elements of a corrected verified address are placed in on the form. It is found in the config directory, and referenced in the Web.config file of the findIT S2 website. The following is an example of the configuration for UK addresses –

```
<formattingConfiguration>
  <country>United Kingdom</country>
  <mappings>
    <Address1>{BUILDINGNAME} {BUILDINGNUMBER} {STREET}</Address1>
    <FlatNo>{SUBBUILDING}</FlatNo>
    <Town>{LOCALITY} {POSTTOWN}</Town>
    <Region>{COUNTY}</Region>
    <Postcode>{POSTCODE}</Postcode>
  </mappings>
</formattingConfiguration>
```

The country node specifies the country that this format applies to – this needs to match the name of the country in the drop down in the UI. The mapping nodes specify which elements in the verified address map to which fields on the form. The name of the node needs to match the field name it is intended for. As can be seen above, the corrected building name, building number and street are all mapped, separated by a space, to the address line 1 field on the form.

There is also an entry in the configuration file for a country labelled 'Default' – This is the format that will be used if an entry for a particular country does not exist.

CasingConfiguration.xml

This file controls the casing for each element in a verified address. It is found in the root directory of the findITS2AddressingService, and is referenced in its Web.config. The following is an example of the casing format for a UK address -

```
<casingConfiguration>
  <country>United Kingdom</country>
  <enabled>True</enabled>
  <casingOptions>
    <street>Proper</street>
    <depstreet>Proper</depstreet>
    <depllocality>Proper</depllocality>
    <locality>Proper</locality>
    <posttown>Proper</posttown>
    <county>Proper</county>
    <postcode>Upper</postcode>
  </casingOptions>
</casingConfiguration>
```

The country node specifies the country that the casing applies to, and the enabled node is used to switch the configuration on or off. The casingOptions sub nodes need be named according to the corrected element that they apply to – Available options are Upper, Proper and Lower. There is also a default settings entry.

Configuring the findIT S2 Interface

The **ControlConfig.xml** file, by default found in the 'config' sub folder of the findIT S2 installation, controls the content of the findIT S2 Web UI. Most elements are configurable, from cosmetic placement of controls to the fields of information you need to enter or return when considering matches. Each section of the ControlConfig.xml file, and what it controls, is described in detail below.

New Entry Settings

In the case of an install for MS Dynamics integration, the first section of the Control Config file is the `<newentrysettings>` node. This controls whether or not the 'Add New' button is displayed on the findIT S2 Web UI form (which when clicked will bring up the 'Add New' Dialog). The button can be enabled / disabled by setting the `enabled="true"` attribute accordingly. It is also possible with this node to specify which actions are available (as radio buttons) on the 'Add New' Dialog, namely Add as Contact, Lead or Account. There are again enabled / disabled in the same way. The `<addascontact>` action is a special case in that it is possible to specify a parent account to link to when adding a contact, so it is necessary to specify the data source set up in the findIT S2 Service that contains this data. By default, this is set to `parentdatasourceid="3"`.

Button Shortcuts

The next node in the ControlConfig.xml file is the `<buttonshortcuts>` node. Here it is possible to specify what keys execute the different buttons / controls on the findIT S2 form when holding down the 'Alt' key on the keyboard. These can be amended to suit the user.

Modules

The `<modules>` section of the configuration determines what findIT S2 UI module are enabled. It is possible to enable / disable the `<addressing>` modules (explained further on), as well as determine whether AJAX is enabled / disabled for both the `<matching>` and `<addressing>` modules.

Unique Fields

This node is used to define text boxes that can be used to do unique searches on particular fields of particular data sources. The `<uniquefield>` tags can contain a collection of sub `<ds>` tags that define the mappings of the unique field for specific data sources –

```
<uniquefields>
  <uniquefield name="ID" label="ID" enabled="true">
    <ds id="1" table="contacts" column="id" />
  </uniquefield>
</uniquefields>
```

In the example above there is a single mapping for the unique field in question for data source '1' – The field to query is the 'ID' column in the 'Contacts' table. You can have multiple unique field lookups.

Contact and Address Input Fields

The next 2 sections of the configuration file, `<contactinputfields>` and `<addressinputfields>`, control the text boxes on the form used to input search data. Both the contact and address input field nodes share the following common parameters :

- `name` – This is the reference name for the particular control, these are mapped to specific matchIT fields and should not be changed.
- `label` – This is what appears beside the input box or as a watermark inside the box itself. This describes what data element is being captured to the user.
- `enabled` – This controls if the box is actively displayed on the screen.
- `interactive` – This allows you to turn on or off the interactive functionality outlined in the modules section independently. The modules section has priority, so this parameter allows you to turn off interactive lookup if it was actually set to true in the relevant modules section.

For the address input field node, there is an additional parameter

- `addressfield` – This is how we map the address fields to the US address correction service, these will be set up for US address correction users by default.

The `<addressinputfields>` section also contains another type of node, the `<addressingserviceselector>` node, which is described in more detail in the Addressing Modules section.

Results Grid

Within the `<results>` node it is possible to specify multiple `<resultsgrid>` nodes, each of which correspond to a tab in the results pane of the findIT S2 UI. A `<resultsgrid>` node takes the following parameters –

- `datasource` – This attribute can be used to specify a particular data source for this form to query. If left blank, all data sources will be queried.
- `title` – This determines how the data set tab in the findIT S2 screen is labelled.
- `tabcolor` – This determines the colour of the text on the grid tab.
- `enabled` – True/false, you can choose to hide a tab you have set up as required.

The only other type of node in the `<resultsgrid>` section is the `<gridfield>` node. This node is used to define a results column in the given results grid. All `<gridfield>` nodes share the following common attributes -

- `name` – This is the name of the field you wish to display in the column. It should match the name of the XML attribute that is returned from the FindMatches() Web service method. If a field is specified that doesn't exist, a blank value will appear.
- `type` – This determines the type of column that is displayed. The types available are `FIWResultsGridColumn` (a standard column), `FIWLinkedResultsGridColumn` (a hyper-linked column) and `FIWRelevanceMeterGridColumn` (a column that displays a relevance meter).
- `header` – This is the title that will appear at the top of the column.

- `width` – This controls the width of the particular column, by default setting it to 0 will auto-size the column width according to the data displayed.
- `enabled` – True/False, display or hide the configured column.

When using a type `FIWLinkedResultsGridColumn` column, it is necessary to also specify the following attribute -

- `LinkUrl` – This is the value that is used for the 'href' of the rendered 'a' tag. It is possible to specify a JavaScript call as well as a standard URL. It is also possible to use the field value of the column in the URL itself, by using the {0} place holder, as per the following - `"JavaScript:GoToDetail('contact','{0}')`"

Output Fields

The final section in the config file, the `<outputfields>` node, contains definitions for hidden fields on the form that contain cleaned and normalised data relating to the search data entered into the form. This cleaned data is produced by taking the input data and processing it with the matchIT API (through the findIT S2 Web Service), and US Addressing Web Service if installed. The cleaned fields are then used, in the case of a Microsoft Dynamics Implementation, to be entered into a system. The following attributes should be specified for an `<outputfield>` node.

- `name` - The name of the specific mapping against the findIT S2 information. By default a complete list has been added to ControlConfig.xml.
- `label` – The label property is for internal use, and has no functionality relevant to the user.
- `enabled` – This property specified whether or not to output the hidden field to the page.
- `type` – Internal mark up to designate where the data element is returned from, these should remain unchanged.

Addressing Modules

The addressing modules, although part of the modules section, require an explanation on their own as the subject is quite in depth. There are currently 3 modules on offer – US addressing, UK addressing and ROW addressing, each with their own configuration node.

For US addressing (only available to US customers) there are simply 2 boolean settings, `enabled` and `interactive`, that specify whether the addressing module is turned on, and whether the fields are Ajax Enabled.

For both UK and ROW Addressing, you can enable / disable addressing through the `enabled` attribute. Both the `<ukaddressing>` and `<rowaddressing>` nodes contain the same following sub node –

```
<credentials dataset="" ipaddress="" account="" client="" />
```

In the case of UK addressing, the `dataset` and `ipaddress` attributes should not need to be changed from their default values of PAF and localhost respectively – the `account` and `client` attributes are not used in this case.

In the case of ROW addressing, the `dataset` and `ipaddress` attributes should again not need to be changed from their default values of WORLD ADDRESSING and blank, however in this instance you will need to fill in the relevant `account` and `client` attributes, encrypted using the findIT S2 Administration Program – If you provided these details on installation, they should already be populated.

At this point it should be noted that, in the case of UK and ROW addressing, it is necessary to enable the `<addressingserviceselector>` node in the `<addressinputfields>` section. This node enables a control on the form with a country drop down and lookup button necessary to perform the address validation. The `<addressingserviceselector>` node has the following attributes –

- `matchitfield` – This should not be changed, as it is used by the service to indicate that it is of type Country.
- `enabled` – Specifies whether or not the control is enabled or hidden.
- `defaultcountry` – This property sets the country in the drop down list that will appear at the top, and will be defaulted to.
- `enabledefaultcountrycookie` – This property can be used to specify that, whenever a selection is made in the drop down, the selected country becomes the new default, and overrides the `defaultcountry` attribute. This value is persisted in a cookie and preserved between visits to the site.
- `enablebydefault` – This specifies whether or not the lookup button is enabled by default to begin with, so that if any issue occurs on the form, the control can still be used to perform address validation.

The findIT S2 Web Service

Both the findIT S2 Administration Program and Web Site make use of the web methods that the findIT S2 service has to offer. You can view the methods that are available by browsing to the web service directly on <http://{your local domain}/findITS2Service/Service1.aspx>. You will be presented with the following screen, which displays a list of the web methods available, which will be described in detail next.

Service1

The following operations are supported. For a formal definition, please review the [Service Description](#).

- [AdminBulkGenerateKeys](#)
- [CancelAdminBulkGenerateKeys](#)
- [CancelIndexKeysTable](#)
- [CompareRecords](#)
- [ConfigurationErrorList](#)
- [DeleteAllDatabaseObjects](#)
- [DeleteKeys](#)
- [EnableService](#)
- [FindMultiRecords](#)
- [FindRecords](#)
- [GetBulkGenerateProgress](#)
- [GetDatasourcesWithKeys](#)
- [GetIndexingProgress](#)
- [GetLastError](#)
- [GetOutputFields](#)
- [IndexKeysTable](#)
- [InsertKeys](#)
- [IsServiceEnabled](#)
- [LoadConfiguration](#)
- [PerformPendingKeyUpdates](#)
- [ReturnKeys](#)
- [UpdateKeys](#)
- [ValidateAdmin](#)

AdminBulkGenerateKeys (List of Data source IDs, Encrypted Admin Password)

This is the method that is used by the administration program when clicking the 'Generate Keys' button. It accepts a list of IDs that are IDs of data sources that do NOT need their keys generating. The method will therefore generate the keys for all data sources specified in the configuration other than the ones specified in the argument passed to the AdminBulkGenerateKeys() method.

There are some important points to note about this method. The first is that the process is multithreaded to provide the best performance. By default, the number of threads to use is set to 4, which is controlled through the 'KeyGenMaxThreads' app setting in the Web.config

file of the web service. If this setting is removed or left blank, this method will automatically use the same number of threads as there are processors on the hardware. The second point to mention is that, for the purpose of stability, this method keeps a log of its progress in the FIS2_Log table for the data source in question. If for some reason this method errors and terminates, then the last logged progress is used to re-initiate the key generation process when calling PerformPendingKeyUpdates(), which is discussed in more detail below. PerformPendingKeyUpdates() is actually called periodically by the Keys Synchronisation Windows Service, so there are regular checks for the event of a key generation failure.

CancelAdminBulkGenerateKeys()

This is the method that is used by the administration program when clicking the 'Cancel Generate Keys' button. It will cause the AdminBulkGenerateKeys() method to terminate if it is running.

CancelIndexKeysTable()

This is the method that is used by the administration program when clicking the 'Cancel Rebuild Indexes' button. It will cause the CreateCompoundIndexes() method to terminate if it is running.

CompareRecords()

This method can be used when wanting to compare records in a batch fashion. Records are submitted to the method as an XML string in the following format -

```
<records>
  <record uniqueref="1" fullname="John Smith" />
  <record uniqueref="2" fullname="Jonathan Smith" />
  <record uniqueref="3" fullname="Johan Smithson" />
</records>
```

These records will then be compared to each other, with each pair being compared exactly once (i.e. in the above example 1 vs 2, 1 vs 3 and 2 vs 3). The results returned depend on the batch datasource settings in the findITConfiguration.xml file (described in the web service configuration section). An example is shown below.

```
<results>
  <comparison record1="1" record2="2" totalscore="116.4" maxscore="120"
  minscore="84" namescore="116.4" />
  <comparison record1="1" record2="3" totalscore="99.6" maxscore="120" minscore="84"
  namescore="99.6" />
</results>
```

Note that each comparison node refers to a comparison result (which is only returned if the threshold score is reached) and contains the unique refs of each record, plus a breakdown of the score components for the match (which again is configurable in the batch datasource settings in the configuration file).

ConfigurationErrorList()

This method simply returns a list of any current errors with the configuration, that typically occur on trying to load the XML configuration file.

DeleteAllDatabaseObjects()

This is the method that is used by the administration program when clicking the 'Delete DBOs' button. It will delete any database objects created by findIT S2 in all databases in the current configuration.

DeleteKeys(Unique record reference, Data source ID)

This method deletes the entry in the keys table for the specified record in the specified data source. The method will check to see if the keys exist, returning an integer result code of -5 (keys don't exist) if they do not exist, or on success, an integer result code of 1 (method succeeded) is returned.

EnableService(Boolean status)

This method sets the status of the configuration class to whatever is provided as the argument. It is used in the administration program to disable the configuration status when clicking the 'Disable' button. Typically, subsequent calls to web methods of the web service will re-enable the configuration providing the configuration xml file has not changed – To prevent this from happening the configuration file either needs to change or the configuration file path / hash values stored under the findIT S2 registry key need to be blanked (The latter is done when clicking the disable button in the administration program).

FindMultiRecords(XML Search String)

This method is essentially the same as the FindRecords() method described in more detail below. The main difference is that no specific data source has to be specified in the query sent to FindMultiRecords - all data sources are queried concurrently. However, if the user wishes to only query a select number of data sources, it is possible to specify a specific data source attribute (shown below) with a comma separated list of data sources to query.

FindRecords(XML Search String)

This is the method that the sample website uses to query the data source(s) and retrieve matching results to what was input. The method accepts an XML string in the following format

```
<contact specificdatasource="" fullname="" company="" address1="" town=""
region="" postcode="" />
```

The attributes in the node above are examples of the fields that can be used – In fact, any field that can be mapped in the findITConfiguration.xml file can be submitted as an attribute to be used in a query to the web service. As an example, if a search on fullname only was being done, the search string would look like the following –

```
<contact specificdatasource="" fullname="" />
```

The 'specificdatasource' attribute is used to specify a single particular datasource to use for the query. For example, if you had three data sources set up, with IDs of 1, 2 and 3 respectively, but only want to query datasource 2, you would set the attribute specificdatasource="2" in your input string. In the FindRecords Method, a specific data source must always be specified. To query all data sources (or a specific range) use the FindMultiRecords() method.

Another two attributes which go hand in hand that can be included in the input string are 'uniqueref' and 'uniquerefcolumns'. The former contains a unique value to search on and the

latter contains the datasources/columns to search. The `uniqueid` value should be in the format `datasourceID/table/column:datasourceID/table/column:datasourceID/table/column` etc, where `datasourceID` is the ID of the datasource to query, `table` is the name of the table that the column exists in, and `column` is the name of the column to query. Note that multiple data sources can be specified by using a comma separator. If the `uniqueid` attribute is supplied without the `uniqueidcolumns` attribute, a result code of -6 (InsufficientUniqueColumnInformation) will be returned in the XML results, described below.

If no attributes are specified, the result will be a warning that no appropriate search key could be determined for the search. The results themselves are returned as XML also, in the following format

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<results>
  <resultCode>(integer result code)</resultCode>
  <data>(XML input string)</data>
  <searchinputfields>(search inputs used)</searchinputfields>
  <dataProcessed>(Processed data nodes)</dataProcessed>
  <count>(Number of results returned)</count>
  <matches>
    (Match Nodes)
  </matches>
  <elapse>(Total process time (ms))</elapse>
</results>
```

A match node (nodes that exist between the 'matches' node) contains data for a match, in the following format

```
<match datasourceID="" uniqueid="" fullname="" address1="" town="" region=""
postcode="" score="" />
```

Where `datasourceID` is the ID of the datasource that the match exists in, `id` is the unique id of the record in the main table of data, `fullname` is the full name of the record, `address1` is the first address line, `town` is town, `region` is region, `postcode` is postcode and `score` is the matching score that the match has achieved.

A processed data node (that exists between) the 'dataProcessed' node) is a node that contains cleaned / normalised elements generated from the input data for the query. The following is an example of a Salutation Generated for a search on `fullname="John Smith"`

```
<pElement name="Salutation" value="Dear Mr Smith" />
```

The search input fields node contains the names of the inputs that were used for this specific query to determine which match keys to use.

Note – More or less attributes can be specified to be returned in the `<match>` xml node by amending the 'outputfields' node in the web service XML configuration file. This is described in section 4 – 'Amending the XML file'.

If an error occurs during the process, the XML is returned in the following format

```
<?xml version="1.0" encoding="iso-8859-1"?>
<results>
  <resultCode>(integer result code)</resultCode>
  <data>(XML input string)</data>
  <searchinputfields>(search inputs used)</searchinputfields>
  <count>(Number of results returned)</count>
  <warning>(Warning message)</warning>
  <matches>
    (Match Nodes)
  </matches>
  <elapse>(Total process time (ms))</elapse>
</results>
```

Which as you can see is in exactly the same format as that of a successful search, only with an extra 'warnings' node containing the error / warning message. You can test this method

and view the XML output by clicking on the link for the method and submitting an XML string in the form `<contact fullname="" />` with the full name of somebody that you know exists in the database. Note – in order to get a successful result, you will need to make sure the configuration is loaded, which can be done with the `LoadConfiguration()` method described below.

GetBulkGenerateProgress()

This method is called at regular intervals during the Generate Keys process in the administration program to retrieve two variables – the total number of records to be processed, and the total number of records processed so far (so 2 integers). The two integers retrieved are used to display output to the job list box and to give a relevant value to the progress bar.

GetDatasourcesWithKeys()

This method returns a list of the datasources in the configuration that have populated keys tables in them. This is used by the Administration Program when clicking on the 'Generate Keys' button to get a list of the data sources with keys and prompt the user with a message box asking if they want to overwrite the keys for each datasource (during which process a list of data sources to NOT generate keys for is made, and passed to the `AdminBulkGenerateKeys()` method described above).

GetIndexingProgress()

This method is called at regular intervals during the Create Compound Indexes process in the administration program to retrieve two variables – the total number of indexes to be created, and the total number of indexes processed so far (so two integers). The two integers retrieved are used to display output to the job list box and to give a relevant value to the progress bar.

GetLastError()

This method returns a string in the form of the last error that occurred in the Web service, specifically with the `AdminBulkGenerateKeys()` method. It is used in the Administration program to display the error on screen in the event of a failure during the 'Generate Keys' process.

GetOutputFields()

This method returns a list of fields (or xml attributes) that are available in each result returned from the `FindRecords()` method.

IndexKeysTable(Encrypted Admin Password)

This method creates indexes on the keys table in each data source in the configuration from the search key combinations that are described in the configuration for each datasource. For example, if a search key is defined to use 'mkName1' and 'mkPostOut', a compound index will be created in the keys table using both of these columns, plus individual indexes for each too.

InsertKeys(Unique record reference, Data source ID)

This method inserts the entry in the keys table for the specified record in the specified data source. The method will check to see if the keys exist, returning an integer result code of -4 (keys already exist) if they do exist. On success, an integer result code of 1 (method succeeded) is returned.

IsServiceEnabled()

This method simply returns a Boolean (true or false) value reflecting the status of the configuration – If the configuration is active (true) then the result is 'true', otherwise the result is false. This method can be used to check if the configuration is available before calling other methods such as FindRecords().

LoadConfiguration(File path to XML Configuration File)

This function loads / instantiates the configuration class from an XML configuration file located from the path passed as an argument to the function. It returns a Boolean value, true on success, false on failure. Any errors that occur in the event of a failure can be retrieved using the ConfigurationErrorList() method. Calling this method also deploys all the necessary database objects to each data source to implement dynamic key updates, which are carried out by the findITS2UpdateKeys Windows Service which periodically calls the **PerformPendingKeyUpdates()** method, described below.

PerformPendingKeyUpdates()

This function simply checks for and performs any pending key updates that exist for all data sources loaded in the current configuration. All updates to data being pointed at by the web service are logged in an updates table in each data source. This method reads those tables for entries that may exist, updates any that do and then deletes the entries that it has performed from the updates table.

Another key part of this method is that, before running updates, it runs a check for each data source to see whether a key generation is in progress, or whether a key generation was in progress that had failed. In both these cases, updates will not be performed, and specifically in the case of the latter, it will re-initiate the key generation process for the data source in question on the remaining records that do not have keys generated for them.

ReturnKeys(Unique record reference, Data source ID)

This method returns the keys for the specified record in the specified data source in the form of an XML string with the keys as attributes.

UpdateKeys(Unique record reference, Data source ID)

This method updates the entry in the keys table for the specified record in the specified data source. The method will check to see if the keys exist, returning an integer result code of -5 (keys don't exist) if they do not exist, or on success, an integer result code of 1 (method succeeded) is returned.

ValidateAdmin(Encrypted Admin Password)

This method checks that the user has the correct level of administrative privileges to open up the findIT S2 Admin tool. To use a password, you will need to enter a password of your choice into the findIT S2 admin tool encryption tool and store this in the

C:\{findITS2 Path}\website\findITS2Service\web.config file

Under the node : `<add key="adminPassword" value="" />`

By default this is left blank.

The findIT S2 Service WCF Interface

As well as the traditional SOAP based web service described above, findIT S2 also has a WCF interface offering access to the searching functionality of the product through use of strongly typed objects, rather than the loose schema offered by the string xml inputs described above. Integrations into certain CRM systems may be better suited to this interface.

The name of the Service Interface is the findITS2SearchService.svc. It can be referenced through the following URL

`http://{your local domain}/findITS2Service/findITS2SearchService.svc`

For a fuller description of this interface and its types / methods, please see Appendix E.

The US Addressing Web Service

The US Addressing Web Service is a distinct web service and can be used after installing findIT S2 and the USA address data, however you do not need to load the configuration file using the administrative application prior to getting results.

The US Addressing Web Service uses the local address data resource files, so these will need to be installed on the machine. Also please note for some Windows users there is a problem with accessing the data files. By default the files will be located at :

C:\addressIT_Data

To fix this issue on Windows Server 2008, you will need to add permission for the IIS User running the findIT S2 app pool to the ACMDATA directory in the folder where the US address data was installed.

Web Service Methods

The following methods are exposed within the US addressing web service and explained in the section below.

Service1

The following operations are supported. For a formal definition, please review the [Service Description](#).

- [ConfirmAddress](#)
- [ConfirmAddressXML](#)
- [GetCityState](#)
- [GetCityStateXML](#)
- [GetDPV](#)
- [GetDPVXML](#)
- [ReturnTIGERData](#)
- [Reverse9Lookup](#)
- [TypeDownCityStateXML](#)
- [TypeDownPrimaryRangeXML](#)
- [TypeDownSecondaryRangeXML](#)
- [TypeDownStreetsFromCityStateXML](#)
- [TypeDownStreetsFromZIPXML](#)
- [getTypeDown](#)

ConfirmAddress – This method returns a \$ delimited list of address elements for a supplied address.

Service1

Click [here](#) for a complete list of operations.

ConfirmAddress

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
zipcode:	<input type="text" value="95118"/>
address1:	<input type="text" value="5020 russo"/>
secondpremise:	<input type="text"/>
city:	<input type="text"/>
state:	<input type="text"/>

This will return the following results string :

```
<string xmlns="http://tempuri.org/">
  Y$95118-4007$Address was DPV confirmed for both primary and (if present)
  secondary numbers$5020 Russo Dr$San Jose$CA$5020$$Russo$Dr$$$$$San
  Jose, CA 95118-4007$Santa Clara$085$4007$0162$A$16$C026$N$
</string>
```

The delimited parts are as follows :

- DPV RESULT CODE
- ZIP CODE WITH +4
- DPV ANSWER DESCRIPTION
- STREET LINE
- CITY
- STATE
- COMPANY
- HOUSE NUMBER
- PRE DIRECTIONAL
- STREET NAME
- SUFFIX
- POST DIRECTIONAL
- POSTAL MAIL BOX
- SECOND STREET LINE
- ADDRESS LEFTOVERS
- SUITE NUMBER/NAME
- SUD
- UNIT NUMBER
- URBANIZATION CODE
- LASTLINE OF ADDRESS
- COUNTY NAME
- COUNTY NUMBER
- ZIP4
- RDI
- LINE OF TRAVEL
- LOT OF TRAVEL DIRECTIONAL
- LACS CODE
- CONGRESSIONAL DISTRICT
- CRRT
- DPV CMRA

ConfirmAddressXML – This method accepts/returns the same data as ConfirmAddress, however it formats the results as an XML string, as follows :

```
<string xmlns="http://tempuri.org/">
  <results>
    <dpv_answer value="Y" />
    <zip value="95118-4007" />
    <dpv_answer_text value="Address was DPV confirmed for both primary and
(if present) secondary numbers" />
    <street value="5020 Russo Dr" />
    <city value="San Jose" />
    <state value="CA" />
    <company value="" />
    <houenum value="5020" />
    <predir value="" />
    <streetname value="Russo" />
    <suffix value="Dr" />
    <postdir value="" />
    <pmb value="" />
    <street2 value="" />
    <leftovers value="" />
    <suite value="" />
    <sud value="" />
    <unitnum value="" />
    <urb value="" />
    <lastline value="San Jose, CA 95118-4007" />
    <countyname value="Santa Clara" />
    <countynum value="085" />
    <zip4 value="4007" />
    <rdi value="" />
    <lot value="0162" />
    <lot_dir value="A" />
    <lacs value="" />
    <congress_dist value="16" />
    <crnt value="C026" />
    <dpv_cmra value="N" />
  </results>
</string>
```

If an error or warning is encountered with the address lookup, an XML string of the following format will be returned by ConfirmAddressXML

```
<string xmlns="http://tempuri.org/">
  <results>
    <error value="7: There are no street name matches in the given ZIP code
or in any geographically-related ZIP code." />
  </results>
</string>
```

A complete list of these error values can be found in Appendix C.

GetCityState – This method returns a \$ delimited list of the city and state for a selected ZIP code.

Examples as follows :

Service1

Click [here](#) for a complete list of operations.

GetCityState

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
zipCode:	<input type="text" value="95128"/>
key:	<input type="text" value="1"/>
<input type="button" value="Invoke"/>	

For a ZIP located wholly in a city or state, you would get the following response

```
<string xmlns="http://tempuri.org/">San Jose$CA</string>
```

For a ZIP located partially in a city, we use the return value, Multiple

```
<string xmlns="http://tempuri.org/">Multiple$NH</string>
```

GetCityStateXML – This method returns an XML string containing a list of possible cities and states for the selected ZIP code. This function has additional functionality, to allow a user to see the actual cities or states returned in full, without use of the keyword 'Multiple'. Here is an example return string for multiple results.

```
<string xmlns="http://tempuri.org/">
  <results>
    <zipCityState zip="03579" city="Errol" state="NH" />
    <zipCityState zip="03579" city="Wentworths Location" state="NH" />
    <zipCityState zip="03579" city="Wntwrths Lctn" state="NH" />
  </results>
</string>
```

GetDPV – This method returns a \$ delimited list of elements to allow a user to establish what the correct Delivery Point Verification code is for a selected address.

Service1

Click [here](#) for a complete list of operations.

GetDPV

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
zipcode:	<input type="text" value="95128"/>
address1:	<input type="text" value="5020 russo"/>
city:	<input type="text" value="san jose"/>
state:	<input type="text"/>
<input type="button" value="Invoke"/>	

```
<string xmlns="http://tempuri.org/">95118-4007$Y$San Jose$CA$5020 Russo
Dr$</string>
```

The output delimitation is as follows :

- ZIP+4 Code
- DPV Correction Result
- City Value
- State Value
- Street Value (N.B. That apartment or sub-premise information is passed and returned in this line)

GetDPVXML – Identical to the above, however the return details are presented as an XML string :

```
<string xmlns="http://tempuri.org/">
  <results>
    <zip4 value="95118-4007" />
    <dpv_answer value="Y" />
    <city value="San Jose" />
    <state value="CA" />
    <street value="5020 Russo Dr" />
  </results>
</string>
```

ReturnTIGERData – TIGER Data is a USPS data set that allows ZIP codes to be matched to geographical location. A user will submit a ZIP+4 code and the additional data files will work out the location of the centroid for this location, and then present longitude and latitude data back in the XML return value.

Service1

Click [here](#) for a complete list of operations.

ReturnTIGERData

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
ZIP:	<input style="width: 90%;" type="text" value="95118-4007"/>

If an invalid ZIP+4 is entered, the lookup will attempt to locate the centroid for just the ZIP code portion, ignoring the +4 element, This will compromise the accuracy somewhat, so it's important if consistent accurate results are required that an well formatted and verified ZIP+4 is supplied to this web method.

The output is as follows :

```
<string xmlns="http://tempuri.org/">
  <results>
    <Count value="2" />
    <Latitude value="+037.257643" />
    <Longitude value="-121.881765" />
    <Block value="1005" />
    <CMSA value="7362" />
    <LatFrom value="37257643" />
    <LatTo value="37258186" />
    <LongFrom value="-121881765" />
    <LongTo value="-121882630" />
    <Plus4 value="2701" />
    <PMSA value="7400" />
    <Side value="82" />
    <Tract value="502910" />
  </results>
</string>
```

count

The number of data points contained at this ZIP+4 location.

latitude

The angular distance north or south from the equator of a point on the earth's surface, measured on the meridian of the point.

longitude

The angular distance east or west on the earth's surface, measured by the angle contained between the meridian of a particular place and some prime meridian, as that of Greenwich, England, and expressed either in degrees or by some corresponding difference in time.

block

Blocks are numbered uniquely within each census tract with a 3-character number that identifies the collection block used in the census and a character block suffix. This character block suffix is often blank.

cmsa

A 4-digit code assigned to areas that consist of primary metropolitan statistical areas.

latFrom

The north/south measurement indicating the beginning point of the address.

latTo

A north/south measurement indicating the ending point of the address.

lonfrom

The east/west measurement indicating the beginning point of the address.

lonTo

The east/west measurement indicating the ending point of the address.

plus4

Describes the last four positions of a ZIP+4 Code. Most delivery addresses are assigned a single ZIP+4 Code. However, large companies may be given a range of ZIP+4 Codes that can be used to route mail to a specific department.

pmsa

A 4-digit code assigned to areas that comprise one or more counties, including a major population nucleus and nearby communities that have a high degree of interaction.

side

Sides indicate what side of the line segment the location occurs at, this is why Lat/Long From and To are returned, to allow identification of the location based on the range of the location. The side is loosely related to the side of a

street or highway the ZIP+4 is located, but translating this to an actual side of a road is unreliable.

tract

Small, locally delineated statistical areas within selected counties, generally having stable boundaries and, when first established by local communities, designed to have relatively homogeneous demographic characteristics.

Reverse9Lookup – We have added the ability to look up an address based on just the ZIP+4 details in light of customer feedback.

Service1

Click [here](#) for a complete list of operations.

Reverse9Lookup

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
fullZIP:	<input type="text" value="951284007"/>

This function is independent from the findIT S2 interface, as there are some USPS restrictions on how we can return data from this service. We appreciate however that this web method may be useful in certain situations.

By entering a full ZIP+4 code you get the following XML return string

```
<string xmlns="http://tempuri.org/">
  <results>
    <zip value="95128" />
    <zipPlus4 value="4007" />
    <city value="San Jose" />
    <state value="CA" />
    <company value="" />
    <street value="11?? Greenbriar Ave" />
    <suite value="" />
    <housetnumber value="11??" />
    <predir value="" />
    <streetname value="GREENBRIAR" />
    <postdir value="" />
    <streetsuffix value="AVE" />
    <sud value="" />
    <unit value="" />
  </results>
</string>
```

Note the USPS will work out the address down to the level of granularity the DPV data supports, in most cases, this will not allow a full street name and street number to be returned. The above examples demonstrate how question marks are used as wildcards, which will then have to be validated manually.

However there are definite keying benefits to certain systems when using a Full ZIP+4 reverse lookup, and so is good source of lookup if the source information is available.

getTypeDown – This web method is used to return a candidate list of potential street names and premise details from a ZIP code.

The method asks for a ZIP code then the first x letters of address line 1 to lookup a list of possible streets.

You need to pass the first x letters for address line 1, the number of results you want to return, the contextKey (which is the reference to your current lookup) to access the specific results and the ZIP Code.

Service1

Click [here](#) for a complete list of operations.

getTypeDown

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
prefixText:	<input type="text" value="5020"/>
count:	<input type="text" value="50"/>
contextKey:	<input type="text" value="1"/>
zipCode:	<input type="text" value="95118"/>

This will return the results in XML format as follows :

```
<ArrayOfString xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://tempuri.org/">
  <string>5020 Durango Ct</string>
  <string>5020 El Roble Ct</string>
  <string>5020 Howes Ln</string>
  <string>5020 Hawley Ct</string>
  <string>5020 Jarvis Ave</string>
  <string>5020 Joseph Ln</string>
  <string>5020 Gelia Way</string>
  <string>5020 Carter Ave</string>
  <string>5020 Corbin Ave</string>
  <string>5020 Las Cruces Ct</string>
  <string>5020 Lenelle Ct</string>
  <string>5020 Russo Dr</string>
  <string>5020 Rio Verde Dr</string>
  <string>5020 Rebel Ct</string>
  <string>5020 Sutcliff Ave</string>
  <string>5020 Trenary Way</string>
  <string>5020 Tifton Way</string>
  <string>5020 Wayland Ave</string>
  <string>5020 Yucatan Way</string>
</ArrayOfString>
```

The main usage of this method is for a typedown interface to allow auto fill of the first line of address. The only restriction is that you need to capture the ZIP code first, to allow the candidate list to be populated. The more information that is submitted with the first line of address the smaller the list of candidates. Within the findIT S2 interface we restrict this in a couple of ways. Firstly we only trigger the event after the first space in the address1 line using the AJAX control, then we limit the results to the number specified in the `<maxrecords value="???" />` node of findITConfiguration.xml, to maintain a manageable group of possible candidates, but aid keying speed at the same time.

TypeDownCityStateXML

This is the first of the 'TypeDown' style methods offered by the US Addressing Web Service. With TypeDownCityStateXML, it is possible to enter a city, city and state, or just state, to get a complete list of city / state combinations that match the input. For example, if you wanted to retrieve a complete list of cities in Texas, you would simply leave the city argument blank, and pass TX (or Texas) as the state argument. The XML Returned would look as follows –

```
<results>
  <citystatedata city="Red Lake" state="TX" />
  <citystatedata city="Bagwell" state="TX" />
  <citystatedata city="Gilmer" state="TX" />
  <citystatedata city="Dike" state="TX" />
  ...(Not all results shown)
</results>
```

TypeDownStreetsFromCityStateXML

A filtered result from the previous method, TypeDownCityStateXML, can be used in this method to retrieve a complete list of streets in the city. For example, using the result Red Lake, TX, we get the following

```
<results>
  <streetdata zip="75855" zipindex="0" streetindex="0" predir="" street="RR 1"
  suffix="" postdir="" city="Oakwood" state="TX" />
  <streetdata zip="75855" zipindex="0" streetindex="1" predir="" street="RR 1"
  suffix="" postdir="" city="Oakwood" state="TX" />
  <streetdata zip="75855" zipindex="0" streetindex="2" predir="" street="RR 2"
  suffix="" postdir="" city="Oakwood" state="TX" />
  <streetdata zip="75855" zipindex="0" streetindex="3" predir="" street="RR 2"
  suffix="" postdir="" city="Oakwood" state="TX" />
  <streetdata zip="75855" zipindex="0" streetindex="4" predir="" street="RR 3"
  suffix="" postdir="" city="Oakwood" state="TX" />
  <streetdata zip="75855" zipindex="0" streetindex="5" predir="" street="RR 3"
  suffix="" postdir="" city="Oakwood" state="TX" />
  ...(Not all results shown)
</results>
```

From these results, it is important to take note of the `zip`, `zipindex` and `streetindex` in particular for use in the 'TypeDownPrimaryRangeXML' method.

TypeDownStreetsFromZIPXML

This method returns exactly the same results as TypeDownStreetsFromCityStateXML, however the argument required by this method is the zip. Looking at the example above, passing the ZIP 75855 to this method would have yielded exactly the same result.

TypeDownPrimaryRangeXML

This method is used to get a list of primary premise ranges for a given street for a particular zip. The arguments required are the zip, zip index and street index from a 'TypeDownStreetsXML' call. Using the first result in the example above (75855, 0, 0) you will get the following result –

```
<results>
  <premisedata zip="75855" zipindex="0" streetindex="0" primaryindex="0"
  lowprem="" highprem="" parity="" outcrrt="R001" outzip4="9801" outcompany=""
  secondaryavailable="False" />
  ...(First Result Only Shown)
</results>
```

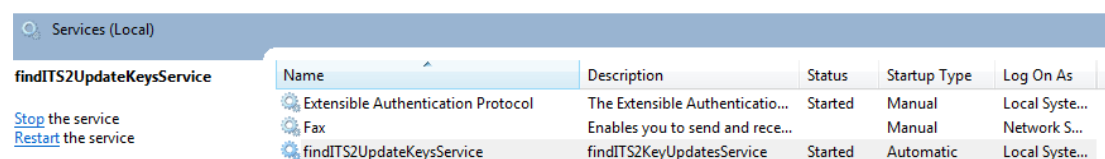
Note that from this result, as well as keeping hold of the zip, zip index and street index, we now need to make note of the primary index, for use in the TypeDownSecondaryRangeXML method.

TypeDownSecondaryRangeXML

This method is used to get a list of secondary premise ranges for a primary premise on a given street for a particular zip. The arguments required are the zip, zip index, street index and primary index from a 'TypeDownPrimaryRangeXML' call. As can be seen in the example above, the secondaryavailable attribute is set to false, indicating that no secondary premise range is available, so a call will not yield any results. If it was set to true however, a call could be made to further verify the secondary premise of an address.

The Key Updates Windows Service

As mentioned in the introduction, one of the components that comes with findIT S2 is a windows service. It is installed when installing findIT S2 and appears in the service manager under the name findITS2UpdateKeysService. The service is started up on install, and is set to automatically start up on system start up.



The screenshot shows the Windows Services console for the local machine. The 'findITS2UpdateKeysService' is selected. The table below represents the data visible in the screenshot.

Name	Description	Status	Startup Type	Log On As
Extensible Authentication Protocol	The Extensible Authenticatio...	Started	Manual	Local Syste...
Fax	Enables you to send and rece...	Manual	Manual	Network S...
findITS2UpdateKeysService	findITS2KeyUpdatesService	Started	Automatic	Local Syste...

This service periodically makes a call to the PerformPendingKeyUpdates() web method. By default, the interval of these calls is set to 30 seconds, however this can be configured to any interval by amending the following section in the file findITS2UpdateKeys.exe.config which is located in the `c:\{Path to findIT S2}\bin` directory.

```
<appSettings>
  <!-- The interval between webservice calls to PerformPendingKeyUpdates in ms
(30000 = 30 Seconds) -->
  <add key="serviceinterval" value="60000" />
</appSettings>
```

Simply change the value attribute of the `serviceinterval` key node to the length in time (in ms) you wish the intervals to be.

CRM System Integration

This section may require a basic understanding of HTML / CSS. It describes the simplest ways to create a link from your CRM system to the findIT S2 Site.

If this particular installation of findIT S2 was for a CRM system, then an extra directory will have been installed to the findIT S2 Site directory, namely 'CRM Resources', which can be found on the path

`c:\{findIT S2 app path}\website\findITS2Site\CRM Resources`

In this directory you will find the relevant files used in the following instructions to link your CRM system to findIT S2.

Microsoft Dynamics 4.0 Installation

The easiest way to integrate MS Dynamics with findIT S2 is to add a custom button to the global navigation through the ISV Config file. Information on what this file is and how to use it / enable customisations made from it can be found on the following link –

<http://msdn.microsoft.com/en-us/library/bb928136.aspx>

If you look in the CRM Resources folder in the findIT S2 Site directory, there should be a file called `fiw-logo-mscrm.gif`. This file should be dropped in the `_imgs` folder in the root of your MS Dynamics installation to be referenced in the button definition in the ISV Config file. The following is an example of a button node in the 'Toolbar' section of the ISV Config –

```
<Button Icon="/_imgs/fiw-logo-mscrm.gif"
JavaScript="window.open('http://localhost/findITS2Site/Default.aspx','findITS2S
earch','width=1100,height=430,scrollbars=1,location=0,toolbar=no');">
  <Titles>
    <Title LCID="1033" Text="findIT S2 Search" />
  </Titles>
  <ToolTips>
    <ToolTip LCID="1033" Text="findIT S2 Search" />
  </ToolTips>
</Button>
```

In the above example, the URL of the `window.open()` method may have to be amended accordingly depending on your installation of MS Dynamics.

Microsoft Dynamics 2011 Installation

As standard, findIT S2 ships with a managed solution that can be imported into your deployment of Dynamics 2011 in that standard manor. The managed solution itself can be found in the following location after install –

`c:\{findIT S2 app path}\website\findITS2Site\CRM Resources\Dynamics2011RibbonButton.zip`

Note that if findIT S2 was not installed using the default port in IIS (53000) then you will need to amend the findIT S2 JS web resource that is installed with the managed solution – namely the location specified in the `window.open()` call – the url will need to be changed accordingly.

Once installed and published, you should be able to see the findIT S2 button appear on the Ribbon in within the contacts, leads and accounts entities.

Appendix A – Troubleshooting

Below is a list of typical errors that may occur during the use of the administration program, coupled with explanations of what to do. Note that most errors that occur with findIT S2 will be logged in the custom event log called 'findIT S2 Log', which can be found in the event viewer - Start > Settings > Control Panel > Administrative Tools > Event Viewer. Generally more detail can be found about specific errors in the Event Log.

Installation Errors

1. Virtual directory could not be created error

If the virtual directories failed to be created during the installation process, they will have to be created manually in IIS.

Web Service Virtual Directory – create a virtual directory under the default website in IIS called 'findITS2Service' and point it to the relevant findITS2Service folder that was installed (`c:\{findIT S2 Path}\website\findITS2Service`), making sure that 'Read' and 'Run Scripts' were checked. Once created, make sure that the default document is 'Service1.aspx' and that the directory is configured to run on ASP.NET version 2.0 (which covers ASP.NET 3.5). You should then be able to browse to the landing page of the web service in IIS and see a list of the available web methods.

Web Site Virtual Directory – create a virtual directory under the default website in IIS called 'findITS2Site' and point it to the relevant findITS2Site folder that was installed (`c:\{findIT S2 Path}\website\findITS2Site`), making sure that 'Read' and 'Run Scripts' were checked. Once created, make sure that the default document is 'Default.aspx' and that the directory is configured to run on ASP.NET version 2.0 (which covers ASP.NET 3.5). You should then be able to browse to the landing page of the web site in IIS and see the data input form.

Web Browser Errors

1. The web service or sample website cannot be browsed to –

A common cause of this is that IIS is not running, so check that. Otherwise, check whether or not the virtual directories findITS2Site or findITS2Service actually exist in IIS. If not, there was an error on installation that caused the virtual directories to not get created. In this case they will have to be created manually, and point to :

`c:\{findIT S2 app path}\website\findITS2Site`

and

`c:\{findIT S2 app path}\website\findITS2Service` respectively.

2. 'Error with the Configuration Class' message

When performing a search on the sample web site (or just retrieving a results XML string from the web service) and you receive this message, it normally means that the web service configuration is unavailable. A common cause of this is that the XML configuration file has not been loaded, or the configuration status has been disabled in the admin program. To fix this, simply reload the XML configuration in the findIT S2 Administration program.

Administration Program Errors

1. Cannot connect to Web Server

If you receive an error message when trying to load the XML configuration file stating that the web service cannot be connected to, the common cause of this is that either IIS is not running as a service, or the findITS2Service virtual directory is not running in IIS. Check that both of these are indeed running.

2. Configuration File Formatting Error

Errors with the configuration file when trying to load the configuration are normally down to bad formatting of the configuration file – unclosed tags and missing quotation marks are just two examples of common mistakes. The easiest way to check if the XML file formatting is valid is to view it in an internet browser, like internet explorer – If it is not valid an error message will be displayed and an explanation of where the error has occurred.

3. Datasource connection error

If you get an error saying that there is an error with the datasource connection, this means that there is probably an error with your connection string specified in the XML. Check that the data source, initial catalog and other credentials are correct. If you still cannot connect check that the user specified in the connection string has access to the database in question. Another thing worth checking in the case of SQL Server is whether SQL Server is running in mixed mode (Windows and SQL Authentication).

4. Table does not exist error

An error stating that a particular table does not exist normally points to a spelling mistake in a table name in the XML. Double check all the tables in the XML for the datasource in question.

5. Column does not exist error

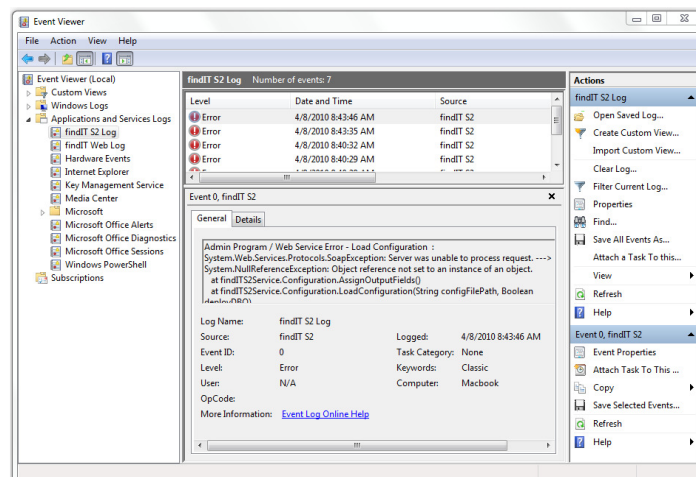
This error means that the column name in question does not exist for the tables that you have specified in your datasource. Double check the field mapping in the XML and check that the database column is spelt correctly, and that the column specified actually does exist.

6. Addition Error Information

In addition to the on screen message, users are encouraged to view the windows event log for any -2 return codes for the findIT S2 admin tool. You can access this via

Control Panel → Administrative Tools → Event Viewer

The below features additional information returned where there is a configuration load error due to a badly formatted findITConfiguration.xml file.



7. Unable to open Accumail database for processing

When calling the address web service, and receiving the following message returned :

```
<?xml version="1.0" encoding="utf-8" ?>  
<string xmlns="http://tempuri.org/">Unable to open Accumail database for processing.</string>
```

normally means either

- a) the addressIT data was not installed on your machine properly.
- b) You have not installed the latest dataset – the addressIT datasets expire every 2 months and the addressing service will not function if your data is out of date.

Appendix B – Quick Start Check List

The following is check list of points for getting findIT S2 up and running –

- Install the product.
- Amend the C:/{findITS2 Path}/findITS2Configuration.xml file as necessary as per the chapter **Amending the Web Service Configuration XML File**.
- Load the configuration file in the findIT S2 Administration program and generate the keys for the datasources.
- Amend the C:/{findITS2 Path}/ControlConfig.xml file as necessary as per the chapter **The findIT S2 Web Service**.
- Browse to the findIT S2 Site and start querying the data source(s).

Appendix C – Warning values returned from ConfirmAddressXML

- 1 – An Error occurred when processing the address.
- 0 – Address was successfully verified.
- 1 – Address is coded but undeliverable (i.e. on side of street known to contain no houses).
- 2 – The zip code was not found and the city and state cannot be used to determine a geographical area to search.
- 3 – Coding would result in changing both the zip and city.
- 4 - The best match would result in too many suspicious changes.
- 5 – The street was identified as an alias, but was out of the range restricted for that alias.
- 6 – No street address was given.
- 7 – There are no street name matches in the given zip code or in any geographically-related zip code.
- 8 – The street may contain superfluous components which cannot be discarded with confidence.
- 9 - The house number could not be matched.
- 10 – The best match was made to a zip move record but was not an exact match.
- 11 – A zip move match was made, but no exact match could be found in the new zip.
- 12 – Insufficient address information. It's not even possible to guess as to what might be correct.
- 13 – There are multiple matches with the same degree of confidence.
- 14 – Incorrect suffix, directional, street name or unit resulted in multiple matches with the same degree of confidence.
- 15 – Incorrect zip, city or urbanization resulted in multiple matches with the same degree of confidence.
- 16 – A Corrected field was too long to fit into the supplied field.
- 17 – Media Error. The database could not be read because of a hardware or system problem.

Appendix D – Addressing Web Service Quick Start Guide

The following is a quick start guide on how to install and get up and running with the Addressing Web Service. The first step is to install the Capscan Matchcode Software, which is a dependency of the findIT S2 Addressing Service –

1. Browse the media with the Capscan Installer on, run the **mcdcd.exe** file and follow these steps
 - Click Here to Continue
 - Install or Update Matchcode Software and Databases
 - Install or Update Matchcode Data Capture and Batch
 - Server Machine
 - Install or Update Server Software and Databases
2. The final step above will launch an installer, during which you will need to follow these steps
 - Use all default values the installer runs with (file paths etc)
 - Do not enter a key, and ignore the warning it gives
 - At the end, uncheck 'Start Capscan Pool Manager Service' as this will not run without a key

3. Run the following executable

C:\Program Files (x86)\Capscan\Server\PoolManAdmin.exe

And follow these steps

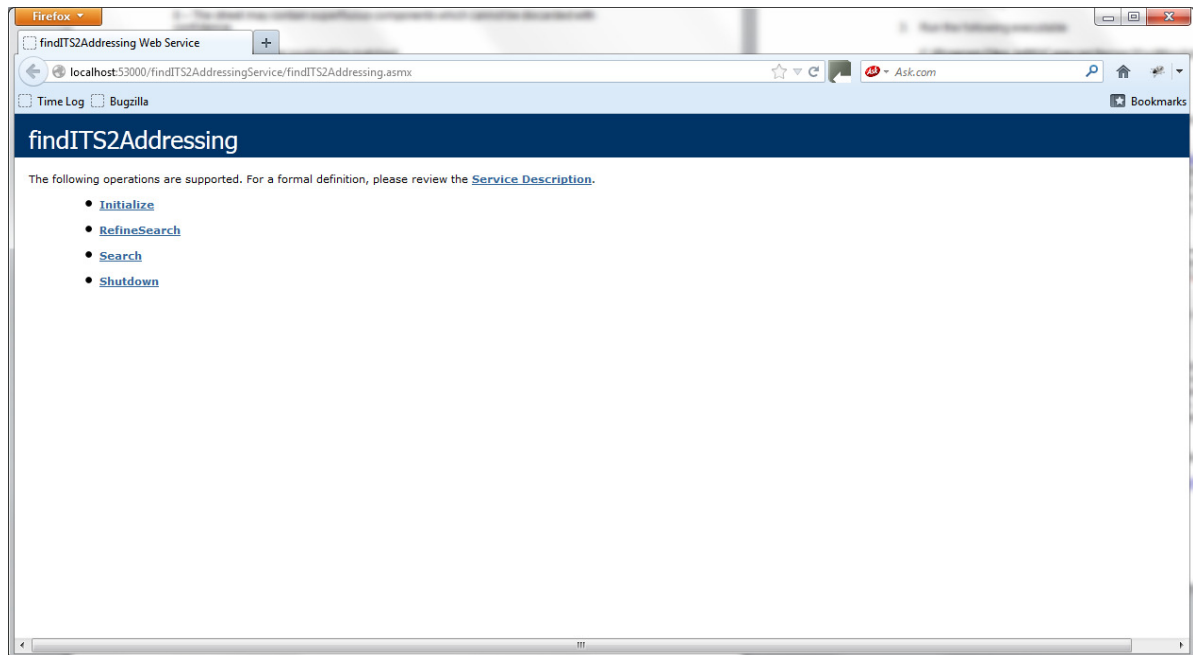
- Ignore any error / warning messages on start up
 - Right Click the available licences on LHS and select 'Create New Licence File'
4. Copy the Reg Key and send to Capscan (support@capscan.com), along with your username and company name so that you can obtain an activation key from Capscan for your specific machine. Note that for ROW addressing, a different key is needed that is entered as part of the findIT S2 installation process – It consists of an Account Number and Client ID, which you will again need to obtain from Capscan depending on your licence.
5. Once you have received the key, run PoolManAdmin.exe > Licences > Create New Licence File again, and enter your credentials (username, company, activation key). This should enable UK addressing in Capscan Matchcode ready to be used by the findIT S2 Addressing Service.

With Capscan Matchcode installed successfully, the next step is to install findIT S2.

6. Run the findIT S2 Installer
7. On the Addressing Section, Select the regions for which you wish to enable Addressing. Note if you select ROW Addressing, you will be prompted to enter the Account and Client ID mentioned above. These will be encrypted and written to the ControlConfig.xml file for use in the findIT S2 UI – for direct calls to the Web Service, these will need to be passed as parameters to the Initialise method, so keep a note of them.
8. On completion, bowse to the following location in a web browser –

<http://localhost:53000/findITS2AddressingService/findITS2Addressing.aspx>

On browsing to the above URL, you should see a screen that looks like the following –



For the purpose of visual representation, we will talk about calling the Web Service directly by using the interface in a web browser. Essentially the following guidelines are exactly the same when you come to call the service programmatically with whatever client technology you are using.

The first call that needs to be performed in order to use the service is to the Initialize() method. This method takes 4 parameters, outlined below –

Initialize

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
dataset:	<input type="text"/>
ip:	<input type="text"/>
account:	<input type="text"/>
client:	<input type="text"/>

The first argument, dataset, will be either 'PAF' or 'WORLD ADDRESSING' depending on whether you wish to perform UK or ROW addressing respectively. The second argument only applies to UK Addressing, and is 'Localhost' by default, unless the matchcode software was installed on a different server to findIT S2. The last 2 arguments, Account and Client, only apply to ROW addressing, and are the Account and Client licence related values mentioned earlier on. Once the relevant parameters have been populated, the call can be made – an example response is shown below.

```

- <string>
  <?xml version="1.0" encoding="UTF-8"?><result><success>true</success><handle>27600</handle></result>
</string>

```

Note that the return string is in an XML format, which can be parsed by the calling client to read the relevant values. A return message will always have a 'success' sub node, indicating true or false. In this case, the call was successful, and so a 'handle' node is also returned (value 27600) which will need to be used in all subsequent calls to the service. An example of what is returned in the case of an error is shown below.

```

- <string>
  <?xml version="1.0" encoding="UTF-8"?><result><success>false</success><error><number>3</number><message>Input/Output error</message></error></result>
</string>

```

Note that in the case of success=false, a 'handle' node is not returned, and instead an error node is in its place with sub nodes of 'number' and 'message' – This is standard practice with calls to the Addressing Web Service, and should be programmed for in the calling client.

Once a handle has been obtained, it is possible to perform address validation calls using the Search() web method. The expected parameters are shown below –

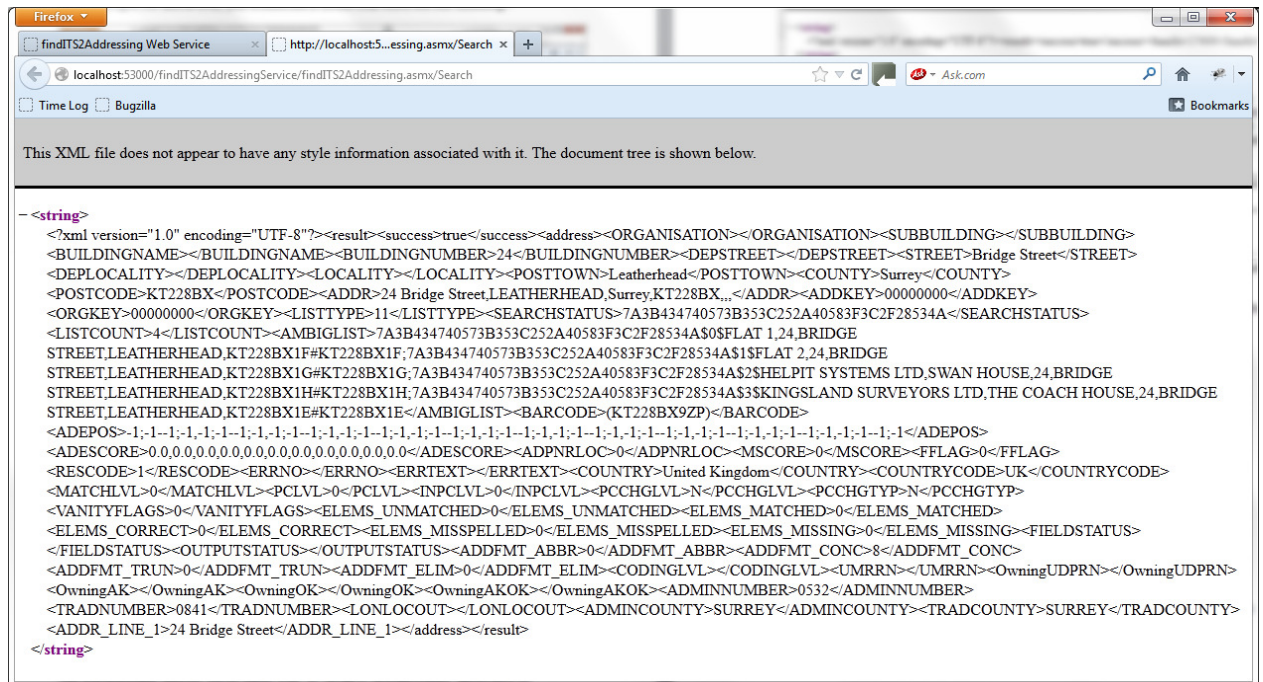
Search

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
handle:	<input type="text"/>
searchType:	<input type="text"/>
data:	<input type="text"/>
user:	<input type="text"/>

The first parameter, 'handle', is the value retrieved from the first call to Initialise(). The second parameter, 'searchType', at the time of writing is recommended to take the value 1 (Single Search). The 'data' parameter is simply the address that you are trying to verify / lookup, and the 'user' parameter is currently not functional, and reserved for future use. Passing the address string '24, KT22 8BX' gives the following result –



As per the first Web Method, the first node is a success node, indicating whether or not the call to the Web Service failed – Had it failed, the only other node present would have been an 'error' node. Instead, we have an address node, that contains many sub nodes with information on the address(es) that have been matched. In this instance, the provided string has produced some ambiguous results that exist at the same premise / postcode. These are listed in the 'AmbigList' node. If the result had been an exact match, no AmbigList values would have existed, and the returned result would have been able to be used directly. Instead, a call is needed to RefineSearch() in order to filter down to the required result. You will notice above that the AmbigList is a list of items delimited by semi-colons. Each item is then made up of the following format –

SEARCH_KEY + \$ + ITEM_NUMBER + \$ + ADDRESS

You will see why this format is important in the next call to RefineSearch(), the parameters of which are described below –

RefineSearch

Test

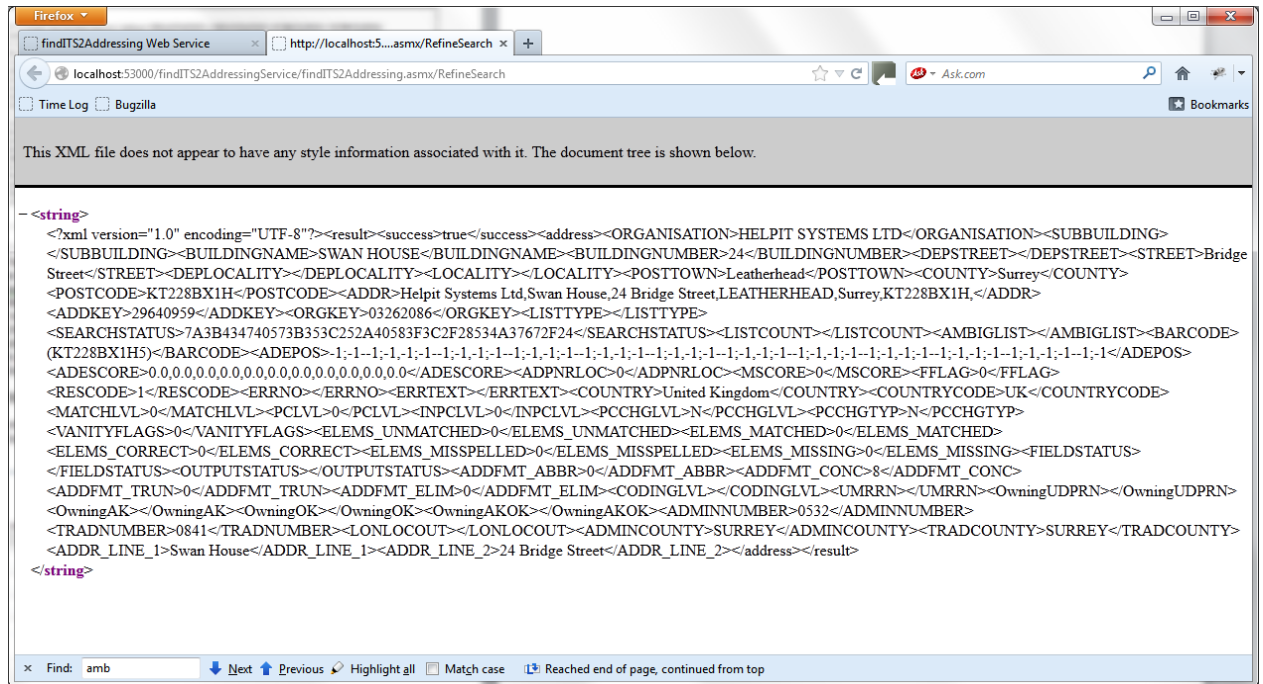
To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
handle:	<input type="text"/>
searchKey:	<input type="text"/>
item:	<input type="text"/>
user:	<input type="text"/>

SOAP 1.1

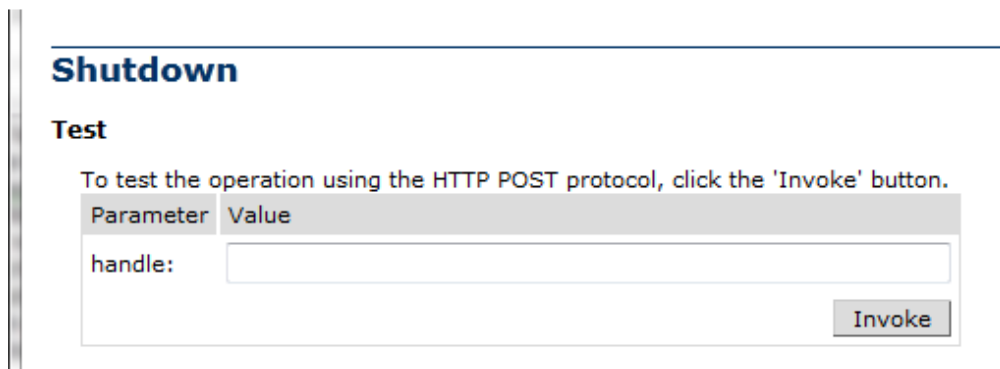
Again, 'handle' takes the same value that was retrieved in the first call to Initialize(), and 'user' is still non-functional. The two remaining paramters, 'searchKey' and 'Item' are the first 2 values in the format described above. The requirement for a calling client then is to, in a call to Search() that produces ambiguous results, parse out the results and allow the user to

select the desired item by passing the SEARCH_KEY and ITEM_NUMBER values to RefineSearch(). Selecting Item 2 in the above example gives the following result –



Notice now that a particular result has been selected, there are no items in the AmbigList.

The final method of the Web Service is the Shutdown() method, shown below –



This method is pretty self-explanatory – It simply takes the handle of the addressing web service that you have finished using and wish to dispose of. It is good practice to call this method once you have finished making use of the addressing service in a client in order to free up resources and keep memory consumption to a minimum.

Appendix E – findIT S2 WCF Search Interface

The following is a technical reference for the types and methods associated with the findIT S2 WCF Search Interface. Where self-explanatory, descriptions have been omitted.

findITS2SearchService

This is the main interface that offers the core searching / comparison functionality of the product. It offers the following 2 methods.

FindRecords(FindRecordsInput input)

This Method takes the parameters provided in the FindRecordsInput object and uses them to perform a search. This method returns a type of **FindRecordsResult**.

CompareRecords(List<Record> records)

This method takes a list of record objects and compares each one to every other. The method returns a type of **CompareRecordsResult**.

ParseRecords(List<Record> records)

This method takes a list of record objects and parses them. The method returns a type of **ParseRecordsResult**.

FindRecordsInput

This object is the input used for a call to the FindRecords method of the findIT S2 search service. It contains the following properties –

List<string> SpecificDataSource

This can be used to specify specific data sources for the search to be performed on. If left as an empty list, all data sources will be queried.

List<UniqueRefSearchInfo> UniqueRefSearches

This can be used to specify a set of unique searches to perform on the data sources. It is possible to specify the data source, table and column name to query. It should not be confused with the UniqueRef property of the Record object, which is not used in searching. All unique ref searches should be configured through this property. See the **UniqueRefSearchInfo** object description for more information.

Record SearchRecord

This is the record that is going to be used to perform the search. Its properties should be filled out accordingly. See the **Record** object description for more information on the properties.

UniqueRefSearchInfo

This object encapsulates all the information necessary for the FindRecords method to perform a unique search. The properties are as follows –

string DataSourceID

This is simply the id of the data source to perform the unique search on. You can only specify one data source per UniqueRefSearchInfo object. To apply the unique search to multiple data sources, simply create multiple instances of the UniqueRefSearchInfo object to add to the FindRecordsInput object.

string TableName

This is the name of the table in the specified data source that contains the data to be queried.

string ColumnName

The name of the column in the specified table that contains the data to be queried.

string UniqueRefValue

This is the value to use when performing the unique ref query.

Record

This object represents a record entity – it is used both as input to query, and as output in matching results. The properties of the Record object are as follows –

string UniqueRef
string FullName
string Salutation
string Contact
string Prefix
string LastName
string FirstNames
string Initials
string Qualification
string Suffix
string DateOfBirth
string SecondPrefix
string SecondLastName
string SecondFirstNames
string SecondInitials
string Organization
string Department
string JobTitle
string Address1
string Address2
string Address3
string Address4
string Address5
string Address6
string Address7
string Address8
string Address9
string FlatNo
string Premise
string Thoroughfare
string Town
string Region
string Postcode
string Country
string Telephone
string Fax
string Email
string Username
string Domain
string CustomField1
string CustomField2
string CustomField3
string CustomField4
string CustomField5
string CustomField6
string CustomField7
string CustomField8
string CustomField9

FindRecordsResult

This object holds the results of a call to FindRecords. The properties are as follows –

bool Success

This indicates whether or not the call to the FindRecords method was successful. If false, there will be an error message indicating what went wrong in the LastError property.

String LastError

This property contains the details of any failure that occurred in the event of the Success property being false.

List<FindRecordsResultItem> Results

This property contains the results of the call to each data source. There will be one FindRecordsResultItem per data source specified in the input – If not data sources were specified, there will be an item for every data source in the configuration. See the description of the **FindRecordsResultItem** object for more information.

int TotalTime

This property simply contains the total time (in ms) elapsed during execution of the FindRecords method call.

FindRecordsResultItem

This object contains the results for a particular data source in a call to the FindRecords method. The properties are as follows –

int ResultCode

This is the result code returned as described in the main documentation.

FindRecordsInput Input

This is simply the input that was provided to the call of the FindRecords method.

string InputFields

This property holds a '+' delimited list of the input fields that correspond to the match keys that were selected and used from the matchkeysettings.xml configuration file. It is helpful in determining the keys being used to search on and tuning them accordingly.

Record NormalisedRecord

This Record object is populated with the parsed and normalised data that was provided in the input record. For example, if the FullName property was populated on the input Record, then the API will parse out the name elements into the appropriate fields (such as prefix, first names and last name).

List<Match> Matches

This property contains a list of matches that were found in the data source in question. For more information see the **Match** object description.

List<string> Warnings

This is simply a list of warnings that occurred during the search against the data source in question.

int Count

This property simply contains the total number of matches that were found in the search.

int Elapsed

This property indicates that total time taken for the search to complete on this particular data source.

Match

This object contains all the information for a match found in the data source being queried. It consists of the following properties –

string DataSourceID

This contains the data source ID that the match came from.

Record Record

This contains the record object that holds all the properties of the match.

double Score

This is the score that was achieved when comparing the input record to the record in question.

double MinScore

This is the minimum score required for the match in question to be considered a match.

double MaxScore

This is the maximum possible score that the match in question could have achieved being compared to the input record.

double NameScore

This is the name score that was achieved when comparing the input record to the record in question.

double OrganizationScore

This is the organization score that was achieved when comparing the input record to the record in question.

double AddressScore

This is the address score that was achieved when comparing the input record to the record in question.

double PostcodeScore

This is the postcode score that was achieved when comparing the input record to the record in question.

double EmailScore

This is the email score that was achieved when comparing the input record to the record in question.

double TelephoneScore

This is the telephone score that was achieved when comparing the input record to the record in question.

double DateOfBirthScore

This is the date of birth score that was achieved when comparing the input record to the record in question.

double CustomField1Score

This is the custom field 1 score that was achieved when comparing the input record to the record in question.

double CustomField2Score

This is the custom field 2 score that was achieved when comparing the input record to the record in question.

double CustomField3Score

This is the custom field 3 score that was achieved when comparing the input record to the record in question.

double CustomField4Score

This is the custom field 4 score that was achieved when comparing the input record to the record in question.

double CustomField5Score

This is the custom field 5 score that was achieved when comparing the input record to the record in question.

double CustomField6Score

This is the custom field 6 score that was achieved when comparing the input record to the record in question.

double CustomField7Score

This is the custom field 7 score that was achieved when comparing the input record to the record in question.

double CustomField8Score

This is the custom field 8 score that was achieved when comparing the input record to the record in question.

double CustomField9Score

This is the custom field 9 score that was achieved when comparing the input record to the record in question.

int SearchIndex

This is the search index, or key index, for which the match was found when comparing the input record to the record in question.

CompareRecordsResult

This object contains all the information returned from a call to the CompareRecords method. It contains the following properties –

bool Success

This property indicates whether or not the call to the CompareResults method was successful. If not, an indication as to what went wrong will be contained in the LastError property.

string LastError

This property contains the details of any failure that occurred in the event of the Success property being false.

List<Comparison> Results

This is a list of the comparison results that were achieved when comparing the list of records provided to the CompareRecords method. For more information, see the **Comparison** object description.

int TotalTime

This property describes the total time taken (in ms) for the call to CompareRecords to complete.

Comparison

This object contains all the information relating to a comparison between two records in the CompareRecords method. It contains the following properties –

string Record1

This contains the unique ref of the first record in the comparison.

string Record2

This contains the unique ref of the second record in the comparison.

Scores Scores

This property contains all the information relating to the scores that were achieved in the comparison. For more information see the **Scores** object description.

Scores

This object contains values of scores at all levels. The different score levels (which are all properties) are as follows –

double TotalScore
double MaxScore
double MinScore
double NameScore
double OrganizationScore
double AddressScore
double PostcodeScore
double EmailScore
double TelephoneScore
double DateOfBirthScore
double CustomField1Score
double CustomField2Score
double CustomField3Score
double CustomField4Score
double CustomField5Score
double CustomField6Score
double CustomField7Score
double CustomField8Score
double CustomField9Score

ParseRecordsResult

This object contains the results of a call to the ParseRecords method. The properties are as follows –

List<Record> ResultRecords

This is a list of records. The records are parsed versions of the records in the list that was passed into the ParseRecords method.

bool Success

This property simply indicates whether or not the call to the ParseRecords method was successful, where True means success and False means failure.

string LastError

If the Success property is False, then this property will provide any errors that the call to the ParseRecords method may have produced.

Appendix F – International Address Verify Service Interface

The following section is a technical reference for the types and methods of the International Address Verify service, available as part of the findITS2AddressService application. Where self-explanatory, descriptions have been omitted.

InternationalAddressVerifyService

This is the main interface that offers the core address verification functionality. It contains one single method -

ProcessResult Process(Address address, ProcessOptions processOptions, Processes processes)

This method takes all the information provided in the input parameters, and performs the verification process accordingly. The return type on this method is a **ProcessResult** – for more information on this and the input parameters, see the object descriptions below.

Address

This object contains all the information relating to an address that requires verification. It is passed as the first argument to the Process method. Its properties are as follows –

string AddressStr – This is a comma delimited string of the whole address

string Address1

string Address2

string Address3

string Address4

string Address5

string Address6

string Address7

string Address8

string Address9

string Address10

string Address11

string Address12

string Country

string SuperAdministrativeArea

string AdministrativeArea

string SubAdministrativeArea

string Locality

string DependentLocality

string DoubleDependentLocality

string Thoroughfare

string DependentThoroughfare

string Building

string Premise

string SubBuilding

string PostalCode

string Organization

string PostBox

string Contact

string Function

string Department

string ThoroughfarePreDirection

string ThoroughfareLeadingType

string ThoroughfareName

string ThoroughfareTrailingType

string ThoroughfarePostDirection

string DependentThoroughfarePreDirection

string DependentThoroughfareLeadingType
string DependentThoroughfareName
string DependentThoroughfareTrailingType
string DependentThoroughfarePostDirection
string BuildingLeadingType
string BuildingName
string BuildingTrailingType
string PremiseType
string PremiseNumber
string SubBuildingType
string SubBuildingNumber
string SubBuildingName
string OrganizationName
string OrganizationType
string PostBoxType
string PostBoxNumber

Note that the applicability of the above properties will depend on the country that the address is associated with.

ProcessOptions

This object contains the properties that describe all the parameters you want to apply to the Process method – It is passed as the second argument to the method. The properties are listed below. Note that if any of the properties are not set, the default value will be used.

string FIS2AddressLineFormattingConfig

Used to specify the custom formatting configuration file for any of the returned address elements. If left blank, the standard formatting provided by the addressing engine is used. The name of this property is prefixed with FIS2 as it has been added and implemented at the wrapper level as part of findIT S2, and does not exist in the native addressing API.

string AddressLineSeparator

Used to specify the string which will separate the output address lines within the output Address field (default "
").

string CasingIgnoreFields

Used to specify the fields that should not be cased. Default "ID,PostalCode,PostalCodePrimary,PostalCodeSecondary,ISO3166-2,ISO3166-3,PostBox,PostBoxNumber,PostBoxType,_L,Latitude,Longitude,GeoAccuracy,GeoDistance,AVC".

string CertifiedCountryList

Certified option which allows testing of a SERP compliant output for Canada (SERP Recognition still pending).

int ConfidenceThreshold

Used to specify the threshold at which to stop returning parse interpretations, as a percentage of the confidence of the top result. Default 90.

int ContextResultCacheSize

Used to specify how many sets of context results should be cached in memory. This cache is a cost-based cache based on processing time, meaning that the slowest context analyses have their results cached. Default 0. Values up to around 5,000 are relevant, but the memory usage and performance gain will be dependent on the supplied input data.

int ContextCountryCacheSize

Used to specify the number of MRU countries to generate ContextResult caches for. On an unsorted multi-country file a value of around 10 may be relevant, but the memory usage and performance gain will be dependent on the supplied input data.

string CountryFields

Used to specify the comma delimited list of fields to be searched for country information.

string CustomFields

Used to specify the comma delimited list of custom fields to be searched for during the parse process.

int CustomFieldConfidence

Used to specify the confidence to be applied to the recognition of any fields recognized through the CustomFields option.

string DataDirectory

Used to specify the directory in which to find the Global Knowledge Repository.

string DefaultCountry

Used to specify the ISO 3166-1 alpha-3 code which should be used if no identifiable country can be found in an input record.

int FieldNameWeight

Used to specify the default field name weight, between 1 and 255. Default 10. A larger value will give more precedence to the supplied field name, a smaller value will give more precedence to the GKR matches.

string ForceCountry

Used to specify the ISO 3166-1 alpha-3 code which should be used for all input records.

string GeocodeCountryList

Used to specify a comma-delimited list of ISO 3166-1 alpha-3 codes of acceptable countries to geocode. Input records from other countries will not return geocodes. The default is to process data from all countries.

bool IgnoreUnmatched

Allow unmatched input data within the address lines to be ignored during a Search process, meaning that a search can proceed matching only partial input data. Default false.

string LogFileName

Used to specify a filename for debug output.

int MatchScoreAbsoluteThreshold

Used to specify the minimum matchscore a record must reach in the Match process to be taken into account as a candidate for matching. Default value is 60 and the maximum value is 100.

int MatchScoreThresholdFactor

This is a value that is used as a factor of the highest matching result. The resulting value is then used as a cut-off for considering candidates for the results. The higher the value of the factor, the higher the chance of getting a good verification result. Default value is 95 and the maximum value is 100.

int MaxResults

Used to specify the maximum number of results to return in the ProcessResults structure.

int MinimumMatchscore

Used to specify the minimum matchscore a record must reach in order to avoid reversion. Default is 0 (zero), valid values are 0-100.

int MinimumPostcode

Used to specify the minimum postcode status a record must reach in order to avoid reversion. Default is 0 (zero), valid values are 0-8.

int MinimumVerificationLevel

Used to specify the minimum verification level a record must reach in order to avoid reversion. Default is 0 (zero), valid values are 0-5.

bool OutputAddressFormat

Used to specify whether to output an extra field for the Address block fields with the suffix 'Format' showing how the hierarchical field positioning has occurred. Default false.

CasingOptions OutputCasing

Used to specify the letter case to use for output fields. Enum values are 'Upper', 'Title', 'Lower' or 'UseDefault'. Default 'Title' (which is also used when using the Enum val 'UseDefault').

string OutputScript

Related to transliteration - Used to specify the ISO 15924 Code in which the output should be encoded, if possible. Alternatively 'Native' should be specified to choose the correct country specific script value. At the time of writing, valid values are -

Latn – Latin (English transliteration wherever possible)

Cyrl – Cyrillic (Russia)

GreK – Greek (Greece)

Hebr – Hebrew (Israel)

Hani – Kanji (Japan)

Hans – Simplified Chinese (China)

Arab – Arabic (United Arab Emirates)

Thai – Thai (Thailand)

Hang – Hangul (South Korea)

Native – Output in the native script wherever possible

int QueryResultCacheSize

Used to specify how many sets of query results should be cached in memory. This cache is a cost-based cache based on query time, meaning that the slowest queries have their results cached. Default 0. Values up to around 5,000 are relevant, but the memory usage and performance gain will be dependent on the supplied input data.

RangeDecoposeOptions RangeDecompose

Used to specify how to expand ranges such as premise numbers and sub building numbers. Valid values are 'Match' (returning only the value matching the input query), 'Range' (returning a human-readable range, e.g. '1-9 Odd'), or 'Full' (expanding out the range into its constituent numbers, e.g. '1', '2', '3', '4', '5'). It is also possible to specify 'UseDefault'.

int ReferenceDatasetCacheSize

Used to specify the number of MRU reference datasets to cache. Default value is 5.

int ReferencePageCacheSize

Used to specify the number of MRU pages to cache from each individual reference dataset. Default value is 12, each page equates to approximately 5Mb of memory use.

string SuppressAdditionFields

Used to specify the list of fields that should not be added to an output result if they do not appear in the input record.

string SuppressAddressFields

Used to specify the list of fields that should be suppressed from the Address field output.

string SuppressFields

Used to specify the list of fields that should be suppressed from the output.

string TransliterationIgnoreFields

Used to specify the fields that should not be transliterated. Default "ID,ISO3166-2,ISO3166-3,_L,Latitude,Longitude,GeoAccuracy,GeoDistance,AVC".

bool ToolInfo

Specify true to output extra debug information viewable using the getFieldInfo method.

bool UseCustomLexicons

Specify true to force the API to search for custom lexicons (lx__c.lfs files)

bool UseSymbolicTransliteration

Specify false to limit transliteration to complete field phrase matches. Default true.

string VerifyCountryList

Used to specify a comma-delimited list of ISO 3166-1 alpha-3 codes of acceptable countries to process. Input records from other countries will return the unprocessed Address Verification Code 'U00-U00-P0-000'. The default is to process data from all countries.

string LogInput

Record the process input data to the specified debug log file.

string LogOutput

Record the process output data to the specified debug log file.

Processes

This is an enumeration of processes that are available from the Process method. The enumeration is a bit field, so in order to combine processes in one call, simply OR the values together as the third argument to the process method. Note that not all processes can run at the same time. Values are as follows –

Country = 1

Parse = 2

Match = 4

Format = 8

Verify = 16

Search = 32

Geocode = 64

Query = 128

GKRInfo = 256

Note that, whilst using strongly typed code it is easy to OR these values (such as using the pipe delimiter in .NET), when manually writing a text based SOAP request, the values are ORed simply by leaving a space between them. For example, using Verify with Geocode in SOAP would be achieved with the following (note the exact format of the nodes below may vary) -

```
<web:processes>Verify Geocode</web:processes>
```

ProcessResult

This object contains the results of a call to the Process method. The properties are as follows -

string AccuracyCode

This represents the Address Verification Code (AVC) of a call to the Process method.

List<ProcessResultItem> ResultItems

This property contains a list of results / matches that were found in the call to the process method. Depending on the Processes submitted as the third argument, and the address data, this list can contain any number of results. For instance, the 'Verify' process will only ever return one single result, whereas the 'Search' process can return multiple matches if ambiguities were detected.

ResultStatus Status

This property describes the status of the call to the Process method. Possible values are -

psException

psInvalidInputRecord

psOK

psServerUninitialized

In the case of psException, the specific reason for the failure will be contained in the LastError property.

string LastError

This property will contain information about any exceptions that occur in the case of a failure.

ProcessResultItem

This object contains all the information relating to a match found during a call to the Process method. The information for each field is contained in a **ProcessResultItemField** object, which is described further below, as is the type of the MatchingInfo property, **MatchInfo**.

ProcessResultItemField AVC

ProcessResultItemField AddressStr

ProcessResultItemField Address1

ProcessResultItemField Address2

ProcessResultItemField Address3

ProcessResultItemField Address4

ProcessResultItemField Address5

ProcessResultItemField Address6

ProcessResultItemField Address7

ProcessResultItemField Address8

ProcessResultItemField Address9

ProcessResultItemField Address10

ProcessResultItemField Address11

ProcessResultItemField Address12

ProcessResultItemField DeliveryAddress

ProcessResultItemField DeliveryAddress1
ProcessResultItemField DeliveryAddress2
ProcessResultItemField DeliveryAddress3
ProcessResultItemField DeliveryAddress4
ProcessResultItemField DeliveryAddress5
ProcessResultItemField DeliveryAddress6
ProcessResultItemField DeliveryAddress7
ProcessResultItemField DeliveryAddress8
ProcessResultItemField DeliveryAddress9
ProcessResultItemField DeliveryAddress10
ProcessResultItemField DeliveryAddress11
ProcessResultItemField DeliveryAddress12
ProcessResultItemField Country
ProcessResultItemField CountryName
ProcessResultItemField ISO3166_2
ProcessResultItemField ISO3166_3
ProcessResultItemField ISO3166_N
ProcessResultItemField SuperAdministrativeArea
ProcessResultItemField AdministrativeArea
ProcessResultItemField SubAdministrativeArea
ProcessResultItemField Locality
ProcessResultItemField DependentLocality
ProcessResultItemField DoubleDependentLocality
ProcessResultItemField Thoroughfare
ProcessResultItemField DependentThoroughfare
ProcessResultItemField Building
ProcessResultItemField Premise
ProcessResultItemField SubBuilding
ProcessResultItemField PostalCode
ProcessResultItemField PostalCodePrimary
ProcessResultItemField PostalCodeSecondary
ProcessResultItemField Organization
ProcessResultItemField PostBox
ProcessResultItemField Unmatched
ProcessResultItemField Contact
ProcessResultItemField Function
ProcessResultItemField Department
ProcessResultItemField ThoroughfareType
ProcessResultItemField ThoroughfarePreDirection
ProcessResultItemField ThoroughfareLeadingType
ProcessResultItemField ThoroughfareName
ProcessResultItemField ThoroughfareTrailingType
ProcessResultItemField ThoroughfarePostDirection
ProcessResultItemField DependentThoroughfarePreDirection
ProcessResultItemField DependentThoroughfareLeadingType
ProcessResultItemField DependentThoroughfareName
ProcessResultItemField DependentThoroughfareTrailingType
ProcessResultItemField DependentThoroughfarePostDirection
ProcessResultItemField BuildingLeadingType
ProcessResultItemField BuildingName
ProcessResultItemField BuildingTrailingType
ProcessResultItemField PremiseType
ProcessResultItemField PremiseNumber
ProcessResultItemField SubBuildingType
ProcessResultItemField SubBuildingNumber
ProcessResultItemField SubBuildingName
ProcessResultItemField OrganizationName
ProcessResultItemField OrganizationType
ProcessResultItemField PostBoxType
ProcessResultItemField PostBoxNumber

ProcessResultItemField GeoAccuracy
ProcessResultItemField GeoDistance
ProcessResultItemField Latitude
ProcessResultItemField Longitude
MatchInfo MatchingInfo
FormatInfo FIS2FormattingInfo

ProcessResultItemField

This object represents all the information pertaining to a field of a match found in the Process method. The properties are as follows –

string Name

This property holds the name of the field as a string

string Value

This property holds the value of the field.

int Confidence

This property holds a measure of how confident the match of this particular field is in the matched record.

string Info

This property can contain further information on the matched field.

float MatchScore

This property contains the score achieved for the field in the matched record.

FieldStatus Status

This is a description of the what actions were taken on this field during the process. Potential values are

fsAdded
fsEmpty
fsIdentifiedAlias
fsIdentifiedContext
fsIdentifiedNoChange
fsNotApplicable
fsUnrecognized
fsVerifiedAliasChange
fsVerifiedLargeChange
fsVerifiedNoChange
fsVerifiedSmallChange

MatchInfo

This object represents the match information available for a particular match found during a call to the Process method. It gives an overall view of the various properties of a match, rather than the individual view shown for each field in the ResultItemField object. The properties are as follows –

int Confidence
int Start
int End
string Type
string Info
float MatchScore

FormatInfo

This object represents the format information available for a particular match found during a call to the Process method. It gives an indication of whether or not custom formatting was applied, and if so which formatting (file / country) was used. The properties are as follows –

bool CustomFormattingUsed
string FormattingConfigurationName
string FormattingCountryName
string FormattingInfoMessage