
QAS Pro

API

Disclaimer

E&OE. Information in this document is subject to change without notice. QAS Limited reserves the right to revise its products as it sees fit. This document describes the state of this product at the time of its publication, and may not reflect the product at all times in the future.

Use of this Product is subject to the terms of the QAS evaluation licence in the case of an evaluation, and to the QAS Licence Terms & Conditions in the case of full commercial use of the product, and will also be subject to Data Provider terms. By downloading, installing or using this product, you agree to comply with all the relevant terms. Please refer to these terms for all permitted uses and applicable restrictions on the use of the product.

The liability of QAS Limited with respect to the documentation and the licensed programs referred, are set out in that software licence agreement. QAS Limited accepts no liability whatsoever for any use of the documentation or the licensed programs by any person other than a permitted user under the software licence agreement.

Copyright

All copyright and other rights in this manual and the licensed programs described in this manual are the property of QAS Limited, save for copyright in data in respect of which the copyright belongs to the relevant data provider. For details of data ownership, see the Data Guides located on the Data installation CDs.

No part of this manual may be copied, reproduced, translated or reduced to any electronic medium or machine readable form without the written consent of QAS Limited.

Microsoft, Word and Windows are trademarks of Microsoft Corporation.

© QAS Ltd. 1999 - 2011

For resolutions to common issues, answers to Frequently Asked Questions, and hints and tips for using our products, visit the Experian QAS Support Site at:
<http://support.qas.com>

To contact Experian QAS Technical Support, use the details provided below for your region.

Experian QAS Support Website:
support.qas.com

Experian QAS Global Website:
www.qas.com

UK

QAS Ltd
George West House
2-3 Clapham Common North Side
LONDON
SW4 0QL
UNITED KINGDOM
Tel: +44 (0) 20 7498 7777
Fax: +44 (0) 20 7498 0303
Technical Support
Tel: +44 (0) 20 7498 7788
E-mail: uk.support@qas.com

USA and Canada

QAS
125 Summer St Ste 1910
Boston MA 02110-1615
USA
Tel: +1 888 322 6201
Fax: +1 888 882 7082
Technical Support
Tel: +1 888 712 3332
E-mail: us.support@qas.com

France

Experian QAS
Tour Europlaza
20 avenue André Prothin
92927 Paris la Défense cedex
FRANCE
Tel: +33 (0) 1 70 39 45 55
Fax: +33 (0) 1 70 39 43 21
Technical Support
Tel: +33 (0) 1 70 39 43 43
E-mail: fr.support@qas.com

Australia

QAS Pty Ltd
L 15 100 Miller Street
NORTH SYDNEY NSW 2060
AUSTRALIA
Tel: +61 (0) 2 8907 7200
Fax: +61 (0) 2 8907 7298
Technical Support
Tel: +61 (0) 2 8907 7272
E-mail: ap.support@qas.com

Singapore

QAS Ltd
1 Maritime Square
#10-33A/B HarbourFront Centre
Singapore 099253

Tel: +65 6593 7500
Fax: +65 6593 7598
Technical Support
Tel: +61 (0) 2 8907 7272
E-mail: ap.support@qas.com

Netherlands

Experian QAS
Kantoorgebouw 't Schip
Verheeskade 25
2521 BE DEN HAAG
THE NETHERLANDS
Tel: +31 (0) 70 440 4700
Fax: +31 (0) 70 440 4710
Technical Support
Tel: +44 (0) 20 7498 7788
E-mail: nl.support@qas.com

For all other countries

QAS Ltd
George West House
2-3 Clapham Common North Side
LONDON
SW4 0QL
UNITED KINGDOM
Tel: +44 (0) 20 7498 7777
Fax: +44 (0) 20 7498 0303
Technical Support
Tel: +44 (0) 20 7498 7788
E-mail: support@qas.com

Experian QAS Global Website
www.qas.com
Experian QAS Support Website
support.qas.com

Version 6.85 Revision 1
December 2011

Contents

Introduction.....	1
Conventions.....	2
Accompanying Documentation.....	2
Data Guide.....	2
API Manual.....	3
Getting Started Guide.....	3
Help Files.....	3
Client/Server Documentation.....	3
Upgrade Guide.....	4
Licences.....	5
Expiry Warnings.....	5
Evaluations.....	5
Street Level Validation.....	5
Installing QAS Pro API.....	7
System Requirements.....	7
Windows Installation.....	8
UNIX Installation.....	11
Installing And Updating Data.....	11
Windows.....	11
UNIX.....	12
Data Updates.....	13
Getting Started With QAS Pro API.....	14
Which Version Of The API Should I Use?.....	16
Sample Code.....	17
Testing Your Primary API Installation.....	17
Searching With A Test Harness.....	18
Testing Your UI API Installation.....	20

Running The Test Harness.....	21
A Typedown Search.....	22
A Single Line Search.....	23
A Key Search.....	24
Searching With QAS Pro.....	25
Which Search Method Should I Use?.....	26
Typedown Searching.....	26
Searching For A Residential Address.....	27
Searching For An Organisation Address.....	28
Searching For A PO Box Address.....	28
Typedown Troubleshooting.....	28
Single Line Searching.....	28
Wildcard Searching.....	29
Searching With Partial Addresses.....	31
Identifying Address Elements.....	32
Key Searching.....	32
Searching On A Utility Meter Number.....	33
Searching For A UPRN.....	33
Alias Matching.....	34
United Kingdom.....	34
Australia.....	35
New Zealand.....	36
Retrieving DataPlus Information.....	37
Retrieving Multiple DataPlus Values.....	37
Barcoding.....	40
QAS Pro User Interface.....	41
Menu Bar.....	42
Toolbar.....	43
Search Area.....	44
Results Area.....	45
Partial Address Bar.....	45
Status Bar.....	46
Select Button.....	47
Picklist of Returned Addresses.....	47
Picklist Symbols.....	49

Typedown Search Results.....	50
Single Line Search Results.....	50
Selecting a Picklist Item.....	51
Order of Picklist Items.....	51
Displayed Postcodes.....	51
Returning An Unrecognised Address.....	52
Address Edit Screen.....	53
Setting QAS Pro Options.....	53
Selecting A Search Method.....	54
Setting The Search Options.....	55
Selecting A Dataset.....	55
Selecting An Address Layout.....	56
QAS Healthcoder.....	59
Using QAS Healthcoder.....	59
AddressBook.....	60
+<search prefix>:<data file name>.....	60
AddressBookData=<file location>.....	61
Data Types.....	63
Function Return Values.....	63
Parameters (Input).....	63
Parameters (Output).....	64
Calling Functions From Languages Other Than C.....	64
NULL Termination.....	64
Passing By Value Or By Reference.....	65
Returned Strings.....	65
Example Of Data Types.....	66
Primary API Reference.....	67
Pseudocode Example Of QAS Pro API.....	67
Main Function.....	67
Display Error Function.....	69
User Action Function.....	70
Display Results Function.....	70
Select Result Function.....	71
Return Address Function.....	71
Handling Errors.....	71

API Instances.....	73
Flags Returned.....	73
Automatic Stepping And Formatting.....	75
Asynchronous Searching.....	76
API Function Reference.....	78
General Error Scenarios (All Functions).....	82
QA_CancelSearch.....	83
QA_Close.....	84
QA_EndSearch.....	85
QA_ErrorMessage.....	86
QA_FormatExample.....	87
QA_FormatResult.....	89
QA_GenerateSystemInfo.....	91
QA_GetActiveData.....	92
QA_GetActiveLayout.....	93
QA_GetData.....	94
QA_GetDataCount.....	96
QA_GetEngine.....	97
QA_GetEngineOption.....	98
QA_GetEngineStatus.....	100
QA_GetExampleCount.....	101
QA_GetFormattedLine.....	102
QA_GetLayout.....	104
QA_GetLayoutCount.....	106
QA_GetLicensingCount.....	107
QA_GetLicensingDetail.....	109
QA_GetPrompt.....	112
QA_GetPromptStatus.....	114
QA_GetResult.....	116
QA_GetResultDetail.....	120
QA_GetSearchStatus.....	124
QA_GetSearchStatusDetail.....	127
QA_GetSystemInfo.....	130
QA_Open.....	131
QA_Search.....	133
QA_SetActiveData.....	134

QA_SetActiveLayout	135
QA_SetEngine.....	137
QA_SetEngineOption.....	138
QA_Shutdown.....	141
QA_StepIn.....	142
QA_StepOut.....	143
User Interface API Reference	145
Handling Client/Server Errors.....	145
Pseudocode Example Of QAS Pro API.....	146
API Function Reference.....	148
QAProWV_UICountryCount.....	150
QAProWV_UIGetActiveCountry.....	151
QAProWV_UIGetActiveLayout.....	153
QAProWV_UIGetCountry.....	154
QAProWV_UIGetFlags.....	156
QAProWV_UIGetLayout.....	157
QAProWV_UIGetResult.....	159
QAProWV_UIGetResultDetail.....	161
QAProWV_UILayoutCount.....	163
QAProWV_UILayoutLineElements.....	164
QAProWV_UIResultCount.....	167
QAProWV_UISearch.....	168
QAProWV_UISetActiveCountry.....	169
QAProWV_UISetActiveLayout.....	170
QAProWV_UISetFlags.....	171
QAProWV_UIShutdown.....	172
QAProWV_UIStartup.....	173
Low-Level System Functions.....	177
QAErroRMessage.....	178
QAErroRLevel.....	179
QASystemInfo.....	180
API Configuration.....	185
Overview.....	185
Format Of A Configuration File.....	186
The Configuration Process.....	188

Dataset Installation Settings.....	189
InstalledData.....	189
DataMappings.....	191
Warning Settings.....	192
NotifyDataWarning.....	192
NotifyLicenceWarning.....	193
Output Address Format Settings.....	194
AddressLineCount.....	195
AddressLineN.....	196
CapitaliseItem.....	198
AbbreviateItem.....	199
SeparateElements.....	200
ElementSeparator.....	201
ElementExtras.....	202
TerminateLines.....	203
LineTerminator.....	204
ExcludeItem.....	205
FlattenDiacritics.....	206
CDFVariation.....	207
Comment.....	208
MultiValueDPSeparator.....	209
Error Logging Settings.....	210
LogFile.....	210
LogErrors.....	212
Search Options And Results Settings.....	213
EngineTimeout.....	213
SLMaxMatches.....	214
ShowAllThreshold.....	215
UPIThreshold.....	217
EngineIntensity.....	218
MultiElementLabels.....	219
ForceAccept.....	220
OemCharacterSet.....	221
Informational Prompt Settings.....	222
NoMatchesMessage.....	222
Other INI Keywords.....	223

Error Code Listing..... 225

Utilities..... 245

 Data Checker..... 245

Introduction

The QAS Pro API is a suite of functions that you integrate into your own applications. Once successfully integrated, QAS Pro takes partial address information that you type in and returns a full, valid address.

You have been supplied with two versions of the API: the Primary (low-level) API and the User Interface API. This manual deals with everything which relates to the QAS Pro Primary API and the User Interface API functions, test harness and front-end screens.

If you are using the QAS Pro API as a client with the QAS Pro server, please read the accompanying QAS Pro Client / Server documentation first.

QAS Pro API uses separate datasets and additional datasets, between which you can switch to retrieve address information from several datasets. You can also create groups of multiple additional datasets to search on simultaneously. You can make use of three different search techniques to find the address that you want.

The remainder of this chapter helps you get started with installing and integrating QAS Pro API and datasets. The other chapters in this guide deal with verifying your installation, methods of searching with QAS Pro API, details of the Experian QAS data types, low-level system functions, and QAS Pro API functions, how to use the User Interface, and how to configure QAS Pro API to return addresses in the way that you want.

Conventions

The table below defines the style conventions used to distinguish features of QAS Pro API in this manual.

Example	Convention
<code>int QAProWV_ Open</code>	Sample code, pseudocode, function prototypes and configuration settings look like this.
<i>viHandle</i>	Parameter names are shown in italics (when not part of sample code).
Press Enter Press Cursor up	Key presses are shown in bold (Enter is the same as Return or Carriage return).
QAProWV_ UIStartup	API functions appear in bold type (when not part of sample code).

Accompanying Documentation

This section provides a comprehensive list of the documentation supplied with QAS Pro, and where it can be located.

Data Guide

A Data Guide is supplied with each dataset that you purchase. This guide provides dataset-specific information and search tips for each dataset, and should be used in conjunction with the other documentation supplied with QAS Pro.

Under Windows, you have the option to install the Data Guide during data installation. The guide is installed to C:\Program Files\QAS\Data Guides by default. If you choose not to install the guide, it can be accessed from the Docs folder on the data CD/DVD.

Under UNIX, you should copy across the Docs folder to a location of your choice.

API Manual

The API manual describes all the functions and data types that make up the QAS Pro User Interface API and Primary API, suggests how you can make use of the functions, and describes the searching techniques and configuration options available.

The API manual can be accessed via the **index.htm** file, which is located in the Docs folder on the QAS Pro Installation CD.

Getting Started Guide

If you purchased the standalone version of QAS Pro Plug & Go, a Getting Started Guide is supplied with the product. This guide is aimed at the end-users of QAS Pro, and is an introduction to retrieving and configuring addresses.

The guide can be accessed via the **index.htm** file, which is located in the Docs folder on the QAS Pro installation CD.

Help Files

In addition to this manual, QAS Pro is supplied with a Configuration Editor help file. This file can be accessed in two ways:

- by pressing **F1** when using the Configuration Editor;
- by selecting **Contents and Index** from the **Help** menu when using the Configuration Editor.

Client/Server Documentation

If you have purchased the Client/Server version of QAS Pro API, you are supplied with a Client/Server manual and an Administrator Console Help file.

The Client/Server manual can be accessed via the **index.htm** file, which is located in the Docs folder on the QAS Pro installation CD.

The Administrator Console Help file can be accessed by:

- pressing **F1** when using QAS Pro;
- selecting **Contents and Index** from the **Help** menu on the Administrator Console User Interface.

Upgrade Guide

This document details the new features in this version of QAS Pro. It also highlights key integration and compatibility issues of which you should be aware, and provides a step-by-step guide to upgrading an existing installation.

The Upgrade guide can be accessed via the **index.htm** file, which is located in the Docs folder on the Installation CD.

Licences

You will receive a licence key for each combination of data and product that you purchase. Failure to enter a valid licence key means that the product will be unable to use installed data.

If you are running QAS Pro on Windows, you will be prompted to enter your licence key(s) during the installation process. Alternatively, you can use the Licence Manager dialog of the Configuration Editor to add and delete licence keys. See the Configuration Editor Help program for more information about the Licence Manager.

If you are running QAS Pro on UNIX, you must enter your licence keys into the **qalcn.ini** file. You should edit the file manually and save it.

Expiry Warnings

If you have data installed which is nearing expiry, the **Dataset Expiry Notification** screen will be displayed when you first open the Configuration Editor. This screen will warn you if your licences have expired or are due to expire.

Evaluations

When you have an evaluation version of an Experian QAS product or data, evaluation licence keys are provided that set time limits on the usage of the data. To continue using the product and data after these time limits have been reached, you must purchase a full licence.

Street Level Validation

The restrictions that are put in place by Street Level Validation (SLV) are controlled by the Licence key, meaning that it is quick and easy to upgrade to the full functionality of QAS Pro. When you upgrade, a new license will be sent out to you that will remove the restrictions.

For more information on SLV, see the GBR Data Guide.

Installing QAS Pro API

If you are installing QAS Pro API as a client, please follow the setup instructions in the QAS Pro Client / Server manual. This chapter only describes the process of installing QAS Pro API as a standalone product.

System Requirements

The QAS Pro Primary API is available for the Windows and UNIX platforms listed below. The User Interface API is available for Windows platforms only. For more information see "Which Version Of The API Should I Use?" on page 16.

Windows

- Windows XP, Windows Server 2003, Windows Server 2008, Windows Vista or Windows 7.
- Enough free hard disk space to store all the data files. See the Data Guide supplied with each dataset for details about how much space you need.

UNIX (Primary API only)

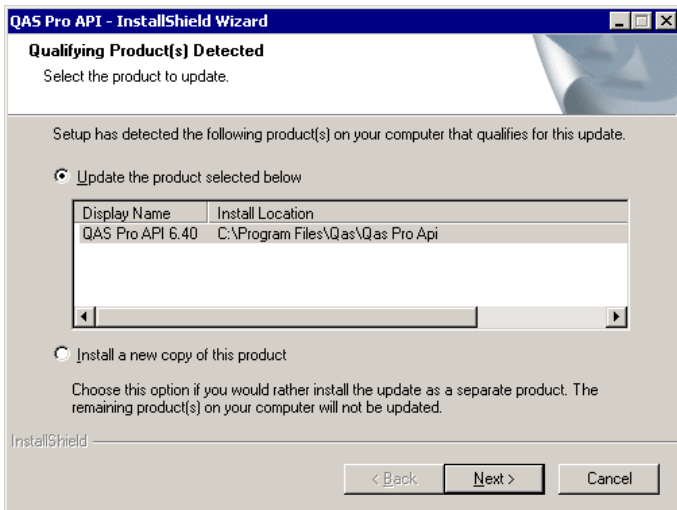
- Most POSIX threaded UNIX-like systems (such as Solaris, Linux etc.)
- Access to a CD-ROM drive (for installation only).
- Enough free hard disk space to store all the data files. See the Data Guide supplied with each dataset for details about how much space you need.

Windows Installation

1. Insert the QAS Pro API CD into your CD-ROM drive.
2. The setup program should start up automatically.

If it does not start, click the **Start** menu and select **Run**. In the dialog box that appears, type **d:\setup**, where **d** is the drive letter of your CD-ROM drive, and press **Enter**.

3. Once the installation program starts, follow the on-screen instructions to install QAS Pro.
4. If you have previously run the QAS Pro installer, the **Qualifying Product(s) Detected** screen will be displayed. This screen allows you to choose to update the version of QAS Pro that is currently installed. Alternatively, you can keep the previous version and install this version alongside it.

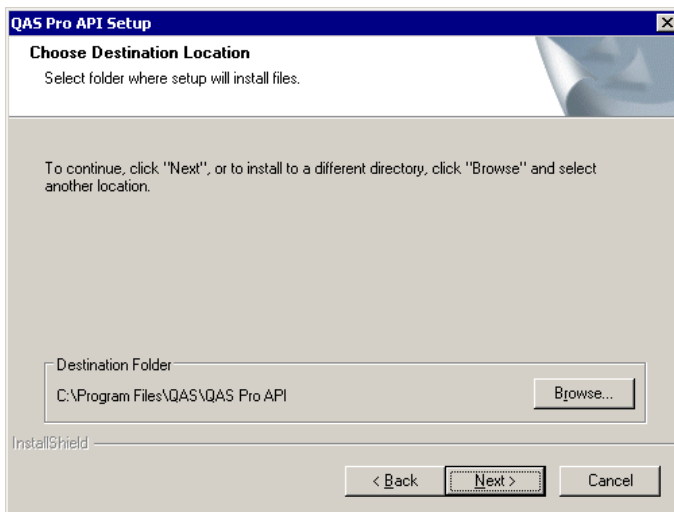


5. On the **Licence Agreement** dialog, the licence agreement is displayed.

You should read the licence agreement, and if you are happy with the terms and conditions select the **I accept the terms of the licence agreement** option.

Click the **Next** button to proceed.

6. If you choose to install a new copy of the product, the **Choose Destination Location** screen is displayed:



If you have previously installed any other version of QAS Pro, the folder where this version is located on your system will automatically be selected as the default destination folder. Therefore, if you do not want to overwrite the contents of this folder, you must browse to a different location.

7. The **Setup Configuration** screen will be displayed if you have previously installed QAS Pro on your system.



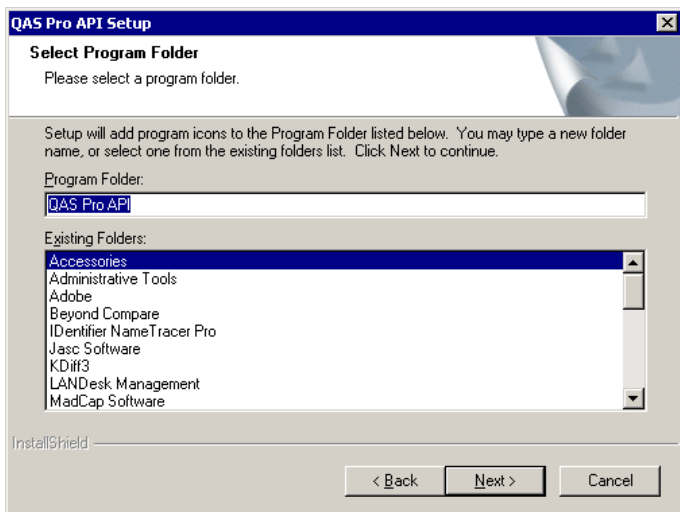
Click the **Copy my setting from** option and choose an existing installation, if you want to use the configuration settings of an existing Experian QAS product.

Click the **Install with default settings** option if you want to use the default configuration settings for QAS Pro.

8. On the **Setup Type** screen, select one of the **QAS API Standalone** options (unless you have already installed the QAS Pro Server). This installs the full program rather than just the QAS Pro client.



9. On the **Select Program Folder** screen, specify the location of the program files by typing a new folder name, or selecting an existing folder from the list.



10. On the **QAS Electronic Updates Screen**, check the box to install **QAS Electronic Updates**, which will allow you to update your Experian QAS programs and data over the internet rather than with CDs or DVDs.
11. After the product has installed, you can choose to install data. Check or uncheck the **Install QAS Data** checkbox as required.
12. Click **Finish**.

If you chose to install data, the data installation process will begin. See "Installing And Updating Data" on page 11 for details.

Otherwise, your installation is complete.

UNIX Installation

To install QAS Pro API on UNIX, just copy the appropriate program files from the CD-ROM.

Installing And Updating Data

If you chose to install Experian QAS Data on the final product installation screen, follow the on-screen instructions to install the data. Alternatively, you can run the data installer from the CD at a later date. To ensure that all datasets are compatible, all data CDs for a country must be the same version, and should be installed at the same time.

A set of files is installed for each dataset you need to support. For example, there are four data files installed for Australia: aus.dts, aus.tpx, aus.zlx and aus.zlb.

Before you begin the data installation process, you should ensure that you have the despatch note that is supplied with QAS Pro. The despatch note contains all of the licence keys for each dataset that you have purchased. When you install data for the first time, you are prompted to insert the licence key for each dataset that you want to install. For more information about licences, see the Licence Manager section of the Configuration Editor online help.

Windows

To install or update data on Windows, follow these steps:

1. Insert the first CD-ROM or the DVD into the relevant drive, click the **Start** menu and select **Run**.
2. In the dialog box that appears, type **d:\setup**, where **d** is the drive letter for your CD-ROM or DVD drive, and press **Enter**.
3. Follow the instructions on the screen to install new or replace existing datasets.

UNIX

You should copy all files from the CD-ROM or DVD **data** folder. Data files must be in lower case and in the following format: usa.zlx, usa.dts, usagcd.dap.

The first time that you carry out a data update, you will need to add `InstalledData` and `DataMappings` settings to the `qawserve.ini` file.

The `qawserve.ini` file is located in the **apps** folder. Open the file using a plain text editor such as vi or emacs. Under the `[QADefault]` section, locate the settings `InstalledData` and `DataMappings`. These settings contains a list of the datasets you have installed. See "Dataset Installation Settings" on page 189 for more information.

Add a line to each setting for each new dataset that you are installing, or update the path for any dataset you are updating.

Installing USA Names Data On UNIX

United States with Names data is supplied on four CDs by default, but may also be supplied on DVD.

If you are installing this data from a DVD, you should copy the relevant files from the **data** folder. Data files must be in lower case and in the following format: usa.zlx, usa.dts, usagcd.dap.

If you are installing this data from a CD, you must re-assemble the `usanam.ads.00x` files from all CDs after installing. This means that you cannot simply copy the required files from a subfolder on the CDs to the appropriate folder on your computer. Instead, QAS Pro for UNIX includes a shell script that automates the copy and reconstruction process, and ensures that you enter the product and data CDs in the correct order. In addition, the script will warn if you are overwriting files, and will remove part-constructed files if you choose to abandon the installation.

To install the United States with Names dataset from a CD on a UNIX platform, follow these steps:

1. Using the command line prompt, locate the contents of the first data CD.
2. Run the following command:


```
sh qascopydata [destination] [source]
```

The source for the data files is optional. The shell script will assume that they are in numbered subdirectories within the current working directory, and will prompt for a new source if a numbered subdirectory is not found; for example if you need to change the CD.

3. The shell script installer will check whether files already exist in the destination folder and will warn the user if they do, giving the option to delete them and continue or to abort the process.
4. If the USA names files were not present in the destination folder or if you have decided to delete them and continue, the installer will begin copying files.
5. Once the shell script has finished appending a sub-component file to the data file that is being reconstructed, it will look for the next sub-component file. If it can find it, the installer will continue without prompting; otherwise it will request the next CD.

If you do not provide the CD, the installer gives you the choice of specifying a new path to the component data files, re-trying the current one or aborting the procedure.

Aborting will delete all files that the installer has placed on the system.

You are still responsible for maintaining correct ini files and updating qawserve.ini and qalcn.ini as appropriate.

Data Updates

Experian QAS provides periodic updates of datasets as and when updated data is available. These updates are supplied to you on CD-ROM; follow the instructions in the installation program to replace your existing datasets.

Datasets expire after a certain period of time, at which point you must install an update. The low level API function **QA_GetLicensingDetail** (see page 109) tells you how many days are left before a dataset expires. By default, a dialog will be displayed when you start QAS Pro, which lists all the datasets that are due to expire soon. You can disable this dialog using the qaattribs_NOLICENSINGDLG flag (see **QAProWV_UIStartup** on page 173).

Getting Started With QAS Pro API

This section provides a few hints on how to start integrating QAS Pro API with your own application.

The basic steps you should take for a successful integration are as follows:

1. Understand How QAS Pro Searches

The chapter "Searching With QAS Pro" (see page 25) describes how QAS Pro API searches on your addresses. Reading this should clarify the values that are returned by some of the API functions.

2. Run The Test Harnesses

Running one of the test harnesses supplied with the API should verify that you have installed the API correctly. It will also give you an idea of what QAS Pro API can do, and the type of results it can produce.

3. Refer to the Pseudocode and Sample Code

The "Pseudocode Example Of QAS Pro API" on page 67 demonstrates a possible interpretation of the API functions. Samples are available in C.

4. Use the API Functions

For a complete listing of QAS Pro API functions see the "API Function Reference" on page 78. You should choose whether the Primary API or the User Interface API is best suited to your requirements.

It is recommended that you integrate the API in stages, beginning with the **QA_Open**, **QA_Close** and **QA_Shutdown** functions, followed by address search and retrieval facilities. Any other functions can be added in the appropriate places.

You should also make use of the system functions, especially **QAErrorMessage** and **QAErrorLevel** or **QA_ErrorMessage**. These functions enable you to see the description and severity of any errors that occur, and as such should be called after any function that has returned an error. See the full "Error Code Listing" on page 225.

When you wish to run your integrated application, you should ensure that the following files are in the same directory as the application executable:

Primary API	UI API
qaworld.ini QAS Pro API configuration file	qaworld.ini QAS Pro API configuration file
qawserve.ini Data file locations and other application configuration settings	qawserve.ini Data file locations and other application configuration settings
qaupied.dll QAS Pro API DLL	qauwved.dll QAS Pro API DLL
qaupied.rev QAS Pro API DLL revision file	qauwved.rev QAS Pro API DLL revision file
qaupied.0xx QAS Pro API DLL language resource file (where 'xx' represents the language code)	qauwved.0xx QAS Pro API DLL language resource file (where 'xx' represents the language code)
	qauwv001.dll QAS Pro API (UI) language GUI resource file (English-United States)
qalcl.dat Locale information file	qalcl.dat Locale information file
qalcn.ini Licence information file	qalcn.ini Licence information file

5. Configure your API

Before running your integrated API, you need to give QAS Pro API the following information:

- The format of your output addresses
- The whereabouts of your datasets(s)

This information must be specified in the configuration file (see "Overview" on page 185).

You can test your integration by trying some of the search examples provided in the Data Guide for your dataset.

Which Version Of The API Should I Use?

If you are running Windows, and the QAS Pro front-end meets your input and output requirements, it is recommended that you integrate the User Interface API. This is the easiest method of gaining maximum functionality with the minimum of effort, as programming with these functions is simpler.

QAS Pro User Interface API can be integrated with a minimum of five functions:

- **QASProWV_UIStartup** (see page 173)
- **QASProWV_UISearch** (see page 168)
- **QASProWV_UIResultCount** (see page 167)
- **QASProWV_UIGetResult** (see page 159)
- **QASProWV_UIShutdown** (see page 172)

These functions start up and shut down the QAS Pro API, perform searches and return full addresses. If you wish to provide additional functionality, such as viewing and selecting address layouts and datasets, the appropriate functions can be added to your program.

The User Interface API is available for Windows Server 2003 SP2, Windows Server 2008, Windows XP SP3 and Windows Vista SP1.

If, however, you need to use your own front-end screens, or you are running UNIX, you should integrate the QAS Pro Primary API.

The QAS Pro Primary API provides you with the functionality you need to integrate QAS Pro seamlessly into your application. It is up to you how searches are performed and results are returned, and you are responsible for any user interface that is required. Although these functions require more programming on your part, they also give you great flexibility in integrating the API.

Even if you choose to integrate the QAS Pro Primary API, you might find it useful to look at the sample code provided for the User Interface API, to get an idea of the available functionality.

The QAS Pro Primary API is available for Windows Server 2003 SP2, Windows Server 2008, Windows XP SP3, Windows Vista SP1 and UNIX.

Sample Code

The sample code should be seen as the starting point for integration, and should be tailored according to the type of integration that you are performing.

Sample code is used to demonstrate best practice and includes the features that are likely to be required by the majority of users.

The single-line sample code is applicable for environments where it is not possible to detect and act upon characters as you type. This means that the picklist is not automatically updated as you enter refinement text, and all processing (searching, stepping in etc.) is performed once you have pressed **Enter**.

If you are running the Primary API test harness in client/server mode, any loss of connection between client and server results in the active search being cleanly aborted. The next search attempt triggers a reconnect, switching to any defined backup servers if available. Refer to the Client / Server documentation for more information.

Testing Your Primary API Installation

The Primary API is supplied with a simple text-based application, called `qs_sl.exe` (`qs_sl` on UNIX), and with sample code. Together these can be used to verify that you have installed QAS Pro API correctly, and to demonstrate some of the API's key functionality.

The test harness is not intended to be used as a commercial application.

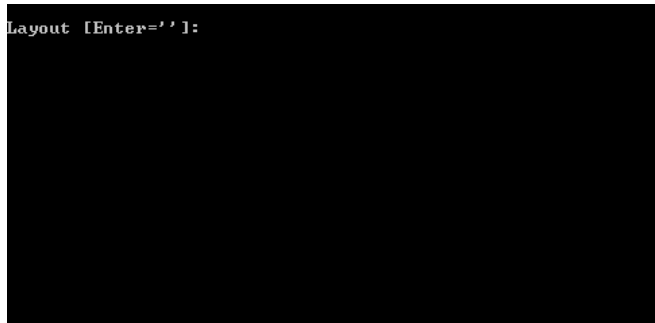
The test harness enables you to obtain matching addresses and picklists from input address information that you type in on the command line.

The examples in this section use the C version of the test harness.

Searching With A Test Harness

If you are using Windows, run the test harness from the shortcut in the Program Group which was created when you installed QAS Pro API.

The test harness appears, looking similar to this:

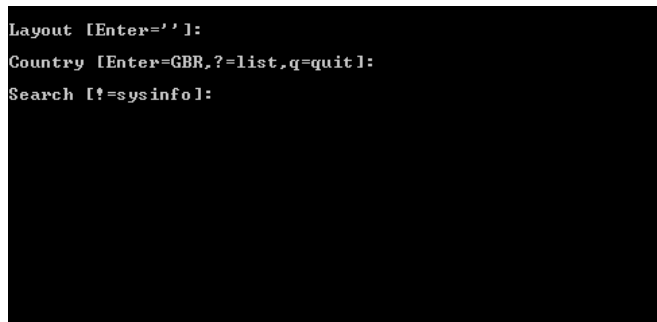


To perform a Single Line search, follow these steps:

1. Press **Enter** to select the default layout.
2. Press **Enter** again to select the default dataset.

Alternatively, type **?** and press **Enter** to display a list of all datasets with their identifiers. Type the relevant code (for example, DEU for Germany, AUS for Australia) and press **Enter** to select the related dataset.

3. Once you have selected the dataset you want to work with, the screen will look like this (in this example, the GBR dataset was selected):



4. Enter a search string, separating each part from the next with a comma, and press **Enter**. For example (if you are using the GBR dataset):

linden gardens, london

For examples of searches with other datasets, see the Data Guide supplied with each dataset.

The test harness returns the following:

```
Layout [Enter=' ']:
Country [Enter=GBR,?=list,q=quit]:
Search [!=sysinfo]: linden gardens, london
Level 1: "Linden Gardens, LONDON" - 181 matches
1 : + Continue typing <or select to show all matches>
Enter building number/name or organisation [#n or text]: _
```

There are 181 possible matches, as shown by the Match Count.

5. Type **30** and press **Enter** to refine the picklist and to therefore reduce the number of matches. Press **Enter** at any prompt to remove the refine and return the list to the way it was.

```
Layout [Enter=' ']:
Country [Enter=GBR,?=list,q=quit]:
Search [!=sysinfo]: linden gardens, london
Level 1: "Linden Gardens, LONDON" - 181 matches
1 : + Continue typing <or select to show all matches>
Enter building number/name or organisation [#n or text]: 30
Level 1: "Linden Gardens, LONDON" - 2 matches
1 : 30      W2 4ES
2 : 30      W4 2EN
Enter building number/name or organisation [#n or text]: _
```

There are now two matches displayed. Note that they are numbered from 1 to 2 in the picklist.

6. Type **#1** and press **Enter** to select the first picklist entry.

The full address is returned:

```
Level 1: "Linden Gardens, LONDON" - 181 matches
1 : + Continue typing (or select to show all matches)
Enter building number/name or organisation [#n or text]: 30
Level 1: "Linden Gardens, LONDON" - 2 matches
1 : 30          W2 4ES
2 : 30          W4 2EN
Enter building number/name or organisation [#n or text]: #1
          : 30 Linden Gardens
          :
          Town: London
          County:
          Postcode: W2 4ES
Country [Enter=GBR,?=list,q=quit]:
```

If your text appears odd, this may be because any diacritics in the text (for example, accents and umlauts) are not displaying correctly in the DOS Console. You can remove diacritics with the `OemCharacterSet` setting (see "Search Options And Results Settings" on page 213).

Testing Your UI API Installation

The QAS Pro User Interface API is supplied with a very simple application called `qs_ui.exe`, and with sample code. Together these can be used to verify that you have installed QAS Pro API correctly, and to demonstrate some of the API's key functionality.

The test harness enables you to obtain matching addresses and picklists from input address information that you type in.

See "Sample Code" on page 17 for more information about the purpose of the sample code. The UI integration scenario is the preferred solution on Windows because it uses the standard QAS Pro dialog, thereby minimising the size and complexity of the integration, and meaning that you have access to all available search modes, as well as the standard User Interface functionality (selection of layouts and databases, menus, toolbars etc.).

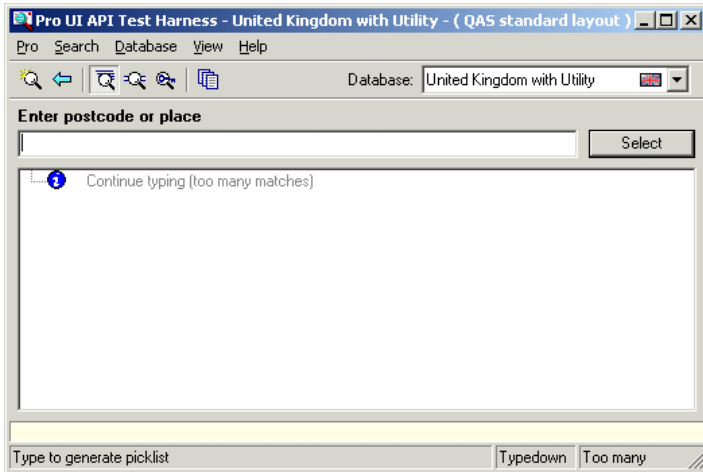
If you are running the UI API test harness in client/server mode, any loss of connection between client and server results in the display of appropriate warnings. Refer to the Client / Server documentation for more information.

The examples in this section use the C version of the test harness.

Running The Test Harness

Run the test harness from the shortcut in the Program Group which was created when you installed QAS Pro API.

The test harness appears, looking similar to the example shown below:



This is the QAS Pro User Interface. For a comprehensive description of all of its aspects and options, see the chapter "QAS Pro User Interface" on page 41.

The following sections describe using the test harness to undertake a Typedown search, a Single Line search and a Key search.

A Typedown Search

In the example below, you are searching for 7 Sand Lake Road in Orlando, FL, USA.

For examples of searches from other datasets, see the Data Guide supplied with your data.

1. Select the **Typedown search** button on the toolbar or press **Ctrl+T**.
2. Enter this search string:

orlando

The test harness starts searching as soon as you type the first character. After typing 'orlando', this is what the test harness returns:



There is only one match for this search string.

3. Click the **Select** button (or press **Enter**) to step into the picklist item.
4. Type **sandl**.

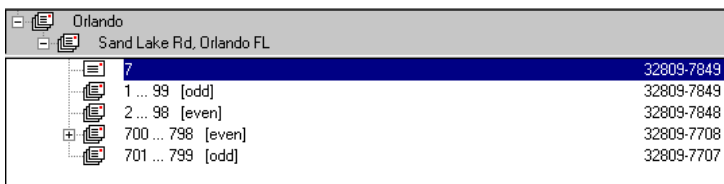
In this example it is the second match in the list that you want.

5. Use the **arrow down** cursor key to move the focus to Sand Lake Rd Orlando FL. Click the **Select** button (or press **Enter**) to step into Sand Lake Rd.

There are too many matches to display in a picklist.

6. Type **7**.

This picklist is returned:



The match you want is at the top of the picklist.

QAS Pro also returns several other matches, and, where appropriate, has split the matches into odd and even number ranges.

7. Click on the **Select** button or press **Enter**.

The full address is returned to the address edit screen.

8. Accept the search to return the result from the API.

A Single Line Search

1. Click on the **Single Line search** button on the toolbar, or press **Ctrl+S**.
2. Enter a search string, separating each part from the next with a comma, and select the **Search** button (or press **Enter**). For example (if you are using the Australia dataset):

65 Rushton St, Carnarvon, WA

This is what the test harness returns:

Address	65 Rushton St
	CARNARVON WA 6701

As the test harness has only found one address which precisely matches your search criteria, it returns it directly to the address edit screen.

3. Accept the search to return the result from the API.

For examples of searches from other datasets, see the Data Guide supplied with your data.

A Key Search

In the example below, you are searching for an address using a gas meter number.

1. Select the **Key search** button on the toolbar or press **Ctrl+K**.
2. Type in the gas meter number, **2481849308**, and press **Enter**.
3. The correct address is returned:

Address	6 Valley View
	Highley
Town	BRIDGNORTH
County	Shropshire
	WV16 6EF

Searching With QAS Pro

QAS Pro has the following search modes:

- Typedown searching
- Single Line searching
- Key Searching

Any of these methods allow you to capture a full address with just a few keystrokes.

Typedown searching starts with the most general address element and, once that has been found, moves on to more specific parts of the address.

Typedown is the more useful search option if you are sure about the validity of the address information. For example, if you are taking address details over the phone, you can enter the caller's postcode and then, if required, you can search for the correct street and building number.

See "Typedown Searching" on page 26 for more information.

Single Line searching requires you to enter one or more address elements, each separated by a comma, in the order that they would appear on an envelope (for example, the street name followed by the town).

Single Line searches can use a variety of techniques to return the correct address from incomplete or misspelled information. For example, if you were verifying illegible handwritten addresses, Single Line searches may return the best results.

See "Single Line Searching" on page 28 for more information.

Key Searching allows you to search on key elements of the data. Using Key Searching, you can, for example, search on gas and electricity meter numbers or serial numbers using the United Kingdom Utility dataset, or on a Unique Property Reference Number (UPRN) using the Gazetteer dataset.

See "Key Searching" on page 32 for more information.

Which Search Method Should I Use?

It is strongly recommended that you integrate Typedown searching as the primary searching method, and use Single Line searching as a secondary option, especially if you have multiple datasets. This is because Typedown provides a consistent method of entering search information across all datasets (for example, a postal/ZIP code followed by a street name), and reduces the likelihood of errors by verifying data as it is entered.

While Single Line searching allows you to enter any combination of address elements, you need some knowledge of each dataset's address information in order to decide which part of the address will return the most search results, and hence provide optimum performance. For example, entering a postal code for a UK address will return an average of fifteen possible full address matches. However, an Australian postal code could cover several localities (which corresponds to thousands of addresses).

Key searching allows you to search on key elements of the data. Key searching can only be used with certain datasets that contain a logical reverse search key. For example, United Kingdom with Gas data contains a Meter Number (MPRN) that can be searched on.

Typedown Searching

In Typedown searching, you start by typing the most general address element (for example, a postcode, county, town or locality in the United Kingdom) and, once that has been found, you move on to more specific parts of the address (for example, the street name, property number, organisation name or PO Box).

QAS Pro API searches on the string that you enter, which can be as little as one character. When you use Typedown searching, QAS Pro will always look for an exact match first. If an exact match is found, and other close matches are below the match threshold, all of the matches are returned. However, if the close matches exceed the match threshold, only exact matches are returned.

Note that any mixture of upper and lower case characters can be used, as QAS Pro API does not differentiate between upper and lower case text.

It is usually faster and more efficient to type in the postcode or ZIP code, than it is to use other address elements, such as a town name.

The following sections describe the types of search you can perform using the Typedown mode.

Searching For A Residential Address

This type of search usually involves three stages.

Typically, after an initial search on a place or postal/ZIP code, QAS Pro will look for street names. Once you have selected the street name that you want, you can enter a property number to return the full address.

You cannot type in the property number followed by the street name, as you can with Single Line searching, because Typedown searches cannot match on numbers until you have selected a street name.

Searching For An Organisation Address

This type of search usually involves two stages.

After the first stage of searching on a place name or postal/ZIP code, QAS Pro API will look for organisation and street names. Once you have found the organisation that you are looking for, QAS Pro API returns the name and full address.

Searching For A PO Box Address

This type of search usually involves three stages.

After the first stage of searching on a place name or postal/ZIP code, QAS Pro will look for the PO Box type. Once you have stepped into this, you can enter a PO Box number to return the final address.

Typedown searching for PO Box addresses is not recommended for large datasets. For example, searching United States data for a PO Box address can be slow if your first stage search contains a large number of matches (for example, a state).

Typedown Troubleshooting

Typedown searching does not use pattern matching. Therefore, if any address element is misspelled, QAS Pro will not be able to find it.

If you type in a place or street level combination which does not match anything in the dataset, a 'No Matches' message appears in the **Results** area. Incorrect text can be deleted with the **Backspace** key.

Single Line Searching

Single Line searching works in the opposite way to Typedown searching: you type in address elements starting with the most specific (for example, a house number and/or street name) and move on to more general elements (for example, a town or postcode in the United Kingdom).

QAS Pro allows you to search on any address element or combination of address elements that are separated by a comma. When you have entered the information, you must activate the search manually.

Note that any mixture of upper and lower case characters can be used, as QAS Pro API does not differentiate between upper and lower case text.

Single Line searching is also equipped with facilities designed to make searching easier and more efficient. These facilities fall into two categories:

- "Wildcard Searching" (see page 29)
- "Identifying Address Elements" (see page 32)

Wildcard Searching

Single Line searching can use wildcards to replace one or more missing letters in your address information.

There are two wildcards available. You can use a combination of wildcards in a single search line.

- Question mark wildcard (?)

This wildcard replaces a single character in an address or postcode and can be placed anywhere within an address element. For example:

Le?land Rd, London

Hawthorn Ave, ?N21 ?HA

The Question Mark wildcard cannot be used with ZZ Postcode searching.

- Asterisk wildcard (*)

This wildcard replaces any number of characters at the end of an address element. For example:

Popes *, Twickenham

South St, Peter*

For more information on using the Asterisk wildcard with ZZ Postcode searching, please see the QAS Pro Getting Started Guide.

Use multiple wildcards sparingly. If too many wildcards are used in a search, there is a risk of considerably extending the search time, and not returning any matches.

Question Mark Wildcard

The question mark represents a single character and can be placed anywhere within an address element.

When searching with a question mark wildcard, QAS Pro API produces a list of all matching streets and postcodes. For example:

If you enter:	QAS Pro retrieves:
?146?	A list of all US streets with zip codes beginning 114, 214, 314 etc. Similarly, all zip codes ending with 461, 462 and so on.
6 victoria crescent, 3?67	The full address: "6 Victoria Cres, Abbotsford, VIC 3067" in Australia.

Asterisk Wildcard

The asterisk can stand for any number of characters and must be placed at the end of an address element.

When searching with an asterisk wildcard, QAS Pro API returns a list of every possible match. For example:

If you enter:	QAS Pro retrieves:
Woodhaven*,Portland, ct	Woodhaven Drive and Woodhaven Road in Portland, America.

Keyword Searching

It is possible to search for certain key words within an address using a keyword search. This is implemented with the asterisk wildcard.

If you enter:	QAS Pro retrieves:
*park, nottingham	All addresses in the city of Nottingham with a first address element that contains the word "park".

In a keyword search, the asterisk can be anywhere in the address element. Keyword searching is therefore especially useful if you are looking for a particular type of organisation or institution, such as banks, colleges, hospitals, etc. For example, the search "*university,dundee" looks for any university in Dundee.

Searching With Partial Addresses

A partial address consists of one or more address elements, separated by commas. One such partial address element might be sufficient to identify the complete address if the organisation, house or street name is unusual. Capital letters are not required.

The address elements available for the datasets that you have purchased are shown in the Data Guide that is supplied with each dataset.

You can use any combination of address elements in any order. If there are discrepancies between the spelling of the entry and the spelling as recorded in the dataset, QAS Pro usually manages to find the required address.

If a definition is not specific enough, the search for matching addresses can take too long or can result in too many addresses to be useful. For example:

Green*

followed by a town name would retrieve:

- all matching organisations (Greenwood Motors, Green Acres Health Farm, etc.);
- all matching house names (Green Cottage, Green Trees, etc.);

- all matching street names (Green Lane, Greenfield Street, etc.);
- all matching place names (Greenslopes, Greenwich, etc.).

Identifying Address Elements

You can limit a search to make it more efficient, by identifying which address elements one or more parts of your address represents.

Thus a part of an address can be identified as, for example, a street name or an organisation name. By tagging part of the address as a post town, QAS Pro only looks through its list of post towns when it endeavours to match that part of the address.

See the Data Guide provided with your dataset(s) for details of the address components that can be identified in this way.

Key Searching

Key searching works in a similar way to Single Line searching. You type in the search term, and QAS Pro returns the address that matches the search in the picklist. By clicking on the **Search** button or pressing **Enter**, the typed-in information becomes a subject of the search.

Using Key Searching, you can search on the following data:

Dataset	Element
United Kingdom with Gas	Meter Point Reference Number (MPRN)
	Meter Serial Number
United Kingdom with Electricity	Meter Point Administration Number (MPAN)
	Meter Serial Number
United Kingdom with Gazetteer Data	Unique Property Reference Number (UPRN)
AddressBase™ Premium	Unique Property Reference Number (UPRN)
	Unique Delivery Point Reference Number (UDPRN)
	Topographic Identifier (TOID)


Searching On A Utility Meter Number

To search for a gas or electricity meter number or serial number, you type in the meter number. QAS Pro API returns the address(es) that match the search.

Searching For A UPRN

To search for an address using the unique property reference number (UPRN), you type in the UPRN. QAS Pro API returns the address that matches the search term.

Alias Matching

 This symbol indicates an alias match or a recode message. If this symbol is returned against an address, this means that although Pro has recognised the information entered by the user as a possible version of the name or address, it has replaced it with the official (postally-correct) version from the dataset.

United Kingdom

Alias Matches

An alias match would occur, for example, if you entered a street and locality, where the locality is not necessary to the address and has been replaced by the town. It would also occur if the postcode has recently been changed and QAS Pro is aware of this: if you enter the old postcode, it will automatically be replaced by the new one.

Here are several examples of why an alias match may occur with United Kingdom data:

Changed Postcode	The Post Office may change the postcode of an existing address.
Street/Town/Name Changes	The town council and the post office may be out of synchronisation about changes to the name of a street or town, or to the names of residents.
Disputed Address	For example, someone living in Londonderry might say that they live in Derry. In this case, the address is matched exactly since Derry is stored as an alias, but the returned formatted address will be in Londonderry.
Name Abbreviation	For example, Michael Smith may prefer to be known as Mike Smith, and may use this name in correspondence. In cases like this, "Mike" would be stored as an alias for "Michael".

Recode Messages

When an alias match results in the recoding of a postcode, locality or street, this change is highlighted. An alias match would also occur if the postal code, locality or street has recently been changed and QAS Pro is aware of this; for example, if you enter the old postal code, it will automatically be replaced by the new one.

Australia

Alias Matches

An alias match would occur, for example, if you entered the name of a thoroughfare or locality which has changed, or which has a local variation.

Bordering Localities

When you search for a street, you may not know the correct postal locality in which the street is situated. In these cases, QAS Pro also searches for the specified street in **all** of the localities which border the input locality and the input postcodes.

For example, all addresses that fall within the locality of Parramatta will return a number of bordering localities, including Mays Hill, North Parramatta, N Parramatta, Rydalmere, etc. When the street is found in a locality that borders the input locality or postcode, the entry is marked as an alias in the resulting picklist. The accuracy score is reduced if a bordering locality is used to match an address.

When a picklist entry is in a bordering locality, this is also prominently displayed in the status line when that entry is highlighted.

New Zealand

Alternative Names

A street, town / city or a suburb may have an alternative name to the official postal name. The alternative name will only be returned if it has been supplied in the search and if the appropriate submitted element has been fixed in the address layout. Refer to the New Zealand Data Guide that was shipped with your data for detailed information about the address elements, including submitted elements.

Suburb Adjacencies

When you are searching for a street using suburb information, you may not know the correct postal suburb in which the street is situated. In such cases, QAS Pro also searches for the specified street in all of the suburbs which border the input suburb.

Suburb adjacencies are used for searching purposes only. They are not returned in the final address.

Retrieving DataPlus Information

QAS DataPlus can provide a wide range of information relating to an address, as a supplement to the QAS Pro API. Currently, DataPlus information is only supplied with certain datasets; if you do not have DataPlus, you can skip this entire section.

DataPlus information is contained in datasets. Each piece of information relates to a general area, such as a locality or postal/ZIP code, or, when the data requires higher resolution, it can be related to the delivery point (letter box). DataPlus handles the information in terms of a code and its related description (if there is one). For example, a dataset containing grid references would only include codes.

DataPlus details can only be viewed once you have selected and displayed a full address from QAS Pro API, and configured your layout to display DataPlus items. For example, if you have the Australia dataset with the associated latitude and longitude DataPlus set, and you perform a Typedown search on **north sydney**, followed by **miller** and then **314**, QAS Pro finds this address:

314 Miller Street,
NORTH SYDNEY NSW 2060
-33.828965 151.20882

If you want to retrieve DataPlus information with your addresses, you should configure your address layout so that it contains lines specifically for DataPlus. See the `AddressLineN` keyword under "Output Address Format Settings" on page 194 or the Configuration Editor Help for further details about setting up DataPlus information.

Refer to the relevant Data Guide for a complete listing of DataPlus sets.

Retrieving Multiple DataPlus Values

If your address contains multiple DataPlus values, such as several gas meter numbers for one property, and if you have configured your address layout to contain this DataPlus item, QAS Pro API returns all DataPlus values in one string, separated by a delimiter.

You can specify the delimiter you want to use with the `MultiValueDPSeparator` configuration setting (see "Output Address Format Settings" on page 194), or using the Configuration Editor. The default delimiter is |. For more information about configuring the delimiter using the Configuration Editor, see the 'Format Options Pane' topic in the Configuration Help program.

The following example shows you how to retrieve multiple gas meter numbers using the User Interface API.

1. Search for the required address using the appropriate search method.
2. In the address return screen, you can see the correct address:

The screenshot shows a software window titled "QAS Pro - United Kingdom with Gas - < Default >". The window has a menu bar with "Pro", "Search", "Dataset", "View", and "Help". Below the menu bar is a toolbar with icons for search, navigation, and data management. A "Dataset:" dropdown menu is set to "United Kingdom with Gas". Below this is a text input field labeled "Enter flat/unit number or organisation" with an "Accept" button to its right. The main area of the window displays search results for three fields: "Address" (47 Redland Road), "Town" (BRISTOL), and "County" (BS6 6AG). At the bottom left, there is a label "Address" next to a "DataPlus" button. At the bottom right, there is a "Typedown" button.

- Click on the **DataPlus** tab.

QAS Pro - United Kingdom with Gas - < Default >

Pro Search Dataset View Help

Dataset: United Kingdom with Gas

Enter flat/unit number or organisation

Accept

DataPlus set	Value
Meter Number	4180040202
Serial Number	5185930S
Customer Barcode	
Sub Building	
Building Name	
Building Number	47
Dependent Street	
Principal Street	REDLAND ROAD
Double Dependent Locality	
Dependent Locality	
Post Town	BRISTOL
Gas Addresses	BS6, 6AG
Gas	< 1/4 > <input checked="" type="checkbox"/> Return this

Address DataPlus

Typedown

The name of the DataPlus set appears on the left, with the related value on the right. Note that in this case, the property has four gas meters.

- You can scroll through the different meters by using the < and > arrow buttons in the **Gas** field. The gas meter number for each meter changes accordingly.
- When you click the **Accept** button, both address and DataPlus information are pasted back to your underlying application:

Doc1.txt - Notepad

File Edit Format View Help

47 Redland Road

BRISTOL

BS6 6AG 1E

4180040000|4180040202|4180040303|4180039801

110509S|5185930S|110923|L9521434715S

(BS66AG1E9)

47|47|47|47

REDLAND ROAD|REDLAND ROAD|REDLAND ROAD|REDLAND ROAD

BRISTOL|BRISTOL|BRISTOL|BRISTOL

BS6|BS6|BS6|BS6, 6AG|6AG|6AG|6AG

Depending on the line width you have configured, all of the characters in the returned elements may not be returned when you paste the address into an application. This occurs because the configured DataPlus line width is too short. For more information about configuring DataPlus lines, see "Output Address Format Settings" on page 194 or the Configuration Editor Help program.

Barcoding

If you have the Australia, United States, Netherlands or United Kingdom datasets installed and you wish to add barcodes to your addresses, this functionality is available as a DataPlus set.

The font used for the barcodes is installed with QAS Pro. If the barcode appears as a list of numbers, ensure that the relevant font from the following list is in your standard Windows font directory:

- the 'QAS 4State Barcode' (for Australia);
- the 'USPS POSTNET Barcode' (for United States);
- the 'CustomerCode Plain (True Type)' (for the Netherlands and the United Kingdom).

For the Primary API, you will see the barcode as a list of numbers.

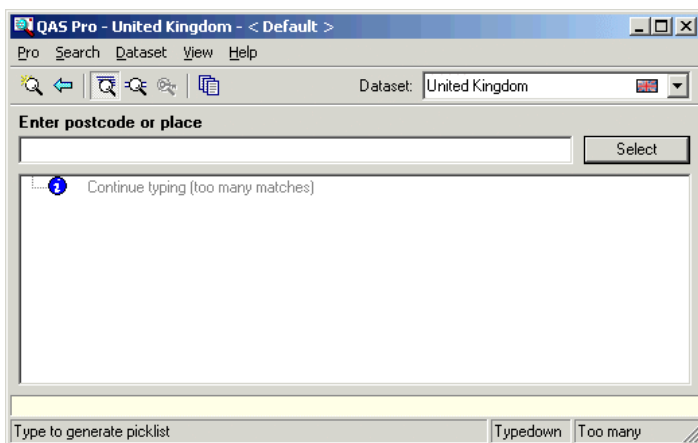
QAS Pro User Interface

This chapter is relevant for Windows users working with the User Interface API only.

All address searches are performed on the user interface. You can select which searching facility you want to use, which address layout to work with and which dataset to search against.

This section covers the ways in which you can set up the user interface, and details every option available.

When you first run QAS Pro, the user interface should look similar to this:



Some of the options on the user interface might not be available to you. The options available depend on what has been specified in the *v/Flags* parameter of **QAProWV_UIStartup** (see page 173), or what has been specified in the Configuration Editor.

There are six parts to the QAS Pro interface, which are described in this section.

Menu Bar

The menu bar consists of five drop-down menus, from which every option available with QAS Pro can be selected. The menu options can be summarised as follows:

Pro	Minimise QAS Pro to the taskbar or Close QAS Pro to the system tray. To fully exit QAS Pro, right-click on the QAS Pro icon in the system tray and select Exit from the menu.
Search	Start a new search, step back to the previous search level, select the searching method and specify advanced search options.
Dataset	Select a dataset. This menu option is available only if more than one dataset is installed.
View	Select an address layout to work with, show or hide the menu bar, toolbar and partial address bar, and select the user interface language.
Help	Get help on all aspects of QAS Pro from this Help program.

To view the options on a menu, either click on the menu name with your mouse, or hold down the **Alt** key and press the key of the underlined letter in the menu name. For example, to select the **Search** menu, press **Alt+S**.

Similarly, once the menu options are displayed, you can select an option by either clicking on the option name with your mouse, or by holding down the **Alt** key and pressing the key of the underlined letter in the menu option name.

For example, to select the **New Search** option using the menus instead of the hotkeys, press **Alt+S** to select the **Search** menu, and then **Alt+N** to select the **New Search** option.

To hide the menu bar, select **Menu** from the **View** menu. To make the bar visible again:

1. Click on the envelope on the title bar.
2. Select **Show menu** from the drop-down menu that appears to make the bar visible again.

Toolbar



The toolbar contains the following options, as shown in the screenshot above (from left to right):

New Search	Clears the existing search and starts a new one.
Back	Steps back to the previous search level.
Typedown Search	Selects Typedown searching.
Single Line Search	Selects Single Line searching.
Key Search	Selects Key searching.
Select Layout	Allows you to choose the address layout you want to use.
Dataset	If you have more than one dataset installed, you can select the dataset you want to search with from this dropdown list.

When using QAS Pro, you can find out the function of a button on the toolbar by placing the cursor over it and holding it there for a few seconds. A short description of the button (known as a tooltip) appears.

To hide the toolbar, select **Toolbar** from the **View** menu. Repeat this action to make the toolbar visible again.

Search Area

The Search area is used to type in address elements to search on.

The Search area prompt depends on the search method you select.

Search area for Single Line searching:

A screenshot of a search input field. The field has a light gray header bar with the text "Enter search" in bold. Below the header is a white text input area with a small downward-pointing arrow on the right side, indicating a dropdown menu.

Search area for Typedown searching:

A screenshot of a search input field. The field has a light gray header bar with the text "Enter postcode or place" in bold. Below the header is a white text input area.

Search area for Keyfinder searching:

A screenshot of a search input field. The field has a light gray header bar with the text "Enter Keyfinder Search" in bold. Below the header is a white text input area with a small downward-pointing arrow on the right side, indicating a dropdown menu.

As you work through the stages of searching, the refinement prompts above the search field provide specific pointers to the type of information that should be entered at each stage; for example, 'Enter postcode or place'.

The Typedown prompt will vary according to the dataset that you are using.

For further details of the types of search you can do with QAS Pro, see "Searching With QAS Pro" on page 25.

Results Area


The main part of the user interface, the Results area, displays the matches returned in response to your search. If there are several alternatives, this area displays a picklist. If there is only one match, or a number of picklists have been worked through to reach a final selection, a full address will be displayed on the "Address Edit Screen" (see page 53).

The results area also displays informational prompts to clarify the current state of the search, for example, 'Continue typing (or select to show all matches)'.

An address can be returned to the address edit screen at any point during a search by pressing **Ctrl+Enter**. Please note that in doing so the returned address will not be fully validated by QAS Pro and, as such, its accuracy can not be guaranteed.

See "Picklist Symbols" on page 49 for further details of picklists.

Partial Address Bar



The partial address bar displays the information that is currently selected in the results picklist as the most complete address possible. For example, the screen shot above shows a partial address from the Netherlands Typedown search results shown below:



Pressing **Ctrl+Enter** returns the address that you last stepped into. However, you should be aware that this will not produce a fully postally correct address from a dataset.

To hide the partial address bar, select 'Partial Address' from the **View** menu. Repeat this action to make the bar visible again.

Status Bar

The status bar indicates the current action and status of the QAS Pro Dialog. When QAS Pro is first run, the status bar displays the searching method – Single Line, Typedown or Key search – that is currently selected, as shown in the screenshot below.



Other information that can be viewed on the status bar is described below:

Stop button	This allows you to abort a search.
Number of matches	Displays the current number of matches during any search. During a Typedown search, you might see 'Too many' on the status bar. This means that the number of matches found so far exceeds the specified match threshold, and you need to enter more information.
Search in progress	During a Single Line search, a blue bar travels from side to side on the status bar, indicating that the search is in progress. This line disappears when QAS Pro has found all possible matches, or when a search has been stopped.
Elements have overflowed	Displayed when an address has been returned to the address edit screen, but one or more address elements are not visible due to lack of space. Existing lines can be widened or more address lines can be added with the Configuration Editor. Refer to the Configuration Editor help for more information.
Elements are truncated	Displayed when an address has been returned to the address edit screen, but the information on an address line is too long for the line. Address lines can be widened with the Configuration Editor. Refer to the Configuration Editor help for more information.
Select Back for close matches	Displayed when QAS Pro has found one 100% match in Single Line searching, and returned a full address to the address edit screen. Click the Back button or press Ctrl+Z to see a picklist of every other possible match found by QAS Pro.

Bordering Localities	When conducting an Australia search, this is displayed when the picklist contains bordering localities. For example, when searching on 'Parramatta', QAS Pro will return a number of bordering localities, including 'Parramatta East', 'Parramatta North', 'North Parramatta', etc.
Alias match	Displayed when the current selection on the picklist is an alias match – i.e. QAS Pro has recognised the information that was typed in, but returned a different, postally-correct version.
Postcode recoded or Locality recoded	Displayed when either the postcode or locality has been recoded. If you select a recoded entry from the picklist, the message will be displayed in white text on a blue background. This message will also be displayed on the final address screen.
Warning – address not verified	Displayed when an incomplete address is returned to the address edit screen.
Exact matches shown – continue typing for more matches	Displayed when there are exact matches, and a higher number of close matches than the limit of the threshold allows. Type in a * or press Select to view the complete list of matches.

Select Button

When you use Typedown searching, QAS Pro will always look for an exact match first. If an exact match is found, and other close matches are below the match threshold, all of the matches are displayed. However, if the close matches exceed the match threshold, only exact matches are displayed.

You can click the **Select** button to display a full list of all results.

Picklist of Returned Addresses

When more than one address is returned from a search, a picklist of the available selections will be displayed. Various symbols are used to display the nature and state of the addresses.

The picklist will vary slightly depending on whether Single Line or Typedown searching have been used. These two picklists are covered in the following section.

Picklist Symbols

There are five symbols that you might see next to matches in a picklist:



This symbol of a letter and a magnifying glass in the grey box at the top of a picklist in Single Line searching shows what QAS Pro is currently searching on.




Single full address

A single white envelope indicates a single match. Select this to return a full address directly to the address edit screen.



Expandable item

A  sign and a symbol of multiple white envelopes next to the item signifies an expandable item. By selecting an expandable item you will see a further picklist containing every component of that item. For example, if the item is a road, expanding it will cause all the premises in that road to be displayed.



Names entry symbol

The symbol of a person indicates a name entry. If you select a name entry, both the name and full address are returned to the address edit screen.

This symbol can only be seen if your dataset includes name data.

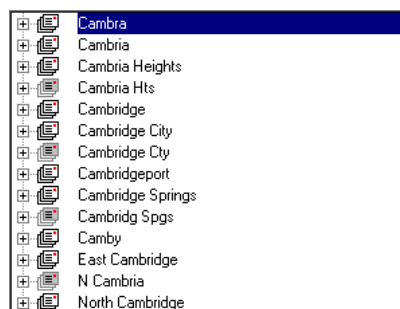


Alias match symbol

A symbol of single or multiple grey envelopes indicates an alias match. This means that although QAS Pro has recognised the address information as a possible version of the address, it has replaced it with the official version from the dataset. See "Alias Matching" on page 34 for more information.

Typedown Search Results

The results of a Typedown search (in this case looking for a place in the US beginning with "camb") are returned in a picklist similar to this:

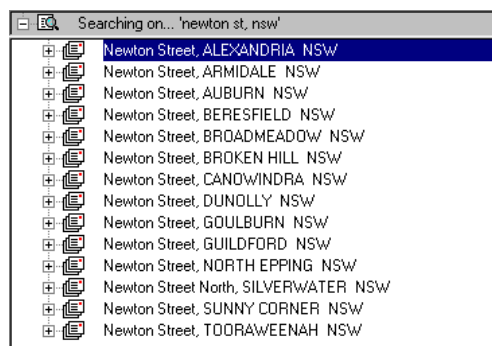


As this is the first stage of a Typedown search, every match is an expandable item.

Every item in a Typedown picklist precisely matches what you have typed in so far.

Single Line Search Results

The results for a Single Line search (in this case, on an Australian street name and state code) are returned in a picklist similar to this:



Selecting a Picklist Item

There are a number of ways to select a picklist item:

- Double-click the item.
- Highlight the item and click the **Select** button.
- Highlight the item and press **Enter**.
- Type enough letters in the **Select item** box (which replaces the **Enter search** box) to identify the item you want, then click **Select** or press **Enter**.

Order of Picklist Items

In Single Line, picklist items are displayed in match percentage order, with the highest at the top of the list, then sorted by address.

Displayed Postcodes

If there is only one postal/ZIP code which covers the whole of a picklist item, it is displayed in the picklist. However, if a street or other expandable item contains more than one postal/ZIP code, the postal/ZIP codes are not displayed. Select the item to see the postal/ZIP codes available.

Returning An Unrecognised Address

When you enter property information, you can return an unrecognised address by pressing **Ctrl+Enter**. For example, using GBR data:

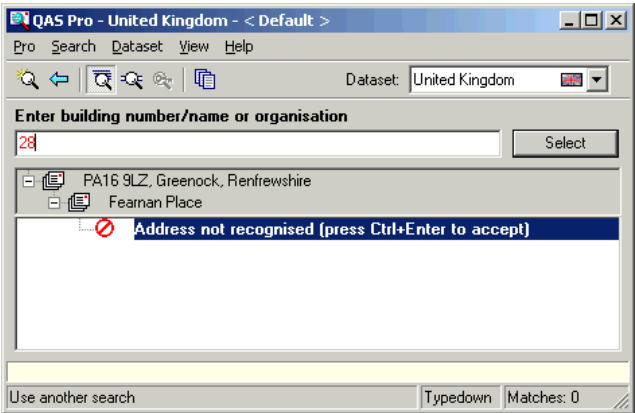
- 1. Click the **Typedown** button to use the Typedown facility. Alternatively, you can select **Typedown** from the **Search** menu, or press **Ctrl+T**.
- 2. Type **PA16 9LZ** into the **Enter postcode or place** box and press **Enter**. When the street address is displayed, press **Enter** again to display a list of premises.

QAS Pro returns two matches, and identifies the second match as an even range:



- 3. Type **28** into the **Enter building number/name or organisation** box.

The picklist area displays the following informational prompt:



Although the address you typed is not valid, and the suggestion of the highlighted prompt at the bottom of the screen is to 'Use another search', you can press **Ctrl+Enter** to return the full address.

- 4. For the purpose of this example you are confident that the input address is correct. Press **Ctrl+Enter** to return the address to the address edit screen.

The address edit screen will display a warning that the returned address has not been verified.

QAS Pro will not display the **Ctrl+Enter** prompt if it has been disabled by your system administrator. See the Help program for further details.

Address Edit Screen

When you select an address from a picklist, the address is returned to the address edit screen, which replaces the picklist in the **Results** area.

The format of the address edit screen depends on the layout you select (see "Selecting An Address Layout" on page 56). If the current dataset has additional DataPlus information, the screen will have two tabs labelled 'Address' and 'DataPlus' in the lower left-hand corner. See "Retrieving DataPlus Information" on page 37 for more details.

Once you have a full address, you can edit it directly on this screen if you wish by clicking in the boxes or using the cursor keys. However, editing a postally correct retrieved address is not recommended.

You may prefer an address element to be on the next line. However, you cannot transfer it to the second line by pressing **Enter**. This is because pressing **Enter** is the same as pressing the **Accept** button, and causes the address to be pasted to the underlying application as it is. Instead, the command to insert a hard return (that is, to move all the text to the right of the cursor position on to the next line) is **Ctrl+Enter**.

Setting QAS Pro Options

There are a number of options that you can specify on the QAS Pro user interface. As well as choosing whether to view or hide the menu bar, toolbar, partial address bar, and informational prompts, you can:

- choose a search method;
- change the searching options;
- specify the dataset you want to search with (if you have more than one dataset installed);
- select the address layout that you want to work with.

It is possible to control the options that a user can change with the functionality flags used in the call to **QAProWV_UIStartup**, and also in the User Interface Options pane on the Configuration Editor. Therefore, you might not be able to change some of these options from the QAS Pro User Interface.

Selecting A Search Method


There are several search methods in QAS Pro. It is easy to switch between them as necessary.

To select Typedown searching:

Either select **Typedown** from the **Search** menu, press **Ctrl+T** or click the Typedown search toolbar button: .


Each of these actions calls the function **QAProWV_UISetFlags** (see page 171) with the `qaattribs_TYPEDOWNSEARCH` flag.

To select Single Line searching:

Either select **Single Line** from the **Search** menu, press **Ctrl+S** or click the Single Line search toolbar button: .

Each of these actions calls the function **QAProWV_UISetFlags** (see page 171) with the `qaattribs_SINGLELINESEARCH` flag.

To select Key searching:

Either select **Key Search** from the **Search** menu, press **Ctrl+K** or click the Key Search toolbar button: .

Each of these actions calls the function **QAProWV_UISetFlags** (see page 171) with the `qaattribs_KEYFINDERSEARCH` flag.

When you start up the QAS Pro API, the search method will default to whatever was used last, or to the flag that has been set with **QAProWV_UIStartup** (see page 173). If the flag `qaattribs_NOCHANGEMODE` has been set, you cannot change the search mode from the user interface.

Setting The Search Options

There are a number of advanced options that can be set for each searching method. To view the **Options** dialog, select **Options...** from the **Search** menu.

Selecting A Dataset

You can select which installed dataset to search on using the **Dataset** list.

Either click on the arrow to the right of the **Dataset** list, or press **Alt+D** followed by the **arrow down** cursor key to see the list of available datasets.



This action performs the equivalent of calling the following functions:

Primary API Functions	UI API Functions
QA_GetDataCount	QAProWV_UICountryCount
QA_GetData	QAProWV_UIGetCountry
QA_GetActiveData	QAProWV_UIGetActiveCountry

To select a dataset, either click on it or use the cursor keys to highlight it and press **Enter**. This performs the equivalent of calling the following functions:

Primary API Function	UI API Function
QA_SetActiveData	QAProWV_UISetActiveCountry

A dataset can be selected by pressing **Ctrl+Shift** and typing in the first letter, or couple of letters of the dataset code to select a dataset. For example, **Ctrl+Shift+DN** selects the Denmark dataset (if the Denmark dataset is installed).

If a different dataset is selected in the middle of a search, the current search is stopped. The search term you entered is retained in readiness for a new search using the new dataset.

Selecting An Address Layout

An address layout is a collection of settings that determine how the final address is formatted when you complete a search in QAS Pro.

For example, a layout could specify that the address will be on five lines, each of width 50 characters, with the town and postal/ZIP code always on the fourth and fifth lines respectively.

Each dataset comes pre-configured with a standard address layout that follows appropriate standards.

When QAS Pro is started, it automatically uses the default layout for the dataset that is selected. See the relevant Data Guide for details on the standard layout. However, you can have several layouts set up for each dataset. See "Overview" on page 185 for full details of creating and setting up new layouts.

Click on the **Select layout** button or choose **Select layout...** from the **View** menu to see a list of available layouts for the currently selected dataset.

This action performs the equivalent of calling the following functions:

Primary API Functions	UI API Functions
QA_GetLayoutCount	QAProWV_UILayoutCount
QA_GetLayout	QAProWV_UIGetLayout
QA_GetActiveLayout	QAProWV_UIGetActiveLayout

To select another layout, either click on the layout name to highlight it and click **OK**, or double-click on the layout name, or use the cursor keys to move to the layout name and press **Enter**. This performs the equivalent of calling the following functions:

Primary API Function	UI API Function
QA_SetActiveLayout	QAProWV_UISetActiveLayout

The Preview area at the bottom of the Select layout dialog shows the address format of any layout that you highlight.

When you create a layout, you can set the layout comment that appears at the bottom of the Select Layout dialog using the `Comment` setting (see "Output Address Format Settings" on page 194).

QAS Healthcoder

QAS Healthcoder is a layer of functionality that sits on top of QAS Pro and contains additional information and features that may be of use to the health sector.

If you are using QAS Healthcoder, the following functionality is accessible by default:

- The Health DataPlus set
- ZZ Postcodes.

For more specific information on the QAS Healthcoder functionality, please see the QAS Pro Getting Started Guide. For more information on integrating QAS Healthcoder, see Using QAS Healthcoder (below).

Using QAS Healthcoder

If you are integrating QAS Healthcoder rather than QAS Pro, there are some changes that need to be made to the configuration files. If you are not using QAS Healthcoder for your integration, then this section can be skipped.

To return ZZ Postcodes, QAS Healthcoder uses a feature called AddressBook. In order for AddressBook to work, three settings in the configuration file need to be set. In QAS Pro Plug and Go, these settings are automatically set up by the installer, but if you are using the API, you have to ensure these are set yourself.

You may want to change these files (for instance if you do not wish to use the zz prefix for ZZ Postcode searching). You can do this by opening your configuration file (QAWSERVE.INI) in a non-formatting text editor such as Notepad under Microsoft Windows or vi under UNIX, and add the setting in the [QADefault] section. The three additional settings are formatted as follows:

AddressBook=Yes [or] No
+<search prefix>:<data file name>
AddressBookData=<file location>

In the majority of cases, the settings you should use will look like this:

AddressBook=Yes
+zz:zzwv.abk
AddressBookData=C:\Program Files\QAS\QAS Healthcoder

AddressBook

The default for this setting in QAS Pro is **No**. The API installer for Healthcoder will set it to **Yes** for use with AddressBook.

+<search prefix>:<data file name>

Type your chosen search prefix and data file name, preceded with a '+' sign, on the line following the configuration setting. The search prefix is the combination of characters that you use to tell QAS Healthcoder that you are undertaking an AddressBook search. It cannot be more than four characters long.

It is recommended that you choose a search prefix that would not be found in standard QAS Healthcoder searching, because it will be used solely for AddressBook. For example, if you set the search prefix to be "99", any search beginning with 99 will tell QAS Healthcoder to look in the AddressBook data file. You would not be able to look for a premise such as '99 Birmingham Road' unless you specifically turn off AddressBook.

For ZZ Postcodes it is recommended you use "zz" as the AddressBook prefix.

After the search prefix, type a colon and then the name of the data file you want to access. The supplied ZZ Postcode file name is "zzwv.abk".

AddressBookData=<file location>

The AddressBookData ini setting allows you to set the file location of the ZZ AddressBook file. Use this setting if you have the AddressBook file in a nondefault location.

Data Types

There are a few Experian QAS-specific data types which appear in the documentation and need some explanation. These types define the parameters that the functions take and values they return.

These can be split into three categories: the values that are returned by the functions, the parameters that go into the functions, and the parameters you get out of the functions.

Function Return Values

QAS Pro data type	Description	Equivalent 'C' data type
INTRET	integer	int
LONGRET	long integer	long
VOIDRET	no return value	void

Parameters (Input)

QAS Pro data type	Description	Equivalent 'C' data type
STRVAL	string	char *
INTVAL	integer	int
LONGVAL	long integer	long
VOIDARG	no arguments	void

Parameters (Output)

QAS Pro data type	Description	Equivalent 'C' data type
STRREF	string	char *
INTREF	integer	int *
LONGREF	long integer	long *

Calling Functions From Languages Other Than C

Whilst C is the language most commonly used when working with these API functions, it is possible to integrate the API with other programming environments. There are, however, a few points which you should note. These are:

- NULL Termination
- Padding
- Passing by Value or by Reference

NULL Termination

QAS Pro API is written in the C programming language. In C, all strings are expected to end with the NULL character (i.e. NULL terminated), which has the absolute value 0 (zero), not ASCII '0'.

For all functions in the API that accept parameters of the type STRVAL, these parameters must be NULL terminated. Furthermore, all return parameters of type STRREF will be NULL terminated.

Passing By Value Or By Reference

In general C programming, function parameters may be passed either by value or by reference.

You must pass a parameter in the way the function expects you to pass it. If you pass a parameter by value when the function is expecting it to be passed by reference then this might crash your program and will certainly produce incorrect results.

- In C programming, strings are always passed by reference, whether they are input or output parameters. However, this is not true of all languages, and strings may be passed by value in the language you are using.
- Numbers are passed by value when they are inputs to the function. They are passed by reference when they are outputs from the function.

Returned Strings

When passing a buffer to an API function parameter that returns a string, the next parameter normally defines the size of the buffer passed. The following example demonstrates this type of function:

```
INTRET QA_GetActiveData ( INTVAL viHandle,  
                          STRREF rsDataID,  
                          INTVAL viDataIDLength );
```

Here, the parameter *viDataIDLength* is the length of the buffer passed with *rsDataID*.

If the buffer length parameter is zero, then the API function will not attempt to populate the buffer with the return string. The parameter that receives the buffer can optionally be passed NULL in combination with a zero buffer length, if the integration language supports this.

If the buffer length parameter is greater than zero, then the integrator should ensure that the buffer size and is large enough to receive the returned string. If this is not the case, then truncation will occur, and will be logged in the error log file (see "Error Logging Settings" on page 210).

Example Of Data Types

This example uses the function **QA_GetData** (see page 94).

This is how the function prototype looks in the documentation:

```
INTRET QA_GetData ( INTVAL viHandle,
                    INTVAL viDataOffset,
                    STRREF rsDataID,
                    INTVAL viDataIDLength,
                    STRREF rsName,
                    INTVAL viNameLength );
```

The parameters *viHandle* and *viDataOffset* are inputs to the function (in the form of integers) and thus are passed by value. Similarly, the parameters *viDataIDLength* and *viNameLength* are also passed by value as they are inputs to the function in the form of integers. The parameters *rsDataID* and *rsName* are output parameters (in the form of strings), and consequently are passed by reference.

In addition, **QA_GetData** returns a status value indicating either the successful execution of the function, or whether an error has occurred (in the form of an error code).

This function can be written in native C as:

```
int QA_GetData ( int viHandle,
                 int viDataOffset,
                 char *rsDataId,
                 int viDataIDLength,
                 char *rsName,
                 int viNameLength );
```

Primary API Reference

Pseudocode Example Of QAS Pro API

This section provides an overview of how a program using the QAS Pro API works on a conceptual level. The pseudocode used is programming language independent.

The example below uses many of the QAS Pro API functions, so that you can see how they work together. In practice, however, you might not want or need to use every function.

The pseudocode does not include all of the available functions.

Six functions are described in the pseudocode. The main function starts an instance of the API, and gives the user a choice of performing a Single Line or Typedown search. It then frees resources after each result has been retrieved. Each search recursively calls five further functions: Display Error, which retrieves and displays the message associated with the returned error code; User Action, which displays prompts (commonly an instruction as to what should be entered by the user); Display Results, which retrieves picklists of results from the search; Select Results, which allows the user to step into picklist items; and Return Address, which formats and returns the final address, using the rules specified in the active layout.

Main Function

The primary API is instance based, where each instance is referenced with a handle. To create a new instance of the API, the function **QA_Open** must be called and the unique handle that it returns used with subsequent API calls. If multiple instances of the API are required, such as for use with multithreading, then **QA_Open** needs to be called multiple times.

Open an instance of the API [**QA_Open**]

As with all API calls, the open could fail for various reasons. The most common reasons for this to occur are that the product is not installed or configured properly. If open fails, address matching will not be available. When writing integrations of the primary API, it will be useful to enable error logging (see "Error Logging Settings" on page 210).

For brevity in the pseudocode we will leave out the error checking for each API function call. However in an integration all API calls should check the error returned.

```
If open failed
  Call Display error
  close API instance [QA_Close]
  shutdown API [QA_Shutdown]
  exit procedure
End If
```

Once an open has been performed, the settings that are going to be used should be applied to the instance. The user may or may not be able to control which dataset and search engine to use. Both of these should be set before searching begins. The active dataset and search engine can be changed between searches if required.

```
set active dataset to search upon [QA_SetActiveData]
set engine to use for searching [QA_SetEngine]
```

The layout to be used for formatting will need to be set, although this defaults to an internal layout that will be valid but probably not suitable. It is wise to set the active dataset before choosing the layout to use with formatting, as some layouts may only be defined for individual data sets.

Any engine options, such as search timeouts or picklist thresholds should also be set at this point if the default values are not desired.

```
set layout to use for formatting [QA_SetActiveLayout]
set engine options [QA_SetEngineOption]
```

The recommended method for handling searching with the API is to recursively display any picklist results, and then allow the user to either search upon the results further, or to step into a result until the user has the address that they require. Formatting can then be applied, and the address returned.

```
Repeat
```


The results should be displayed before the search is performed, as there may be some information returned to the user by the API before searching. This is commonly done by the typedown search engine.

Call Display results

There are two essential requests that a user can make that need to be handled. The first is giving some text to search upon. The second is to select an item to either step into or format. Other actions can be coded at the integrators discretion, such as stepping out of a result to go back, or to exit the address searching.

Call User action

If action was search text

 search with text [QA_Search]

Else If action was to select an item

 Call Select Result

End If

Until user has obtained address

After a search has been completed, it is essential to call the end search function before starting another in the instance.

end the search [QA_EndSearch]

Once all address searching has been performed with the instance, the instance should be closed.

close the instance [QA_Close]

If no more instances are required, the API needs to be shutdown, which will close all instances anyway.

shutdown the API [QA_Shutdown]

Display Error Function

Almost every primary API function call returns an error code which should be checked. If an error occurs then the message associated with the returned error code can be retrieved and displayed to the user.

retrieve error message [QA_ErrorMessage]

display error message to user

Return to calling function

User Action Function

In order to aid the user when they are interacting with the integration, the Primary API can return prompts to display to the user. These will commonly be an instruction of what should be entered, such as 'Enter postcode or place' when using the typedown engine at the beginning of a search. The prompt should be displayed to the user in some manner before requesting input.

In order to aid the user when they are interacting with the integration, the Primary API can 'return' (codes) prompts to 'display' (typed) to the user. These will commonly be an instruction of what should be entered, such as Enter postcode or place when using the typedown engine at the beginning of a search. The prompt should be displayed to the user in some manner before requesting input.

```
retrieve prompt text [QA_GetPrompt]
display prompt to user
obtain action from user
Return to calling function
```

Display Results Function

To display results, first obtain a count of the available results and then retrieve and display each one. The integration may choose to display different type of results in different ways. For example it is common to indicate to the user which results can be stepped into, or which results are full addresses that will be used if selected.

```
get a count of available results [QA_GetSearchStatus]
For Each result
    retrieve the result [QA_GetResult]
    display the result to the user
End For
Return to calling function
```

Select Result Function

When a user selects a result the action that the integration should take is determined by the attributes associated with the result in question. The two most important attributes are whether the result can be stepped into, and whether it is a final address. Other result attributes may be handled at the integrators discretion.

```
get the attributes of the result [QA_GetResult]
If result can be stepped into
    step into result [QA_StepIn]
Else If result is a final address
    Call Return address
End If
Return to calling function
```

Return Address Function

When a user has selected a final address, it should be formatted and returned in the appropriate manner. The address will be formatted using the rules specified in the active layout.

```
format result and retrieve line count [QA_FormatResult]
For Each formatted line
    retrieve the line text [QA_GetFormattedLine]
    return text to the user
End For
Return to calling function
```

Handling Errors

Every Primary API function except for **QA_Shutdown** returns an integer error code value. This value will be within the following ranges:

0	Function succeeded
less than 0	Function encountered errors

Experian QAS strongly recommends that every function call is checked for errors. Although the majority of errors are generated by mistakes in an integration which would be picked up through testing, some are unpredictable. For example, the failure of a client-server link, running out of memory, or a data file expiry.

All Primary API functions will set their output parameters to defined values if an error occurs. Integer output variables will be set to zero, and string output variables will be set to blank.

The `LogErrors` configuration setting is a useful way of locating the cause of an error. Experian QAS recommends that you enable error logging while writing and debugging an integration. See "Error Logging Settings" on page 210 for more information.

The **QA_ErrorMessage** function provides a textual description of error codes returned from the API. See **page 86** for more information.

The handling of errors should typically be split into two categories. These are described below.

1. An error returned from a call to **QA_Open**. This means that address capture is unavailable.

This could be caused by many things, such as configuration or administration errors. Experian QAS recommends that you call **QA_Shutdown** and locate the source of the problem using the `LogErrors` setting.

2. An error returned after a call to **QA_Open** because, although address capture is available, the current search has failed for some reason. This could be due to a lack of resources, or to an administration error. Experian QAS recommends that you call **QA_EndSearch** to end the current search.

Note that you do not have to shut down the system, and that address capture may still be available. For example, a client-server link may have failed, losing the current search. In that case attempting a new search would reestablish the connection and your integration would continue functioning correctly. See "Error Logging Settings" on page 210 for more information.

If you are writing a client-server integration of Pro, it is very important to react to errors after **QA_Open** as described above. An error will be returned from an API function if a connection to the server in use is lost, as the search results will not be retrievable.

API Instances

The Primary API is based on the concept of instances. An instance of the API is created through calling **QA_Open**, and is destroyed by calling **QA_Close**. Once an instance is created, it is referenced through an integer handle that is returned from **QA_Open**. All subsequent functions that use the instance must be passed through this handle. Once all instances have been closed using **QA_Close**, the API should be shut down using **QA_Shutdown**.

Each instance of the API that is created is independent from other instances. Any action that is performed upon one instance will not affect that state of another instance. Actions can be performed upon separate instances simultaneously.

If you want to have a multithreading integration that performs multiple searches concurrently, you should create multiple instances of the API. Each thread will typically perform the following actions:

- Create an instance using **QA_Open**
- Perform one or more searches
- Close the instance using **QA_Close**

Once all threads have finished, call **QA_Shutdown**. It would be uncommon for a single-threaded integration to require multiple instances of the API.

Flags Returned

Many API functions, such as **QA_GetSearchStatus**, pass information back using 'flags'.

Flags are a means of communicating multiple pieces of information back to the caller using a single parameter.

The value returned consists of multiple values ORed together.

Example:

QA_GetResult returns information about a picklist item. If a picklist item is informational and can be stepped into, the following flags are returned:

<code>qaresult_CANSTEP</code>	4
<code>qaresult_INFORMATION</code>	1024

The value of *rlFlags* will be:

4 OR 1024 = 1028.

To check for a specific flag the integrator should AND the returned set of flags with the flag they wish to test for. If the result is zero, the flag was not present. If the result is non-zero, the flag was present.

Example:

Testing for *qareult_CANSTEP* (4) in returned value 1028:

1028 AND 4 = 4 (non-zero)

Therefore the picklist item can be stepped into.

Example:

Testing for *qareult_ALIASMATCH* (8) in returned value 1028:

1028 AND 8 = 0 (zero)

Therefore the picklist item is not an alias match.

In C:

```
if (rlFlags & 1024)
{
    ...
}
```

Some functions have accompanying 'Detail' functions; for example, **QA_GetResult** has the accompanying function **QA_GetResultDetail**. These allow the caller to inquire about a single specific attribute and is useful for languages that cannot perform bitwise operations such as AND, or for integrations that require extra information.

If *rsDetail* or *rlDetail* are not used in a particular call to a 'Detail' function, it will return a blank string or zero as appropriate.

Example:

```
QA_GetResultDetail ( iHandle,  
                    iResult,  
                    qaresultint_ISINFORMATION,  
                    &lDetail,  
                    NULL,  
                    0 );
```

This will return `qavalue_TRUE` or `qavalue_FALSE` in parameter *lDetail*, depending on whether the given picklist item is informational or not.

Automatic Stepping And Formatting

To help you locate addresses faster and more effectively, the Primary API supports automatic stepping into and formatting of picklist items generated from an initial search. For example, if you were to perform a search that generated only a single exact result, the integration can automatically step into that item instead of prompting you to do it manually, which speeds up the address capture process.

Automatic stepping in and/or formatting is not required for a successful integration, and it is therefore at your discretion whether you act upon the returned flags.

The Primary API will output one of the following four flags from the function **QA_GetSearchStatus**.

The flag `qastate_AUTOSTEPINSAFE` is returned when a search or step in has produced a picklist containing only one match that itself can be stepped into. Experian QAS recommends that the integration should call **QA_StepIn** upon the first item in the picklist when this flag is returned, as it will speed up the address capture process.

The flag `qastate_AUTOSTEPINPASTCLOSE` is returned when a search or step in has produced a picklist containing only one exact match, and also multiple non-exact matches. You may choose to call **QA_StepIn** on the first item in the picklist when this flag is returned, as it may speed up the address capture process.

The flag `qastate_AUTOFORMATSAFE` is returned when a search or step in has produced a picklist containing only a single match that is a final address item. If the integration allows you to go back to the picklist after selecting a final address, then Experian QAS recommend that **QA_FormatResult** should be called upon the first picklist item and the address returned.

The flag `qastate_AUTOFORMATPASTCLOSE` is returned when a search or a step in has produced a picklist containing only one exact match that is a final address item, and also multiple non-exact matches. If the integration allows you to go back to the picklist after selecting a final address, then you can choose to call **QA_FormatResult** on the first item in the picklist and on the address returned.

Asynchronous Searching

The Primary API supports two different threading models for performing searches: synchronous and asynchronous.

Synchronous searching is the default threading model, and is used by the majority of integrations. Using synchronous searching, the functions **QA_Search** and/or **QA_StepIn** return to the caller once the search has been completed. You can then immediately access the results.

Asynchronous searching is an alternative threading model where the calls to the searching functions are returned as soon as the search has begun. The search is performed using a background thread.

Asynchronous searching is not available for single threaded versions of the Primary API, as supplied for UNIX platforms without the required POSIX multithreading standard.

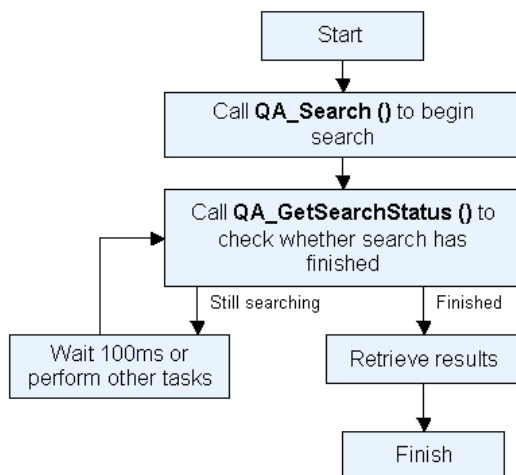
There are three engine options that control asynchronous searching:

1. The `qaengopt_ASYNCSEARCH` engine option allows the integrator to control whether the initial search using **QA_Search** on the single line engine is asynchronous. This is the most commonly used option for asynchronous searching.
2. The `qaengopt_ASYNCSTEPIN` engine option allows the integrator to control whether stepping into picklist items using **QA_StepIn** is asynchronous.

3. The `qaengopt_ASYNCREFINE` engine option allows the integrator to control whether picklist refinement using **QA_Search** after the first result for the single line engine, and all searches using the typedown engine, are asynchronous. This will be used less frequently, as picklist refinement is a fast process.

The search must be complete before you can retrieve the results. The caller needs to check periodically whether the search has been completed by calling **QA_GetSearchStatus**, and checking whether the flag `qastate_STILLSEARCHING` is present. Typically, you should check every 100ms to see whether the search has completed.

The following diagram demonstrates polling:



Asynchronous searching is useful, although not essential, for graphical user interface integrations where the caller wants to be able to perform tasks while the search is in progress, and does not want to use multithreaded code. For example, this could be used to react to search cancel requests.

API Function Reference

The QAS Pro Primary API functions are split into the following groups. A full list of the Primary API functions is provided below, along with details of where you can find each one:

General Functions

QA_Open (see page 131)

Opens an instance of the API.

QA_Close (see page 84)

Closes an instance of the API.

QA_ErrorMessage (see page 86)

Translates error codes into descriptions.

QA_Shutdown (see page 141)

Closes down the API.

Common Search Functions

QA_SetEngine (see page 137)

Sets the current search engine to Typedown, Single Line, Verification or Keyfinder.

QA_GetEngine (see page 97)

Retrieves the current search engine.

QA_SetEngineOption (see page 138)

Sets search engine-related attributes for the current engine.

QA_GetEngineOption (see page 98)

Retrieves the value for a particular engine-related attribute.

QA_GetEngineStatus (see page 100)

Retrieves the status current search engine.

QA_GetPrompt (see page 112)

Returns the prompt text to direct the user action.

QA_GetPromptStatus (see page 114)

Returns status information about the current prompt.

QA_Search (see page 133)

Performs a search.

QA_CancelSearch (see page 83)

Cancels a search in progress.

QA_EndSearch (see page 85)

Ends the current search, and resets all results.

QA_GetSearchStatus (see page 124)

Provides information about the state of a search. Can be called at any time.

QA_GetSearchStatusDetail (see page 127)

Provides detailed information about the state of a search. Can be called at any time.

QA_StepIn (see page 142)

Steps into a picklist item.

QA_StepOut (see page 143)

Steps out of a picklist item.

Result Functions

QA_GetResult (see page 116)

Obtains limited information regarding a picklist item.

QA_GetResultDetail (see page 120)

Obtains detailed information regarding a picklist item.

QA_FormatResult (see page 89)

Selects a picklist item and applies formatting.

QA_GetFormattedLine (see page 102)

Returns a given formatted line.

QA_GetExampleCount (see page 101)

Returns the number of example addresses for the given dataset.

QA_FormatExample (see page 87)

Selects an example address and applies formatting.

Layout Functions

QA_GetLayoutCount (see page 106)

Retrieves the number of available layouts that can be used for formatting.

QA_GetLayout (see page 104)

Returns information about a given layout.

QA_GetActiveLayout (see page 93)

Returns the name of the layout currently in use for formatting returned addresses.

QA_SetActiveLayout (see page 135)

Sets the layout that will be used for formatting.

Dataset Functions

QA_GetDataCount (see page 96)

Provides a count of the available datasets.

QA_GetData (see page 94)

Provides information about a given dataset.

QA_GetActiveData (see page 92)

Returns the data ID of the dataset currently in use.

QA_SetActiveData (see page 134)

Sets the dataset to be searched on.

QA_GetLicensingCount (see page 107)

Provides a count of the total available data and dataplus sets.

QA_GetLicensingDetail (see page 109)

Provides detailed information about a given dataset.

System Functions

QA_GenerateSystemInfo (see page 91)

Generates information regarding the state of the system.

QA_GetSystemInfo (see page 130)

Interrogates the system information provided by **QA_GenerateSystemInfo**.

General Error Scenarios (All Functions)

Bad Parameter

One of the API parameters has been passed an invalid value. Use the `LogErrors` configuration setting (see "Error Logging Settings" on page 210) to determine where the problem lies.

Bad Handle

The parameter *viHandle* has been passed an invalid handle. This should only be passed values returned from **QA_Open** that have not since been closed.

Server Error

An error has occurred on the server. This may be due to the server running out of resource, or to connection issues. This error may be returned in a non client-server integration if the 'server' side of the application runs out of resource, such as memory.

Licensing Error

The active dataset you are attempting to use may not be licensed properly and will return an error. Check that data is the data is correctly installed and licensed, and use the `LogErrors` configuration setting to determine where the problem lies.

See also "Handling Errors" on page 71 for more information about what to do in response to an error.

QA_CancelSearch

Cancels a search that is still in progress. This can be called if using either asynchronous or synchronous searching, although it is more common to use it with asynchronous searching.

If cancelling a synchronous search, then this function will have to be called using a separate thread created by the integration. If the search has finished before this function is called, then nothing will happen.

Prototype

```
INTRET QA_CancelSearch ( INTVAL viHandle,  
                        LONGVAL vlFlags );
```

Arguments

- viHandle* Handle for this instance of the API
- vlFlags* Flags to control function behaviour

Return Value

- Either:** 0 if call is successful
- Or:** Negative error code

Comments

The following flags, passed into *vlFlags*, can be used to specify how to stop a search in progress (either to return immediately or to wait until the search has been cancelled):

Symbolic Name	Decimal Value	Description
qacancelflag_NONE	0	Returns immediately after cancelling
qacancelflag_BLOCKING	1	Does not complete function until the search has been successfully cancelled

QA_Close

Shuts down an instance of the API that has been opened with **QA_Open**. Once an instance has been closed, the instance handle will be invalid and cannot be passed to subsequent API functions.

If the call to **QA_Open** failed, then the returned instance handle will be set to zero, which can be safely used with this function.

If "Asynchronous Searching" (see page 76) is in use, then calling **QA_Close** will safely cancel it before closing the instance.

Prototype

```
INTRET QA_Close ( INTVAL viHandle );
```

Arguments

viHandle Handle to be closed

Return Value

Either: 0 if call is successful

Or: Negative error code

QA_EndSearch

Ends the search. This function is called once all search information has been returned. It must be called before a new search is started.

If an asynchronous search is in progress, then calling **QA_EndSearch** will safely cancel it before ending the search.

Prototype

```
INTRET QA_EndSearch ( INTVAL viHandle );
```

Arguments

viHandle Handle for this instance of the API

Return Value

Either: 0 if call is successful

Or: Negative error code

QA_ErrorMessage

Used to translate an error code into a description.

Prototype

```
INTRET QA_ErrorMessage ( INTVAL viError,  
                          STRREF rsBuffer,  
                          INTVAL viBufferLength );
```

Arguments

viError Error code to translate
rsBuffer Returned error description
viBufferLength Length of *rsBuffer*

Return Value

Either: 0 if call is successful
Or: Negative error code

QA_FormatExample

Each dataset has example addresses, which can be used to preview layouts. See **QA_GetExampleCount** on page 101 for more information about example addresses.

QA_FormatExample formats an example in the current active layout. Sample address lines can be retrieved by calling **QA_GetFormattedLine**.

Prototype

```
INTRET QA_FormatExample ( INTVAL viHandle,  
                          INTVAL viExample,  
                          STRREF rsComment,  
                          INTVAL viCommentLength,  
                          INTREF riLineCount,  
                          LONGREF rlInfo );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>viExample</i>	The index of the example to format
<i>rsComment</i>	A comment describing the example address
<i>viCommentLength</i>	The size of the <i>rsComment</i> buffer
<i>riLineCount</i>	The number of lines that have been formatted
<i>rlInfo</i>	Information about the formatted example

Return Value

Either: 0 if call is successful
Or: Negative error code

The 'Zero' example number always exists and is a blank example address, useful for getting the number of layout lines.

Comments

The parameter *viExample* should be passed an index into the count of example address from **QA_GetExampleCount**, between 0 and the count - 1.

The parameter *rlInfo* can return the following flags ORed together:

Symbolic Name	Decimal Value	Description
qaformat_OVERFLOW	1	There are not enough address lines configured to display the full address
qaformat_TRUNCATED	2	Truncation has occurred on one or more address lines in the final address

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Bad Index	The value passed to parameter <i>viExample</i> was not a valid example address offset. The range should be between 0 and the count of example addresses - 1 from QA_GetExampleCount .
Bad Layout	The active layout that is set for the instance is invalid. The list of valid layouts can be obtained by using the QA_GetLayoutCount and QA_GetLayout functions. A layout may be defined for a specific dataset only, and so changing the active dataset may invalidate the active layout.

QA_FormatResult

Selects a picklist item and applies the formatting routines to it. This is done according to the current active layout.

Prototype

```
INTRET QA_FormatResult ( INTVAL viHandle,  
                          INTVAL viResult,  
                          STRVAL vsExtra,  
                          INTREF riLineCount,  
                          LONGREF rlInfo );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>viResult</i>	The picklist item to be formatted
<i>vsExtra</i>	Extra text to add to the formatted address
<i>riLineCount</i>	The number of lines that have been formatted
<i>rlInfo</i>	Information about the formatted result

Return Value

Either:	0 if call is successful
Or:	Negative error code

Comments

The parameter *viResult* should be passed an index into the count of available picklist items from **QA_GetSearchStatus** or **QA_GetSearchStatusDetail**, between 0 and count - 1.

The parameter *rlInfo* can return the following flags ORed together:

Symbolic Name	Decimal Value	Description
qaformat_OVERFLOW	1	There are not enough address lines configured to display the full address.
qaformat_TRUNCATED	2	Truncation has occurred on one or more address lines in the final address.

The parameter *vsExtra* allows you to pass through text that you want to add to the formatted address. This is typically used when you have refined on text that is not in the data, but that you wish to use in the final address.

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Out of Sequence	The API cannot format a result of a picklist if a search has not yet been performed.
Bad Index	The value passed to parameter <i>viResult</i> was not a valid picklist item offset. The range should be between 0 and the picklist result count - 1.
Bad Layout	The active layout that is set for the instance is invalid. The list of valid layouts can be obtained by using the QA_GetLayoutCount and QA_GetLayout functions. A layout may be defined for a specific dataset only, and so changing the active data set may invalidate the active layout.

QA_GenerateSystemInfo

Generates information regarding the state of the system. You can then interrogate this with the function **QA_GetSystemInfo**.

Prototype

```
INTRET QA_GenerateSystemInfo ( INTVAL viHandle,  
                               INTVAL viType,  
                               INTREF riCount );
```

Arguments

- viHandle* Handle for this instance of the API
- viType* Type of information to generate
- riCount* Count of lines generated

Return Value

- Either:** 0 if call is successful
- Or:** Negative error code

Comments

The following values can be passed to *viType*:

Symbolic Name	Decimal Value	Description
qasysinfo_SYSTEM	1	Generate the current state of the system

Error Scenarios

- Busy Handle** The parameter *viHandle* has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.

QA_GetActiveData

Obtains the data ID of the active dataset. This is the dataset that will be used to search on.

The data ID is a short string that uniquely identifies the dataset. The available data IDs for all installed datasets can be returned using **QA_GetDataCount** and **QA_GetData**.

Prototype

```
INTRET QA_GetActiveData ( INTVAL viHandle,  
                           STRREF rsDataID,  
                           INTVAL viDataIDLength );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>rsDataID</i>	Buffer returning data ID of active dataset
<i>viDataIDLength</i>	Length of <i>rsDataID</i>

Return Value

Either: 0 if call is successful
Or: Negative error code

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
----------------	---

QA_GetActiveLayout

Retrieves the layout that is currently active. This will be used for formatting the returned address.

Prototype

```
INTRET QA_GetActiveLayout ( INTVAL viHandle,  
                             STRREF rsName,  
                             INTVAL viNameLength );
```

Arguments

viHandle Handle for this instance of the API
rsName Name of layout
viNameLength Length of *rsName*

Return Value

Either: 0 if call is successful
Or: Negative error code

Error Scenarios

Busy Handle The parameter *viHandle* has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.

QA_GetData

Returns information about a given dataset. You can call this multiple times to get information about all available datasets, which can be manually specified using the `DataMappings` setting in the `qawserve.ini` file (see "Dataset Installation Settings" on page 189).

The function **QA_GetDataCount** must be called before this function.

Prototype

```
INTRET QA_GetData ( INTVAL viHandle,
                    INTVAL viDataOffset,
                    STRREF rsDataID,
                    INTVAL viDataIDLength,
                    STRREF rsName,
                    INTVAL viNameLength );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>viDataOffset</i>	Index into dataset count
<i>rsDataID</i>	Buffer returning the ID of the dataset
<i>viDataIDLength</i>	Length of <i>rsDataID</i>
<i>rsName</i>	Buffer returning dataset
<i>viNameLength</i>	Length of <i>rsName</i>

Return Value

Either:	0 if call is successful
Or:	Negative error code

Comments

The parameter *viDataOffset* should be passed an index into the available data count from **QA_GetDataCount**, between 0 to the count - 1.

The data ID returned from the parameter *rsDataID* is a short string identifying the dataset. This can be passed to **QA_SetActiveData** in order to change the current dataset being searched upon.

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Out of Sequence	The function QA_GetDataCount must have been called prior to this function.
Bad Index	The value passed to parameter <i>viDataOffset</i> was not a valid data offset. The range should be between 0 and the count of data - 1 from QA_GetDataCount .

QA_GetDataCount

Returns a count of the available datasets, which can be manually specified using the `DataMappings` setting in the `qawserve.ini` file.

This must be called before the data accessor function **QA_GetData** is used.

The count returned by this method is the number of separate datasets that can be passed to **QA_SetActiveData**, which sets the active dataset to be searched on.

Prototype

```
INTRET QA_GetDataCount ( INTVAL viHandle,  
                        INTREF riCount );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>riCount</i>	Count of datasets

Return Value

Either:	0 if call is successful
Or:	Negative error code

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
-------------	---

QA_GetEngine

Retrieves the current search engine. Refer to "Searching With QAS Pro" on page 25 for information about the differences between the three search engines and how to use them.

Prototype

```
INTRET QA_GetEngine ( INTVAL viHandle,  
                      INTREF riEngine );
```

Arguments

- viHandle* Handle to the API
- riEngine* The current active search engine

Return Value

- Either:** 0 if call is successful
- Or:** Negative error code

Comments

The possible search engines that can be returned from parameter *riEngine* are as follows:

Symbolic Name	Decimal Value	Description
qaengine_SINGLELINE	1	Single line engine
qaengine_TYPEDOWN	2	Typedown engine
qaengine_KEYFINDER	5	Keyfinder engine

Error Scenarios

- Busy Handle** The parameter *viHandle* has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.

QA_GetEngineOption

Retrieves the current settings used with a search engine.

Prototype

```
INTRET QA_GetEngineOption ( INTVAL viHandle,
                             INTVAL viEngOption,
                             LONGREF rlValue );
```

Arguments

- viHandle* Handle for this instance of the API
- viEngOption* Engine option to be returned
- rlValue* Value for the given engine option

Return Value

- Either:** 0 if call is successful
- Or:** Negative error code

Comments

The possible engine options that can be returned are listed below:

Symbolic Name	Decimal Value	Description
Options		
qaengopt_ASYNCSEARCH	1	Single line searches will be asynchronous.
qaengopt_ASYNCSTEPIN	2	Calls to QA_StepIn will be asynchronous
qaengopt_ASYNCREFINE	3	Picklist refinement will be asynchronous
qaengopt_THRESHOLD	6	Get the current picklist size threshold

Symbolic Name	Decimal Value	Description
qaengopt_TIMEOUT	7	Get the current search timeout (ms)
qaengopt_SEARCHINTENSITY	8	Get the current search intensity

The engine option types qaengopt_ASYNCSEARCH, qaengopt_ASYNCSTEPIN, and qaengopt_ASYNCREFINE will return one of the following values in parameter *rlValue*:

Boolean Values		
qavalue_FALSE	0	False
qavalue_TRUE	1	True

The engine option type qaengopt_SEARCHINTENSITY will return one of the following values in parameter *rlValue*.

Search Intensity Values		
qaintensity_EXACT	0	Exact searching
qaintensity_CLOSE	1	Close searching
qaintensity_EXTENSIVE	2	Extensive searching

The other engine option types will return an integer value.

The engine options that control asynchronous searching are as follows:

- qaengopt_ASYNCSEARCH
- qaengopt_ASYNCSTEPIN
- qaengopt_ASYNCREFINE

These can only be used for multithreaded versions of the library: the single threaded version of the library will not accept them. This is because they require the API to create threads if the options are enabled.

Error Scenarios

Busy Handle The parameter *viHandle* has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.

QA_GetEngineStatus

Retrieves the current status of a search engine. This allows you to check whether a particular engine is available for the data mapping you have selected.

Prototype

```
INTRET QA_GetEngineStatus ( INTVAL viHandle,  
                             INTVAL viEngine,  
                             INTRET riAvailable );
```

Arguments

viHandle Handle for this instance of the API
viEngine Engine to be selected
riAvailable Availability of the engine

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The possible search engines that can be passed to *viEngine* are:

Symbolic Name	Decimal Value	Description
qaengine_SINGLELINE	1	Single line engine
qaengine_TYPEDOWN	2	Typedown engine
qaengine_KEYFINDER	5	Keyfinder engine

The parameter *riValue* will return one of the following values:

Boolean Values		
qavalue_FALSE	0	False
qavalue_TRUE	1	True

QA_GetExampleCount

Returns the number of available example addresses for the current data set. The example addresses can be formatted and retrieved using **QA_FormatExample** and **QA_GetFormattedLine**.

Prototype

```
INTRET QA_GetExampleCount ( INTVAL viHandle,  
                           INTREF riExampleCount );
```

Arguments

viHandle Handle for this instance of the API
riExampleCount The count of example addresses that exist for the given dataset

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The Zero example number always exists and is a blank example address, useful for obtaining the number of layout lines.

Error Scenarios

- Busy Handle** The parameter *viHandle* has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
- Bad Layout** The active layout that is set for the instance is invalid. The list of valid layouts can be obtained by using the **QA_GetLayoutCount** and **QA_GetLayout** functions. A layout may be defined for a specific dataset only, and so changing the active data set may invalidate the active layout.

QA_GetFormattedLine

Returns the given formatted address line.

Pre-call Conditions

Either **QA_FormatResult** or **QA_FormatExample** must have been called for the current search.

Prototype

```
INTRET QA_GetFormattedLine ( INTVAL viHandle,  
                             INTVAL viLine,  
                             STRREF rsFormattedLine,  
                             INTVAL viFormattedLineLength,  
                             STRREF rsLabel,  
                             INTVAL viLabelLength,  
                             LONGREF rlContents );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>viLine</i>	The layout line to return
<i>rsFormattedLine</i>	The buffer to receive the formatted line
<i>viFormattedLineLength</i>	The size of <i>rsFormattedLine</i>
<i>rsLabel</i>	Label for the line
<i>viLabelLength</i>	The size of <i>rsLabel</i>
<i>rlContents</i>	Flags describing the contents of the line

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The parameter *viLine* should be passed an index into the count of formatted lines from **QA_FormatResult** or **QA_FormatExample**, between 0 and the count - 1.

The parameter *rsLabel* will return the name of any individual address element fixed to the line. If there are no elements (or more than one element) fixed to the line, this will be blank.

The parameter *rlContents* can return the following flags ORed together:

Symbolic Name	Decimal Value	Description
qaformatted_NAME	1	There are name elements fixed to this line.
qaformatted_ADDRESS	2	There are address elements fixed to this line.
qaformatted_ANCILLARY	4	There are ancillary elements fixed to this line.
qaformatted_DATAPLUS	8	There are dataplus elements fixed to this line.
qaformatted_TRUNCATED	16	Truncation occurred on this line, due to the width being too small.
qaformatted_OVERFLOW	32	Some elements were lost from this line, due to the width being too small and not enough lines being defined.
qaformatted_DATAPLUSSYNTAX	64	Dataplus is badly configured upon this line due to invalid syntax.
qaformatted_DATAPLUSEXPIRED	128	Dataplus is badly configured upon this line as it has expired.
qaformatted_DATAPLUSBLANK	256	Dataplus is blank on this line as there was no appropriate value in the data to return.

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Bad Index	The value passed to parameter <i>viLine</i> was not a valid data offset. The range should be between 0 and the count of data - 1 from QA_FormatResult or QA_FormatExample .

QA_GetLayout

Returns information about the given layout. You can call this multiple times to get a list of all layouts for the currently active dataset.

Pre-call Conditions

The function **QA_GetLayoutCount** should be called prior to this.

Prototype

```
INTRET QA_GetLayout ( INTVAL viHandle,  
                      INTVAL viLayout,  
                      STRREF rsName,  
                      INTVAL viNameLength,  
                      STRREF rsComment,  
                      INTVAL viCommentLength );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>viLayout</i>	Layout to return
<i>rsName</i>	Name of layout
<i>viNameLength</i>	Length of <i>rsName</i>
<i>rsComment</i>	Layout comment
<i>viCommentLength</i>	Length of <i>rsComment</i>

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The parameter *viLayout* should be passed an index into the count of available layouts from **QA_GetLayoutCount**, between 0 and the count - 1.

You can set the layout comment when you create a layout (see "Output Address Format Settings" on page 194). You may choose to display this comment to users when they change layout.

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Bad Index	The value passed to parameter <i>viLayout</i> was not a valid data offset. The range should be between 0 and the count of data - 1 from QA_GetLayoutCount .

QA_GetLayoutCount

Retrieves the number of available layouts in the configuration file for the active dataset.

Prototype

```
INTRET QA_GetLayoutCount ( INTVAL viHandle,  
                           INTREF riCount );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>riCount</i>	Number of available layouts in the configuration file

Return Value

Either:	0 if call is successful
Or:	Negative error code

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
-------------	---

QA_GetLicensingCount

Returns the total number of licensed datasets, including DataPlus sets, that are available. This total includes all servers that the API is using. Data duplicated across servers will be reported multiple times. This function can be used to report the most serious warning that applies to the set of licensed data. This is useful to check that there are no immediate or impending issues with the installed data, without interrogating each set individually using **QA_GetLicensingDetail**.

If a count of datasets available to search with is required, use the **QA_GetDataCount** function instead.

Prototype

```
INTRET QA_GetLicensingCount ( INTVAL viHandle,  
                              INTREF riCount,  
                              LONGREF rlWarningLevel );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>riCount</i>	The count of licensed datasets
<i>rlWarningLevel</i>	Warning level for the set of licensed data

Return Value

Either:	0 if call is successful
Or:	Negative error code

Comments

The parameter *rlWarningLevel* will return one of the following values (sorted in increasing order of urgency and importance):

Symbolic Name	Decimal Value	Description
qalicwarn_NONE	0	No licence warning
qalicwarn_DATAEXPIRING	10	A dataset is about to expire
qalicwarn_LICENCEEXPIRING	20	A licence is about to expire
qalicwarn_EVALUATION	30	Data is on an evaluation licence
qalicwarn_DATAEXPIRED	40	A dataset has expired
qalicwarn_EVALLICENCEEXPIRED	50	An evaluation licence has expired
qalicwarn_FULLLICENCEEXPIRED	60	A full licence has expired
qalicwarn_LICENCENOTFOUND	70	Licence information cannot be found for a dataset
qalicwarn_DATAUNREADABLE	80	A dataset is not readable

Error Scenarios

Busy Handle The parameter *viHandle* has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.

QA_GetLicensingDetail

Returns detailed information about a given licensed dataset. The count of licensed datasets can be obtained from the function **QA_GetLicensingCount**.

Prototype

```
INTRET QA_GetLicensingDetail (  INTVAL viHandle,
                                INTVAL viLicence,
                                INTVAL viType,
                                LONGREF rlDetail,
                                STREF  rsDetail,
                                INTVAL viDetailLength );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>viLicence</i>	Index into count of licensed datasets
<i>viType</i>	Type of information detail to return
<i>rlDetail</i>	Integer detail
<i>rsDetail</i>	String detail
<i>viDetailLength</i>	Length of <i>rsDetail</i>

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The parameter *viLicence* should be passed an index into the count of licensed datasets from **QA_GetLicensingCount**, between 0 and the count - 1.

If the symbolic name of the result detail type passed to *viType* is prefixed with *qalicensestr_* then a string will be returned through the *rsDetail* parameter. If the symbolic name is prefixed with *qalicenseint_* then an integer will be returned through the *rlDetail* parameter.

The parameter *viType* is used to specify what piece of information you want to be returned. It can take one of the following values:

Symbolic Name	Decimal Value	Description
qalicensestr_ID	1	Returns the dataset name
qalicensestr_DESCRIPTION	2	Returns a brief description of what the dataset represents
qalicensestr_COPYRIGHT	3	Returns the copyright information for the dataset
qalicensestr_VERSION	4	Returns the version of the data
qalicensestr_BASECOUNTRY	5	Returns the data ID of the country to which the dataset is an extension
qalicensestr_STATUS	6	Returns the string describing the state of the data; for example if it is about to expire.
qalicensestr_SERVER	7	Returns the server name of where the dataset is being used
qalicenseint_WARNINGLEVEL	8	Returns the warning level for the dataset. This can take one of the values listed in the following table.
qalicenseint_DAYSLEFT	9	Returns the number of days left before the dataset is unusable
qalicenseint_DATADAYSLEFT	10	Return the number of days left before the data expires
qalicenseint_LICENCEDAYSLEFT	11	Return the number of days left until the dataset licence expires

The following warning levels can be returned for the qalicenseint_WARNINGLEVEL detail type:

Symbolic Name	Decimal Value	Description
qalicwarn_NONE	0	No licence warning
qalicwarn_DATAEXPIRING	10	The dataset is about to expire

Symbolic Name	Decimal Value	Description
qalicwarn_LICENCEEXPIRING	20	The licence for the dataset is about to expire
qalicwarn_EVALUATION	30	The data is on an evaluation licence
qalicwarn_DATAEXPIRED	40	The dataset has expired
qalicwarn_EVALLICENCEEXPIRED	50	The evaluation licence for the dataset has expired
qalicwarn_FULLLICENCEEXPIRED	60	The full licence for the dataset has expired
qalicwarn_LICENCENOTFOUND	70	Licence information cannot be found for the dataset
qalicwarn_DATAUNREADABLE	80	The dataset is not readable

Error Scenarios

Busy Handle

The parameter *viHandle* has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.

QA_GetPrompt

Returns the prompt text. This is a short sentence that can be displayed to prompt the user to search using the appropriate address items. For example when beginning a search using the typedown engine the prompt may be 'Enter place or postcode'.

Prototype

```
INTRET QA_GetPrompt (  INTVAL viHandle,
                        INTVAL viLine,
                        STRREF rsPrompt,
                        INTVAL viPromptLength,
                        INTREF riSuggestedInputLength,
                        STRREF rsExample,
                        INTVAL viExampleLength );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>viLine</i>	The prompt line to be returned. Always pass 0.
<i>rsPrompt</i>	Buffer to receive a prompt line
<i>viPromptLength</i>	The length of the receive buffer
<i>riSuggestedInputLength</i>	For future expansion
<i>rsExample</i>	For future expansion
<i>viExampleLength</i>	For future expansion

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The *riSuggestedInputLength*, *rsExample* and *viExampleLength* parameters should be treated as any other parameter, except that any returned values should be ignored for this release of QAS Pro.

Error Scenarios

Busy Handle The parameter *viHandle* has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.

QA_GetPromptStatus

Returns status information about the current prompt.

Prototype

```
INTRET QA_GetPromptStatus ( INTVAL viHandle,  
                             INTVAL viType,  
                             INTREF riStatus,  
                             STRREF rsStatus,  
                             INTVAL viStatusLength );
```

Arguments

viHandle Handle for this instance of the API
viType Type of status information to receive
riStatus Integer status information
rsStatus String status information
viStatusLength Length of *rsStatus*

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

If the symbolic name of the result detail type passed to *viType* is prefixed with *qapromptstr_* then a string will be returned through the *rsStatus* parameter. If the symbolic name is prefixed with *qapromptint_* then an integer will be returned through the *riStatus* parameter.

The parameter *viType* is used to specify what piece of information you want to be returned. It can take one of the following values:

Symbolic Name	Decimal Value	Description
qapromptint_DYNAMIC	2	Returns whether or not the current prompt is dynamic. See comments below.

The prompt status type `qapromptint_DYNAMIC` will return one of the following values in parameter *riStatus*.

Boolean Values		
<code>qavalue_FALSE</code>	0	False
<code>qavalue_TRUE</code>	1	True

The prompt status affects the way in which a GUI should display the text box in which the user enters the search string. A non-dynamic prompt should be blanked after every call to **QA_Search** and **QA_StepIn** as the search text should not be retained for the next stage. A dynamic prompt should only be blanked when **QA_StepIn** is called, and should retain the text after a call to **QA_Search**.

The prompt status is designed to be obtained before the call to **QA_Search**, not afterwards.

Error Scenarios

Busy Handle The parameter *viHandle* has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.

QA_GetResult

Obtains limited information about a given picklist item. The result count comes from **QA_GetSearchStatus**.

The flags that are returned for a given picklist item indicate what actions can be performed upon it; for example, you can only step into a result using **QA_StepIn** if it has a `qaresult_CANSTEP` flag; otherwise an error will be returned.

Prototype

```
INTRET QA_GetResult ( INTVAL viHandle,  
                      INTVAL viResult,  
                      STREF rsDescription,  
                      INTVAL viDescriptionLength,  
                      INTREF riConfidence,  
                      LONGREF rlFlags );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>viResult</i>	Index into available results
<i>rsDescription</i>	Result text that will be displayed in the picklist
<i>viDescriptionLength</i>	Size of <i>rsDescription</i>
<i>riConfidence</i>	Confidence score of match
<i>rlFlags</i>	Flags describing the type of the given picklist item

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The parameter *viResult* should be passed an index into the count of available picklist items from **QA_GetSearchStatus**, between 0 and the count - 1.

The value returned in *riConfidence* is an indication of how good a match is. The maximum value is 100%.

The parameter *rsDescription* returns the text of each result item. The result shows only enough information for your users to be able to distinguish between different matches. This element should be shown in a picklist to your users to allow them to choose the result to step in to.

The parameter *rlFlags* can return the following flags ORed together. See "Flags Returned" on page 73.

Symbolic Name	Decimal Value	Description
qareult_FULLADDRESS	1	You have reached the full deliverable address and you can now call QA_FormatResult (see page 89).
qareult_MULTIPLES	2	This item represents multiple address lines.
qareult_CANSTEP	4	This item can be stepped into, using QA_StepIn .
qareult_ALIASMATCH	8	The match is an alias. See "Alias Matching" on page 34 for more information.
qareult_POSTCODERECODED	16	The picklist item has a recoded postcode. This is currently only available with GBR data; please refer to your Data Guide for more information.
qareult_CROSSBORDERMATCH	32	The picklist item represents a nearby area outside the strict boundaries of the initial search. This applies to bordering localities, which are only relevant to AUS data. If you are using AUS data, refer to your Australia Getting Started guide.
qareult_DUMMYPOBOX	64	The item is the dummy PO Box item. See comments below this table.
qareult_NAME	256	The picklist item is a Names result.

Symbolic Name	Decimal Value	Description
qareult_INFORMATION	1024	The result item is an informational prompt (see comments below).
qareult_WARNINFORMATION	2048	Warning informational prompt item (see comments below).
qareult_INCOMPLETEADDR	4096	The dummy item in premise-less countries (see comments below).
qareult_UNRESOLVABLERANGE	8192	A static range item that cannot be expanded. This applies to USA data and some European datasets only (see comments below).

A dummy PO Box picklist item is one that can be stepped into and contains PO Box type addresses beneath. This should not be handled as a special case in an integration, but a GUI may choose to use a different icon to display.

An informational picklist item is one that does not correspond to an address, but instead conveys useful information to the user. Some informational prompts can be stepped into, and so must never be filtered by the integration. The item itself should not be handled as a special case in an integration, but a GUI may choose to use a different icon to display.

A dummy item in a premise-less dataset will be returned for datasets where premise-level detail is not available. When you step into a picklist item that corresponds to a street, then this item may be returned in the resulting picklist. The result description of this item will prompt the user to enter the building details. This can then be used to refine the picklist and generate the desired address. The item itself should not be handled as a special case in an integration, but a GUI may choose to use a different icon to display.

An unresolvable range item will be returned for a premise range that cannot be stepped into using **QA_StepIn**. Instead, you must type text to refine the picklist further to generate the desired complete address. The item itself should not be handled as a special case, but a GUI integration may choose a different icon to display.

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Bad Index	The value passed to parameter <i>viResult</i> was not a valid picklist item offset. The range should be between 0 and the picklist result count - 1.

QA_GetResultDetail

Obtains limited information about a picklist item. The result count comes from **QA_GetSearchStatus**.

Prototype

```
INTRET QA_GetResultDetail ( INTVAL viHandle,  
                             INTVAL viResult,  
                             INTVAL viType,  
                             LONGREF rlDetail,  
                             STRREF rsDetail,  
                             INTVAL viDetailLength );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>viResult</i>	Index into available results
<i>viType</i>	Type of result detail requested
<i>rlDetail</i>	Returned detail integer
<i>rsDetail</i>	Returned detail string
<i>viDetailLength</i>	Length of <i>rsDetail</i>

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The parameter *viResult* should be passed an index into the count of available picklist items from **QA_GetSearchStatus**, between 0 and the count - 1.

If the symbolic name of the result detail type passed to *viType* is prefixed with *qaresultstr_* then a string will be returned through the *rsDetail* parameter. If the symbolic name is prefixed with *qaresultint_* then an integer will be returned through the *rlDetail* parameter.

The parameter *viType* is used to specify what piece of information you want to be returned. It can take one of the following values:

Symbolic Name	Decimal Value	Description
qareultstr_DESCRIPTION	2	Return the text description for the given item, suitable for displaying in a picklist
qareultstr_PARTIALADDRESS	3	Return the partial address for the given result, formatted in a single line. See comments below.
qareultint_ISFULLADDRESS	13	Return whether the given item is a full deliverable address and you can now call QA_FormatResult (see page 89).
qareultint_ISMULTIPLES	14	Return whether the given item represents multiple address lines.
qareultint_ISCANSTEP	15	Return whether the given item can be stepped into, using QA_Stepln .
qareultint_ISALIASMATCH	16	Return whether the given item is an alias. See "Alias Matching" on page 34 for more information.
qareultint_ISPOSTCODERECODED	17	The picklist item has a recoded postcode. This is currently only available with GBR data; please refer to your Data Guide for more information.
qareultint_ISCROSSBORDERMATCH	18	Return whether the given item represents a nearby area outside the strict boundaries of the initial search. This applies to bordering localities, which are only relevant to AUS data. If you are using AUS data, refer to your Australia Getting Started guide.

Symbolic Name	Decimal Value	Description
qaresultint_ISDUMMYPOBOX	19	Return whether the given item is the dummy PO Box item (see comments below).
qaresultint_ISNAME	20	Return whether the given item is a Names result.
qaresultint_ISINFORMATION	21	Return whether the given item is an informational prompt (see comments below).
qaresultint_ISWARNINFORMATION	22	Return whether the given item is a warning informational prompt (see comments below).
qaresultint_ISINCOMPLETESADDR	23	Return whether the given item is a dummy item in premise-less countries (see comments below).
qaresultint_ISUNRESOLVABLERANGE	24	Return whether the given item is a static range item that cannot be expanded. This applies to USA data only (see comments below).

For the types that are prefixed qaresultint_IS, the following values can be returned from the parameter *r/Detail*:

Boolean Values	
qavalue_FALSE	0 False
qavalue_TRUE	1 True

A dummy PO Box picklist item is one that can be stepped into and contains PO Box type addresses beneath. This should not be handled as a special case in an integration, but a GUI may choose to use a different icon to display.

An informational picklist item is one that does not correspond to an address, but instead conveys useful information to the user. Some informational prompts can be stepped into, and so must never be filtered by the integration. The item itself should not be handled as a special case in an integration, but a GUI may choose to use a different icon to display.

A dummy item in a premise-less country will be returned for datasets where premise-level detail is not available. When you step into a picklist item that corresponds to a street, then this item may be returned in the resulting picklist. The result description will prompt you to enter the building detail which will refine the picklist and generate the desired complete address. The item itself should not be handled as a special case in an integration, but a GUI may choose to use a different icon to display.

An unresolvable range item will be returned for a premise range that cannot be stepped into using **QA_StepIn**. Instead, you must type text to refine the picklist further to generate the desired complete address. The item itself should not be handled as a special case, but a GUI integration may choose a different icon to display.

The partial address for a given result is a single line showing the most complete address information that can be returned at that stage of the search. For example, if the user has typed down to street level, and the street has a unique postcode and locality, the partial address would consist of the street name, locality and postcode, without a premise name or number.

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Bad Index	The value passed to parameter <i>viResult</i> was not a valid picklist item offset. The range should be between 0 and the picklist result count - 1.

QA_GetSearchStatus

Obtains the status of a search. This function is an alternative to **QA_GetSearchStatusDetail**. Using this function, all of the search status is returned together through flags. See "Flags Returned" on page 73.

This function is used to retrieve the number of results matched by the engine; and also if anything unexpected has occurred, such as if the search times out or if there are too many matches to display.

This function can be called while a search is still in progress. This situation could occur if "Asynchronous Searching" (see page 76) was active, or if the integration uses a different thread to call the function.

Prototype

```
INTRET QA_GetSearchStatus (  INTVAL viHandle,
                             INTREF riPicklistSize,
                             INTREF riPotential,
                             LONGREF rlSearchState );
```

Arguments

viHandle Handle for this instance of the API
riPicklistSize Size of the picklist used for the index
riPotential Number of total potential items
rlSearchState The state after a search

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The flags with the prefix `qastate_AUTO` are related to auto stepping and formatting.

The parameter *riPotential* can be useful when the search has returned a number of matches over the threshold, so *riPicklistSize* is 1. *riPotential* then gives an idea of how far the results exceed the threshold. The value that is returned from *riPotential* is only an approximate estimate of the number of potential matches, which may indicate the extent of the refinement that should be performed in order to return a single result. This estimate should only be used as a guide to the user, and not relied upon in the integration.

The returnable flags (see "Flags Returned" on page 73) in the *riSearchState* parameter are ORed together, and are as follows:

Symbolic Name	Decimal Value	Description
qastate_NOSEARCH	1	No search has been started.
qastate_STILLSEARCHING	2	The search is still in progress.
qastate_TIMEOUT	4	The search has timed out.
qastate_SEARCHCANCELLED	8	The search has been cancelled using QA_CancelSearch .
qastate_MAXMATCHES	16	The maximum number of results has been reached, and there are too many to return: therefore, no results are returned. The search is finished and you must call QA_EndSearch before beginning a new one.
qastate_OVERTHRESHOLD	32	There are too many matches to display, as there are more picklist items than the threshold value. The threshold value is set in QA_SetEngineOption (see page 138).
qastate_LARGE POTENTIAL	64	Potentially, there are too many results to display, so you must keep typing (i.e., search with a larger string) in order to refine the search.
qastate_MORE OTHER MATCHES	128	There are additional other matches that can be displayed.

Symbolic Name	Decimal Value	Description
qastate_REFINING	256	Any text passed into QA_Search will be used to refine the current picklist rather than search.
qastate_AUTOSTEPINSAFE	512	The current picklist is trivial, therefore it is suggested that you immediately step into the first picklist item.
qastate_ AUTOSTEPINPASTCLOSE	1024	There are other close matches present, but only one exact match. Integrators may choose to step immediately into the first picklist item with QA_StepIn .
qastate_CANSTEPOUT	2048	Can step out of the picklist by calling QA_StepOut .
qastate_AUTOFORMATSAFE	4096	The current picklist is trivial, therefore it is suggested that you immediately format the first picklist item with QA_FormatResult .
qastate_ AUTOFORMATPASTCLOSE	8192	There are other close matches present, but only one exact match. Integrators may choose to format immediately the first picklist item with QA_FormatResult .

QA_GetSearchStatusDetail

Obtains detailed information about the status of a search. This function is an alternative to **QA_GetSearchStatus**. Using this function you can inquire about individual aspects of the search status separately, instead of obtaining all the information through the use of flags.

This function can be called while a search is still in progress. This situation could occur if you use asynchronous searching, or if the integration uses a different thread to call the function.

Prototype

```
INTRET QA_GetSearchStatusDetail ( INTVAL viHandle,  
                                  INTVAL viType,  
                                  LONGREF rlDetail,  
                                  STRREF rsDetail,  
                                  INTVAL viDetailLength );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>viType</i>	Type of detail requested
<i>rlDetail</i>	Returned detail integer
<i>rsDetail</i>	Returned detail string
<i>viDetailLength</i>	Length of <i>rsDetail</i>

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

If the symbolic name of the result detail type passed to *viType* is prefixed with *qassstr_* then a string will be returned through the *rsDetail* parameter. If the symbolic name is prefixed with *qassint_* then an integer will be returned through the *rlDetail* parameter.

The detail types with the prefix `qassint_ISAUTO` are related to auto stepping and formatting.

The parameter *viType* is used to specify the information that you want to be returned. It can take one of the following values:

Symbolic Name	Decimal Value	Description
<code>qassint_PICKLISTSIZE</code>	1	Returns the size of picklist.
<code>qassint_POTENTIALMATCHES</code>	2	Returns the number of potential matches.
<code>qassint_SEARCHSTATE</code>	3	Returns the search state flags (as returned by QA_GetSearchStatus).
<code>qassint_ISNOSEARCH</code>	4	Returns whether a search has not yet been started.
<code>qassint_ISSTILLSEARCHING</code>	5	Returns whether a search is currently in progress.
<code>qassint_ISTIMEOUT</code>	6	Returns whether the search timed out. The timeout limit is set in QA_SetEngineOption .
<code>qassint_ISSEARCHCANCELLED</code>	7	Returns whether the search has been cancelled using QA_CancelSearch .
<code>qassint_ISMAXMATCHES</code>	8	Returns whether the maximum number of results has been reached and there are too many to return.
<code>qassint_ISOVERTHRESHOLD</code>	9	Returns whether there are more picklist items than the threshold value, and therefore too many matches to display.
<code>qassint_ISLARGEPOENTIAL</code>	10	Returns whether there are potentially too many results to display and the search must be further refined.

Symbolic Name	Decimal Value	Description
qassint_ ISMOREOTHERMATCHES	11	Returns whether there are additional other matches that can be displayed.
qassint_ ISREFINING	12	Returns whether any text passed into QA_Search will be used to refine a picklist rather than to search.
qassint_ ISAUTOSTEPINSAFE	17	Returns whether the current picklist is trivial and you should step into the first item.
qassint_ ISAUTOSTEPINPASTCLOSE	18	Returns whether there are some close matches in the picklist but only one exact match that could be automatically stepped into using QA_StepIn .
qassint_ CANSTEPOUT	19	Returns whether the current picklist can be stepped out of using QA_StepOut .
qassint_ ISAUTOFORMATSAFE	20	Returns whether the current picklist is trivial and you should format the first item.
qassint_ ISAUTOFORMATPASTCLOSE	21	Returns whether there are some close matches in the picklist but only one exact match that could be automatically formatted.

For the types that are prefixed qassint_IS and for qassint_CANSTEPOUT, the following boolean values can be returned from the parameter *riDetail*:

Boolean Values		
qavalue_FALSE	0	False
qavalue_TRUE	1	True

QA_GetSystemInfo

Gets a line of the system information produced by **QA_GenerateSystemInfo**.

Prototype

```
INTRET QA_GetSystemInfo ( INTVAL viHandle,  
                           INTVAL viLine,  
                           STRREF rsBuffer,  
                           INTVAL viBufferLength );
```

Arguments

<i>viHandle</i>	Handle for this instance of the API
<i>viLine</i>	Line number to extract
<i>rsBuffer</i>	Returned system information line
<i>viBufferLength</i>	Size of <i>rsBuffer</i>

Return Value

Either:	0 if call is successful
Or:	Negative error code

Comments

The parameter *viLine* should be passed an index into the count of generated system information lines from **QA_GenerateSystemInfo**, between 0 and the count - 1.

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Bad Index	The value passed to parameter <i>viLine</i> was not a valid line of generated system information. The range should be between 0 and the count of lines - 1 from QA_GenerateSystemInfo .

QA_Open

Opens an instance of the API allowing you to specify the name of the configuration file to use and the section to use within that file.

The handle returned from *riHandle* is the one you will pass to all other functions. If the call to **QA_Open** fails for any reason, then the handle returned will not be valid to pass to other functions, except **QA_Close**.

There is no inherent limit on the number of instances that can be created with **QA_Open** and held simultaneously. System resources such as memory can limit the number, although this would never typically be reached for an integration upon a suitable machine.

For more information on handles and instances, see "API Instances" on page 73.

Prototype

```
INTRET QA_Open ( STRVAL vsIniFile,  
                 STRVAL vsSection,  
                 INTREF riHandle );
```

Arguments

vsIniFile Name of configuration file to open
vsSection Section of the configuration file to use
riHandle Instance handle returned by the API

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The layout for the instance will default to an internal layout that will be valid but may not be suitable. However, Experian QAS strongly recommends that you should call **QA_SetActiveLayout** after **QA_Open**.

If no configuration file is specified, the standard ini file qaworld.ini will be used. If no section is specified, the default 'QADefault' will be used.

A call to **QA_Open** must have a corresponding call to **QA_Close**.

Error Scenarios

INI File Error	The configuration file specified in parameter <i>vilniFile</i> cannot be opened.
Open Failure	The API was unable to create a new instance. Use the <code>LogErrors</code> configuration setting to determine where the problem lies (see "Output Address Format Settings" on page 194).

QA_Search

Performs a search using the current active dataset.

This function should be called when you have entered text to be searched upon. This will be in two cases:

- You are entering the initial search using the single line engine.
- You are entering text to refine the picklist further.

To finish an old search and perform a new one, you must first call **QA_EndSearch**.

See "Searching With QAS Pro" on page 25 for more information about how to search and refine addresses.

Prototype

```
INTRET QA_Search ( INTVAL viHandle  
                  STRVAL vsSearch );
```

Arguments

viHandle Handle for this instance of the API

vsSearch The search string

Return Value

Either: 0 if call is successful

Or: Negative error code

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Bad Layout	The active layout that is set for the instance is invalid. The list of valid layouts can be obtained by using the QA_GetLayoutCount and QA_GetLayout functions. A layout may be defined for a specific dataset only, and so changing the active dataset may invalidate the active layout.

QA_SetActiveData

Sets the active dataset to be searched upon. Data IDs for datasets are retrieved using **QA_GetData** which can be used to list all available datasets.

If the active dataset is changed once a search has begun, this will end the current search and reset all results.

Some layouts are defined only for specific datasets, and so changing the active dataset may invalidate the active layout.

Prototype

```
INTRET QA_SetActiveData ( INTVAL viHandle,  
                          STRVAL vsDataID );
```

Arguments

viHandle Handle for this instance of the API

vsDataID Data ID of the dataset to use

Return Value

Either: 0 if call is successful

Or: Negative error code

Comments

If you pass a blank string to *vsDataID*, the active data will be set to the first usable dataset found by the system.

Error Scenarios

Data Not Available The dataset specified in parameter *vsDataID* is unavailable. This should be passed dataset IDs as returned by **QA_GetDataCount** and **QA_GetData**.

QA_SetActiveLayout

This sets the layout that is used to format a final address. Layout names are retrieved using **QA_GetLayout** which can be used to get all available layouts.

Some layouts are defined only for specific datasets and so changing the active dataset may invalidate the active layout. The integration must ensure that either:

- All layout names are defined for all available datasets (advised)
- or
- The integration checks that the active layout is still valid when the active dataset is changed by calling **QA_GetLayoutCount** and **QA_GetLayout**.

You can call this function once a search has begun, although it will not affect an address that has already been formatted using **QA_FormatResult** or **QA_FormatExample**. If you wish to reformat an address using a different layout then you must recall the appropriate formatting function after changing the active layout.

Prototype

```
INTRET QA_SetActiveLayout ( INTVAL viHandle,  
                           STRVAL vsLayout );
```

Arguments

viHandle Handle for this instance of the API

vsLayout Layout name as retrieved by **QA_GetLayout**

Return Value

Either: 0 if call is successful

Or: Negative error code

Comments

The parameter *vsLayout* can optionally be passed a blank string instead of a valid layout name. This will set the layout to the default layout as specified in the [QADefault] section of the configuration file, or to an internal layout if this does not exist. This allows the integrator to specify a layout that is guaranteed to be valid for the active dataset.

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Bad Layout	The active layout that is set for the instance is invalid. The list of valid layouts can be obtained by using the QA_GetLayoutCount and QA_GetLayout functions. A layout may be defined for a specific dataset only, and so changing the active dataset may invalidate the active layout.

QA_SetEngine

Changes the current search engine. If the search engine is changed during a search, this will end the current search.

Prototype

```
INTRET QA_SetEngine ( INTVAL viHandle,  
                      INTVAL viEngine );
```

Arguments

viHandle Handle for this instance of the API
viEngine Engine to be selected

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The possible search engines that can be passed to *viEngine* are:

Symbolic Name	Decimal Value	Description
qaengine_SINGLELINE	1	Single line engine
qaengine_TYPEDOWN	2	Typedown engine
qaengine_KEYFINDER	5	Keyfinder engine

Error Scenarios

Busy Handle The parameter *viHandle* has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.

QA_SetEngineOption

Sets engine-related attributes for the given engine. The attributes that can be modified include threshold, timeout, search intensity and asynchronous operation.

Prototype

```
INTRET QA_SetEngineOption (  INTVAL viHandle,
                             INTVAL viEngOption,
                             LONGVAL vlValue );
```

Arguments

viHandle Handle to the API
viEngOption The engine option to be set
vlValue Value for this option

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

The possible engine options that can be passed to *viEngOption* are as follows:

Symbolic Name	Decimal Value	Description
qaengopt_DEFAULT	0	Resets the engine options to the defaults.
qaengopt_ASYNCSEARCH	1	Single line searches will be asynchronous. The default is qavalue_FALSE.
qaengopt_ASYNCSTEPIN	2	Calls to QA_StepIn will be asynchronous. The default is qavalue_FALSE.

Symbolic Name	Decimal Value	Description
qaengopt_ASYNCREFINE	3	Picklist refinement will be asynchronous. The default is qavalue_FALSE.
qaengopt_THRESHOLD	6	Allows you to set the result size threshold. The default is 25. The minimum limit is 5 and the maximum limit is 1000.
qaengopt_TIMEOUT	7	Allows you to set the timeout limit (ms). The default is 0 milliseconds (never times out) and the limit is 600000 (10 minutes).
qaengopt_SEARCHINTENSITY	8	Single-line mode edit distance level. The default is qaintensity_CLOSE.

The engine options that control asynchronous searching are as follows:

- qaengopt_ASYNCSEARCH
- qaengopt_ASYNCSTEPIN
- qaengopt_ASYNCREFINE

These can only be used for multithreaded versions of the library: the single threaded version of the library will not accept them. This is because they require the API to create threads if the options are enabled.

The type qaengopt_DEFAULT does not use the parameter *vIValue*.

The types qaengopt_ASYNCSEARCH, qaengopt_ASYNCSTEPIN, and qaengopt_ASYNCREFINE can have the following values passed to parameter *vIValue*:

Boolean Values		
qavalue_FALSE	0	False
qavalue_TRUE	1	True

The engine option type `qaengopt_SEARCHINTENSITY` can have the following values passed to parameter *v/Value*:

Search Intensity Values		
<code>qaintensity_EXACT</code>	0	Exact searching
<code>qaintensity_CLOSE</code>	1	Close searching
<code>qaintensity_EXTENSIVE</code>	2	Extensive searching

The search intensity setting represents how hard the Single Line search engine will search for an address with respect to the accuracy of matches. If the search intensity is set higher, then searches may take longer to perform, although more inexact matches will be returned.

When using `qaengopt_THRESHOLD` and `qaengopt_TIMEOUT`, an integer value should be passed to *v/Value*.

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Bad Value	The value passed to parameter <i>v/Value</i> is not valid for the given engine option.

QA_Shutdown

Closes down the API, and must be called as the final function, after all instances have been closed with **QA_Close**.

The shutdown function frees all resources associated with the Universal API and must be the last function to be called in the API before the calling program ends or unloads the API from memory.

Prototype

```
VOIDRET QA_Shutdown ( VOIDARG );
```

QA_StepIn

Selects a picklist item from a picklist to expand further.

You can only step into a result if it has a `qaresult_CANSTEP` flag; otherwise an error will be returned. See **QA_GetResult** on page 116 and **QA_GetResultDetail** on page 120 for more information.

Prototype

```
INTRET QA_StepIn ( INTVAL viHandle,  
                  INTVAL viResult );
```

Arguments

viHandle Handle for this instance of the API

viResult Index of picklist item

Return Value

Either: 0 if call is successful

Or: Negative error code

Comments

The parameter *viResult* should be passed an index into the count of available picklist items from **QA_GetSearchStatus**, between 0 and the count - 1.

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Bad Index	The value passed to parameter <i>viResult</i> was not a valid picklist item offset. The range should be between 0 and the picklist result count - 1.
Bad Step	The picklist result item that was passed into <i>viResult</i> is not an item that can be stepped into. Only picklist items that have the flag <code>qaresult_CANSTEP</code> from QA_GetResult can be stepped into.

QA_StepOut

Steps back a stage in a search, returning to the previous list of items.

This is not possible if a step in has not been performed. If **QA_StepOut** is called before **QA_StepIn**, an error will be returned. The search state `qastate_CANSTEPOUT` can be used to determine whether **QA_StepOut** can be called.

Prototype

```
INTRET QA_StepOut ( INTVAL viHandle );
```

Arguments

viHandle Handle for this instance of the API

Return Value

Either: 0 if call is successful

Or: Negative error code

Error Scenarios

Busy Handle	The parameter <i>viHandle</i> has been passed a handle that is already in use in another thread. Each handle can only be used by one thread at any point in time.
Out of Sequence	The API cannot 'step out' of a picklist if a search has not been yet performed.
Bad Step	The picklist cannot be stepped out from, as it is at the top level. This can be checked through the API by calling QA_GetSearchStatus and checking for the flag <code>qastate_CANSTEPOUT</code> .

User Interface API Reference

Handling Client/Server Errors

This section is only applicable if you have configured the QAS Pro client to use the QAS Pro server.

There are some network communication errors from which it is not always possible to recover. These error conditions are listed below:

- Server connection full

Whilst connecting to a server, the server's maximum connection count has been reached.

- Connection cancelled

This occurs when there is a failure to find a server, or when the connection to the server is lost.

- Connection timeout

The server has failed to respond within the timeout period.

If any of the above error conditions occur, the QAS Pro API will return `qaerr_NOLIVESERVER`. This error is fatal - you must therefore close the API by calling **QAProWV_UIShutdown**. (Any other operations will continue to return the error.)

You may continue operation by attempting to open the API with **QAProWV_UIStartup**.

If you have multiple servers configured, you will only need to call **QAProWV_UIStartup** once, as it will attempt to connect to each server in turn before returning `qaerr_NOLIVESERVER`.

Pseudocode Example Of QAS Pro API

This section provides an overview of how a program using the QAS Pro User Interface (UI) API works at a conceptual level. The pseudocode used is independent of any programming language.

The example below uses the main QAS Pro UI API functions to clarify how they work together. The pseudocode does not use all of the available functions.

The concept of the User Interface API is that a single instance of the QAS Pro application can be controlled through a small set of API functions. To create the application, the function **QAProWV_UIStartup** must be called. At this point, no graphical display will be visible.

The function takes a flag parameter *vFlags* which controls the overall behaviour of the user interface. See **QAProWV_UIStartup** on page 173 for a detailed explanation of each of the individual flags that can be passed.

Start the Pro application [**QAProWV_UIStartup**]

As with all API calls, the Open call could fail for various reasons. The most common reasons for this to occur are that the product is not installed or configured properly. If Open fails, address matching will not be available. When integrating the User Interface API, it is useful to enable error logging. For more information, see "Error Logging Settings" on page 210.

Repeat

The text that you want to search on should be obtained from the user from within the integration environment and passed to the function **QAProWV_UISearch**. If you do not want to obtain the text from the user integration, but would instead rather just popup the QAS Pro graphical interface in order to allow the user to enter the search directly into QAS Pro, then pass a blank string to **QAProWV_UISearch**.

Search with the provided text [**QAProWV_UISearch**]

This function call will return to the caller either once a final address has been selected, or when the application has been manually closed. The return value from the function **QAProWV_UISearch** can be tested to determine whether the user completed a search. We will assume in the pseudocode that a search has been successfully completed.

The next step is to retrieve the final address result which will have been formatted using the rules specified in the selected layout. First obtain a count of formatted lines, and then retrieve each one in turn:

```
Obtain a count of formatted lines [QAProWV_UIResultCount]
For Each formatted line
    Retrieve the line text [QAProWV_UIGetResult]
    Return text to integration
End For
Until no more searches are required
```

Once all required searches have been performed, the function **QAProWV_UIShutdown** must be called to free all allocated resources. For performance reasons, it is advised to only call **QAProWV_UIShutdown** once all searches have been performed, rather than between separate searches.

```
Shutdown the application [QAProWV_UIShutdown]
```

API Function Reference

The User Interface API functions can be split into the following groups:

- **General Functions**
Starting up and shutting down the API.
- **Search Functions**
Performing a search, and retrieving picklists of results and formatted addresses.
- **Housekeeping Functions**
Viewing and selecting available datasets, address formats and functionality flags.

Below is a full list of the QAS Pro User Interface API functions and where you can find them:

General Functions

QAProWV_UIStartup (see page 173)

Initialises the API.

QAProWV_UIShutdown (see page 172)

Closes down the API.

Search Functions

QAProWV_UISearch (see page 168)

Performs a search on an input string.

QAProWV_UIResultCount (see page 167)

Returns a count of the number of lines in a selected address.

QAProWV_UIGetResult (see page 159)

Retrieves a full matched address.

QAProWV_UIGetResultDetail (see page 161)

Retrieves detailed information regarding an address.

Housekeeping Functions

QAProWV_UILayoutCount (see page 163)

Returns a count of available layouts.

QAProWV_UIGetLayout (see page 157)

Retrieves the name of one layout.

QAProWV_UIGetActiveLayout (see page 153)

Retrieves the name of the currently selected layout.

QAProWV_UISetActiveLayout (see page 170)

Selects a new layout to use.

QAProWV_UILayoutLineElements (see page 164)

Retrieves the element fixed to a specific layout line.

QAProWV_UICountryCount (see page 150)

Returns a count of available datasets.

QAProWV_UIGetCountry (see page 154)

Retrieves the name of one dataset.

QAProWV_UIGetActiveCountry (see page 151)

Retrieves the name of the currently selected dataset.

QAProWV_UISetActiveCountry (see page 169)

Selects a new dataset to use.

QAProWV_UIGetFlags (see page 156)

Retrieves the currently-set functionality flags.

QAProWV_UISetFlags (see page 171)

Sets new functionality flags.

QAProWV_UICountryCount

Retrieves the number of installed datasets available to the API.

Pre-Call Conditions

The API has been started with **QAProWV_UIStartup**.

Prototype

```
INTRET QAProWV_UICountryCount ( INTREF riCount );
```

Arguments

riCount Number of datasets available

Return Value

Either: 0 if call is successful

Or: Negative error code

Possible Error Codes

-3407 qaerr_UIAPINOTSTARTED

Comments

This function tells you how many datasets are configured for the API. The availability of datasets is determined by your qawserve.ini file.

Once you have the number of datasets, you can call **QAProWV_UIGetCountry** as many times as is necessary to retrieve a description of each dataset.

QAProWV_UIGetActiveCountry

Retrieves the identifier of the currently active dataset.

Pre-call Conditions

The API has been started with **QAProWV_UIStartup**.

Prototype

```
INTRET QAProWV_UIGetActiveCountry (  STRREF rsBuffer,  
                                     INTVAL viLength );
```

Arguments

rsBuffer Buffer to receive country identifier

viLength Length of *rsBuffer*

Return Value

Either: 0 if call is successful

Or: Negative error code

Possible Error Codes

-3407 qaerr_UIAPINOTSTARTED

Comments

The identifier returned by this function is that of the dataset currently open for searching. Any search submitted to the API at this point with **QAProWV_UISearch** will be checked against this dataset.

This dataset is the one specified in your call to **QAProWV_UIStartup**, unless it has been changed since with **QAProWV_UISetActiveCountry** or via the user interface.

As an identifier is three characters long, you should use a minimum buffer of size of 4 to allow for a null terminating character.

QAProWV_UIGetActiveLayout

Retrieves the name of the current configuration layout.

Pre-call Conditions

The API has been started with **QAProWV_UIStartup**.

Prototype

```
INTRET QAProWV_UIGetActiveLayout ( STREF rsBuffer,  
                                   INTVAL viLength );
```

Arguments

rsBuffer Buffer to receive layout name

viLength Length of *rsBuffer*

Return Value

Either: 0 if call is successful

Or: Negative error code

Possible Error Codes

-3407 qaerr_UIAPINOTSTARTED

Comments

The layout name returned by this function is that of the configuration layout currently in use. Any address returned from a search submitted to the API at this point with **QAProWV_UISearch** will be formatted according to this layout.

The layout is the one specified in your call to **QAProWV_UIStartup**, unless it has been changed since with **QAProWV_UISetActiveLayout** or via the user interface.

QAProWV_UIGetCountry

Retrieves one dataset name and identifier, in conjunction with **QAProWV_UICountryCount**.

Pre-Call Conditions

QAProWV_UIStartup has been called to start the API. **QAProWV_UICountryCount** has been called to return a count of the available datasets.

Prototype

```
INTRET QAProWV_UIGetCountry ( INTVAL viIndex,  
                               STRREF rsIsoBuffer,  
                               STRREF rsNameBuffer,  
                               INTVAL viNameLength );
```

Arguments

viIndex Number of datasets (from 0 to **QAProWV_UICountryCount** - 1)
rsIsoBuffer Buffer to receive identifier of dataset
rsNameBuffer Buffer to receive name of dataset
viNameLength Maximum length of *rsIsoBuffer* and *rsNameBuffer*

Return Value

Either: 0 if call is successful
Or: Negative error code

Possible Error Codes

-3400 qaerr_INVALIDCOUNTRYINDEX

-3407 qaerr_UIAPINOTSTARTED

Comments

This function, in conjunction with **QAProWV_UICountryCount**, is useful if you want to confirm the number, names and identifiers of available datasets for a particular instance of the API. For example, you might want to use this functionality prior to the first call of **QAProWV_UISearch**, so that you know which datasets are available to search on.

You should call this function as many times as required to retrieve dataset details. For example, if **QAProWV_UICountryCount** returned a count of 4, you would call this function a maximum of four times to retrieve details of each dataset, setting *vilIndex* to 0, 1, 2 and 3.

The parameter *vilIndex* contains the number of the dataset whose details you want to retrieve – for example, inputting 0 retrieves the name of the first installed dataset, 1 returns the name of the second dataset, and so on.

The output parameters *rsIsoBuffer* and *rsNameBuffer* contain the identifier and name respectively of a dataset. An identifier is a unique three-letter descriptor for a dataset, which appears in the Data Guide supplied with your data. For example, the Australia dataset has the identifier AUS. The *rsIsoBuffer* parameter should always have at least a four-character buffer.

QAProWV_UIGetFlags

Returns the current configuration flags.

Pre-Call Conditions

The API has been started with **QAProWV_UIStartup**.

Prototype

```
INTRET QAProWV_UIGetFlags ( LONGREF rlFlags );
```

Arguments

rlFlags Line description flags

Return Value

Either: 0 if call is successful

Or: Negative error code

Possible Error Codes

-3407 qaerr_UIAPINOTSTARTED

Comments

This function tells you which functionality flags are currently set. The flags are originally set with **QAProWV_UIStartup**.

To add or remove flags, call **QAProWV_UISetFlags**.

QAProWV_UIGetLayout

Retrieves the name of a particular configuration layout.

Pre-Call Conditions

QAProWV_UIStartup has been called to start the API. **QAProWV_UILayoutCount** has been called to return a count of the available layouts.

Prototype

```
INTRET QAProWV_UIGetLayout ( INTVAL viIndex,  
                             STRREF rsBuffer,  
                             INTVAL viLength );
```

Arguments

viIndex Number of layout to retrieve (from 0 to **QAProWV_UILayoutCount** -1)
rsBuffer Buffer to receive layout name
viLength Maximum length of *rsBuffer*

Return Value

Either: 0 if call is successful
Or: Negative error code

Possible Error Codes

-3401 qaerr_INVALIDLAYOUTINDEX

-3407 qaerr_UIAPINOTSTARTED

Comments

This function, in conjunction with **QAProWV_UILayoutCount**, is useful if you want to confirm the number and names of available configuration layouts prior to searching.

You should call this function as many times as required to retrieve layout names from a configuration file. For example, if **QAProWV_UILayoutCount** returned a count of 6, you would call **QAProWV_UIGetLayout** six times to retrieve each layout name.

The parameter *viIndex* contains the number of the layout whose name you want to retrieve – for example, inputting 0 retrieves the name of the first layout in the configuration file, 1 returns the name of the second layout, and so on. An error will be returned if an invalid number is passed into this parameter.

QAProWV_UIGetResult

Retrieves one line of a returned address.

Pre-Call Conditions

The API has been started with **QAProWV_UIStartup**, and a search has been started with **QAProWV_UISearch**.

Prototype

```
INTRET QAProWV_UIGetResult ( INTVAL viIndex,  
                             STRREF rsBuffer,  
                             INTVAL viLength );
```

Arguments

viIndex Number of the line to retrieve (from 0 to **QAProWV_UIResultCount** -1)

rsBuffer Buffer to receive search result

viLength Maximum length of *rsBuffer*

Return Value

Either: 0 if call is successful

Or: Negative error code

Possible Error Codes

-3407 qaerr_UIAPINOTSTARTED

-3803 qaerr_INVALIDADDRESSLINE

Comments

This function returns one formatted address line from a search. You get the total number of address lines from the function **QAProWV_UIResultCount**.

You should call this function as many times as required to retrieve the full address. For example, if **QAProVV_UIResultCount** returned a count of 5 address lines, you would call this function five times to retrieve each item.

The parameter *viIndex* contains the number of the result line which you want to retrieve – for example, inputting 0 retrieves the description of the first line in the layout, 1 returns the second line, and so on. An error will be returned if an invalid line number is specified.

QAProWV_UIGetResultDetail

Retrieves the address details for an address selected from a returned picklist. This function also returns information to show the integrator whether or not a selected address has been modified by the user, whether the address is validated as a USPS Delivery Point (US data only), and whether the address has been truncated or whether any address details have been lost.

Pre-Call Conditions

The API has been started with **QAProWV_UIStartup**, and a search has been started with **QAProWV_UISearch**.

Prototype

```
INTVAL QAProWV_UIGetResultDetail ( INTVAL viType,  
                                   LONGREF rlDetail,  
                                   STRREF rsDetail,  
                                   INTVAL viDetailLength );
```

Arguments

<i>viType</i>	Type of result detail requested. Please refer to the following table for values for <i>viType</i>
<i>rlDetail</i>	Returned integer detail
<i>rsDetail</i>	Returned string detail
<i>viDetailLength</i>	Length of <i>rsDetail</i>

Return Value

Either:	0 if call is successful
Or:	Negative error code

Comments

Most of the flags relate to the address that was returned to the final address screen. If the user edits that address, then the validity of the flags will be uncertain.

If the symbolic name of the result detail type passed to *viType* is prefixed with *qaresultstr_* then a string will be returned through the *rsDetail* parameter. If the symbolic name is prefixed with *qaresultint_* then an integer will be returned through the *rlDetail* parameter.

The parameter *viType* is used to specify what piece of information you want to be returned (to the final address screen). It can take one of the following Boolean values:

Detail type	Description
qaresultint_ ISDPVVALID	Returns whether the given address is a valid USPS Delivery Point. This value only applies for US address data.
qaresultint_ ISEDITED	Returns whether that the user has manually edited the selected address, so any DPV assessment (returned on the original address) may no longer be valid.
qaresultint_ ISOVERFLOW	Returns whether there are insufficient address lines in the selected layout to accommodate all of the address details.
qaresultint_ ISTRUNCATED	Returns whether any address information has been truncated.
qaresultint_ ISUNVERIFIED	Returns whether the user has selected an unverified address (by pressing Ctrl+Enter prior to arriving at the final address screen).

For the types that are prefixed *qaresultint_IS*, the following values can be returned from the parameter *rlDetail*:

Boolean Values		
qavalue_FALSE	0	False
qavalue_TRUE	1	True

QAProWV_UILayoutCount

Retrieves the number of available layouts in the configuration file.

Pre-Call Conditions

The API has been started with **QAProWV_UIStartup**.

Prototype

```
INTRET QAProWV_UILayoutCount ( INTREF riCount );
```

Arguments

riCount Number of layouts in the configuration file

Return Value

Either: 0 if call is successful

Or: Negative error code

Possible Error Codes

-3407 qaerr_UIAPINOTSTARTED

Comments

This function tells you how many configuration layouts are available in your qawserve.ini file. The number available to use depends on the active country. See "Overview" on page 185 for a detailed description of configuration files and layouts.

Once you have the number of layouts, you can call **QAProWV_UIGetLayout** as many times as is necessary to retrieve the name of each layout.

QAProWV_UILayoutLineElements

Returns a description of the element fixed to a particular line of the currently selected address layout.

Pre-Call Conditions

The API has been started with **QAProWV_UIStartup** and a layout has been selected.

Prototype

```
INTRET QAProWV_UILayoutLineElements (  INTVAL viIndex,  
                                         STRREF rsBuffer,  
                                         INTVAL viLength,  
                                         LONGREF rlType );
```

Arguments

viIndex Address line to retrieve
rsBuffer Buffer to receive line elements
viLength Maximum length of *rsBuffer*
rlType Line type description

Return Value

Either: 0 if call is successful
Or: Negative error code

Possible Error Codes

-3411 qaerr_INVALIDLINEINDEX

Comments

This function tells you which address elements, if any, have been fixed to certain lines of the address layout. This function returns the name of an address element when there is only one element fixed to a line. If there is more than one element fixed to a line, a blank string will be returned.

If you want to retrieve descriptions of each address line from the current layout, you should call this function as many times as required. For example, if your address layout contains 6 address lines (as indicated by a call to **QAProWV_UIResultCount**), you could call this function six times to retrieve each layout line.

The parameter *vilIndex* contains the number of the address line whose description you want to retrieve. This is zero-based – for example, inputting 0 retrieves the description of the first line in the layout, 1 returns the second line, and so on.

The parameter *rsBuffer* contains the results of the function call. For example, if the town was fixed to line 4 of an address layout, you would set the value of *vilIndex* as 3, and *rsBuffer* would return 'Town'. If there are no elements fixed to the line that you have specified, the buffer will be empty.

The *rlType* parameter contains the type of line that is being retrieved. The types that can be returned are as follows:

Type Name	Decimal Value
element_ADDRESS	0
element_NAME	1
element_DATAPLUS	2
element_ANCILLARY	3

For example, you might have a seven-line address layout, where the first line contains name information, the second to sixth lines contain the address, and the final line is reserved for DataPlus data. In this case, the first line of the returned address would return *element_NAME*, the last line would return *element_DATAPLUS*, and the lines in between would return *element_ADDRESS*.

The values assigned to each type are symbolic constants defined by the API, and appear in the prototyped header files for each language.

This function will fail safely with an error if the *viIndex* parameter is out of range (for example, if you specify line 6 in a five line layout).

QAProWV_UIResultCount

Returns a count of the number of address lines in a selected address.

Pre-Call Conditions

The API has been started with **QAProWV_UIStartup**, and a search has been submitted with **QAProWV_UISearch**.

Prototype

```
INTRET QAProWV_UIResultCount ( INTREF riCount );
```

Arguments

riCount Number of address lines

Return Value

Either: 0 if call is successful
Or: Negative error code

Possible Error Codes

-3407 qaerr_UIAPINOTSTARTED

Comments

This function tells you the number of lines in the final selected address, and hence how many times the function **QAProWV_UIGetResult** needs to be called in order to retrieve them.

QAProWV_UISearch

Performs a search on the input string.

Pre-Call Conditions

The API has been started with **QAProWV_UIStartup**.

Prototype

```
INTRET QAProWV_UISearch ( STRVAL vsSearch );
```

Arguments

vsSearch Input string to search on

Return Value

Either: Positive value for successful search
Or: 0 if call is successful
Or: Negative error code

Possible Error Codes

-3407 qaerr_UIAPINOTSTARTED

Comments

If the specified search returns a single matching property, and the qaattribs_NOCONFIRMVIEW and qaattribs_MINDISPLAY flags are in effect, it can be retrieved immediately by calling **QAProWV_UIResultCount** and **QAProWV_UIGetResult**.

Otherwise, the search results will be displayed in the QAS Pro User Interface for the user to interactively confirm, before this function returns.

QAProWV_UISetActiveCountry

Changes the active dataset.

Pre-Call Conditions

The API has been started with **QAProWV_UIStartup**.

Prototype

```
INTRET QAProWV_UISetActiveCountry ( STRVAL vsIsoCode );
```

Arguments

vsIsoCode Identifier of the dataset to change to

Return Value

Either: 0 if call is successful

Or: Negative error code

Possible Error Codes

-3407 qaerr_UIAPINOTSTARTED

-3405 qaerr_CANTCHANGEDATABASE

Comments

This function allows you to switch between datasets without closing down the API. The dataset is originally set in the call to **QAProWV_UIStartup**.

For example, passing AUS in the *vsIsoCode* parameter changes the active dataset to Australia (if you have Australian data installed).

You can find out which datasets are available with the functions **QAProWV_UICountryCount** and **QAProWV_UIGetCountry**. An error will be returned if you specify the code of a database that is not installed.

QAProWV_UISetActiveLayout

Changes the configuration layout.

Pre-Call Conditions

The API has been started with **QAProWV_UIStartup**.

Prototype

```
INTRET QAProWV_UISetActiveLayout ( STRVAL vsLayout );
```

Arguments

vsLayout Layout name to change to

Possible Error Codes

-3402 qaerr_UNKNOWNLAYOUTNAME

-3407 qaerr_UIAPINOTSTARTED

Comments

This function allows you to switch between configuration layouts within an instance of the API. The layout was originally set with your call to **QAProWV_UIStartup**.

You can find out which layouts are available in the configuration file with the functions **QAProWV_UILayoutCount** and **QAProWV_UIGetLayout**.

The text in *vsLayout* must match one of the layout names returned from **QAProWV_UIGetLayout**, or an error will be returned. Note that the function is case-sensitive – for example, to select the layout QADefault, you cannot enter 'qadefault', as the API will not recognise it.

QAProWV_UISetFlags

Adds and removes configuration flags.

Pre-Call Conditions

The API has been started with **QAProWV_UIStartup**.

Prototype

```
INTRET QAProWV_UISetFlags ( LONGVAL vlRemove,  
                             LONGVAL vlAdd );
```

Arguments

vlRemove Flags to remove
vlAdd Flags to add

Return Value

Either: 0 if call is successful
Or: Negative error code

Possible Error Codes

-3407 qaerr_UIAPINOTSTARTED

-3414 qaerr_BADCONFIGFLAGS

Comments

This function allows you to change the flags that were set with **QAProWV_UIStartup**.

Call the function **QAProWV_UIGetFlags** for a list of the currently set flags.

QAProWV_UIShutdown

Closes down the API.

Pre-Call Conditions

The API is initialised.

Prototype

```
VOIDRET QAProWV_UIShutdown ( INTVAL viStatus );
```

Arguments

viStatus Last returned status code to be reported by the API prior to full shutdown.

Return Value

Either: 0 if call is successful
Or: Negative error code

Comments

This function will close down the API completely, and must be called as the final function.

If another function has returned an error, you can specify this as the input parameter in the call to **QAProWV_UIShutdown**. Then, when the API shuts down, a dialog box will appear with a description of the error. If you have received an error which you wish to handle yourself, or are shutting down without errors, pass 0 (zero) into the function.

QAProWV_UIStartup

Opens an instance of the API, specifying the name of the configuration file to be used, the layout to format addresses with, and the available functionality.

Pre-Call Conditions

None.

Prototype

```
INTRET QAProWV_UIStartup ( STRVAL vsTitle,  
                           STRVAL vsIniFile,  
                           STRVAL vsCountry,  
                           STRVAL vsLayout,  
                           STRVAL vsLanguage,  
                           LONGVAL vlFlags );
```

Arguments

<i>vsTitle</i>	Text to display in title bar
<i>vsIniFile</i>	Name of configuration file to open
<i>vsCountry</i>	Name of dataset to open
<i>vsLayout</i>	Name of configuration layout to use
<i>vsLanguage</i>	Reserved for future use (should be set to NULL)
<i>vlFlags</i>	Search functionality

Return Value

Either: 0 if call is successful
Or: Negative error code

Possible Error Codes

- 1031 qaerr_BADINIFILE
- 3405 qaerr_CANTCHANGEDATABASE
- 3406 qaerr_UIAPIALREADYSTARTED
- 3413 qaerr_NOLIVESERVER
- 3414 qaerr_BADCONFIGFLAGS

Comments

When you open an instance of the QAS Pro API, you need to specify five things:

- The title which appears at the top of the dialog together with the active dataset (defaults to blank) in *vsTitle*.
- A configuration file (default qaworld.ini) in *vsIniFile*.
- A layout within the configuration file (default QADefault) in *vsLayout*.
- The dataset to open for searching (defaults to the alphabetically first installed dataset) in *vsCountry*.
- The initial functionality to use in *vlFlags*.

If NULL or an empty string is passed into any of the above STRVAL input parameters, the API uses the defaults listed.

The flags that can be passed into *vlFlags* are as follows:

Symbolic Name	Decimal Value	Description
qaattribs_NONE (Default)	0	All options set from Configuration Editor. You do not need to specify any other flags.
qaattribs_NOCONFIRMVIEW	4	When a full address is found, it is returned directly to your application rather than to QAS Pro's address edit screen (see "Address Edit Screen" on page 53).
qaattribs_NOLAYOUTCHANGE	8	Disables ability to change layout. In order to enable this ability, the flag and the .ini setting must be set.
qaattribs_NOHELP	16	Disables Help facility. Note that the UShowHelp INI setting has similar functionality, and that there is no Help provided for the UI API.

Symbolic Name	Decimal Value	Description
qaattribs_NOCHANGEMODE	32	Disables ability to change searching mode. This needs to be set in conjunction with either qaattribs_SINGLELINESEARCH or qaattribs_TYPEDOWNSEARCH.
qaattribs_NOCHANGECOUNTRY	128	Disables ability to switch between datasets. In order to enable this ability, the flag and the .ini setting must be set.
qaattribs_SINGLELINESEARCH	256	Starts API in Single Line search mode.
qaattribs_TYPEDOWNSEARCH	512	Starts API in Typedown search mode.
qaattribs_KEYFINDERSEARCH	262144	Starts API in Key search mode.
qaattribs_NOPROBITMAP	8192	Displays the User Interface without the QAS Pro bitmap.
qaattribs_MINDISPLAY	16384	Only displays the QAS Pro User Interface if necessary - i.e. all searches are done from your application and the User Interface only appears if there is a picklist of results or an address to be confirmed/edited.
qaattribs_READONLY	65536	If this flag is set, users will be unable to edit addresses returned to the address edit screen. This overrides the Allow Address Editing option in Configuration Editor.
qaattribs_NOLICENSINGDLG	131072	Disables the 'Dataset Due to Expire' dialog, which is displayed by default when QAS Pro is started up. This lists all the datasets that are due to expire soon.

The value given with each flag represents that flag's hexadecimal value. If you want to specify more than one of these flags (for example, Single Line searching and no QAS Pro bitmap), the values should be ORed together.

If you attempt to set more than one search mode flag, the search engine will default to Typedown. For example, if you set `qaattribs_SINGLELINESEARCH` and `qaattribs_TYPEDOWNSEARCH` (by setting the *vFlags* parameter to 0x00000300), the API will default to Typedown mode. To change the search engine, call **QAProWV_UISetFlags** with only one of the search flags set.

You will get an error if QAS Pro API cannot find the specified configuration file, or if one of the datasets has expired or has been moved from its default location.

Fail-over logic is included so that **QAProWV_UIStartup** will loop through all of the defined servers until a successful connection can be established. See the Client/Server Guide or Client/Server Help for more information on fail-over logic.

Low-Level System Functions

The User Interface API also includes three low-level system functions.

The first of these is **QAErrorMessage**. This function translates a numeric error code into a simple textual explanation of that error. For a full list of error codes see the "Error Code Listing" on page 225.

Next is **QAErrorLevel**, which indicates the severity of an error and whether you should take action on it.

Finally comes **QASystemInfo**, which lists system usage details, such as what resources the API has taken from your operating system.

These functions are described on the following pages.

QAErroRMessage

This function translates an error code to a text message.

Prototype

```
VOIDRET QAErroRMessage ( INTVAL viStatus,  
                          STRREF rsBuffer,  
                          INTVAL viBuffLen );
```

Arguments

viStatus Error code

rsBuffer Buffer to receive error text message

viBuffLen Maximum buffer length (including room for NULL terminator)

Comments

This function is useful for converting an error code to a short text message that can be displayed to the user for informational purposes.

It is advised that **QAErroRMessage** is called after any function which returns an error code, as the text message might help you identify the cause of the error.

QAEErrorLevel

Returns the severity of an error.

Prototype

```
INTRET QAEErrorLevel ( INTVAL viStatus );
```

Arguments

viStatus Error code

Return Value

Either: 0 for a fatal or serious error

Or: 1 for a warning

Comments

This function indicates the severity of an error returned by the API. A fatal or serious error should be flagged to the user and must be dealt with. A warning should be handled in a manner appropriate to the condition and can, if desired, be ignored.

QASystemInfo

Returns detailed information about the system usage of QAS Pro.

Prototype

```
INTRET QASystemInfo ( INTVAL viLineNo,  
                      STRREF rsBuffer,  
                      INTVAL viBuffLen );
```

Arguments

viLineNo Line number being accessed (or negative if resetting)

rsBuffer Buffer to receive text line

viBuffLen Buffer length (including room for NULL terminator)

Return Value

Either: 0 if call is successful

Or: negative error code for an invalid line number

Comments

You must call **QAProWV_UIStartup** before calling this function in the UI API.

The system information text contains detailed information about QAS Pro, how it is configured, and the resources that it has taken from the operating system. The text is split over several lines and so has to be read one line at a time. A buffer size of 80 bytes will be sufficient to guarantee that no lines are truncated.

When the first line is read, the library generates an internal copy of the system information text. It is important that this copy is reset once all the lines have been read otherwise the allocated memory will not be freed. This is done by calling this function with the first parameter, *viLineNo* as -1 i.e. QASystemInfo (-1, NULL, 0).

Below is a C example that prints out **QASystemInfo** text:

```
void PrintSystemInfo(VOIDARG)
{
    char sBuffer[80];
    int iLineNo;
    /* read each line in turn */
    for (iLineNo = 0;
        QASystemInfo(iLineNo, sBuffer, sizeof(sBuffer)) == 0;
        iLineNo++)
    {
        puts(sBuffer);
    }
    /* reset in order to free memory */
    QASystemInfo(-1, NULL, 0);
}
```

If you run the previous example, you get a result similar to this:

Program:	QAUPIED
Copyright:	QAS Ltd
Release:	6.00 (131)
Platform:	Windows 32-bit
Libraries:	QAKRNXF 2.23(227)
	QAALSXD 2.0(13)
	QAUFWXF 5.00(244.3)
	QACOMXD 1.01(168)
	QAHSGXD 3.00(112)
	QAGENXD 1.00(27)
	QAHSVXD 3.30(114)
	QALICXD 1.00(29)
	QAHCLXD 3.00(74)
	QACDIXD 4.50(355)
	QADC2XD 2.40(68)
	QADS2XD 2.00(23)
	QAWD2XD 2.40(84)
	QAUSGXD 4.00(134)
	QALL2XD 2.40(106)
	QAUTDXD 4.00(139)
	QAUTDXD 4.00(142.4)
	QATIGXD 4.00(124.4)
	QATIXXD 4.00(120)

QAZLCXD 1.01(42)
QAZLGXD 4.02(71)
QAZLSXD 4.02(57.2)

Config: C:\Program Files\QAS\QAS Pro API\qaworld.ini
Section: QAUPIED

Dongle: <none>

Prog Dir: C:\Program Files\QAS\QAS Pro API
Home Dir: C:\Program Files\QAS\QAS Pro API
Data Dir: C:\Program Files\QAS\QAS Pro API
Temp Dir: C:\Temp
Log File: <disabled>

Memory: 1099010 (allocs=134)

Data: 969928 (free=11216)
Blocks: 1465 (free=410)

Server-side INI file: C:\Program Files\QAS\QAS Pro API\qawserve.ini

Data ID: AUS
Country Name: Australia
Copyright: Australia Post
Version: 30 October 2004 (PAF v2005.1)
Data Expiry: 64 days
File path: Q:\data\AUS.dts

Data ID: GBR
Country Name: United Kingdom
Copyright: The Post Office
Version: 20 September 2004
Data Expiry: 267 days
File path: Q:\data\GBR.dts

Data ID: USA
Country Name: United States of America
Copyright: United States Postal Service
Version: 15 November 2004

Data Expiry: 63 days
File path: Q:\data\usa.dts

API Configuration

Overview

Before you can perform any searches with the QAS Pro API, you need to specify where QAS Pro will search for addresses, and the format in which output addresses are returned. This can be done using the Configuration Editor (Windows only), or by editing the configuration files.

Configuration Editor

QAS Pro API is supplied with a Win32 Configuration Editor. This enables you to do the following:

- create, edit and manage formatted address layouts for QAS Pro;
- install and manage QAS Pro datasets;
- set QAS Pro configuration options;
- control user access options in QAS Pro.

The Configuration Editor contains the ability to specify Pasting Options, but these have no effect on the QAS Pro API.

For more information about the Configuration Editor, see the Configuration Editor help.

Configuration Files

QAS Pro bases processing decisions on configuration (INI) files. There are two configuration files supplied with QAS Pro: qaworld.ini and qawserve.ini.

- The qawserve.ini file is used to specify which datasets will be searched against, and to define layouts, which will be used to format the addresses

found by QAS Pro.

In Client/Server mode, the qawserve.ini file is held by the Server, and is not required at the Client. Please see the Client/Server documentation for further information on editing this file.

The qawserve file is automatically used when the API is initialised. You should not rename this, move it or attempt to call it with any of the API functions.

- The qaworld.ini file is used to specify the behaviour of the application; for example, search timeouts and picklist size thresholds.

If you are running the QAS Pro API in Client/Server mode, there are a number of additional settings that must be specified in the qaworld.ini file. For more information, refer to the Client/Server manual that was shipped with the Client/Server version of QAS Pro.

The remainder of this chapter explains how to use the available configuration settings in the configuration files.

Format Of A Configuration File

A configuration file can contain several sections, each comprising a number of settings, which may or may not specify a layout. To view a configuration file, such as the standard qawserve.ini file, use a plain text editor. Do not use a formatting editor such as Microsoft Word because it will corrupt the configuration file with its own formatting codes. Instead, use something like Notepad (under Windows) or vi (under UNIX).

The sections within the configuration file have their titles in square brackets:

[section name]

The following section should be included in both configuration files:

[QADefault]

QADefault is, as its name suggests, the default section. You should not alter anything within this section; to create alternative settings, copy QADefault, rename it and then edit the copy.

Within INI files, section names define the beginning of each section. Section names must be enclosed within square brackets and left-justified. A section ends when a new section name is declared. The final section is terminated by the end of the file.

Each section comprises a set of instructions in the form of keyword assignments, like this:

keyword=value

A keyword is the name of a setting. It can consist of any combination of letters and digits in uppercase or lowercase, and it must be followed immediately by an equals sign (=), which introduces the value assigned to the keyword. The value can be an integer, a string, or a quoted string, depending on the type of setting. There should be no space between the = and *value*.

You must not alter any keyword assignments in qawserve.ini apart from those documented in this manual.

The keyword assignments can come in any order. A typical keyword assignment looks like this:

```
USAAddressLineCount=4
```

This tells QAS Pro to create an output address consisting of four lines for the USA dataset.

Not all entries have to be keyword assignments. You can add comments by prefixing the comment with a semi-colon (;).

The ; character must be the first character on the line that is being commented.

You can also add a replacement for any commented-out setting. For example,

```
;EngineTimeout=0  
EngineTimeout=10
```

allows you to switch between settings by swapping the one which is commented out as required.

The Configuration Process

When setting up QAS Pro API for the first time, it is recommended that you follow the steps listed below:

1. Ensure that one (or more) dataset has been copied to a known location.
2. Configure an output format for the API to return a matched address (optional).
3. Specify whether error logging is enabled.

The first step involves checking settings in `qawserve.ini`. The second requires you to specify keyword values in `qawserve.ini`, and the third step involves specifying keywords in `qaworld.ini`.

The 'Default' documented for each setting refers to the product default. If a setting is not defined in the `qawserve.ini` file, it will assume this default. Some settings have been configured in the `.ini` file; these configurations will override the product default.

To return the product to its shipping state, any changes that you have made to the `qawserve.ini` file need to be discarded, and the file returned to its original condition. Therefore, before you make any changes to this file, you should make a copy of it so it can be recovered in the event that you need to return the product to its shipping state. Alternatively, you can reinstall the product, which will have the same effect.

If you are running the QAS Pro API in Client/Server mode, there are a number of additional settings that must be specified in the `qaworld.ini` file. For more information, refer to the Client/Server manual that was shipped with the Client/Server version of QAS Pro.

Dataset Installation Settings

The file `qawserve.ini` is used to specify which datasets are available to QAS Pro, and where they are installed. To modify these settings, open the file `qawserve.ini` in a plain text editor. For Windows, this file can be found in the same directory as the library files; for UNIX, this file can be found in the `apps` directory.

You can manually specify the location of the `qawserve.ini` file using the `QAWSERVE` environment variable `QAWSERVE = path-name`.

The configuration keywords used to configure datasets are `InstalledData` and `DataMappings`, located in the default section `[QADefault]` of the `qawserve.ini` file. You may need to add these settings if they do not already exist.

InstalledData

Format:

`InstalledData={identifier},{path}`

Default:

Blank

Purpose:

This keyword lists the installed datasets by a three letter identifier and location. These datasets are the ones installed by the setup program or copied across from the supplied data CDs/DVDs. If you wish to change or add to them, you should run the setup program again or copy them from the supplied medium. Note that if you are also using Additional Datasets, they do not need to be listed in this setting.

If you have more than one dataset installed, the first dataset appears directly after the = sign, and each subsequent database appears on a new line preceded by a + sign.

For every line that you have specified here, you should also add a line in the DataMappings setting.

Example:

If you have installed the UK, Australia and Netherlands datasets in C:\Program Files\QAS\Data, this setting would appear as follows:

```
InstalledData=GBR,C:\Program Files\QAS\Data\  
+AUS,C:\Program Files\QAS\Data\  
+NLD,C:\Program Files\QAS\Data\
```

If you need to move one or more of your datasets, you should update this setting accordingly.

DataMappings

Format:

DataMappings={identifier},{dataset/group name},{dataset+additional datasets}

Default:

Blank

Purpose:

This allows you to specify which datasets or groups of additional datasets you are using, and what you want the identifier and name of the dataset or group of additional datasets to. These are groupings are referred to as datamappings. The identifier is a 3-digit alphanumeric code. You can specify both the dataset/group name and the datamapping identifier.

Note that when you call the **QA_GetData** function for the primary API, the identifier and names that are returned are the values specified in this setting.

If you add or remove datasets in the `InstalledData` setting, you should update this setting accordingly.

Example:

If you have the `InstalledData` setting set up to include the UK, Australia and Netherlands datasets in `C:\Program Files\QAS\Data`, and if you also have United Kingdom Names, Gas and Electricity additional datasets and want to create groups to search on different combinations of data simultaneously, this setting might appear as follows:

```
DataMappings=GBR,United Kingdom,GBR
+GBN,United Kingdom With Names,GBR+GBRNAM
+UTI,United Kingdom With Utility,GBR+GBRGAS+GBRELC
+AUS,Australia,AUS
+NLD,Netherlands,NLD
```

Warning Settings

The warnings settings in qawserve.ini specify how long before a dataset or licence expires the user will be warned. See "Licences" on page 5 and "Data Updates" on page 13 for more information.

NotifyDataWarning

Format:

NotifyDataWarning={Integer}

Default:

62

Purpose:

This sets a threshold number of days in advance of the data expiry which, when reached, triggers a warning to the user that they need to renew their data.

Example:

NotifyDataWarning=31

NotifyLicenceWarning

Format:

NotifyLicenceWarning={Integer}

Default:

0

Purpose:

This sets a threshold number of days in advance of the licence expiry which, when reached, triggers a warning to the user that they need to renew their licence.

Example:

NotifyLicenceWarning=31

Output Address Format Settings

These settings are found in the qawserve.ini file.

Each dataset has its own default address format, which conforms to local standards. For example, the default American address contains the house number followed by the street name, town name, state and ZIP code. The default German address, on the other hand, contains the street name followed by house number, postal code and town name. It is strongly recommended that you do not edit the default address format; if you wish to create a different format to fit your application, you should do so within a new configuration layout.

Identifiers

The settings in this section must all be immediately prefixed by a datamapping identifier.

It is possible to define address formats for more than one datamappings within a single configuration layout. Therefore, each of the settings in this section must be directly prefixed with the identifier of the relevant datamapping. For example, the setting Capitaliseltem for the Australia datamapping would become AUSCapitaliseltem. An address format must be set up for each datamapping that you have set up. Datamappings are set up with the `DataMappings` keyword (see "Dataset Installation Settings" on page 189).

Element Codes

Address elements are specified in the QAS Pro API INI file using element codes. Element codes are specific to (and only meaningful for) a particular dataset. A list of element codes for each dataset can be found in the relevant Data Guide; for example, the street element code in the US is S11, and the street name element code is S112.

It is not necessary to specify every element in order for them to be visible within the returned and formatted address. All standard elements will be returned in a suitable place between any elements which have been fixed.

AddressLineCount

Format:

[identifier]AddressLineCount={integer}

Default:

0

Purpose:

This defines the number of lines in the formatted output address. The format of each individual address line is specified with the `AddressLineN` keyword (see below).

The number of lines you specify should take into account DataPlus information lines that might be returned.

Example:

```
USAAddressLineCount=4
```

This tells QAS Pro to produce formatted output addresses of four lines for the USA.

AddressLineN

Format:

[identifier]AddressLine[N]=W<width>,<element list>

Default:

Blank

Purpose:

This specifies which address elements should appear on which line. You can add as many `AddressLineN` lines as you defined with the `AddressLineCount` keyword (see above), each time replacing N with the appropriate line number. W signifies that the number that follows it is the maximum width of the line in characters, and <element list> is a comma-separated list of dataset-specific element codes. By specifying an element code, you force QAS Pro to place that element on that line (if the element exists in the matched address).

DataPlus elements are formed from the base name and the element name (see example 2).

You can allow QAS Pro to automatically insert other suitable elements before, after or between fixed elements by using the format specifier '...'

Example 1:

```
NLDAddressLine1=W40,S11,...
```

This instructs QAS Pro to give line 1 of a Netherlands output address a maximum width of 40 characters. The street name is fixed to the line, and any subsequent elements can also appear on the line if they fit there.

Example 2:

```
AUSAddressLine6=W40,AUSMOS.Desc
```


This tells QAS Pro to give line 6 of a Australian output address a maximum width of 40 characters, and fix the description part of the MOSAIC DataPlus set to it.

Certain DataPlus elements contain imputed information by default. An imputed field is one where data does not exist for all addresses. In this scenario, the gaps in the data are filled in (imputed) using neighbouring data. Alternatively you can specify that you do not want the data to be imputed.

For example:

GBRWPT.Party	Standard DataPlus item
GBRWPT.Party.NotImputed	If there is no available value in the raw data for this DataPlus set for this address, this element will be blank.
GBRWPT.Party.IsImputed	The value of this item can be either Yes or No. If this item has been taken directly from the raw data, the value is No. Otherwise, the item has been imputed by Experian QAS, and the value is Yes.

CapitaliseItem

Format:

[identifier]CapitaliseItem={element list}

Default:

Blank

Purpose:

This keyword defines which address elements should appear in upper case in the formatted address. The value of the keyword is a list of element codes separated by spaces.

Example:

AUSCapitaliseItem=P12 X11

Means that the building name and country name will be capitalised in Australian output addresses.

AbbreviateItem

Format:

[identifier]AbbreviateItem={element list}

Default:

Blank

Purpose:

This keyword defines which address elements should be abbreviated in the formatted address. The value of the keyword is a list of element codes, which differ from dataset to dataset, separated by spaces.

Example:

AUSAbbreviateItem=L12

Means that the Australian state name will be abbreviated.

SeparateElements

Format:

[identifier]SeparateElements={boolean}

Default:

YES

Purpose:

This keyword specifies whether or not address elements on a single line are separated, usually by commas. Set this keyword to NO if you do not want to comma-separate address elements.

Example:

GBRSeparateElements=YES

Means that a UK street and locality on the same line would look like this:

Chester Road, Ash

If the keyword was set to NO, it would look like this:

Chester Road Ash

ElementSeparator

Format:

[*identifier*]ElementSeparator={separator sequence}

Default:

Blank, but is usually set in the ini file depending on the dataset.

Purpose:

This keyword defines which address elements and should be separated by additional characters in the formatted address.

Example:

```
GBRElementSeparator=C11{ ^ }
```

Here the element separator would be blank before and after United Kindom postcodes.

Element separators take precedence right over left. These settings can be enabled or disabled with the keyword `SeparateElements`.

ElementExtras

Format:

[identifier]ElementExtras={element list}

Default:

Blank

Purpose:

Places additional characters around an address element. The format is: Element code {extra characters^extra characters}.

Example:

```
USAElementExtras=L21{ (^) }
```

This places brackets around element L21 (County Name in the United States dataset).

TerminateLines

Format:

[identifier]TerminateLines={boolean}

Default:

NO

Purpose:

This keyword defines whether or not certain address lines are ended with a comma.

Example:

```
GBRTerminateLines=YES
```

This means that a five-line UK address could look like this:

6 Cedar Grove,
Bisley,
WOKING,
Surrey,
GU24 9EF

If the keyword was set to NO, the commas would not appear.

LineTerminator

Format:

[identifier]LineTerminator={terminator sequence}

Default:

Blank

Purpose:

Use this to end specific address elements with a comma, if `TerminateLines` is set to Yes.

Example:

```
AUSLineTerminator=C11{^} B11{, ^ }
```

In the example, C11 represents the postcode and B11 represents the PO Box for Australian addresses. This setting is not to be applied before or after the postal code or after PO Box (where nothing is used). A comma and space is used before the PO Box. These settings can be enabled or disabled using the `TerminateLines` keyword.

ExcludeItem

Format:

[identifier]ExcludeItem={element list}

Default:

Blank

Purpose:

Prevents an item from appearing in an address if it isn't fixed to a particular line.

Example:

```
DEUExcludeItem=C11
```

If the postcode has not been fixed to a specific line, then it will not appear in the address, even though it is in the list of default address items.

FlattenDiacritics

Format:

[identifier]FlattenDiacritics={boolean}

Default:

NO

Purpose:

The Flatten Diacritics option allows you to replace all diacritic characters, such as accents and umlauts, with their non-diacritic equivalents. For example, the Danish address

Degnsgårdvej 1
7840 Højslev

would change to

Degnsgardvej 1
7840 Højslev

if the `FlattenDiacritics` keyword is set to Yes.

Example:

ESPFlattenDiacritics=Yes

This means that all diacritic characters will be suppressed by QAS Pro.

CDFVariation

Format:

[identifier]CDFVariation={integer}

Default:

1

Purpose:

This keyword tells QAS Pro which Form of address to use when automatically fitting address elements into the returned address. This is for datasets which include more than one Form of address, such as the Netherlands, Belgium and Finland datasets. For more information about Forms of address, see the Data Guide that accompanies your dataset.

Example:

NLDCDFVariation=2

This example selects the second Form of address for the Netherlands dataset. The three Forms of address for Netherlands addresses are Official, NEN and TPG, so this setting would apply the NEN Form to your output addresses.

Comment

Format:

[identifier]Comment={text string}

Default:

Default View

Purpose:

This allows you to add a comment to a layout, which is displayed at the bottom of the Select Layout dialog.

Example:

GBRComment=Custom Layout for the United Kingdom

MultiValueDPSeparator

Format:

[identifier]MultiValueDPSeparator={string}

Default:

|

Purpose:

This allows you set the delimiter used to separate returned multiple DataPlus values.

Example:

```
GBRMultiValueDPSeparator=|
```

Error Logging Settings

Error logging can be used to help diagnose any problems preventing normal operation of the QAS Pro API. Problems such as failure to start up the API can be solved from observing the output to the log file.

Logging can be activated by modifying configuration settings in the [QADefault] section of the qaworld.ini file.

If you are running the QAS Pro API in Client/Server mode, these settings will only activate error logging on the client. For information about error logging on the server, refer to the Client/Server manual.

LogFile

Format:

LogFile={filename}

Default:

None

Purpose:

The `LogFile` setting enables you to specify a log file to which any errors that occur when you call API functions are written. These errors are only written if `LogErrors` is set to YES (see below) as well as specifying the name of the file you want to write to with `LogFile`.

It is recommended that you create a log file when integrating the API.

Example:

```
LogFile=error.log
```

This creates a log file called `error.log` in the same directory as the program files. You can also specify the full path of the file to write logging to.

LogErrors

Format:

LogErrors={boolean}

Default:

NO

Purpose:

This keyword specifies whether or not error logging is enabled. If `LogErrors` is set to YES, the `LogFile` keyword must contain a valid file path and name.

Example:

LogErrors=NO

This means that no errors are logged, even if a file name has been specified in `LogFile`.

Search Options And Results Settings

These settings are found in the qaworld.ini file.

EngineTimeout

Format:

EngineTimeout={integer}

Default:

0

Purpose:

This is the length of time (in milliseconds) that QAS Pro will spend on a search before it returns a timeout flag. The default setting of 0 sets a timeout to infinity. Increasing this limit may lead to long searches taking up system resources.

Example:

To set a timeout period of 30 seconds (30,000 milliseconds), you would use
`EngineTimeout=30000`

SLMaxMatches

Format:

SLMaxMatches={Integer}

Default:

1000

Purpose:

This defines the maximum number of matches (picklist entries) that can be returned by the single-line search engine. The maximum value is 1000. If this is set to 0, the maximum value will be used.

The higher the limit set, the longer the search will be if many suitable results are found, and, correspondingly, more system resources will be used.

Note that a certain amount of approximation applies to this limit due to the way that common matches can get merged together to form single picklist entries.

Example:

SLMaxMatches=100

ShowAllThreshold

Format:

ShowAllThreshold={Integer}

Default:

750

Purpose:

When a picklist is returned in QAS Pro that contains more items than the picklist threshold (set by the ini keyword `UPIThreshold`), one of two informational prompts will be displayed:

 Continue typing (or select to show all matches)

 Continue typing (too many matches)

If the number of potential picklist items is below the `ShowAllThreshold` setting, the first 'Continue typing (or select to show all matches)' prompt is displayed. The user can step into the informational prompt to display all available matches.

If the number of potential matches is above the `ShowAllThreshold` value, the second 'Continue typing (too many matches)' prompt is displayed instead. The user must continue typing to refine the search and so decrease the number of potential matches. When the number of potential matches falls below the `ShowAllThreshold` value, they can then step into the informational prompt to show all matches.

The maximum value for the `ShowAllThreshold` setting is 1000.

The higher the limit that is set when you are running the QAS Pro API in Client/Server mode, the more network resources will be used.

Note that a certain amount of approximation applies to this limit due to the way that common matches can get merged together to form single picklist entries.

Example:

ShowAllThreshold=50

UPIThreshold

Format:

UPIThreshold={Integer}

Default:

25

Purpose:

This defines the number of matches (picklist entries) that can be displayed in a picklist returned by the search engine. If this limit is exceeded, an informational prompt will be displayed, requesting the user to enter refinement text or to select the informational prompt to display all available matches.

The maximum value for this setting is 1000.

Example:

UPIThreshold=35

EngineIntensity

Format:

EngineIntensity={integer}

Default:

1

Purpose:

This keyword controls the degree of 'fuzzy' matching used when searching for addresses as specified by the input search terms. Possible values are as follows

Setting	Description
0	Exact
1	Default fuzzy matching
2	Extensive fuzzy matching

The fuzzy matching engine will start at the default setting, and look for a match to the input search terms. This can be CPU intensive if it is above 0 (Exact). If no matches are found, it will proceed through the levels up to and including level 2 (or until `EngineTimeout` occurs) before giving up.

You can choose between 'standard' and 'intensive' search options using the QAS Pro user interface, but the changes may not appear in the `EngineIntensity` setting. This is because the API checks the presence of the following registry key: `HKEY_CURRENT_USER\Software\QAS Systems\QAS Pro\UserIni\QADefault`. If this key exists, the API writes the settings in the registry key, and does not change the INI file. If the registry key is not found, the API changes the `EngineIntensity` keyword in the INI file.

Example:

EngineIntensity=0

MultiElementLabels

Format:

MultiElementLabels={boolean}

Default:

No

Purpose:

This keyword defines whether or not to return address labels, where multiple address elements have been fixed to a single line. See **QA_GetFormattedLine** on page 102 for more information on how to return the labels for address elements that are fixed to a line.

If this keyword is set to Yes, the address elements are returned separated by commas. If it is set to No, blank labels are returned for such address lines.

Example:

```
MultiElementLabels=Yes
```

ForceAccept

Format:

ForceAccept={character}

Default:

!

Purpose:

The Primary API uses this keyword to define the character that the user can type at the end of the premises/sub-premises information, to force the data to be accepted as a match. If you do not define a character, the feature is disabled.

Alternatively you can use the *vsExtra* parameter of **QA_FormatResult** (see page 89).

Example:

ForceAccept=\$

OemCharacterSet

Format:

OemCharacterSet={text string} [NoDiacritics]

Default:

ANSI

Purpose:

The QAS Pro API includes support for character sets that contain non-standard characters, such as diacritics (for example, accents and umlauts). The API also provides the ability to remove diacritic characters on address output.

The API needs to know which (OEM) character set the calling application is using. The character set can be configured using the following values in the qaworld.ini file:

- Text string — a string that indicates the generic character family. The default value is ANSI code page 1252 (Latin 1).
- NoDiacritics — a string that indicates that all diacritic characters have been replaced with their non-diacritic equivalents.

The following character sets are supported by QAS Pro. They are 8-bit character sets and can support diacritics and multiple code pages:

Family	Description
ANSI	The character sets as defined by the American National Standards Institute.
ASCII	As above but without diacritics.
DOS	DOS code page 850.

Example:

```
OemCharacterSet=DOS NoDiacritics
```

Informational Prompt Settings

These settings are found in the qaworld.ini file.

NoMatchesMessage

If you are using the UI API, you should use the `UINoMatchesMessage` setting instead of this one.

Format:

`NoMatchesMessage={text string}`

Default:

Address not recognised (type '!' to accept).

Purpose:

The Primary API uses this setting to define the prompt to display when returning property level information that cannot be matched against an address. If this setting is left blank, then the feature is disabled. Note that the accept character, '!', by default, is set with the `ForceAccept` setting (see "Search Options And Results Settings" on page 213).

Example:

`NoMatchesMessage=Unrecognised address (type '$' to accept)`

Other INI Keywords

The following INI keywords, although present in either qawserve.ini or qaworld.ini, should not be edited:

- AllowPartialAddress
- CDFVersion
- Configured
- DataplusLines
- InvalidateAncillary
- LastEdited
- PaddingCharacter
- PasteFooter
- PasteHeader
- PasteKeyBegin
- PasteKeyEnd
- PasteLine
- PasteLineX
- PasteMode
- PasteSelect
- PasteSelectX
- PasteSpeed
- ReadOnlyAddress
- UsedByProPnG=No
- UsedByProAPIUI=Yes
- UsedByProAPIPrimary=Yes
- UsedByProWeb=No
- UsedByProServer=No
- UsedByBatch=No
- UsedByBatchAPI=No

Error Code Listing

Below is a full list of error codes and their descriptions. Call the system function **QAErrorMessage** (see page 178) to retrieve the message associated with the returned error code, and **QAErrorLevel** (see page 179) to ascertain whether the error is serious or a warning.

Some of the error codes are documented in more detail later in this chapter.

Code	Message	Explanation
-1000	qaerr_FATAL	Fatal error
-1001	qaerr_NOMEMORY	Out of memory
-1002	qaerr_INITINSTANCE	Invalid multi-threading instance
-1005	qaerr_INITOOLARGE	INI file too large
-1006	qaerr_ININOEXTEND	Cannot extend INI file
-1009	qaerr_FILECHGDETECT	Cannot detect file changes
-1010	qaerr_FILEOPEN	File not found
-1011	qaerr_FILEEXIST	File already exists
-1012	qaerr_FILEREAD	File read failure
-1013	qaerr_FILEWRITE	File write failure
-1014	qaerr_FILEDELETE	Could not delete file
-1015	qaerr_FILERESV	Reserved device
-1016	qaerr_FILEACCESS	File access denied
-1017	qaerr_FILEVERSION	Incorrect version of data file
-1018	qaerr_FILEHANDLE	Maximum number of files open
-1019	qaerr_FILECREATE	Could not create file
-1020	qaerr_FILERENAME	Could not rename file

Code	Message	Explanation
-1021	qaerr_FILEEXPIRED	Data file has expired
-1022	qaerr_FILENOTDEMO	Can only access demonstration data
-1023	qaerr_FILETIMEGET	Failed to obtain file timestamp
-1024	qaerr_FILETIMESSET	Failed to modify file timestamp
-1025	qaerr_READFAIL	Disk read failure
-1026	qaerr_WRITEFAIL	Disk write failure
-1027	qaerr_BADDRIVE	Invalid drive
-1028	qaerr_BADDIR	Invalid directory
-1029	qaerr_DIRCREATE	Could not create directory
-1030	qaerr_BADOPTION	Invalid command line option
-1031	qaerr_BADINIFILE	Could not locate INI file
-1032	qaerr_BADLOGFILE	Could not create log file
-1033	qaerr_BADMEMORY	Invalid memory configuration
-1034	qaerr_BADHOTKEY	Invalid hot key
-1035	qaerr_HOTKEYUSED	Hot key already in use
-1036	qaerr_BADRESOURCE	Could not locate language file
-1038	qaerr_BADTMPDIR	Bad temporary directory
-1040	qaerr_NOTDEFINED	Entry not defined
-1041	qaerr_DUPLICATE	Entry duplicated
-1042	qaerr_BADACTION	Invalid (list) action
-1050	qaerr_CCFAILURE	Copy control failure
-1051	qaerr_BADCODE	Invalid copy control code
-1052	qaerr_CCACCESS	Copy control access denied
-1053	qaerr_CCNODONGLE	Dongle not configured
-1054	qaerr_CCNOUNITS	No units left on meter
-1055	qaerr_CCNOMETER	Meter not initialised
-1056	qaerr_CCNOFEATURE	Feature not supported
-1057	qaerr_CCINVALID	Softkey integrity failure
-1060	qaerr_CCINSTALL	Copy control not installed

Code	Message	Explanation
-1061	qaerr_CCEXPIRED	Allowable time expired
-1062	qaerr_CCDATETIME	Date / time is invalid
-1063	qaerr_CCUSERLIMIT	Number of concurrent users exceeded
-1064	qaerr_CCACTIVATE	Copy control installed but not activated
-1065	qaerr_CCBADDRIVE	Invalid copy control drive
-1066	qaerr_CCREGISTER	Product must be registered
-1070	qaerr_UNAUTHORISED	Not authorised
-1074	qaerr_NOLOCALEFILE	Locale file not found
-1075	qaerr_BADLOCALEFILE	Invalid locale file
-1076	qaerr_BADLOCALE	Unknown language / country
-1077	qaerr_BADCODEPAGE	Unknown code page
-1078	qaerr_RESOURCEFAIL	Resource lookup failure
-1080	qaerr_NOTHREAD	Invalid thread handle
-1081	qaerr_NOTLSMEMORY	Out of thread-local-storage
-1090	qaerr_NOTASK	Could not create task
-2300	qawdperr_BADFUNCTION	Bad function code to library
-2301	qawdperr_NOKERNEL	Kernel must be pre-initialised
-2302	qawdperr_NOTFOUND	One or more DataPlus items not found
-2303	qawdperr_BADLEADIN	Internal or data error
-2304	qawdperr_NOPLUGIN	Plug-in not found for this DataPlus set
-2305	qawdperr_ISOPEN	DataPlus file already open by this DataPlus handle
-2306	qawdperr_BADENTRY	Bad DataPlus entry detected (data error)
-2307	qawdperr_NOTOPEN	DataPlus set is not open
-2308	qawdperr_NOCOUNTRY	No country code specified
-2309	qawdperr_PLUGINIT	Problem initialising plug-in

Code	Message	Explanation
-2310	qawdperr_PLUGBADARG	Bad argument to plug-in - no result returned
-2311	qawdperr_NEWFORMAT	New file format detected
-2400	qallderr_BADFUNCTION	Bad function code
-2401	qallderr_NOKERNEL	Kernel is not initialised
-2402	qallderr_BADHEADER	Bad object header detected
-2403	qallderr_NOOBJSIZE	No object size (internal error)
-2404	qallderr_NOMOREOBJECTS	No more objects (result of attempted move)
-2405	qallderr_BADDICT	Bad dictionary entry detected
-2406	qallderr_BADDATA	Bad compressed data detected in rds
-2407	qallderr_INTDPPROB	More than one DataPlus item returned from lookup
-2408	qallderr_MAXDPLITEMS	Too many DataPlus references
-2409	qallderr_NONAMES	Can't open Names DataPlus set
-2410	qallderr_BEYONDEOF	Offset is beyond end of file
-2411	qallderr_BADOBJCHAIN	Invalid object chain from this object position
-2412	qallderr_BADRANGETYPE	Invalid range type detected
-2413	qallderr_BADRANGESTART	Bad range start value detected
-2414	qallderr_ADSALREADYOPEN	Requested ADS is already open
-2415	qallderr_DELTADeltaTACOMP	Error decompressing delta-delta index
-2416	qallderr_NOADSFORADTS	DTS reference not found in delta-delta index for this ADS
-2417	qallderr_NOADSFORADS	ADS reference invalid for this ADS
-2450	qacerr_BADFUNCTION	Bad function code
-2451	qacerr_BADCCLASS	Comms class not found
-2452	qacerr_NOLISFUNC	Listen function not specified in listen request

Code	Message	Explanation
-2453	qacerr_CCNOTIMP	Specified comms class is not yet implemented
-2454	qacerr_ALRDYLIST	Already listening on the specified port
-2455	qacerr_BADTFUNC	Bad transport function called (internal error)
-2456	qacerr_NOLISTENER	Destination is not listening
-2457	qacerr_BADHANDLE	Invalid handle specified
-2458	qacerr_CANCELLED	Connection was cancelled remotely
-2459	qacerr_NOBUFFER	Attempted to take a size of an uninitialised buffer
-2460	qacerr_GETINCHAR	Get called but there are insufficient characters
-2461	qacerr_NOKERNEL	Kernel is not initialised
-2462	qacerr_NOTXN	Can't call send or receive unless a transaction is in progress
-2463	qacerr_NOTXNUM	A transaction must have a non-zero transaction number
-2464	qacerr_TERMINATED	Talk was cancelled in the transaction callback
-2465	qacerr_NOREGFUNCTION	No registered function was found for this transaction number
-2466	qacerr_NOTXNMESSAGE	Receive called but no message waiting
-2467	qacerr_NOSERVERMEMORY	No server memory during transaction processing
-2468	qacerr_ATMAXBUFFER	Maximum transaction buffer size reached
-2469	qacerr_NOTLISTENHAND	Comms handle specified is not a listen handle!
-2470	qacerr_SERVERFULL	Maximum server connection count exceeded
-2471	qacerr_DELETESELF	Can't delete own connection

Code	Message	Explanation
-2472	qacerr_TIMEDOUT	A connection's transaction has timed out
-2473	qacerr_SCANLOCK	Scan already in progress on another instance
-2474	qacerr_VSNNOTSUPPORTED	This version not supported at server
-2507	TCPIP_CONNECTFAIL	The connect attempt failed
-2508	TCPIP_TRANSTHREADFAIL	Transmission thread failure
-2511	TCPIP_INSTANCENOTFOUND	Using invalid connection instance
-2514	TCPIP_TRANSTHREADFULL	Transmission thread message queue full
-2517	TCPIP_LISTENTHREADNOTINIT	The expected listen thread was not found
-2519	TCPIP_INVALIDMSGTYPE	Received invalid message type
-2520	TCPIP_MSGCORRUPTED	Received corrupted message
-2521	TCPIP_MSGQUEUEFULL	Outgoing message queue full
-2524	TCPIP_GETMOREDATA	Retrieving more data
-2526	TCPIP_CANCELLED	Connection cancelled remotely
-2549	TCPIP_LISTENSTRUCT	Failed to create listen data or administrator
-2900	qadcperr_BADFUNCTION	Bad function code
-2901	qadcperr_NOKERNEL	Kernel is not initialised
-2902	qadcperr_NOREGFUNCTION	Function number not recognised by server glue
-2903	qadcperr_CANTCREATEINSTANCE	Can't create server instance
-2904	qadcperr_REINITIALISE	No initialisation instance found for this client
-2905	qadcperr_BADHANDLE	Bad client dataplus handle specified
-2906	qadcperr_SERVERPROBLEM	Error at server
-2907	qadcperr_INVALIDHANDLE	Dataplus handle invalid

Code	Message	Explanation
-2908	qadcperr_BADCODE	Bad format for dataplus set code
-2909	qadcperr_SETNOTOPEN	Dataplus set is not open
-2910	qadcperr_CLIENTPLUGINERR	Error at client plug-in
-3400	qaerr_INVALIDCOUNTRYINDEX	Invalid index into UIGetCountry
-3401	qaerr_INVALIDLAYOUTINDEX	Invalid index into UIGetLayout
-3402	qaerr_UNKNOWNLAYOUTNAME	Unknown layout name in UISetActiveLayout
-3403	qaerr_UNKNOWNCOUNTRYCODE	UIStartup with unknown ISO code
-3404	qaerr_NOSEARCHENGINESAVAILABLE	Fatal case of the UF reporting no engines available
-3405	qaerr_CANTCHANGEDATABASE	The more common invalid/unknown ISO code error
-3406	qaerr_UIAPIALREADYSTARTED	Multiple calls to UIStartup
-3407	qaerr_UIAPINOTSTARTED	UIStartup not called
-3408	qaerr_UIBADSEARCHSTRING	No longer used
-3409	qaerr_UIBADCDFFORMAT	No longer used
-3411	qaerr_INVALIDLINEINDEX	Invalid index into UIGetLayoutLineElements
-3412	qaerr_NOSELECTEDLAYOUT	No layout in use during call to UIGetLayoutLineElements
-3413	qaerr_NOLIVESERVER	Fatal comms error
-3414	qaerr_BADCONFIGFLAGS	Bad combination of config flags
-3415	qaerr_CANTCHANGEENGINE	Failed to change search engine
-3416	qaerr_NOUFACTIVE	Internal error
-3417	qaerr_UFINVALIDSTATE	Internal sequence error
-3418	qaerr_UILICENSINGERROR	Licensing failure has occurred with one or more data sets
-3450	qaerr_BADFUNCTION	Bad function number
-3451	qaerr_NOKERNEL	No kernel initialised at library call
-3452	qaerr_NOTPRESENT	Shared entity not present

Code	Message	Explanation
-3600	qaerr_INITFAILURE	QAS TCP/IP failure
-3601	qaerr_CLEANUPFAIL	Sockets failed to clean up properly
-3602	qaerr_ACCEPTFAIL	Error accepting remote connection
-3603	qaerr_SOCKETBIND	Failed to bind socket
-3604	qaerr_LISTENFAIL	Failed to initialise the listen
-3605	qaerr_TALKFAIL	Failed to initialise the talk
-3606	qaerr_SOCKETFAIL	Failed to create socket
-3607	qaerr_CONNECTFAIL	(TCP/IP talk socket) failed to connect to target
-3608	qaerr_ADDRESSERROR	Error looking up remote hostname or address
-3609	qaerr_SENDError	Error sending data
-3610	qaerr_RECVERROR	Error receiving data
-3611	qaerr_SELECTERROR	Error during processing of socket select for pending data
-3612	qaerr_TOOLARGE	Message is too large to be transmitted by protocol
-3613	qaerr_PORTINUSE	Specified port number already in use / Non-reusable
-3614	qaerr_CONNECTREFUSED	Could not connect to remote host
-3615	qaerr_CONNECTIONCLOSED	Connection has been lost
-3616	qaerr_SOCKOPTERROR	Error setting socket option
-3617	qaerr_WAITTIMEDOUT	Wait timed out
-3618	qaerr_HOSTNOTFOUND	Host not found
-3622	qaerr_WOULDBLOCK	Socket would block
-3623	qaerr_UNEXPECTED	Unhandled error: please inform Experian QAS Technical Support
-3625	qaerr_WSAEINVAL	Invalid function or argument for socket

Code	Message	Explanation
-3627	qaerr_NOSOCKETS	(The listen thread has) no sockets attached
-3701	qaerr_CDFSyntax	Invalid CDF syntax
-3702	qaerr_BADCDFORDER	Inaccurate item order in CDF
-3703	qaerr_BADCDFITEM	Item in CDF is invalid
-3704	qaerr_INVALIDADDRESSEXAMPLE	Invalid address example
-3705	qaerr_INVALIDCDFOBJECT	Invalid CDF object
-3706	qaerr_INVALIDABBREV	Invalid abbreviation
-3708	qaerr_NOSUCHVARIATION	CDF does not have this many variations
-3709	qaerr_OBJECTDEFINITION	Objects incorrectly defined
-3710	qaerr_UNKNOWNOFFSET	Offset supplied was out of range
-3711	qaerr_INVPARSEERR	Push invalid parse has failed
-3801	qaerr_FORMATSYNTAX	Incorrect formatting syntax
-3802	qaerr_TOOMANYADDRLINES	Too many address lines requested
-3803	qaerr_INVALIDADDRESSLINE	Address line out of range
-3804	qaerr_NOFORMATSPEC	No format spec in INI file
-3805	qaerr_FORMATOVERFLOW	Format(s) have overflowed
-3806	qaerr_FORMATTRUNCATED	Format(s) are truncated
-3807	qaerr_BADCDFVERSION	CDF version incompatible with format
-3808	qaerr_UNKNOWNDPITEM	Incorrect DataPlus item name
-3809	qaerr_DPFAILURE	One or more DataPlus sets failed to open
-3810	qaerr_PREMISENEEDED	Enter premise details
-3811	qaerr_NEEDRANGEOFFSET	You need to enter number details within a range
-4300	qaerr_UFHANDLING	Engine would like UF to take over

Code	Message	Explanation
-4301	qaerr_UFNOTHANDLING	UF would like engine to handle again
-4302	qaerr_NOUFGUEINSTANCE	Glue not initialised
-4303	qaerr_UFNOHK	Housekeeping unavailable
-4304	qaerr_UFNOFUNC	Requested function unavailable
-4306	qaerr_UFCANTSTEP	Can't step in
-4307	qaerr_UFITEMRANGE	Chosen picklist item out of range
-4308	qaerr_UFLAYOUTRANGE	Invalid layout handle given
-4309	qaerr_NOUF	NULL handle given
-4310	qaerr_NOUFSEARCH	No search in progress
-4311	qaerr_ENGINEUNAVAILABLE	Engine cannot be selected
-4312	qaerr_UFCANCELLED	An action has been cancelled
-4313	qaerr_UFNOLAYOUT	No layout matched that supplied
-4314	qaerr_INEXACTUNAVAILABLE	Inexact matches go beyond threshold
-4315	qaerr_NODATAAVAILABLE	Data unavailable at this level
-4316	qaerr_MOREINEXACT	Data unavailable at this level
-4317	qaerr_CANTSTEPIN	Can't step in
-4318	qaerr_CANTSTEPOUT	Can't step out
-4319	qaerr_BADOPTIONS	Options value(s) wrong
-4320	qaerr_NOENGINESUPPORT	Underlying engine doesn't provide this function
-4321	qaerr_BADSEQUENCE	Functions called out of sequence
-4322	qaerr_MONIKER_BADVERSION	UF does not recognise this moniker's version number
-4323	qaerr_MONIKER_BADMESSAGE	UF does not recognise this moniker's message number
-4324	qaerr_MONIKER_WRONGENGINE	Current UF instance is not using this moniker's engine
-4325	qaerr_MONIKER_WRONGCOUNTRY	Current UF instance is not using this moniker's country

Code	Message	Explanation
-4326	qaerr_MONIKER_WRONGDATAVERSION	Current UF instance is not using this moniker's data version
-4327	qaerr_MONIKER_BADOFFSET	This moniker has a zero offset
-4328	qaerr_BADCOUNTRYBASEPATH	The option for the country basepath is bad
-4329	qaerr_BADPROMPTSETINDEX	The prompt set parameter was invalid (not 0..2)
-4330	qaerr_MONIKER_BADFORMAT	UF does not recognise this moniker's format
-4331	qaerr_MONIKER_BADEXAMPLE	The example number in this moniker is out of range
-4361	qaerr_CANTFORMATITEM	The informational item cannot be formatted (eg "No matches").
-4362	qaerr_NODATAMAPPINGS	No (valid) datamappings can be found in the QAWSERVE.ini file.
-4363	qaerr_MISSINGDATAMAP	The QAWSERVE.ini file contains no datamappings for this request.
-4364	qaerr_NOLAYOUTSELECTED	No layout is currently selected, but the API is expecting one.
-4501	qaerr_TOOMANYINSTANCES	Maximum number of open handles have been created
-4503	qaerr_MAXRESOURCES	Maximum resource limit reached for resource store
-4551	qaerr_OPENFAILURE	Failed to create an API instance
-4552	qaerr_APIHANDLE	Instance handle invalid
-4553	qaerr_OUTOFSEQUENCE	Function called out of sequence
-4554	qaerr_INSTANCEBUSY	Instance handle already being used
-4556	qaerr_BADINDEX	Index not within valid range
-4557	qaerr_BADVALUE	A value passed was not valid
-4558	qaerr_BADPARAM	Invalid parameter passed to API
-4559	qaerr_PARAMTRUNCATED	API output truncated

Code	Message	Explanation
-4560	qaerr_NOENGINE	Search engine is unavailable for this dataset
-4561	qaerr_BADLAYOUT	The active layout is invalid
-4562	qaerr_BADSTEP	Step-in/Step-out not allowed on item
-4570	qaerr_DATASETNOTAVAILABLE	Dataset cannot be used
-4571	qaerr_LICENSINGFAILURE	Licensing failure has occurred with one or more data sets
-4580	qaerr_SERVERCONNLOST	Lost connection to the server. Transaction timed out or server error
-4581	qaerr_SERVERFULL	The maximum number of server connections has been reached
-8300	qaerr_CTDIFUNC	Unknown client function
-8301	qaerr_CTDINOKERNEL	No kernel instance on client
-8304	qaerr_CTDIBOTTOM	Item is not steppable
-8305	qaerr_CTDISERVERERR	Typedown server error (reported by client)
-8400	qaerr_NOTDGLUEINSTANCE	No Typedown glue instance
-8401	qaerr_TDIFUNC	Unknown server function
-8402	qaerr_TDINOKERNEL	No kernel instance on server
-8403	qaerr_TDINOMATCH	No match
-8404	qaerr_TDIINDEXERROR	Error in typedown index
-8405	qaerr_THRESHOLDTOOLLOW	Threshold was below minimum
-8406	qaerr_TDINOHK	Housekeeping is not initialised
-8407	qaerr_TDINONAMES	Names cannot be opened
-8408	qaerr_TDCANCELLED	Typedown search was cancelled
-8409	qaerr_TDDISCONNECT	Typedown search was disconnected
-8410	qaerr_TDSSWOPFAIL	Typedown swap failed
-8411	qaerr_NORESULTS	Results unexpectedly not present
-8412	qaerr_EMPTYBITMAP	Bitmap was empty

Code	Message	Explanation
-8413	qaerr_TOOMUCHDATA	Comms exceeded
-8500	qaerr_HOUSEFUNC	Bad function code
-8501	qaerr_HOUSENOKERNEL	Kernel is not initialised
-8502	qaerr_REMOTEINLOCK	Remote INI file is locked
-8650	qaerr_ZLCFUNC	Function unavailable
-8651	qaerr_ZLCNOKERNEL	Kernel not initialised
-8652	qaerr_ZLCDISTANCE	Maximum search range cannot be less than minimum
-8653	qaerr_ZLCBADTERM	Search term must contain at least one letter or number
-8654	qaerr_ZLCNOMEMORY	Client out of memory
-8655	qaerr_ZLCTOOMANYMATCHES	Too many matches found
-8660	qaerr_ZLCABORT	Search cancelled
-8661	qaerr_ZLCTIMEOUT	Search timed out
-8750	qaerr_ZLSFUNC	Unknown function
-8751	qaerr_ZLSNOKERNEL	Kernel not initialised
-8752	qaerr_ZLSABORT	Search aborted
-8753	qaerr_ZLSTOOMANYMATCHES	Too many matches found
-8754	qaerr_ZLSTIMEOUT	Search timed out
-11500	qaerr_LICFILENOTFOUND	Licence File not found
-11501	qaerr_INVALIDLICENCEKEY	Invalid licence key
-11502	qaerr_LICENCECONFLICT	Conflicting licences found in licence file
-11510	qaerr_LICENCENOTFOUND	Licence not found
-11511	qaerr_LICENCEEXPIRED	Licence expired
-11512	qaerr_EVALUATIONEXPIRED	Evaluation licence expired
-11520	qaerr_BADINPUTPARAM	Invalid input parameter
-11521	qaerr_INVALIDDATE	Invalid date

General Errors: -1000 to -1001

qaerr_FATAL

This should not occur unless your application encounters an unexpected condition. The API will abort.

qaerr_NOMEMORY

You will encounter this if QAS Pro API begins a task and subsequently discovers that there isn't enough memory available to complete it. For example, memory may run out during a complicated search, or if insufficient system resources are available.

If your application receives this error, a possible course of action is to produce a message asking the user to close down one or more applications in an attempt to free enough memory for QAS Pro API.

File Errors: -1010 to -1021

qaerr_FILEVERSION

This occurs if the data files that comprise a dataset have different version numbers. Reinstall your data to ensure that you have the latest versions of all files.

qaerr_FILEEXPIRED

This occurs if a dataset has expired. You can check how many days are left before expiry with the function **QA_GetLicensingDetail** (see page 109). If this error occurs, contact Experian QAS for a data update.

Startup Errors: -1030 to -1074

qaerr_BADOPTION

This indicates that an invalid command line parameter has been used. Enter the correct parameter to continue.

qaerr_BADINIFILE

This means that QAS Pro API cannot open the configuration file. This happens when the API fails to find the configuration file in any of the designated search paths. The default configuration file is `qaworld.ini` unless you specify otherwise in your call to **QA_Open** (see page 131).

QAS Pro looks for the configuration file in the following manner:

If a full path or filename was specified in your call to **QA_Open**, then the API looks along this path for the configuration file. If it is unable to find the file, it returns the error `qaerr_BADINIFILE`.

Otherwise the API looks for the configuration file in the current directory and then, if necessary, it searches for it along the path. If it has still not found a configuration file, then it returns this error.

Failure to open the configuration file will prevent QAS Pro from initialising.

`qaerr_BADLOGFILE`

If you get this error, the API has been unable to create the log file that you specified. Check the `LogFile` setting in the configuration file to ensure that the path and filename are valid.

`qaerr_BADRESOURCE`

The API cannot locate the language resource files, or the language resource files are corrupt. Check that they are in the correct location and that they are the correct version for your application. A missing configuration file may also cause this error to be returned. If the INI file is available, ensure that it is in the correct location, and that the `Language` setting is a correct value for this version of the API.

`qaerr_BADDATADIR`

This appears when an invalid data directory has been specified. You will encounter this error if the keyword `InstalledData` includes a directory which does not exist or does not contain a dataset. If your application receives this error, ensure that all directory and file names for the configuration setting `InstalledData` and `DataMappings`, located in `qawserve.ini` (see "Dataset Installation Settings" on page 189) are correct.

`qaerr_NOLOCALE FILE`

This appears if you attempt to start QAS Pro without the localisation file `qalcl.dat` in the program directory. Locate this file on your computer, or copy the file across from the CD.

Pro UI API Errors: -3400 to -3412

`qaerr_INVALIDCOUNTRYINDEX`

This error occurs if the *vilIndex* parameter in a call to **QAProWV_UIGetCountry** (see page 154) is out of range. Ensure that the index number is valid, remembering that all indexes in the API are zero-based (i.e. the first country in a list should be retrieved by setting *vilIndex* to 0, the second by setting it to 1, etc.).

qaerr_INVALIDLAYOUTINDEX

This error occurs if the *vilIndex* parameter in a call to **QAProWV_UIGetLayout** (see page 157) is out of range. Ensure that the index number is valid, remembering that all indexes in the API are zero-based (i.e. the first layout in a list should be retrieved by setting *vilIndex* to 0, the second by setting it to 1, etc.).

qaerr_UNKNOWNLAYOUTNAME

The layout name specified in a call to **QAProWV_UIGetActiveLayout** (see page 153) is not available in the current configuration file. You can check which layouts are available with the function **QAProWV_UIGetLayout** (see page 157).

qaerr_CANTCHANGEDATABASE

The country identifier specified in a call to **QAProWV_UISetActiveCountry** (see page 169) is not valid, either because it does not exist or the relevant dataset is not available. Ensure that you have installed the database that you want, and that the country identifier is correct.

qaerr_UIAPIALREADYSTARTED

The API is already running. This error is returned from **QAProWV_UIStartup** (see page 173) if it has been called previously and not yet shut down.

qaerr_UIAPINOTSTARTED

The API has not been started. Call **QAProWV_UIStartup** (see page 173) to start the API before calling any other functions.

qaerr_UIBADCDFFORMAT

This error is returned from **QAProWV_UIStartup** (see page 173) if none of the datasets can be initialised properly, due to an invalid <country identifier.dts> file. Ensure that your datasets have been installed correctly, and that each file has the same date.

qaerr_UINOCOUNTRIES

No valid datasets have been installed. See "Dataset Installation Settings" on page 189 or use the Configuration Editor to install datasets.

qaerr_INVALIDLINEINDEX

This error is returned from **QAProWV_UILayoutLineElements** (see page 164) if the *vilIndex* parameter is out of range. Ensure that the index number is valid, remembering that all indexes in the API are zero-based (i.e. the first line in a layout should be retrieved by setting *vilIndex* to 0, the second by setting it to 1, etc.).

qaerr_NOSELECTEDLAYOUT

This error is returned from **QAProWV_UILayoutLineElements** (see page 164) if you have not selected a layout to use. You can select a layout with **QAProWV_UISetActiveLayout** (see page 170).

qaerr_NOLIVESERVER

Could not find a valid configuration (or active server in Client Server installation). Can be returned during **QAProWV_UIStartup** (see page 173) or immediately after any Client / Server breakdown.

qaerr_BADCONFIGFLAGS

A nonsensical combination of flags was specified (e.g. deny change to search engine, and not specify an engine explicitly) Can be returned during **QAProWV_UIStartup** (see page 173) or **QAProWV_UISetFlags** (see page 171).

qaerr_CANTCHANGEENGINE

An error possible when unable to change to a specified engine, or keep existing search engine when changing country. Can be returned during **QAProWV_UIStartup** (see page 173), **QAProWV_UISetFlags** (see page 171), or **QAProWV_UISetActiveCountry** (see page 169).

CDF Errors: -3701 to -3809

If you encounter an error starting with -37..., you should contact Technical Support for further assistance.

qaerr_FORMATSYNTAX

qaerr_TOOMANYADDRLINES

qaerr_NOFORMATSPEC

These errors all signify a problem with the address format specified in the configuration file. Check that the configuration `AddressLineN` setting (see "Output Address Format Settings" on page 194) contains a suitable format for returning addresses, and that the number in the `AddressLineCount` setting is the same as the number of lines defined in `AddressLineN`.

qaerr_INVALIDADDRESSLINE

This error means that you are trying to retrieve an address line which is out of range. This might occur if, for example, your address format contains five lines and you call **QA_GetFormattedLine** (see page 102) with the *viLine* parameter set to 5. The range for an address with five lines would be 0 to 4.

qaerr_FORMATOVERFLOW

qaerr_FORMATTRUNCATED

If you encounter either of these errors, the retrieved address does not fit into the address format you have specified in the configuration file. For example, the returned address might be seven lines long whereas you have configured a five-line format, or one line might have 45 characters and cannot fit onto a line configured as 40 characters wide.

If you get several addresses that are truncated in this manner, you should review the address format settings `AddressLineCount` and `AddressLineN` (see "Output Address Format Settings" on page 194) in the configuration file to ensure that they meet your needs.

`qaerr_UNKNOWNDPITEM`

This error means that you are trying to retrieve a DataPlus item which does not exist. Check the DataPlus lines in your `AddressLineN` setting (see "Output Address Format Settings" on page 194) are correct.

`qaerr_DPFAILURE`

This is a warning that one or more DataPlus items has not been returned. This usually occurs if there is no information relating to the address that you have selected, and does not affect the returned address or any other DataPlus details.

`qaerr_BADSEQUENCE`

This error is sent back when the programmer calls functions in the wrong order.

Utilities

The following utility is available with QAS Pro and can be used to check the quality of your data.

Data Checker

You can check the integrity of Experian QAS data files using **quchkn.exe**.

Quchkn.exe is called as follows:

Syntax	QUCHKN (filespec) Show all command line options. QUCHKN -log -?	
Example	QUCHKN H:\QAS\DATA*.*	
Description	Performs an integrity check on selected Experian QAS data files. Uses a CRC (Cyclic Redundancy Check) to verify that the contents of the data files are not corrupt. Can be used to check for problems with the data or file corruption. The buttons on the dialog once the application has been launched can be used to perform the following actions:	
	Add...	Add an Experian QAS data file to the list.
	Remove...	Remove the selected file from the list.
	Check File	Check the currently-selected file.
	Check All	Check all the files in the list.

