

DTS Edition Reference Manual

Nov 2013



Copyright © 1998-2012 Intech Solutions Pty Ltd 2012. All Rights Reserved. Subject to change without notice.

IQ Office is a registered trademark of Intech Solutions Pty Ltd. All other trademarks mentioned herein are the property of their respective owners.

Email info@intechiq.com
Web www.intechsolutions.com.au
Phone +61 2 8305 2100
Fax +61 2 8305 2199

Contents

IQ Office Components	1
Summary	1
SetupEx.....	1
IQ Address Reference File Utility	1
IQ Rapid Address.....	2
Rapid SDK.....	2
IQ Easy Post.....	2
Standard Grammars	2
PAF	2
IQ Standardiser	2
Standardiser SDK.....	3
Fix Address	3
Enhanced Grammars.....	3
Grammar SDK	3
IQ Matcher.....	3
IQ Matcher SDK	3
Data Grammars	3
Country Grammars.....	3
 IQ Office Architecture	 6
API Summary.....	6
Platform	8
Microsoft Windows	8
UNIX	8
Mainframe.....	8
 Standardising Overview	 10
Input and Output Data.....	10
Processes.....	10
Example Processes.....	10
Combined Processes	11

Grammar Files	11
Grammar File Categories	12
Projects	12
Software Components	14
Application Integration	14
Matcher Integration	14
Logging and Reporting	15
Input and Output Data Options	15
IQ Office Installation	18
Installation Overview	18
Installation Options	18
1 Custom Installation Components	18
2 Copy Licence Files	19
3 Register Servers	19
4 Advanced Register Server Options	20
5 Standardising engine	20
IQ Office Configuration	22
Overview	22
Home Directory	22
ARF Directory	22
PAF Directory	23
Open PAF configuration setting	23
Grammar Directory	23
Max Grammars	23
Auto-loading Grammars configuration setting	24
AMAS Certified Date configuration setting	24
IQ Easy Post registry entries	24
IQ Office Database Integration	26
Oracle Database Integration	26
listener.ora	26
tnsnames.ora	27
Defining the Oracle Library	27
Load the Oracle Package	28
Batch Address Validation - Sample	28
Microsoft SQL Database Integration	28

Creating an extended stored procedure	28
Sample Code	28
IQ Office Release History	30
Version 5.7 (November 2013)	30
Version 5.6 (November 2012)	31
Version 5.5.4 (August 2012).....	31
Version 5.5.3 (May 2012).....	32
Version 5.5.2 (February 2012).....	32
Version 5.5.1 (November 2011)	32
Version 5.4.12 (August 2011).....	33
Version 5.4.11 (May 2011).....	33
Version 5.4.10 (February 2011).....	34
Version 5.4.9 (November 2010)	35
Version 5.4.8 (August 2010).....	35
Version 5.4.7 (May 2010).....	35
Version 5.4.6 (March 2010).....	36
Version 5.4.5 (December 2009).....	37
Version 5.4.4 (August 2009).....	37
Version 5.4.3 (May 2009).....	37
Version 5.4.1 (February 2009).....	38
Version 5.4 (November 2008)	38
Version 5.3 (November 2007)	38
Version 5.2.....	39
Version 5.1	39
Version 5.....	39
Version 4.4.....	39
Version 4.2.....	39
Version 4.1	39
Version 4.....	39
Version 3.1	41
Version 2.9.....	41
IQ Address Reference File Utility	44
Overview	44
Using ARFUtil.....	44
Start Options	44

Menus.....	44
IQ Office Component Setup	48
Overview.....	48
Using Component Setup.....	48
Client / Server Setup.....	49
Advanced Registry Settings.....	52
Install the Licence Files.....	53
Install IQ Data Files.....	54
Uninstall IQ Data Files.....	55
Stop and Uninstall Services.....	55
IQ Easy Post	58
Overview.....	58
Workflow.....	59
Australian Address Fields.....	59
New Zealand Address Fields.....	60
Australian Reports.....	60
New Zealand SOA.....	61
Manual Fix Address.....	61
Using IQ Easy Post (Aus).....	61
Starting and Exiting.....	62
Input File tab.....	62
Options tab.....	64
Process tab.....	69
Manual Fix Address tab.....	70
Rapid Tab.....	72
Rapid Address search functions.....	72
Using IQ Easy Post (NZ).....	74
NZ Input File tab.....	75
NZ Options tab.....	75
NZ Process tab.....	76
NZ SOA Tab.....	78
Issued SOA.....	79
NZ Rapid tab.....	80
IQ Rapid Address	82
Overview.....	82

Using IQ Rapid Address	83
Starting and Exiting	83
Rapid Address Window	84
Finding addresses.....	89
Selecting an ARF	98
Passing address data between applications	99
General Options	99
Search Options.....	102
Advanced Options.....	106
Saving options.....	115
IQ Rapid Address SDK	118
Overview.....	118
About this manual	118
Components	118
RapidSvr	120
RapidSvr Installation	120
RapidSvr Registration.....	120
RapidSvr Errors	121
Methods	121
Properties	129
PAF Libraries.....	147
PAF Libraries Installation.....	147
Function summary	147
Function reference	149
PAF Error codes.....	179
Socket Client.....	187
Socket Server	187
Socket Server Installation	187
Calling the socket server.....	188
Function Summary.....	188
SOAP Server.....	190
SOAP Server Installation.....	190
Calling the SOAP Server	191
Function Reference	191
RapidJNI Java class	201
Function Reference	201

IQ Rapid Address.....	207
IQ Rapid Address Installation.....	207
IQRapidForm32	208
Using IQRapidForm32.....	209
IQRapidForm32 Reference.....	209
IQRapidForm	214
IQRapidForm Installation.....	214
Using IQRapidForm.....	215
IQRapidForm Summary.....	215
IQRapidForm Reference.....	215
Reference	221
Barcodes.....	221
Record Formats.....	222
Address Reference Files	228
Configuration Settings	238

IQ Office Grammar File Reference 244

Overview.....	244
About this manual.....	245
Installation of Grammar Files	245
Summary of Grammar Files.....	245
Geographical coordinate accuracy.....	247
Dependencies	247
Customising with DEFINES	248
Codes	249
System Grammars.....	253
_RapidFormat	253
Standard Grammars	253
AddressEnhancer	253
AddressField	254
ArfGeocode.....	254
CustInfoBarcode	255
Dpid3Pass	255
DpidDefault	256
DpidDefault3Pass	256
DpidGeocode	257
DpidToBarcode.....	259

PostcodeToSortPlans	259
IQ Office Fix Address	262
Overview	262
Using Fix Address.....	262
1 Files tab.....	263
2 Mapping tab	266
3 Prompt tab.....	267
4 Display tab	268
5 Go tab.....	268
Rapid Address search functions.....	270
IQ Standardiser	274
Overview	274
Workflow	276
Using IQ Standardiser	277
Starting and Exiting	277
Managing Projects	278
Standardiser screen	279
OLE DB Source Selection Screen	280
Input Text File Format Screen – Delimited	283
Input Text File Format Screen – Fixed-width.....	284
Output Text File Format screen	285
Standardising Specifications screen.....	286
Reporting	288
Report Options	289
Relative Path Names	291
IQ Standardiser SDK	294
Overview	294
About this manual	294
Components	294
Windows Installation	296
Stan Libraries Installation	296
StanRt Installation.....	296
StanConsole Installation	297
StanB Installation.....	297
StanRt.....	297

StanRt object.....	297
Manager object	301
Loading grammar files with StanRt.....	304
Stan Libraries	305
Function summary	305
Loading grammar files with the Stan Libraries	306
Function reference	307
Configuration Settings	321
StanConsole application.....	324
Interactive mode.....	324
Batch mode	324
Running an IQ Standardiser project.....	325
StanJNI Java Class.....	325
StanJNI reference.....	325
Socket Server	330
Socket Server Installation	330
Calling the socket server.....	331
Function Summary.....	331
SOAP Server.....	333
SOAP Server Installation.....	333
Calling the SOAP Server	334
StanB.....	339
StanB object.....	339
Stan Spec File	343

Enhanced Grammars 348

Summary	348
AccountNames.....	348
AliasNames	350
AusPhone	351
Company	351
DecodeBarcodeScanner	352
Email.....	352
FixLastLine.....	353
IndNames	353
IndNames3a	354
Organisation3	356

OrgOrIndividual.....	357
Phone	358
Phone2	358
RemoveAddress.....	359
Rule Grammars	360
Soundex	361
Data-specific Grammars	364
Summary	364
ABSGeocode	364
AdminGeocode	365
Country-specific Grammars	368
Summary	368
NzAddress.....	368
NZValidate.....	369
NZDefault	370
NzAddressTwoArfsCNAR	371
IQ Office Grammar File SDK	374
Overview.....	374
About this manual.....	375
A simple grammar file	375
Tips	376
Debugging with StanConsole	377
Grammar file structure.....	379
Description.....	379
Suggested divider.....	380
Output fields	380
Tables.....	382
Grammar definitions	385
Dynamic Linked Library (DLL) definitions	385
Sub Grammars	390
Initialise and Finalise commands.....	390
Pre and Post-parse commands	391
Variables	391
Global Options.....	394
General syntax rules.....	395

Parsing	399
Token definitions	399
Simple tokens types	399
Composed token type.....	405
Qualifiers of simple types	405
Qualifiers of TABLE types.....	408
Qualifiers of composed type	409
Tolerance methods	412
Output actions	418
Commands in token definition.....	420
How the Parser works	421
Commands	422
When are parsing stage commands performed?	422
Command syntax	423
Command summary	424
Operators.....	426
Functions.....	427
Conditional commands	437
DLL Functions	439
Sub Grammar Calls.....	439
Command examples.....	440
Reference	447
Limits	447
Error messages	449
Glossary	464
Index	470

IQ Office Components

Summary

IQ Office suite components depend on the IQ Office edition as summarised below.

Software Component	IQ Office Edition			
	Rapid Address	ARF	DTS	Enterprise
IQ Office Component Setup	Y	Y	Y	Y
IQ Address Reference File Utility	Y	Y	Y	y
IQ Rapid Address	Y	Y	Y	Y
Rapid SDK	Y	Y	Y	Y
IQ Easy Post	Y	Y	Y	Y
Standard Grammars	Y	Y	Y	Y
PAF	Y	Y	Y	Y
IQ Standardiser		Y	Y	Y
IQ Standardiser SDK		Y	Y	Y
Fix Address		Y	Y	Y
Enhanced Grammars			Y	Y
Grammar SDK			Y	Y
IQ Matcher				Y
IQ Matcher SDK				Y
Data Grammars *				
Country Grammars *				

Two levels of licence apply to each edition. The *Dev* licence provides all SDK components, whereas the *Pro* licence provides only applications.

* There are also additional data licences which limit the supplied files and component functions, for example, geocoding functions and country-specific data. Contact Intech support for details of your supplied edition and licences.

SetupEx

IQ Office Component Setup is a simple utility for configuring, installing and removing IQ Office components after software installation.

IQ Address Reference File Utility

IQ Address Reference File Utility is a command-line utility used to test, build and extract (with appropriate license files) Address Reference Files (ARF) that can be used as the source for address validation and IQ Rapid Address entry.

IQ Rapid Address

IQ Rapid Address is a high performance address capture application that allows the user to quickly and accurately capture, validate, standardise and barcode addresses at the point of data entry, with a significantly reduced number of keystrokes. By accessing the Australian or New Zealand Postal Address Files (PAF) containing all deliverable addresses, or alternate ARF, IQ Rapid Address enhances the integrity of customer data by returning a customer address that is both valid in content (accuracy) and format (standard), whilst reducing data entry costs.

Rapid SDK

The IQ Rapid SDK is an extensive set of Information Quality Application Programming Interfaces and test code which enables rapid and efficient address file searching functions to be integrated into new and existing business systems.

IQ Easy Post

IQ Easy Post is an extremely simple and easy to use software package that provides everything needed to process address lists for barcoded bulk mail lodgement. IQ Easy Post is ideal for small to medium enterprises that lodge their own post directly with Australia or New Zealand Post, as well as mail houses, list brokers and data processing bureaus that lodge post on behalf of their customers.

Standard Grammars

Standard Grammars standardise and validate Australian addresses, including BSP, Barcodes and DPID look up.

PAF

The Australia Post Postal Address File is updated quarterly by AMAS-approved vendors and is used for address validation and barcode assignment for bulk mail discounting. It is supplied with IQ Office in sorted and compressed format for rapid access.

IQ Standardiser

IQ Standardiser is a high performance data transformation engine designed to standardise, enhance, validate and transform "free format" data into a structure that conforms to organisational standards, hence realising the "hidden" value. Developed with maximum flexibility in mind, IQ Standardiser offers a premium level of functionality for enterprise-wide Data Quality.

IQ Standardiser will also validate, enhance and correct address data by matching it to an Address reference file (ARF) such as the data in Australia Post's Postal Address File (PAF). Approved under Australia Post's Address Matching Approval System (AMAS), IQ Standardiser performs DPID and barcode allocation, as well as pre-sorting and generation of manifest reports, enabling access to barcode PreSort mailing discounts.

Integrating IQ Standardiser into existing and new databases increases the accuracy, consistency and referential integrity of customer information. This facilitates strong customer relationships through increased reporting capabilities, improved direct marketing campaigns, and the ability to make better overall strategic decisions. With well-structured and standardised data in place, an organisation can effectively and accurately consolidate customer records, from multiple sources across the organisation, with real time capabilities.

Standardiser SDK

The IQ Standardiser SDK is an extensive set of Information Quality Application Programming Interfaces and test code which enables standardising functions to be integrated into new and existing business systems.

Fix Address

Fix Address is a simple companion to IQ Standardiser which prompts. It will present you with the addresses that weren't automatically allocated a DPID/Barcode, allowing you to manually search for those addresses and allocate the DPID/Barcode.

Enhanced Grammars

Enhanced Grammars standardise names, phone numbers, email addresses and organisations, and generate phonetic algorithm codes for words.

Grammar SDK

The Grammar SDK Reference Manual contains detailed instructions on how to create, edit and run existing and customised grammar files.

IQ Matcher

IQ Matcher is a high performance record-matching engine that standardises, matches, cleanses and integrates customer, marketing and organisational data from multiple sources and/or detects and corrects duplicate information within a data source. Designed specifically for large-scale databases, IQ Matcher enables the integration of enterprise-wide data, whilst also providing the means to quickly and accurately integrate data in real time across many data sources.

With IQ Matcher, an organisation is provided with the ability to quickly and accurately match large volumes of records, even where the type of record fields vary or contain disparate information (such as misspellings, address variations and other inconsistencies). To find matches, IQ Matcher applies an advanced statistical algorithm to determine the similarity between records and assigns a match confidence level, which can be adjusted to meet business requirements.

By seamlessly linking together customer information, IQ Matcher empowers businesses with Quality Information to deliver a consistent and complete view of the customer.

IQ Matcher SDK

The IQ Matcher SDK is an extensive set of Information Quality Application Programming Interfaces and test code which enables matching functions to be integrated into new and existing business systems.

Data Grammars

Data grammars are specific to the licensed and installed data package, and standardise Australian addresses, provide brief or detailed Australian Bureau of Statistics (ABS) statistical boundary information, electoral administrative boundaries and geocodes.

Country Grammars

Country grammars are specific to a country, such as New Zealand, and standardise and validate New Zealand addresses.

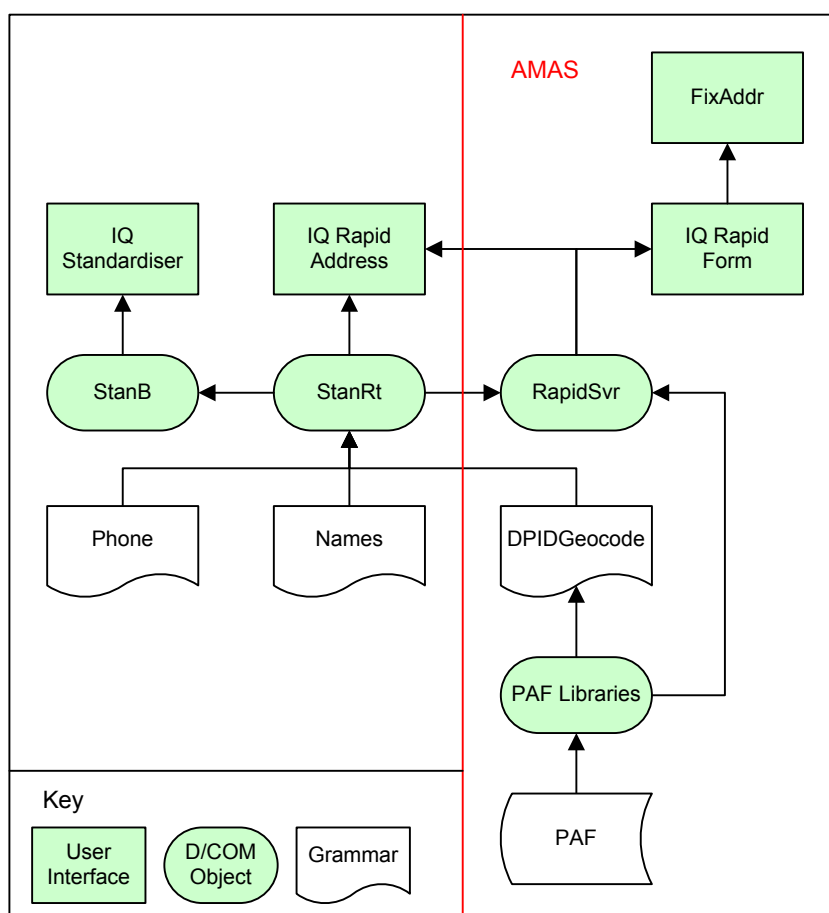
IQ Office Architecture

This document is an introduction to the IQ Office Package, IQ Office SDK and IQ Office APIs.

IQ Office is available on multiple platforms including [Windows](#), [UNIX](#), and [Mainframe](#). Some interfaces listed in the documentation are platform-dependent, however all platforms can make use of the same core IQ Office functions.

The diagram below summarises IQ Office architecture. Arrows show a dependency, with the arrowhead pointing to a dependent component. Components specific to address matching – AMAS (Address Matching Approval System) – are shown on the right of the red line.

Note that the PAF Libraries can be used as an alternative to RapidSvr, and the Stan Libraries can be used as an alternative to StanRt (but not as a client of StanB).



API Summary

Address Matching

Information on the following modules can be found in the [IQ Rapid Address SDK Manual](#):

- | | |
|------------------|--|
| PAF | (Postal Address File) is a file provided by Australia Post containing all deliverable addresses in Australia with their DPID. It is supplied here in a sorted and compressed form. It is used by the DpidGeocode grammar-file and by IQ Rapid Address . |
| RapidSvr | A DCOM / COM server providing functions to search the PAF files. It can run as an NT Service. |
| IQ Socket Server | Address searching functions provided by the PAF libraries are also available via a sockets server |

IQ Soap Server	The address searching functions provided by the PAF Libraries are also available via a SOAP (Simple Object Access Protocol) / XML (Extensible Markup Language) Server.
PAF Libraries	Library functions providing functions to search and access the PAF files. It is available as a Windows API DLL (iPaf32.dll). It is also available as C-function library for several other operation systems. It also contains functions to generate barcode code strings.
IQ Rapid Address	An application that allows entry of addresses using minimal keystrokes, validates the address and appends its. It uses the RapisSvr COM library or the iPaf32.dll library.
IQ RapidForm	A COM object that provides the same screen as the IQ Rapid Address application.
IQ RapidForm32	A library, containing Windows API function calls, providing the same screen as the IQ Rapid Address application.

Standardising

Information on the following modules can be found in the [IQ Standardiser SDK Manual](#):

StanRt COM	(Standardiser Real Time) the standardising engine. It will standardise a single piece of data (a field, a string) at a time. Available as a COM / DCOM object that can run as an NT service.
Stan Libraries	The standardising engine. It will standardise a single piece of data (a field, a string) at a time, in real-time; Available as a Windows API DLL and as C-function library for several other operation systems.
StanB	(Stan Batch) a COM object library that uses StanRt to standardise a whole table of records.
IQ Socket Server	The Standardising functions provided by the Stan libraries are also available via a sockets server
IQ Soap Server	The Standardising functions provided by the Stan libraries are also available via a SOAP (Simple Object Access Protocol) / XML (Extensible Markup Language) Server.
StanConsole	a console application that can standardise data stored in text-files. It can also be used in interactive mode to test the standardising engine.
Grammar Files	these are the rules for standardising that are used by StanRt and StanConsole . Each grammar file will standardise different type of data (eg. addresses, names, and phone numbers). The DpidGeocode grammar-file is used to look up an address against the PAF , validate it and append a DPID. These grammar-files can be customised for any particular need as described in the Grammar SDK Reference Manual .

Grammar Files

Grammar file contain the instructions on how the IQ Standardiser parsing engine is to standardise data. The IQ Grammar SDK enables developers to create custom data transformations, and alter existing data transformations to meet your specific requirements. The [Grammar SDK Reference Manual](#) describes how to use the IQ Grammar SDK and write grammar files. The [Grammar File Reference Manual](#) describes the grammar files supplied with the different IQ Office editions.

Probabilistic Matching

Information on the following modules can be found in the IQ Matcher SDK Manual:

IQMatch COM	Provides a programmatic interface to both batch and real time matching functionality. The IQ Matcher windows GUI application uses this object to perform the matching.
-------------	--

IQMatch Libraries	A c library exposing the low level matching engine. Provides real time and cross platform access to the matching engine. A client application passes records to this library for probabilistic comparison.
IQ Socket Server	The Matching functions provided by the IQMatcher C libraries are also available via a sockets server
MatchConsole	A cross platform console application that executes a preconfigured matching process.

Platform

Microsoft Windows

We recommend the Windows Server family architecture for stability and control, however IQ Office can be run on any Windows platform from Win98 upwards. Also note that Windows IIS is not required for the IQ Soap Server as it runs as a standalone service on a dedicated TCP port.

The Windows deployment provides the utility “Component Setup” with the Intech start menu items. This utility allows for easy client/server configuration with the ability to restart the Intech services.

UNIX

The IQ Office software package requires the use of a windows based system for the MSI installation. This installation process is purely used as a means to extract the required UNIX files. The IQ Office files are provided in both RAW and Zip archive format. The use of a Windows system is only required for the extraction of the binary files. Data updates that are provided do not require a full re-installation or access to a Windows based machine.

UNIX comes in many flavours and we aim to offer binary compatibility with all mainstream versions including AIX, HP-UX, RedHat Linux (for Intel) and Solaris.

Mainframe

The IQ Office APIs are available through both MVS Batch and MVS CICS environments. These use natively compiled C Libraries to access the IQ Office functions. Sample COBOL code is provided that demonstrates an address validation and rapid address process. A JCL is provided that will standardise text file input and produce a standardised output file. For more information regarding a mainframe deployment, please review the IQ Mainframe Reference Manual.

Standardising Overview

Standardising is the process of converting unformatted text data such as addresses, names and telephone numbers into data in a *standard format* performed by IQ Office Standardiser, a suite of applications and library functions supplied with IQ Office DTS and Enterprise editions. This typically cleanses, transforms, enhances and validates the input data, providing greatly improved and high quality output data.

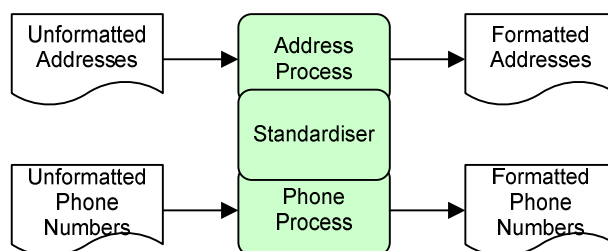
Input and Output Data

The *input data* to be standardised might have different formats, word orders, and contain additional characters to be removed. The *output data* is processed to convert the input data into a defined and consistent format.



Processes

Standardising is performed by one or more standardising *processes*. Each process takes one type of alphanumeric input data such as addresses, names or telephone numbers, *parses* the input data to break it into components, and returns each required component in a specified format.



Example Processes

The input to an *address process* could contain a mixture of upper and lower case words, punctuation, and line breaks, or simply contain one combined address line, for example:

```
7 35 Spring St
Bondi Junction 2022
```

```
Unit 2, 1 Albert Road MOONAH,Tasmania    7009
```

The address standardising process parses the input to recognise each component of the address, removes unwanted characters, replaces different spellings and abbreviations with standard lists, changes the order of words, and outputs a standard format of the combined address components, in this case a unit and street number separated by a comma on the first line, the suburb in title case on the second line, and the state in upper case and postcode on the third line.

```
Level 7, 35 Spring St
Bondi Junction
NSW 2022
```

```
Unit 2, 1 Albert Rd
Moonah
TAS 7009
```

The input to a *phone process* could contain telephone numbers with additional text, state abbreviations and international prefixes, for example:

```
83052100
```

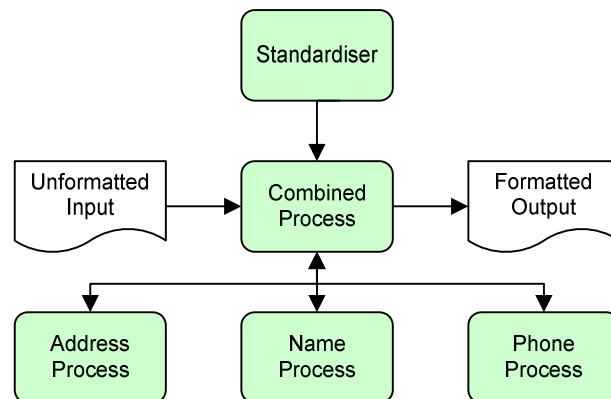
+ (612) 83052199
Tas 6230 5605
Freecall 13 12 13

The process removes all additional text, looks up the state, and returns the area code in the output.

(02) 8305 2100
(02) 8305 2199
(03) 6230 5605
13 12 13

Combined Processes

Processes can be combined, for example a combined “Name, Address and Phone” process might break input data into separate name, address and phone number components, pass each type of data to a corresponding process, and compile the returned outputs.



The input and output of such a combined process might be as shown in the example below.

Tim Smith, 8305 2100
7/35 Spring St Bondi Junction
2022

Attn: Mr T Smith
Tel: 8305 2100
Level 7
35 Spring St
Bondi Junction
NSW 2022

Grammar Files

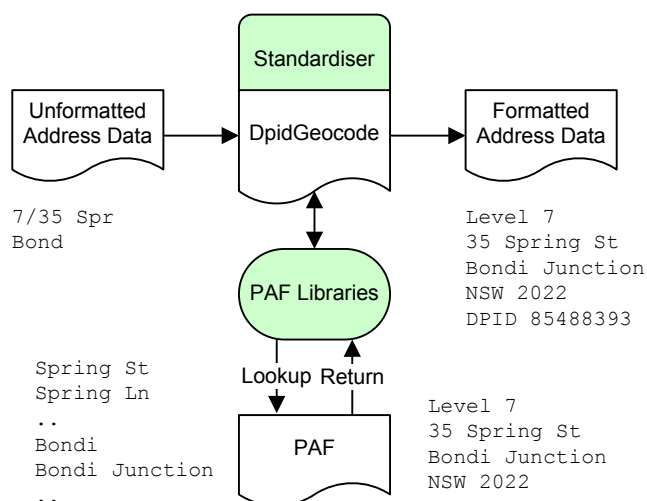
Each standardising process has a *grammar file*, which defines how the data will be standardised and formatted. The Standardiser reads and interprets the grammar file before performing the defined standardising processes.

Grammar files are text files written in a simple programming language. They contain definitions of the number, type and format of input data fields; instructions on how Standardiser is to process input data; references to lookup-lists such as standard abbreviations; and definitions of the number, type and format of output data fields.

Existing grammar files can be easily edited to *customise* them to the specific requirements of an organisation, for example, to process address data received in a variety of formats, and output data in the specific format used by a particular database or automated mailing system. New files can also be written to fulfil the needs of new business processes, for example, if an organisation installs a new account management system.

Grammar files can contain *calls* to other grammar files, for example, the “Name, Address and Phone” grammar file would call three separate grammar files and combine the returned output.

Grammar files can also contain calls to other processes, data sources, libraries and *address reference files*. For example, the *DpidGeocode* grammar file supplied with IQ Office Standardiser uses the IQ Office PAF Libraries to look up an address in the Postal Address File (PAF) containing all Australian postal addresses supplied by Australia Post, and returns the formatted output and DPID.



Grammar File Categories

IQ Office can supply the following categories of grammar file, depending on the software edition and data licence. All except System files can be customised to specific customer requirements.

System	Internal system files providing common functions to other grammar files. These files must not be edited or deleted.
Standard	Supplied with all IQ Office Editions. Standardise and validate Australian addresses, including BSP, Barcodes and DPID look up.
Enhanced	Only supplied with IQ Office DTS and Enterprise Editions. Standardise names, phone numbers, email addresses and organisations, and generate phonetic algorithm codes for words.
Data	Specific to the licensed and installed data package. Standardise Australian addresses, and provide brief or detailed Australian Bureau of Statistics (ABS) statistical boundary information, electoral administrative boundaries and geocodes.
Country	Specific to a country, such as New Zealand. Standardise and validate New Zealand addresses.

Projects

Standardising processes are specified in a *project* which defines the processes performed, and input and output data and fields. A project can be created, edited, saved, and opened again using IQ Standardiser, or run in Batch mode by IQ Stan Console with no user or screen interaction.

A project is defined in two text files.

The *specification* or *spec file* (extension `.sta`) lists the input fields passed to each process, and the output fields of each process, for example, `AddressProcessing.sta`

```

PROCESS

PROCESS Address
OUT 0 3 state

```



```

OUT  1  12  postCode
OUT  2  46  locality
OUT  3  30  streetName
OUT  4   4   streetType
OUT  5   5   houseNo
OUT  6   5   levelNo

```

```

IN   0  90  Address_1
IN   0  50  Suburb
IN   0   3   State
IN   0   4  Postcode

```

Spec files can also specify fields which *pass through* the process intact, for example, a record number.

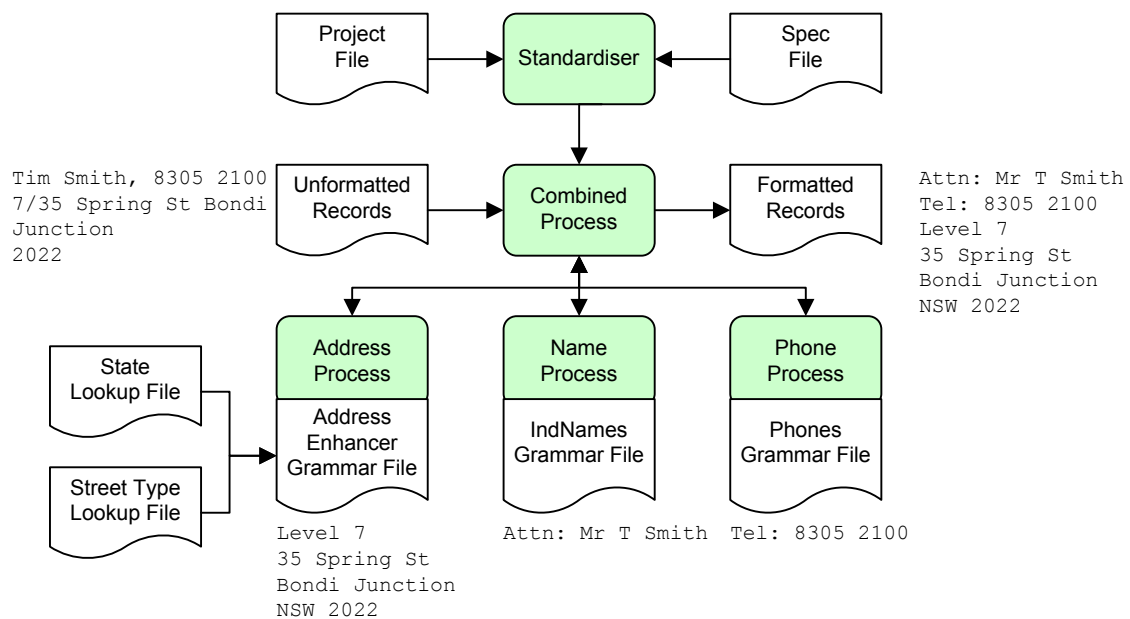
The *project file* (extension `.ist`) lists the input and output files, input and output fields of each process, and other parameters, for example, `AddressProcessing.ist`

```

InputTextFile=C:\Data\RawAddresses.txt
InputFields=Address_1,Suburb,State,Postcode
InputSizes=90,50,3,4
OutputTextFile=C:\Data\ProcessedAddresses.txt
OutputTextDelimiter=,

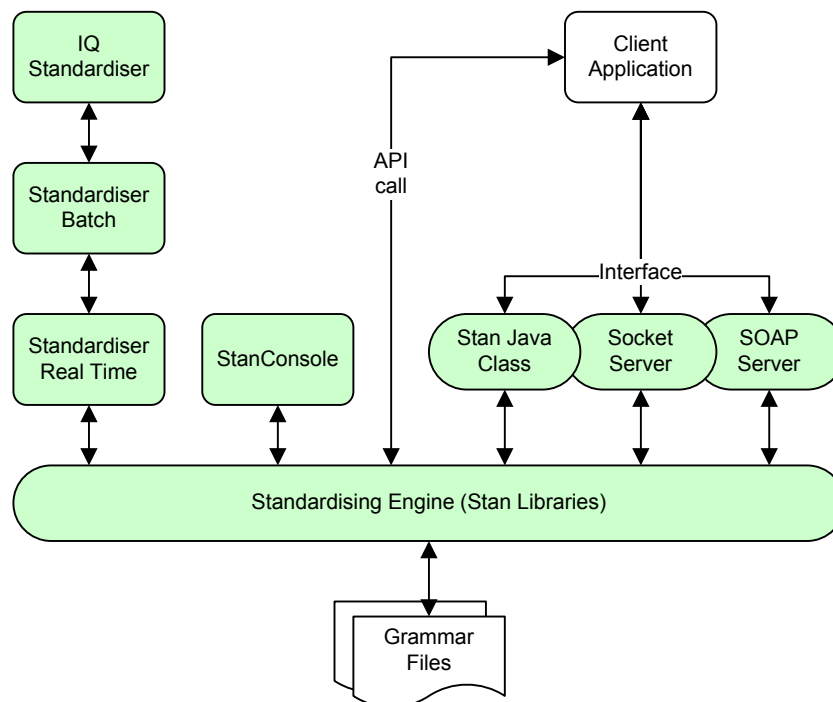
```

The diagram below shows all components of an example Standardiser project.



Software Components

IQ Office Standardiser has several components which can perform standardising functions in real time and batch mode, either as standalone applications, or functions integrated into other systems.



IQ Standardiser	Standalone Windows application for managing and running Standardiser projects.
Standardiser Batch	A COM object library that uses Standardiser Real Time to standardise a batch of records.
Standardiser Real Time	A COM / DCOM object implementation of the standardising engine which can run as a Windows service.
StanConsole	A console application that can standardise data stored in text files, be used in interactive mode to test the standardising engine, and run projects.
Socket Server	A socket server interface to the standardising engine.
SOAP Server	A SOAP / XML server interface to the standardising engine.
Stan Java Class	A Java interface to the standardising engine.

Application Integration

All components use the underlying Standardising Engine (Stan Libraries), which standardise data items in real-time. These are available as a Windows API DLL or Unix-family C function library, which can be called either directly by a client application, or via a SOAP, Socket or Java interface to provide all the functions provided by the Standardising Engine.

For example, a company with an address management database with Windows front end could call API functions to standardise address records as an integral part of the system.

Matcher Integration

Standardising functions can be integrated into the advanced probabilistic matching engine of IQ Office Matcher supplied with IQ Office Enterprise edition, to enable unstructured input data to be cleaned and broken into components suitable for matching.



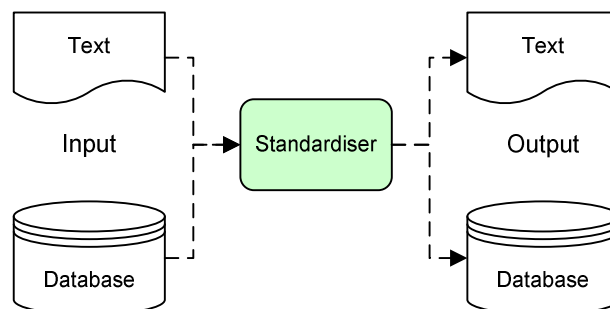
Logging and Reporting

Errors occurring during standardising can be recorded in a *log file* (extension `.log`).

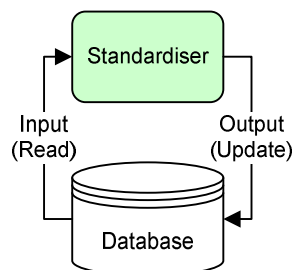
Standardiser can also create *reports* based on the standardised output. Currently the only available reports are for Australian and New Zealand bulk mail lodgements. Output for these reports must be created using a standardising process which assigns an AddressID.

Input and Output Data Options

Standardiser input and output data can be a combination of either *text files* in fixed-width or delimited format, or different types of *database* supporting OLE-DB connections.



Data can be input from one type of data source into another, for example, from a text file to a database or vice-versa. In the case of databases, data inputs and outputs can be a table, view, query, stored procedure or SQL statement. Data can be input from one database and output to another, or output to a different table in the source database either by creating a new table, or by updating the same table from which the data was read, via a direct table write or stored procedure.

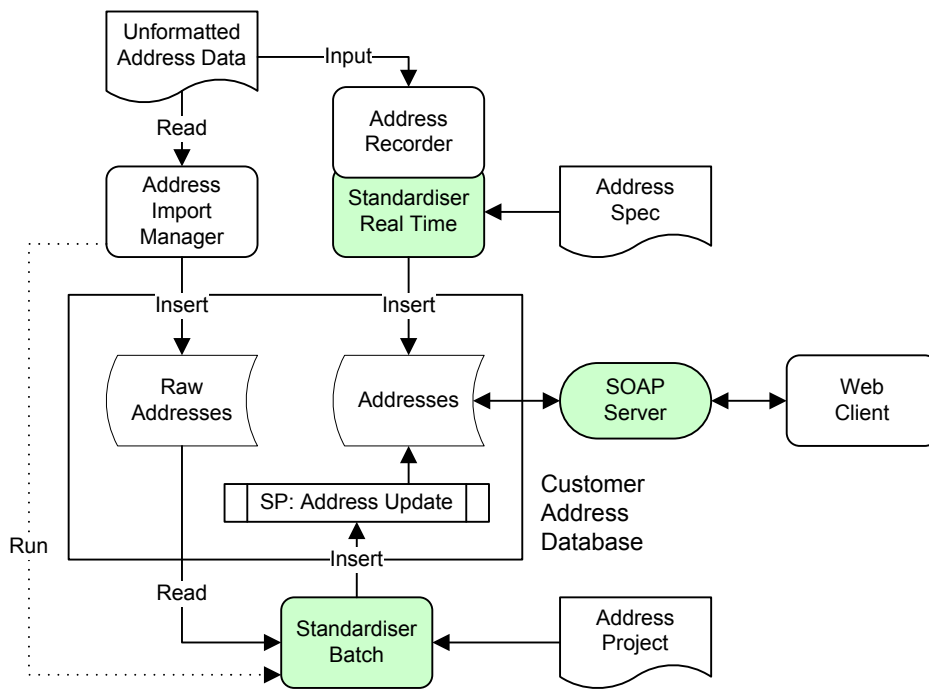


These flexible options, combined with batch mode operation, provide a number of powerful methods to standardise and update data illustrated in the example below.

An Address Import Manager such as a batch file or cron job, periodically reads unformatted address data from a text file and inserts it into a Raw Addresses table in a Customer Address database. The same manager periodically calls Standardiser to run an Address project in batch mode to read and standardise the unformatted address data, and update a formatted Addresses table via a stored procedure.

Unformatted address data is also entered using an Address Recorder application used by a call centre to input new customer details. The application calls integrated Standardiser Real Time library functions to insert formatted addresses directly into the Addresses table, using the format defined in the Address Spec.

At the same time customers can also view their records by typing a free format address string into a web client, which interfaces to the database via a SOAP server.



IQ Office Installation

Installation Overview

Use the following procedure to install IQ Office:

1. Obtain a valid installation key from Intech Solutions for the particular product version and edition that will be installed.
2. Uninstall previous version of IQ Office:
 3. Back up any customer modifications that you may have made, such as to grammar files
 4. Back up current license files (*.lic)
 5. Start -> Settings -> Control Panel -> Add/Remove Programs
6. Run the setup program. If Autoplay is enabled on your DVD drive, it will launch the installation menu when the DVD is inserted. Alternatively double click either the autorun.exe or the following file:

```
/install/setup IQ Office v5 - Windows.exe
```

Installation Options

When installing, you will be given the choice of one of three setup types:

- Typical (recommended for Stand Alone) – if the software will be running entirely on this computer.
- Client Only – if the software will be running in a Client/Server architecture, with this computer acting as the client.
- Custom (recommended for Server) – if the software will be running in a Client/Server architecture, with this computer acting as the server, or to have more control over which components are installed.

1 Custom Installation Components

If you choose a "Custom" installation, you will be presented with a list of the following components to install.

Client Components

The components needed for the client machine in a client/server architecture.

- IQ Standardiser Client – the Graphical User Interface for IQ Standardiser
- IQ Rapid Client – the Graphical User Interface for IQ Rapid Address
- Fix Address – the Graphical User Interface for Fix Address
- IQ Easy Post – the Graphical User Interface for Easy Post

Server Components

The components needed for the server machine in a client/server architecture.

- IQ Standardiser Server – StanRt.exe – the StanRt DCOM object that can run as an Out-of-Process Server or an Windows Service.
- IQ Rapid Server – RapidSvr.exe – the Rapid DCOM object that can run as an Out-of-Process Server or an Windows Service.
- AU Grammars – these files are softcoded logic use by the standardiser for address validation.
- East Post Grammars – grammar files required for the Easy Post process.
- Additional Grammars – additional grammar files

- SOAP Server Files required to provide the IQ Standardiser and IQ Rapid Address as an IIS integrated SOAP server.
- TCP/IP Socket server – interface to IQ Rapid Address and IQ Standardiser functions.
- IQArfUtil – used for building and testing ARF's.

SDK

The Software Development Kit (SDK) is documentation on how to use Intech's extensive API (Application Programming Interface). It also contains sample code (in Visual Basic, Delphi, C and other languages) and tools.

IQ Matcher

IQ Matcher is a high performance record-matching engine that standardises, matches, cleanses and integrates customer, marketing and organisational data from multiple sources and/or detects and corrects duplicate information within a data source.

2 Copy Licence Files

At the end of the setup process, you will be prompted to copy licence files. If you have received licence files from Intech, place them in the floppy disk drive (or other location), click "Continue" and browse for them. If you don't have licence files, click "Skip".

3 Register Servers

Server registration depends on the type of installation:

- Typical – not applicable
- Client Only – you will be prompted for the name of the server machine. If you don't yet know the name of the server machine, you can set it later using the Component Setup utility (`Config\SetupEx.exe`)
- Custom – you will be given the option of how you want the COM objects to run. You may change your choice later by running the Component Setup utility. Depending on the installation chosen you may not have the server options.

Server

- IQ Socket Server - a TCP/IP socket server that will handle requests to the IQ Office APIs.
- IQ Soap Server - a standalone Web service that provides IQ Office functionality via Soap protocol.
- IQ Rapid Address DCOM Server - the DCOM object will run and be configured to autostart with the system.
- IQ Standardiser DCOM Server - the DCOM object will run and be configured to autostart with the system.

Client

- Remote Socket Server - a minimal client installation can use sockets to access the IQ Rapid and IQ Standardiser functions. This minimises the update requirements on client systems. This remote computer must be running the Socket Server on the port specified.
- Remote DCOM Server - an application requiring the COM object will look for it on the remote computer specified. This remote computer must be setup with the Intech software acting as a DCOM service.
- standalone COM (EXE) - the COM object will run in a separate process. It will only run as long as there are applications using the COM object.
- standalone COM (DLL) - the COM object will run in the same process as the application using it.

- standalone without COM - this will make the COM object unavailable. Some applications won't work if this option is chosen.

4 Advanced Register Server Options

If you have chosen a "Custom" setup, then you will be able to click on "Advanced..." from the "Register Servers" screen. This will prompt you with the following options. You may click "Cancel" to leave the options as they are.

5 Standardising engine

When the standardising engine starts, it may preload some grammar files, so that when they are needed, there will be no time delay in providing them. You may select which grammar files are to be preloaded. This option usually only useful if the standardising engine is running as a separate process (an Windows Service or an Out-of-process server COM (EXE)).

The maximum number of grammar files that can be loaded at the same time may be set here. Note that some grammar files such as Dpid3Pass load other grammar files, which increases the number of loaded grammar files.

IQ Office Configuration

Overview

The IQ Office suite uses various configuration settings which are either stored in the Windows Registry or a Unix configuration file. This manual describes global settings, other settings are described in individual IQ Office suite component manuals.

The [IQ Office Component Setup Manual](#) describes how to edit configuration settings.

Windows

Windows Registry settings are shown in Registry File (*.reg) format, e.g.

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech]
"HomeDir"="C:\\Program Files\\Intech"
```

This example shows that under the key

```
HKEY_LOCAL_MACHINE\SOFTWARE\Intech
```

there is a string value named `HomeDir` with value

```
C:\Program Files\Intech
```

The `REGEDIT4` label may be omitted for ease of reading.

UNIX / LINUX

UNIX / LINUX settings are stored in a configuration text file which is searched for in the following order:

1. A file in the current directory named `.intech`
2. A file specified by the environment variable `INTECH_CONFIG_FILE`
3. A file in the HOME directory named `.intech`
4. The file `/usr/etc/.intech`

This file contains one key/value pair per line in the form `key/subkey=value`, e.g.

```
HomeDir=/home/bob/intech
RapidSvr/Properties/TimeLimit=4
```

Home Directory

This setting specifies the Intech home directory, in which licence file(s) must be located. The actual directory may be different to those in the examples below.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech]
"HomeDir"="C:\\Program Files\\Intech"
```

Configuration Text File example

```
HomeDir=/home/intech
```

ARF Directory

This setting specifies the default location of the compressed Address Reference Files (ARF). The actual directory may be different to those in the examples below.

Window Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\ARF]
```

```
"Paf2005_1"="C:\\Program Files\\Intech\\Paf\\Paf2005_1"
```

Configuration Text File example

```
ARF/Paf2005_1=/home/intech/paf/paf2005_1
```

PAF Directory

This setting specifies the default location of the compressed Address Reference Files (ARF/PAF). The actual directory may be different to those in the examples below.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\\SOFTWARE\\Intech]  
"PafDir"="C:\\Program Files\\Intech\\Paf"
```

Configuration Text File example

```
PafDir=/home/intech/paf
```

Open PAF configuration setting

This setting specifies how the PAF Libraries load the PAF files into memory. It can have one of the following three values, or Disk if none is specified.

Disk	(Default) The files are read from the disk and not loaded into memory. This is the simplest option.
Mem	The files are pre-loaded into memory when opened. This option should only be used if there is enough free RAM to contain all the files, but will provide greater performance when batch processing addresses. Pre-loading into memory will cause a small delay while opening the files.
Map	The files are accessed via the File Mapping method, in which the operating system loads them into memory when required, and unloads them when memory runs low.

Note that this setting does not affect RapidSvr, which always opens the PAF files with the “Disk” option, unless otherwise specified via the ChangeARF method.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\\SOFTWARE\\Intech]  
"OpenPaf"="Map"
```

Configuration Text File example

```
OpenPaf=Map
```

Grammar Directory

This setting specifies the location of the grammar files. The actual directory may be different to those in the examples below.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\\SOFTWARE\\Intech]  
"GrammarPath"="C:\\Program Files\\Intech\\Grammars"
```

Configuration Text File example

```
GrammarPath=/home/intech/grammars
```

Max Grammars

This setting specifies the maximum number of grammars that can be simultaneously loaded by the Standardising Engine. The example below sets this to 20, which is the default if this setting is missing.

Window Registry example

```
[HKEY_LOCAL_MACHINE\\SOFTWARE\\Intech\\StanRt]  
"MaxGrammars"="20"
```

Configuration Text File example

```
StanRt/MaxGrammars=20
```

Auto-loading Grammars configuration setting

This setting specifies which if any grammar files are to be loaded when StanRt or the Stan Libraries are started. These will remain loaded, as if the KeepLoaded method or KeepLoaded function was called.

When running StanRt Com as an NT Service, opening of grammar files will be logged in the Windows Event Log.

The example below specifies that the `DpidGeocode.grm` and `AusPhone.grm` grammar files are loaded. The names of the entries (“1” & “2” in this example) are ignored, and anything can be used.

Window Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\StanRt\Autoload]
"1"="DpidGeocode.grm"
"2"="AusPhone.grm"
```

Configuration Text File example

```
StanRt/Autoload/1=DpidGeocode.grm
StanRt/Autoload/2=AusPhone.grm
```

AMAS Certified Date configuration setting

This setting specifies the date Australia Post certified the software according to its AMAS program, which is usually renewed annually.

This information is used in creating Address Summary Matching Reports. The actual certified date may be different to that in the examples below.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech]
"AmasCertifiedDate"="Nov 2004"
```

Configuration Text File example

```
AmasCertifiedDate=Nov 2004
```

IQ Easy Post registry entries

This setting specifies the grammar files used by IQ Easy Post for the “Strict AMAS Compliant” and “Pre process/clean input record before AMAS match” options.

This registry entry is presented here in “Registration Files (.reg)” format.

```
REGEDIT4

[HKEY_CURRENT_USER\Software\VB and VBA Program Settings\IQ Easy
Post\GrammarFiles]
"AMAS"="DpidDefault.grm"
"Enhanced"="Dpid3PassDefault.grm"
```


IQ Office Database Integration

Oracle Database Integration

Oracle v8 onwards provides an external procedure interface (extproc) that can be used to connect to externally compiled libraries. Using standard PL/SQL packages the IQ Office libraries can be called from any Oracle application that allows PL/SQL.

The first step in using IQ Office from within Oracle is to configure the Oracle Listener enabling the external procedure interface. This requires modifying the following two Oracle configuration files:

```
listener.ora
tnsnames.ora
```

The following is a brief outline of the changes required to combine the external procedure interface into your current listener service. For alternative configurations please consult your Oracle DBA and refer to the Oracle documentation.

listener.ora

Add the following line to the listener address list

```
(ADDRESS = (PROTOCOL=IPC) (KEY=EXTPROC))
```

Add the following block to the SID_LIST:

```
(SID_DESC=
(SID_NAME=PLSExtProc)
(ORACLE_HOME=C:\oracle\ora92)
(PROGRAM=extproc)
#Use the following for windows
#(ENV= "EXTPROC_DLLS=C:\program files\intech\sdk\oracle\ipaf32cdecl.dll:
C:\program files\intech\sdk\oracle\stan32cdecl.dll")
#Use the following for UNIX
#(ENV= "EXTPROC_DLLS=/Intech/IQ_Office/lib/libiqpaf.so:/Intech/IQ_Office/li
b/libiqstan.so,INTECH_CONFIG_FILE=/Intech/IQ_Office/.intech")
(ENV=EXTPROC_DLLS=ANY)
)
```

Note that a few of the lines have been commented out. Make sure that the path to the IQ Office libraries has been appropriately set for the correct installation directory. It is also important that under UNIX based platforms that the `INTECH_CONFIG_FILE` is either set within the `ENV` parameter or configured through the oracle users `SHELL`.

Assuming that the Oracle system is running on Windows server called `viper.intechsolutions.com.au` and that you are not going to restrict the external procedure to using only named DLLs you could use the following `listener.ora` file contents.

```
LISTENER =
(DESCRIPTION_LIST =
(DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL=IPC) (KEY=EXTPROC))
(ADDRESS = (PROTOCOL = TCP) (HOST =
viper.intechsolutions.com.au) (PORT = 1521))
)
)
)
```

```

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = viper.intechsolutions.com.au)
      (ORACLE_HOME = C:\oracle\ora92)
      (SID_NAME = ORCL)
    )
    (SID_DESC=
      (SID_NAME=PLSExtProc)
      (ORACLE_HOME=C:\oracle\ora92)
      (PROGRAM=extproc)
      (ENVS=EXTPROC_DLLS=ANY)
    )
  )
)

```

tnsnames.ora

Add the following block to the file:

```

extproc_connection_data =
  (DESCRIPTION=
    (ADDRESS_LIST =
      (ADDRESS= (PROTOCOL=IPC)
      (KEY=EXTPROC) )
    )
    (CONNECT_DATA=
      (SID=PLSExtProc)
      (PRESENTATION = RO)
    )
  )
)

```

Note that this block has the KEY and SID parameters that match what was configured in the listener.ora configuration.

Once changes have been made the listener.ora and tnsname.ora files the Oracle listener services should be restarted.

Defining the Oracle Library

Use the following SQL commands to register the external libraries for Windows:

```

sql>
create or replace library IQStanLib
as
' C:\program files\intech\sdk\oracle\stan32cdecl.dll ';
/

create or replace library IQRapidLib
as
' C:\program files\intech\sdk\oracle\ipaf32cdecl.dll ';
/

```

Use the following SQL commands to register the external libraries for UNIX:

```

sql>
create or replace library IQStanLib
as
'/Intech/IQ_Office/lib/libiqstan.so';
/
create or replace library IQRapidLib
as
'/Intech/IQ_Office/lib/libiqpaf.so';
/

```

NOTE: Remember to update the path setting it to the location of the installed library

Load the Oracle Package

Run the relevant SQL script in the installation home directory under SDK\Oracle subdirectory:

```
IQOfficePackage.sql  
IQOfficePackage_Unix.sql
```

These scripts contain the PL/SQL wrappers to the IQ Office C libraries within an Oracle package.

Batch Address Validation - Sample

Run the SQL script `IQOfficeValidateSample.sql`

This script will create two tables and populate one of them with address data.

It will then create a procedure `IQBatchValidate` that will use the `IQRapid` package to validate the addresses in one table and output cleaned and validated results including barcode37 and flag codes to the output table.

The `IQBatchValidate` function will be called as part of the script execution.

NOTE: For multiple calls to the `IQBatchValidate` procedure it is important to first clear the output table of any previous results.

Microsoft SQL Database Integration

An extended stored procedure library has been included with the SDK. This library (`IQxp.dll`) will act as a wrapper to the standard IQ Office APIs allowing access to the functions from within a Microsoft SQL server.

Creating an extended stored procedure

Copy the file `IQXP.dll` from the Intech sub-directory `SDK\SQLSvr` to the SQL Binn directory:

```
C:\Program Files\Microsoft SQL Server\MSSQL\Binn
```

Register `IQXP.dll` as an extended stored procedure using "SQL Server Enterprise Manager" in the master database. This extended stored procedure must be called `xp_stanrt`

Sample Code

Run the SQL script `IQ_ValidateAddress.sql`

This will create the stored procedure `IQ_ValidateAddress` that parses and validates addresses using the `DpidGeocode.grm` configuration file. It also provides other examples on how to batch validate addresses from a database table.

IQ Office Release History

The sections below list new features in previous versions of the IQ Office Suite.

Version 5.7 (November 2013)

2014 Cycles

- The IQ Office package has been approved by Australia Post for the 2014 AMAS Cycle and New Zealand Post for the 2014 SendRight Cycle.

Standardiser

The batch size for using soap can now be configured from the GUI as a per project setting via the soap server options screen.

IQ Matcher

Bug Fix: List comparisons types were only comparing the first 10 items. This has now been increased and could effect the score calculated on records with long list items.

Bug Fix: Matching score calculated based on prefix is now limited by default weight.

IQ Soap

Bug Fix: Invalid XML characters could previously be passed through a standardisation process causing an invalid XML response.

SDK

- IQ Matcher SDK
 - New function: StandardiseAndWrite
- IQ Rapid Address SDK
 - "Level" and "Units" details were sometimes lost from input when matching on building Name.
 - New Properties: AddressFilter, ExpandLevelType, ExpandStreetType, ExpandStreetSuffix, ExpandUnitType, ListDeeperOnBestMatch, TitleCaseFields
 - New ReadOnly Properties: LicenceRemote, LastResult, LastResultLength
 - address record duplicating the RD details in formatting some addresses.
- IQ Office Grammar File SDK
 - ONLY_SPACE & NO_NUMBERS added as new tolerance rule qualifiers
 - Encrypted sections; allows for partially or fully encrypted grammar files and included tables.

Version 5.6 (November 2012)

2013 Cycles

- The IQ Office package has been approved by Australia Post for the 2013 AMAS Cycle and New Zealand Post for SendRight Cycle G.

IQ Matcher

- Bug Fix: Use of missing value scores set within the match spec was not being properly attributed to the overall record score. This may effect match tuning on projects that specifically set missing value scores.

SDK

- IQ Standardiser SDK
 - BatchStan & GetGrammarInfo SOAP functions
 - StanStanW
- IQ Matcher SDK
 - MatchRecord SOAP function
- IQ Rapid Address SDK
 - Formatting options updated.
 - New properties: AbbreviateStreet & AbbreviateLevel, ArfLibVersion*
 - AllowWithinRangeMatch, HouseNumResolveLocality, LoneNumberIsPostal, PolygonLongLatOrder, RapidCodeDump, ReturnFormattedAddressOptions, SwitchAlternateStreetPosition, SwitchSynonymLocalityPosition
 - New property: ValidateAddressOptions, can set to "CorrectNzSuburb=True"
 - GetAddresses – to allow return of all streets in a given postcode
 - Error codes: 6400-6499, 4900-4910, 4341
- IQ Office Grammar File SDK
 - "\CHARSET UTF8"
 - If an output field or variable doesn't have a string length, then it is now variable.
 - Error codes 191-196.
- IQ Office Grammar File Reference
 - DpidGeocode flags updated.

Version 5.5.4 (August 2012)

iqArfUtil

- iqArfUtil can now use spatially formatted GIS shape files as an input to the GIF building process. This eliminates the need to externally use GIS software to calculate spatial overlays prior to creating GIFs.

Address Validation

- Architectural changes have been made to the IQ Office libraries to allow version separation between AU Address validation, NZ Address validation and other countries.

Version 5.5.3 (May 2012)

Standardiser

- Custom reports based on field outputs can now be produced.
- Outputs optionally as both OLE DB and Text file where previously there could only be one output per project.

IQ Soap

- Bug Fix: Content-Type of "application/xml" for will now be used when serving up xml files instead of the "Unknown" Content-Type when the IQ Soap Server is acting as an httpd.

Version 5.5.2 (February 2012)

Standardiser

- State Electorate (SE) Name field size has been modified from 40 to 50 characters to prevent some electoral name truncations.

Version 5.5.1 (November 2011)

2012 Cycles

- The IQ Office package has been approved by Australia Post for the 2012 AMAS Cycle and New Zealand Post for SendRight Cycle F.

Standardiser / (Beta)

- IQ Standardiser has matured with some needed documentation and is no longer considered a beta product.
- The input specification can now replace newline characters within the input and substitute with any given character.
- Decoupled from library dependencies so it can now be deployed on a system with only the stanconsole available and connect to a soap server to perform batch configuration and execution.

Matcher

- Probabilities can now optionally be created with more than 100 values per field.
- Bug Fix: tolerance2 parameter can now be configured even when using prefixlist comparison type.
- Bug Fix: certain conditions could previously cause the set merging process in a two-file match project to fail.

Soap Server

- Bug Fix: ValidateAddressWithOptions previously didn't return a list if incorrect locality information provided.

NZ Address Validation

- Tightened rule on matching to a dpid via updating the suburb and postcode based on street and houseNo details where the input suburb and postcode exist. This will now prevent false positive results where addresses may be missing from the PAF.

Version 5.4.12 (August 2011)

IQ Easy Post

- Bug Fix: Correction to mixed file formatting where input was selected as delimited and output was selected as fixed width. Under these conditions the SOA creation process previously resulted in an error when basing the statement of accuracy on the cleaned addresses in the 'processed file'

Standardiser (Beta)

- Introduced functionality to produce post validation address reports.

Rapid Address

- IQ Rapid Address can now optionally be configured to be a client of an IQ Soap Server.
- Bug Fix: index error was occasionally being thrown after the popup hint box appeared.

Version 5.4.11 (May 2011)

Standardiser (Beta)

- Bug Fix: Correction for text files that use Unix Line feed characters which were creating an additional blank output record.
- Introduced functionality to optionally standardise via a soap server connection.

Matcher

- Records for a match block selection can now be retrieved via stored procedures to give more flexibility to the blocking criteria.

Soap Server

- Bug Fix: validateAdressWithOptions method will now return an empty Addresses array instead on an unpopulated Addresses element when the "Options" have been set for "Never" or "IfNeeded" and no address options are to be returned.

- Bug Fix: Increased buffer limits of address options as the use of large GeographicCodes in the return was sometimes raising an error.

NZ Address Validation

- Tightened rule on matching based on synonym city to prevent false positive results.
- Corrections are now made for suburb misspellings even though the SendRight rules do not place any value in the correctness of the suburb field.

General Address Validation

- Bug Fix: House Number suffixes were not been taken into account when matching to a primary point address.
- The Geocode reliability flag is now demoted by the address validation engine when dump coding so that a consistent result is provided across the interfaces. (The demotion previously only occurred within the grammar files).
- Improvements to flagString for flag codes containing 6 or 10 (some items amended and some not).

Version 5.4.10 (February 2011)

2011 Cycles

- The IQ Office package has been approved by New Zealand Post for SendRight Cycle E.

NZ Address Validation

- Correction to counter delivery addresses (Grammar).
- Improvements for Super City / Region name validation.
- Added Synonym City amendedFlag
- Rural Delivery numbers are now used within the Rapid Address searching.

General Address Validation

- Better separation of flagString "Some items amended & some not" to allow post validation analysis.
- Allow optional swapping of SynonymLocality and StreetAlias fields within the PAF record output, via two new properties, so that the synonym names can be easily used in place of gazetted values.
- The AdminGeocode.grm interface has been changed, increasing the SE_PID field output to varchar(10), SLA_Name to varchar (40), LGA_Name to varchar(40) and SE_Name to varchar(30).

Standardiser Beta

- Added support for reviewer and administrator modes to be based on project file configuration in addition to command line options.

Rapid Address

- IQ Rapid Address List tab will now display New Zealand Lobbies or Cities.

Version 5.4.9 (November 2010)

2011 Cycles

- The IQ Office package has been approved by Australia Post for the 2011 AMAS Cycle.

General Address Validation

- Bug fix: When using `validateAddressWithOptions` the rapid search results has now been limited to display those addresses matching the given locality where the primary address match has indicated a "matched to locality only".
- Bug fix: Internal buffer sizes have been increased to allow reverse searching of Geographic Index File (GIF) values that are longer than 20 characters.

Version 5.4.8 (August 2010)

NZ Address Validation

- `FormatAddressId` doesn't repeat lobby name anymore.

AU Address Validation

- Allow swapping position of alpha in unit number. Eg B1 -> 1B.

General Address Validation

- a new Property had been added "ReturnFormattedAddressOptions" which allow the configuration of street abbreviations and other such formats in the address line output that is returned from Rapid functions.
- Bug fix: Postbox search in postcode when there are more than 100 localities was not finding final address.
- Bug fix: Distance was not available in the confidence field output of the first result of a reverse LatLong search.

Version 5.4.7 (May 2010)

NZ Address Validation

- Better parsing of hospital names in addresses (Grammar)
- Better recognition of Region names in the input as opposed to matching as a city name (Engine)
eg Wellington in "WAIKANAE, WELLINGTON"
- Tighter restrictions on changing street type (Engine)
- Better matching to similar street names (Engine)

eg Alexandra to Alexander

- Improved matching using an ARF without postcodes
- Recognising more street name abbreviations, eg GR for Grove.
- Bug fix: "Street number can resolve street name" option on even if set to off.
- Bug fix: FormatAddressId function was not always outputting in an SendRight compliant format effecting the cleansed SOA result of Easy Post.

AU Address Validation

- Better parsing of addresses with unknown localities. (Grammar)
- Sort plans now returned by default (Grammar)
- Bug fix: a long GIF list was being truncation and not used in its entirety. (Grammar)
- Bug fix: some building names were being repeated in address lines (Engine)
- Bug fix: wrong Sort Plan Date being returned (Engine)

General Address Validation

- Bug fix: didn't allow drilling down into some addresses (IQ Rapid Address).

Version 5.4.6 (March 2010)

2010 Cycles

- The IQ Office package has been approved by New Zealand Post for CycleD of the sendRight program.

IQ Rapid Address

- A "hide on close" option has been added within the General options to prevent the accidental closing of the Rapid Address application. When this option is chosen the close will minimise the application to the task bar. The application can be closed by right clicking the task bar icon and choosing "exit"

IQ Rapid SDK

- Provided sample c# sample code.

NZ Address Validation

- Less importance given to old postcodes (Engine).
- Allow incorrect position of street suffix (Engine)
ie. streetsuffix before streetType
- Lobby name matching improvements (Grammar)
- Recognition of Region names (eg. Auckland) (Engine)
- Allow abbreviations in City names

eg "MT COOK" -> "Mount Cook"

- Allow unit number before building name (Grammar)
eg. "4J Stanford Apartments, 189 Hobson Street, Auckland 1001"
- Allow Search for NZ State Highway in IQ Rapid Address (Engine & Grammar)
- Bug fix: INVALID flag status no longer incorrectly given to some inputs (Grammar)

General Address Validation

- Address line formatting improvements (Grammar)
- Fixed address line truncation (Grammar)

Grammars

- UK postcode formatting improvement (Grammar & Stan Engine)
- Names Parsing improvements including allow spaces around apostrophes and/or hyphens (Grammar)
eg. "Paul & Jane O ' Sullivan"

Version 5.4.5 (December 2009)

2010 Cycles

- The IQ Office package has been approved by Australia Post for the 2009 AMAS Cycle.

Version 5.4.4 (August 2009)

IQ Easy Post

- The IQ Easy Post AU and IQ Easy Post NZ have been incorporated into a single application. The country data set and process can be changed via the country menu option.

Version 5.4.3 (May 2009)

Help Files

- All help file documents are now provided in Microsoft's HTML Help file format (.chm). This enables the documents to be read from Vista and Windows 2008 machines.
- Microsoft's HTML Help files should be stored and accessed locally. If they are accessed across a network they are unlikely to work due to IE security constraints.

IQ Office 64bit Files

- IQ Office 64bit compiled libraries are shipped within the \SDK\64bit directory.
- There have been some name changes for these files since their first release which have been performed to remove the legacy use of 64 in the name to indicate 64bit versions. Additionally we have taken the opportunity to prefix the libraries with "IQ" so that they are more easily found within the system

directories. The files can be copied with the previous names to support prior integrations without modifying any integration code.

Version 5.4.1 (February 2009)

Help Files

- The SDK documents are now provided in Microsoft's HTML Help file format (.chm). This enables the documents to be read from Vista and Windows 2008 machines.
- Microsoft's HTML Help files should be stored and accessed locally. If they are accessed across a network they are unlikely to work due to IE security constraints.

StanConsole

- Added Support for batch projects to use database sources and stored procedures.

IQ Soap Server

- Added Support for logging to a database via a stored procedure.

Version 5.4 (November 2008)

- IQ Office has been approved by Australia Post for the 2009 AMAS cycle.
- IQ Office has been approved by New Zealand Post for CycleC of the sendRight program.

Version 5.3 (November 2007)

IQ Matcher SDK

- The Matcher can standardise File A and File B data internally. These standardised fields can then be used as match fields for the field comparison process. See “Standardiser Integration” in the IQ Matcher SDK.
- Wild-card characters can be defined and used in matching.
- Regeneration of fields (such as phonetic codes) based on truncation at wild-card character position.
- New functions to match one record against a client-read file: `MoBegin`, `MoBeginPass`, `MoAddRecord`, `MoGetResult`.
- New function `CompareTwoRecords` – useful in debugging a match spec, to see how record-comparisons are scored.
- New comparison types: `InString`, `InStringWords`.

IQ Rapid SDK

- An address can be looked up via its ID (eg. DPID). Use the `GetAddress` function.
- When using an Address Reference file (ARF) with available geographical information file (GIF) indexes, addresses can be searched by GIF value. Use the `GetAddress` function. This could include searching for addresses within a given distance of a given latitude/longitude.
- The socket server can open specified Address Reference Files (ARFs) at start up – “Autoloading”.

IQ Stan SDK

- Fields that have been generated through a standardiser process are available as input to a subsequent process within the same project.

- New function `GetOutputFieldDescription`.

SOAP Server

- Will handle a HTTP GET for “?wsdl” and “/iqoffice.wsdl”.

Version 5.2

- IQ Office has been approved by Australia Post for the 2007 & 2008 AMAS cycle.
- Rapid Address can now use curly brackets {} around the locality information to activate an exact partial match on the locality input.

Version 5.1

- Additional Flag codes have been introduced for both Flag and AmendedFlag fields.
- METAPHONE phonetic codes can be created via the IQ Standardier engine.
- The PAF Record Format now returns 42 Fields (fields have been added to the end to remain backward compatible)

Version 5

- IQ Office has been approved by Australia Post for the 2006 AMAS cycle.
- Additional interfaces for TCP/IP Sockets and Soap servers

Version 4.4

- Multiple Address reference files can be opened by RapidSvr.
- Geographical Information files (GIF) can be accessed by the RapidSvr to provide additional information about an address or location.
- Additional amend flag code of 1 to represent dump coding when using GIF lookups.

Version 4.2

- IQ Office has been approved by Australia Post for the 2005 AMAS cycle.
- Error flag code of 64 introduced to represent that additional sub address information is available for the standardised address.

Version 4.1

- IQ Matcher has been enhanced to allow deterministic field comparisons to be included as ‘override options’ within a probabilistic match specification.
- The Matcher API has also been extended to allow for access to higher level functions. Including the `RunOnePass` and the `RunFullMatch` functions that can both make use of a pre-configured specification file as well as functions to create the probability and index files.

Version 4

The 2004 AMAS cycle has introduced various changes to address matching rules used to allocate DPID values and validate addresses. As part of these rules it is now possible to allocate a GroupDID using the “Below DPID matching rule 1” or a LocalityDID using the “Below DPID matching rule 2”

It is also possible to match addresses using street alternates and bordering localities. These changes have increased the information required to be returned by some of the API's and grammar files.

Overview of changes:

- Support for PAF 2004.
 - Faster searches when no locality level information is entered into the search criteria.
 - 'Popup hints' - a new feature (available via API and/or GUI) used to assist a user in entering valid search criteria. For example, a user can enter 'Kil', and then display a 'popup' screen, to show all building names and/or street names starting with 'Kil'.
 - **Additional fields** returned (via API and/or GUI) include:
 - Enhanced address accuracy flags – indicating how a PAF match was achieved (or why a match was not achieved), and which address elements were changed automatically after manual editing of addresses (see Manual editing of address below).
 - Bordering Locality – indicates a match was achieved via searching in a bordering locality.
 - Synonym street name – indicates an alternative name for the street.
 - Group Delivery ID (GroupDID) – an 8 digit identifier indicating a match to a PAF Group (street, locality, state and postcode combinations).
 - Locality Delivery ID (LocalityDID) – an 8 digit identifier indicating a match to a Locality (locality, state and postcode combinations).
- Note: DIDs start their 8 digit identifier with 26, 27, or 28
- **Auto searching** – GUI facility to allow searches to be triggered as the user types, rather than requiring a direct user search action.
 - **Auto Popup's** – GUI facility to allow 'popups' (see 'Popup Hints' above) to automatically be displayed while the user types the search criteria.
 - **Manual editing of address** – a user can now manually alter, or add to an address prior to accepting this address via the IQ Rapid Address GUI.

Grammar Files

The DpidGeocode & Dpid3Pass grammar files now return a locality with a 46 character length instead of 30 characters. This allows the locality field to become Australian Standard 4590 compliant. The grammar files can be manually altered to output 30 characters for backward compatibility. The flag field returned by the grammar files has also increased in size from 2 to 10 characters. This change allows for the necessary space to represent the address match conditions and if a DPID, GroupDID or LocalityDID has been allocated.

IQ Rapid Address

The rapid GUI and API have undergone extensive enhancements that now allow for superior speed when performing street level searches from the postal address file (PAF). As part of these enhancements a few new functions are now available including the GetPopupHint and GetNextPopupHint methods. The PAF record format has also changed from 26 to 35 fields, the additional fields being added to the end of the record. The input record format now allows for partial matching of address and locality fields. The partial address can be trailed by an asterisk. (Eg. For a street match "SPRIN*" will return Spring, Sprint etc.). To control these enhancements new properties SortPlanDate, PafEngineVersion, ListSynonyms, ListAlternateStreetNames, ListDeliveryIDParent, PutDIDinDpid have been made available under the properties method.

Additional error codes have also been added.

See IQRapidSDK.hlp for more detailed information on the above changes.

Version 3.1

- The DpidGeocode & Dpid3Pass grammar files return an amended flag, indicating what was amended (if anything) in the address.
- The Dpid3Pass grammar file will correct locality information even if a DPID is not found.
- Support for PAF Version 2003.
- IQ Rapid address can represent the found addresses in a Tree like structure.

May 2003 update

- PAF 2003.3 supplied
- IQ Matcher added to installation of IQ Office Enterprise
- New comparison types for IQ Matcher
- The maximum number of records returned by PAF libraries and RapidSvr can be changed via the MaxAddressesReturned property or set in the registry.
- New commands available for Writing Grammar Files: FOR ... NEXT, ELSEIF (see IF... ELSEIF ... ELSE ... END IF) and TRIM.
- Improved table look-up used by the standardiser, resulting in faster Grammar Files.

Version 2.9

The “IQ Rapid Address” application’s screen is now available via a COM object. See IQRapidForm.

- For RapidSvr and the PAF Libraries:
- Additional fields returned: *SynonymName* and *FinalAddress*; see Record format.
- Additional options for formatting an address via FormatAddress function or FormatAddress method.
- Functions to generate barcodes and sort plan numbers now available from **RapidSvr** and the **PAF Libraries**.
- New properties, allowing customisation of the searching and return format. See Properties property and SetProperty function.
- A new format for the ReadSynonymRecord function. See Direct PAF access functions.

IQ Address Reference File Utility

Overview

IQ Address Reference File Utility (ARFUtil) is a command-line utility used to test, build and extract the Address Reference Files (ARF) which provide the address records used by other IQ office suite applications. The available ARFUtil functions and ARFs depend on the purchased and installed IQ Office Edition, data files and licence. ARFs may include Australian and New Zealand Postal Address Files (PAF), and Australian Geocoded National Address File (G-NAF).

Using ARFUtil

To use ARFUtil, run it from the command line where it is installed, usually in the IQ Office Home Directory, e.g. C:\Program Files\Intech

Start Options

Use no options to run ARFUtil in interactive mode. Use `-h` to show all command line options.

```
C:\Program Files\Intech>iqArfUtil.exe -h
IQ ARF Utility - Version 5.4.80.2 (c) Intech Solutions Pty Ltd 2005-2010
Licenced to: Admin, (13/12/2004-31/12/2010) [ARFU1]
USAGE: iqarfutil
        iqarfutil Dir
        iqarfutil -a ArfSpec
        iqarfutil -d GnafSpec
        iqarfutil -o OledbGnafSpec
        iqarfutil -g GifSpec1 [GifSpec2 [GifSpec3 ...]]
        iqarfutil -e ExtractSpec
        iqarfutil [-v|-V|-h]
```

```
Dir   is the directory containing the raw Address Files
-a    will batch format the ARF according to the spec
-d    will batch format the GNAF files from multiple directories
-o    will batch format the GNAF files from an OLE DB data source
-g    will batch format the GIF, according to list of spec files
-v    will just show the header (with version and licence)
-V    will just output the version number
-h    will just show this help message
```

Menus

ARFUtil is menu-driven. The main menu is displayed in interactive mode. Type a menu number then press Enter to display sub-menus or execute a function. Available functions depend on the licence.

```
C:\Program Files\Intech>iqArfUtil
IQ ARF Utility - Version 5.4.80.2 (c) Intech Solutions Pty Ltd 2005-2010
Licenced to: Admin, (13/12/2004-31/12/2010) [ARFU1]
```

```
Main menu:
1-Find records in Address Reference File...
2-Test Address Reference Files...
3-Format PAF Menu
4-Extract ARF Menu
7-Geographical Index File (GIF) menu
8-Format Generic ARF menu
9-Format G-NAF menu
10-Help
11-Show command line options
12-Other utilities
```


0-Quit

IQ Office Component Setup

Overview

IQ Office Component Setup is a simple utility for configuring, installing and removing IQ Office components after software installation.

Functions include:

- Install and remove Address Reference Files (ARFs), and select the default ARF
- Edit configuration settings of all IQ Office components stored in the registry (Windows) or a text configuration file (Unix). See also the [IQ Office Configuration Manual](#).
- Install system component licence files.
- Configure machines as a Rapid Address or Standardiser socket, SOAP or DCOM client or server, and install, remove or manage services.

Using Component Setup

To start Component Setup, select it from the Intech | Other Utilities program menu, or from the command line:

```
C:\Program Files\Intech\Config\SetupEx.exe
```

This opens the Setup panel.



Use the Setup panel to manage installation and removal of IQ Office components and configuration settings. Click on a link to open a window to manage each component or setting.

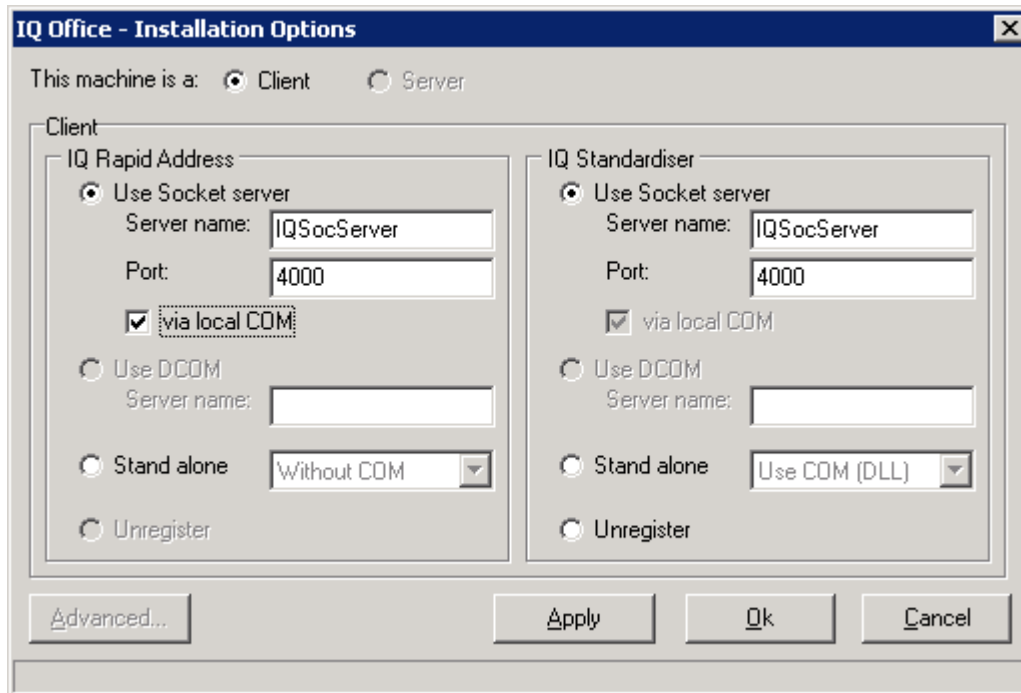
- [Client / Server Setup](#) – configure Rapid Address and Standardiser client and server
- [Advanced Registry Settings](#) – edit configuration settings of all IQ Office components
- [Install the Licence Files](#) – install system component licence files
- [Install IQ Data Files](#) – install ARFs and select default
- [Uninstall IQ Data Files](#) – uninstall ARFs
- [Stop and Uninstall Services](#) – stop and uninstall all services installed in [Client / Server Setup](#)

Click Close to exit Component Setup.

Client / Server Setup

Use the Client / Server Setup screen to configure a machine as a Rapid Address or Standardiser socket, SOAP or DCOM [Client](#) or [Server](#) in various configurations.

Client



Use this panel to configure Rapid Address and Standardiser clients.

IQ Rapid Address

Select an option to specify the address-searching engine which IQ Rapid Address will use.

Use Socket Server

IQ Rapid Address will use the address-searching engine located on the socket server.

This also applies to any other application that calls the PAF Libraries (`iPaf32.dll` or `IQPaf.dll`). All calls to the libraries will be redirected to the socket server.

Server name Enter a machine name on the local network (e.g. MACHINE2), an IP address (e.g. 192.168.0.50), or a network address (e.g. server.mycompany.com)

Port The port number must also be specified.

via Local COM

Check this option to configure IQ Rapid Address to use the RapidSvr COM server (`RapidSvr.dll` or `RapidSvr.exe`) on the local machine. RapidSvr will in turn call the PAF Libraries, which will call the socket server. To choose which of `RapidSvr.dll` or `RapidSvr.exe` are to be used, choose “Stand Alone” and select “Use COM (DLL)” or “Use COM (EXE)”. Click on “Apply”, and then choose “Use Socket Server”.

Clear this option to configure IQ Rapid Address to use the PAF Libraries, which will call the socket server.

Place the mouse pointer over this option to display the file name of the RapidSvr COM server.

This option will be ignored if the [Advanced Registry Settings](#) IQ Rapid Address | Use Dll option is set.

Use DCOM

IQ Rapid Address will use the address-searching engine located on the DCOM server. Specify the name of the machine running the IQ Rapid Address DCOM server.

Place the mouse pointer over this option to display the file containing the type library information for the RapidSvr DCOM object.

Stand alone

Select this option to configure IQ Rapid Address to use the address-searching engine on the local machine by one of the following methods:

- Without COM Use the address-searching engine (iPaf32.dll) directly.
- Use COM (DLL) Use the COM object in RapidSvr.dll. This is called an In-Process Server, whereby each application loads the COM server into its memory space and therefore each application will run its own instance of the engine.
- Use COM (EXE) Use the COM object in RapidSvr.exe. This is called an Out-of-Process Server, whereby the COM server will run in its own process and therefore all applications using the COM server will share one address-searching engine.

Without COM Do not use the COM object.

Place the mouse pointer over the “Stand Alone” label to display the location of iPaf32.dll.

Place the mouse pointer over the drop-down list to display the file name of the COM server.

IQ Standardiser

Select an option to specify the address-searching engine which IQ Standardiser will use.

Use Socket Server

IQ Standardiser will use the standardising engine located on the socket server.

This also applies to any other application that calls Stan32.dll (or IQStan.dll). All calls to Stan32.dll (or IQStan.dll) will be redirected to the socket server.

- Server name Enter a machine name on the local network (e.g. MACHINE2), an IP address (e.g. 192.168.0.50), or a network address (e.g. server.mycompany.com)
- Port The port number must also be specified.

Use DCOM

IQ Standardiser will use the standardising engine located on the DCOM server. Specify the name of the machine running the IQ Standardiser DCOM server.

Place the mouse pointer over this option to display the file containing the type library information for the StanRt DCOM object.

Stand alone

Select this option to configure IQ Standardiser to use the address-searching engine on the local machine by one of the following methods:

- Use COM (DLL) Use the COM object in StanRt.dll. This is called an In-Process Server, whereby each application loads the COM server into its memory space and therefore each application will run its own instance of the engine.
- Use COM (EXE) Use the COM object in StanRt.exe. This is called an Out-of-Process Server, whereby the COM server will run in its own process and therefore all applications using the COM server will share one address-searching engine.

Place the mouse pointer over the “Stand Alone” label to display the location of Stan32.dll.

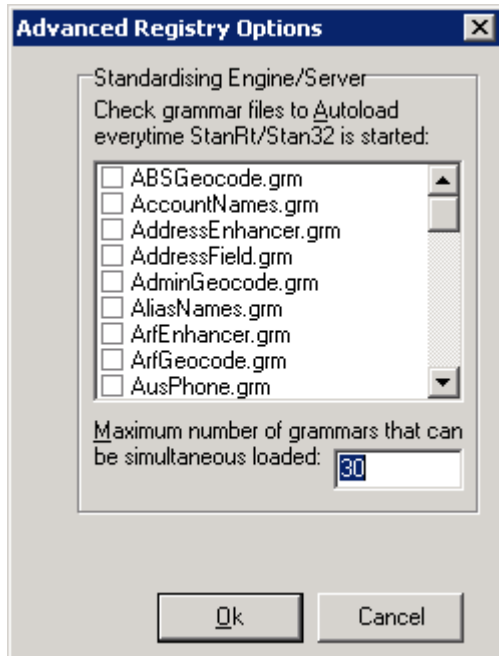
Place the mouse pointer over the drop-down list to display the file name of the COM server.

Unregister

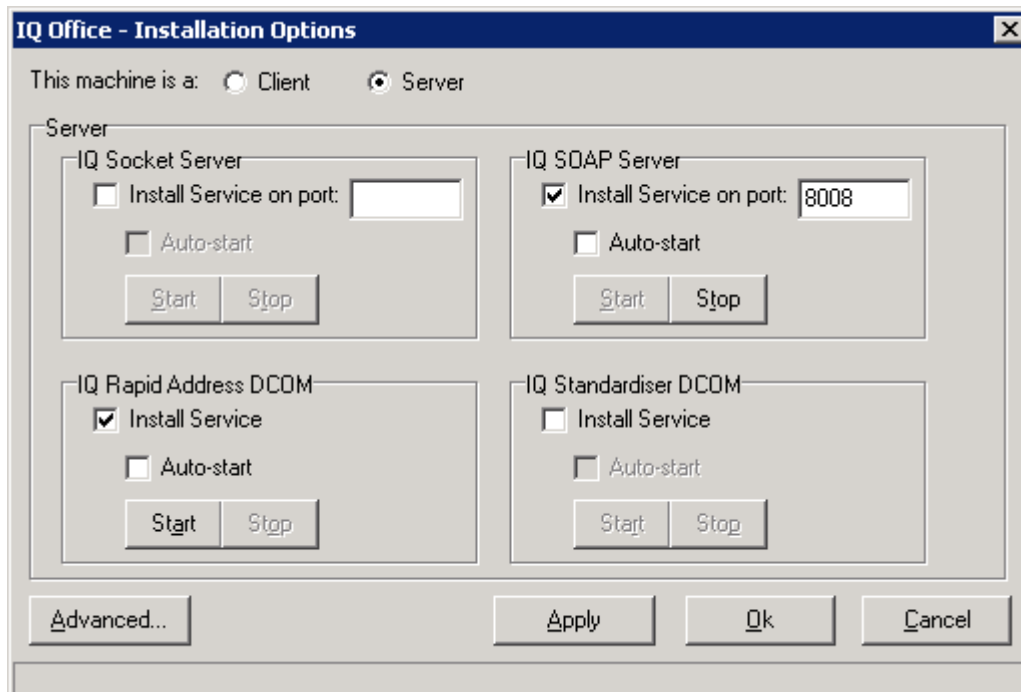
Select this option to unregister the StanRt COM server. IQ Standardiser will no longer work.

Advanced

Click to open the Advanced Registry Options panel. Check the names of grammar files to automatically load every time StanRt is loaded. You can also specify the maximum number of grammars which can be simultaneously loaded, which is the same as the [Advanced Registry Settings](#) Stan | Max Grammars option.



Server



Use this panel to configure Rapid Address and Standardiser servers (services).

Check a box then click OK to install a server. The Socket and SOAP server both require a port number to listen on, which must be the same as the client port.

Use the buttons to manage the installed servers.

- Start** start an installed server
- Stop** stop a running server
- Auto-start** automatically start server when the system starts

Place the mouse pointer over one of the servers to see the server file name.

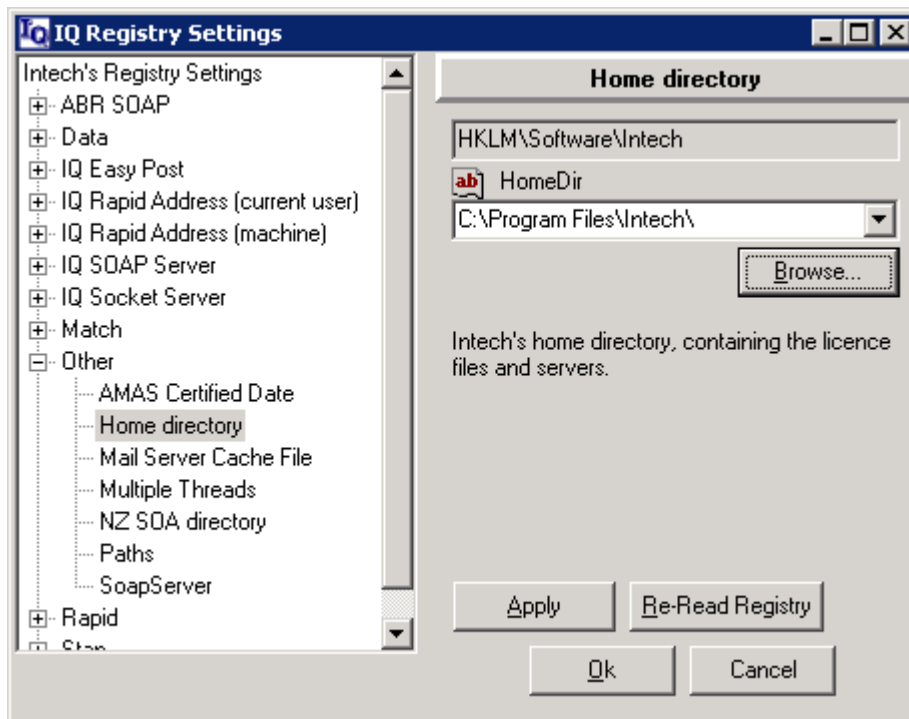
Use [Stop and Uninstall Services](#) to stop and uninstall all services installed here.

DCOM servers

The default security permissions of the more recent Microsoft Windows Server platforms may not allow remote users to connect to DCOM services unless they are also Administrators of the server.

To change this security use Windows “Component Services” manager to add the users or users groups that will need access to the services to the RapidSvr and StanRt DCOM objects with at least remote access and remote activation permissions. When setting these permissions on the individual DCOM objects it may also be necessary to edit the COM security limits for the server to also allow these permissions.

Advanced Registry Settings



Use this panel to edit configuration settings of all IQ Office components. Configurations are stored in the registry (Windows) or a text configuration file (Unix) as described in the [IQ Office Configuration Manual](#). Other specific configuration settings are described in the reference manual for each component.

To edit configuration settings, expand the tree on the left to find the setting, then enter the details in the panel on the right, which contains a description of each setting, and may also contain drop down lists, check boxes or browse buttons.

Configuration settings are summarised under each category below.

- ABR SOAP – proxy server for ABN lookup service
- Data – default ARF directories and memory loading options
- IQ Easy Post – grammar files used
- IQ Rapid Address (current user) – current user configuration settings

- IQ Rapid Address (machine) – machine-wide configuration settings
- IQ SOAP Server – authentication, logging and other advanced options
- IQ Socket Server – socket limit, timeout and audit file
- Match – IQ Matcher paths and parameters
- Other – home directory, paths and AMAS certification date
- Rapid – audit, socket and library settings
- Stan – audit, socket, limits and grammar directory

Use the buttons at the bottom of the panel to load or save changes.

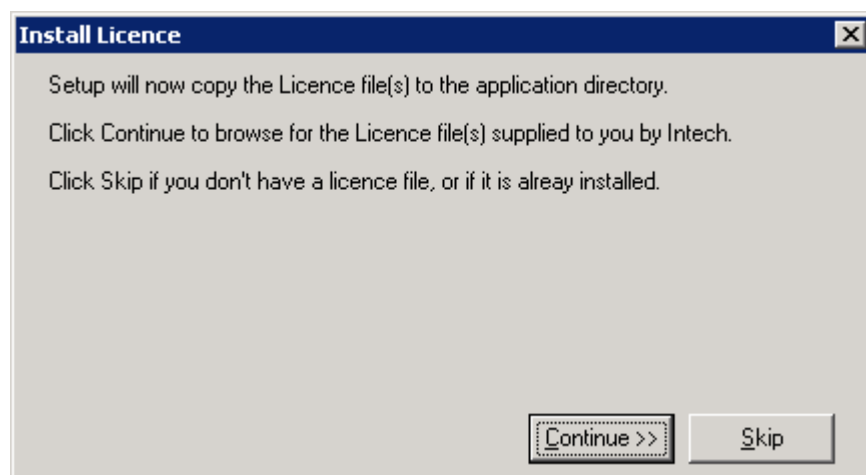
Apply Save the entered or selected setting to the registry or configuration file.

Re-Read Registry Load the existing setting from the registry or configuration file.

OK Apply all changes and close the panel.

Cancel Discard all changes and close the panel.

Install the Licence Files



This panel prompts to copy the licence files to the Intech Home Directory, which is defined in the configuration settings below, see [Home Directory](#).

Windows Registry

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech]
"HomeDir"="C:\\Program Files\\Intech"
```

Unix Configuration File

```
HomeDir=/home/intech
```

To change the Home Directory, edit the [Advanced Registry Settings](#) Other | Home Directory.

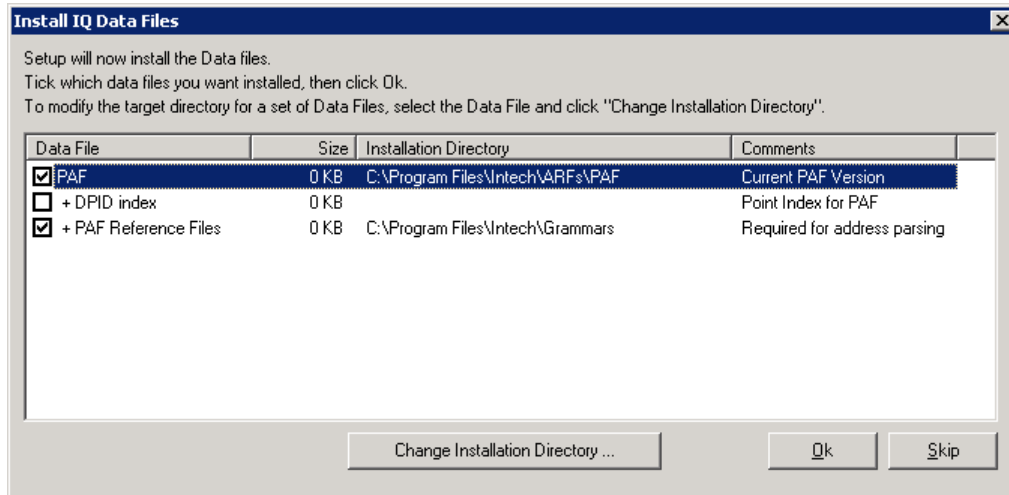
All licence files are listed below. The available files depend on the IQ Office Edition and purchased licence.

iStan.lic	Required for IQ Standardiser and for the standardising engine (Stan32.dll or IQStan.dll).
iRapid.lic	Required for IQ Rapid Address and for the address-searching engine (iPaf32.dll or IQPaf.dll).
IQMatch.lic	Required for IQ Matcher.

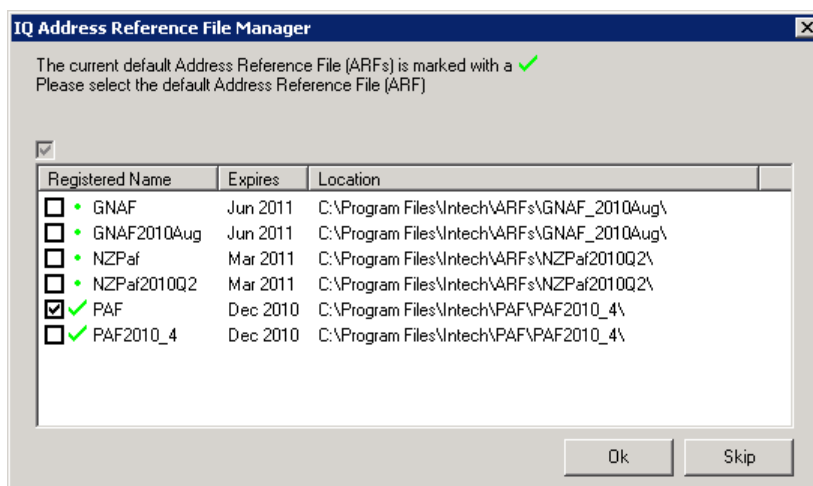
iPost.lic Required for IQ Easy Post if there is no iStan.lic.

ArfUtil.lic Required for advanced functionality of iqArfUtil. If there is more than one such licence all are copied, e.g. ArfUtil1.lic, ArfUtil2.lic

Install IQ Data Files

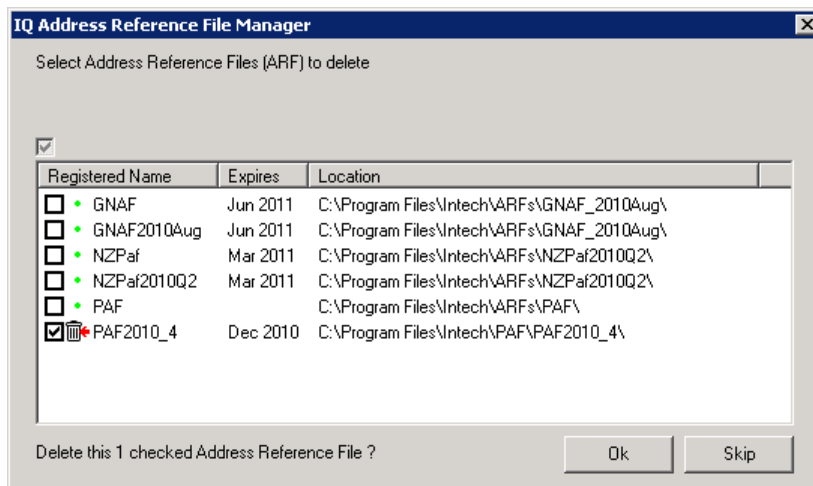


Use this panel to install data files. The available files depend on the IQ Office Edition and purchased licence. Check the files to install then click OK to display the ARF Manager panel.



Check a box next to the file to be the default ARF then click OK.

Uninstall IQ Data Files



Use this panel to uninstall data files. The available files depend on the IQ Office Edition and purchased licence, and those that have been installed by [Install IQ Data Files](#). Check the files to uninstall then click OK.

Stop and Uninstall Services

Click this link to stop and uninstall all services installed in [Client / Server Setup](#).

IQ Easy Post

Overview

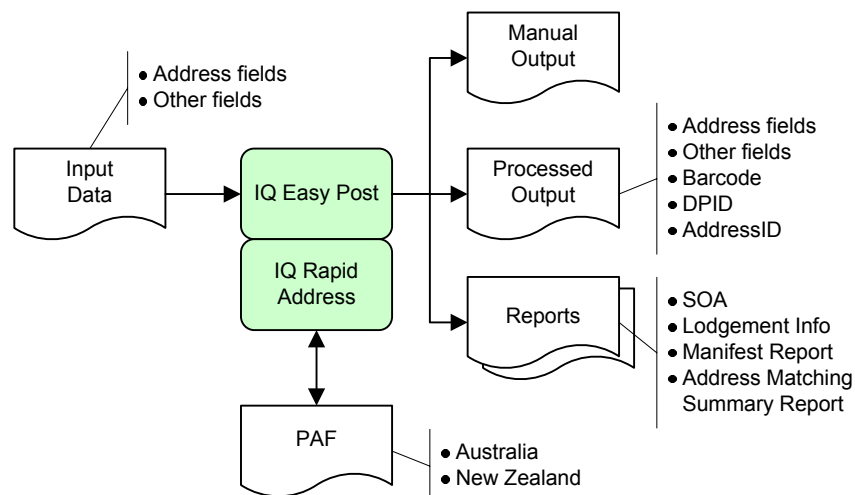
IQ Easy Post is an easy-to-use application that prepares addresses for posting using the Australia Post barcoding system or New Zealand Post bulk mailing system. It is ideal for small to medium enterprises that lodge their own post directly, as well as mail houses, list brokers and data processing bureaus that lodge post on behalf of their customers.

IQ Easy Post reads an input text file containing addresses, allocates a Barcode and BSP number or AddressID to each address, creates an output file with fields required for mailing, sorts files according to various options, and produces all required lodgement reports.

Addresses which could not be validated during address processing can be written to a file which can subsequently be reviewed by looking them up in the Australia Post or New Zealand Postal Address File (PAF), which contain all postal delivery addresses in the country. When the address and corresponding DPID / AddressID is found, the output file is updated.

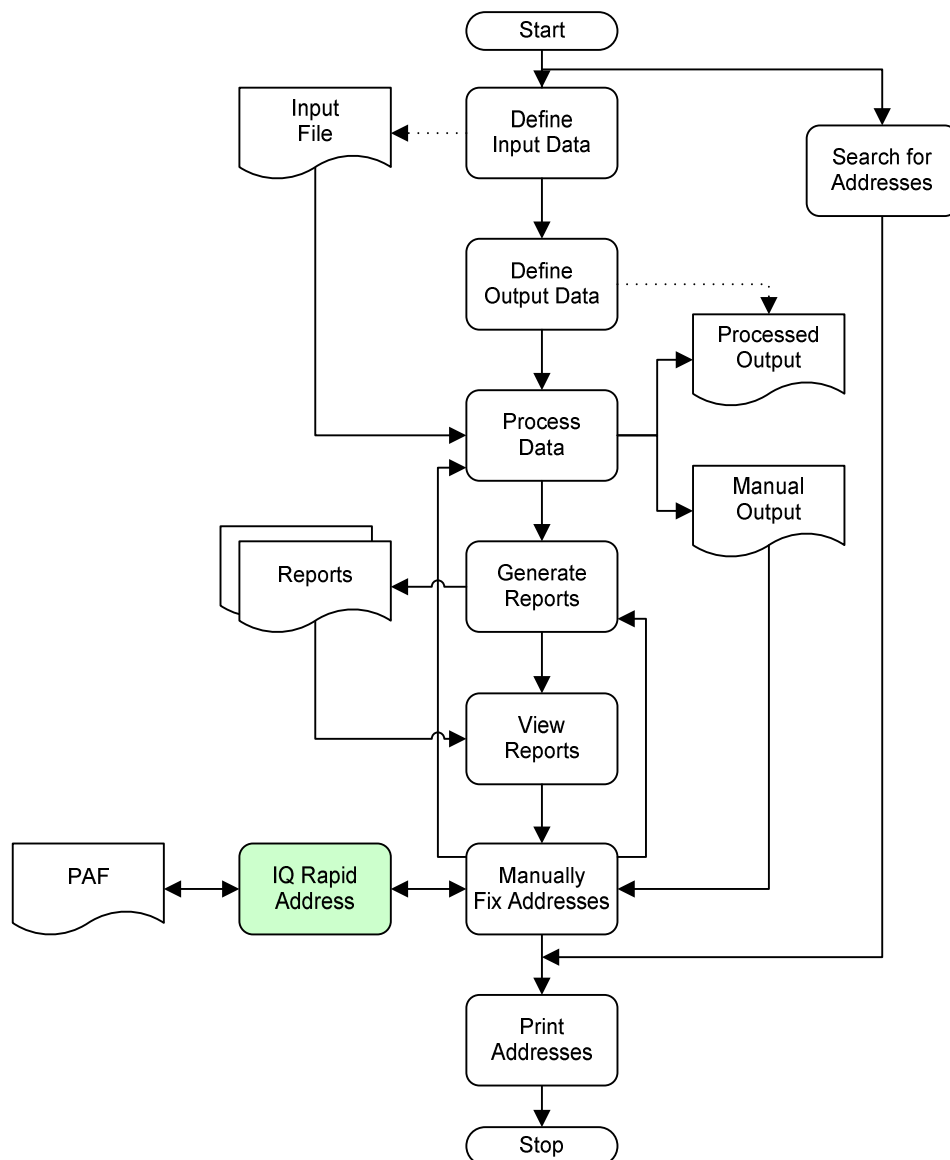
Postal addresses can be looked up in the PAF at any time using the Rapid Address panel.

The diagram below shows a summary of IQ Easy Post functions and files.



Workflow

The diagram below shows a typical EasyPost workflow and associated files for [Using IQ Easy Post \(Aus\)](#) and [Using IQ Easy Post \(NZ\)](#).



Australian Address Fields

IQ Easy Post generates an output file with the following fields from the address input file. You can also use the [Input File tab](#) to select additional fields in the input file to be output directly.

Address1, Address2, Address3, Locality, State, Postcode

These fields make up the address to be printed on the mail item.

Flag

Indicates whether the input address was modified. The number of modified addresses is needed for the Address Matching Summary Report.

BSP

Used for creating the Manifest Report and for sorting the addresses ready to qualify for Barcode PreSort mail discounts.

Barcode37

A 37-digit barcode string generated from the DPID of the address. The string is composed of digits 0 to 3 which are printed as bars to form the barcode on the mail item.

New Zealand Address Fields

IQ Easy Post generates an output file with the following fields from the address input file. You can also use the [Input File tab](#) to select additional fields in the input file to be output directly.

PreAddress, Address1, Address2, Address3, Suburb, City, Postcode

These fields make up the address to be printed on the mail item. If IQ Easy Post corrected the address, these fields are updated.

Flag

Indicates whether the input address was correct, corrected, or why it could not be corrected.

ValidFlag

Indicates whether the unchanged address is valid according to New Zealand Post standards. It can have a value of VALID_U (unique match), VALID_B (base address match) or INVALID.

AddressId

Number assigned by New Zealand Post that uniquely identifies the address.

Australian Reports

IQ Easy Post automatically creates the three reports required for Australian barcoded bulk mail lodgement. Reports options can be specified in the [General Options tab](#) and [Report Options tab](#).

Address Matching Summary Report

The Address Matching Summary is a declaration of the software used to append the DPID and Barcodes, that is, IQ Easy Post. It also records the number of addresses for which a DPID was found, the number of addresses that needed to be modified in order to append a DPID according to AMAS rules, and the number of addresses for which a DPID could not be found.

The report is created in both text and HTML formats, with the following file names:

```
AddressMatchingSummaryReport.htm
AddressMatchingSummaryReport.txt
```

Manifest Report

The Manifest Report contains the number and weight of letters and the number of trays for each BSP number. It divides the BSP numbers into states, providing state totals and grand totals, and will handle mixed barcoded and unbarcoded lodgements.

The report is created in text, CSV, and HTML formats, with the following file names:

```
ManifestReport.csv
ManifestReport.htm
ManifestReport.txt
```

Lodgement Info

The Lodgement Info report contains a summary of the Manifest Report, which is the information required by Australia Post for the lodgement document when lodging PreSort letters.

The report is created in both text and HTML formats, with the following file names:

```
LodgementInfo.htm
LodgementInfo.txt
```


New Zealand SOA

A Statement of Accuracy (SOA) certificate shows the percentage of addresses in a mailing list or database that are valid matches against the New Zealand Post Postal Address File (PAF). The SOA is valid for one year from the date of issue and must be sent to New Zealand Post to obtain bulk mail discounts. IQ Easy Post automatically generates and issues the SOA, and a log file summarising of all issued SOAs.

Each address is marked with the ValidFlag address field as a unique match, base address match, or invalid. The SOA shows the number and percentage of valid matches (unique, base and combined).

Manual Fix Address

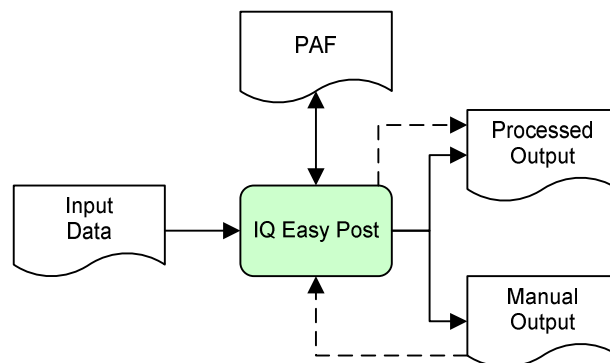
During processing, addresses may be found which are invalid (NZ), or could not be allocated a DPID or barcode (Aus). Such addresses can be manually fixed from the [Manual Fix Address tab](#) or [Process tab](#) by searching the PAF, enabling found addresses to be assigned DPIDs (Aus) or validated (NZ).

Manual fix works by reading each record in the input file, copying valid records directly to the output file, and prompting to search for the invalid addresses in the PAF. As a result of searching:

- A found final address can be accepted to copy the record to the output file with the address fields validated and updated.
- If an address cannot be found it can be skipped to copy the invalid record to the output file as is.

Manual fixing can be paused at any time and later resumed, even if Easy Post has been closed.

A copy of the processed output file incorporating all corrections is created after manually fixing addresses. The diagram below summarises the manual fix process, showing initial address processing as solid lines, and subsequent fixes as dashed lines.



Using IQ Easy Post (Aus)

Use Easy Post by filling in each tab in the order below. To select the next tab either click the tab name at the top of the screen, or the Next button at the bottom of the screen. If an item dependency is missed or not valid you will be unable to move to the next tab until valid options have been selected.

7. Start Easy Post, see [Starting and Exiting](#).
8. Select Australia from the Country menu.
9. Use the [Input File tab](#) to select the input data file and fields to process, then the [Input Text File Format screen](#) to define the format of the input fields. File formats can be loaded or saved using the [Saved Text File Formats](#) dialog box. Either click the Quick Start button to process addresses, or proceed to the other tabs.
10. Use the [File Options tab](#) to define the location and name of the output files, then the [Output Text File Format screen](#) to define the format of the output fields.
11. Use the [General Options tab](#) to define the AMAS address compliance level, and sorting of the processed output.

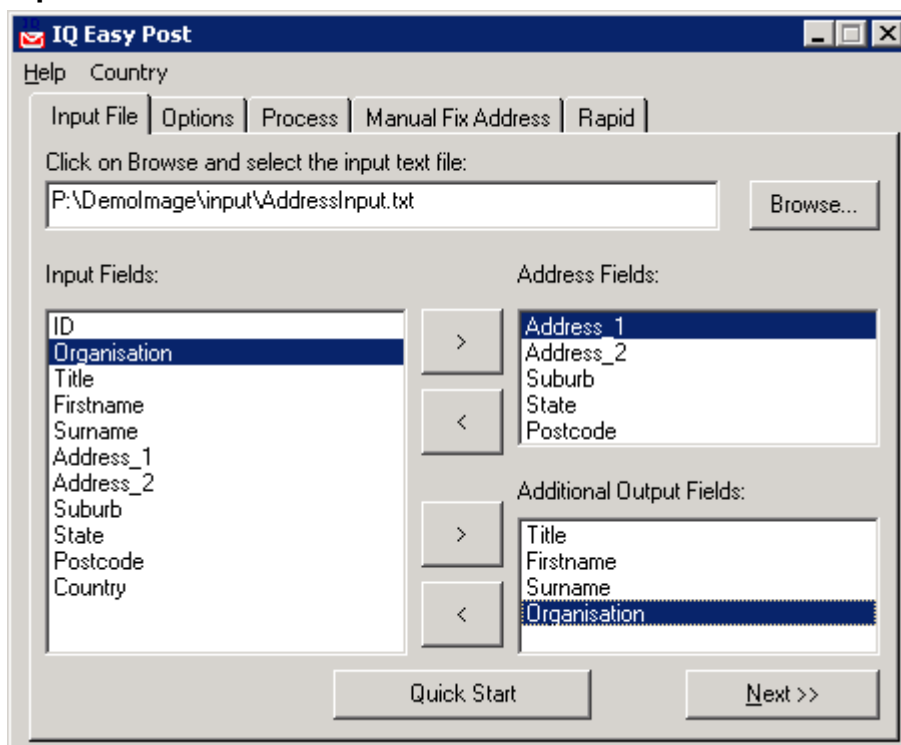
12. Use the [Report Options tab](#) to define options for the [Manifest Report](#) and [Address Matching Summary Report](#).
13. Use the [Process tab](#) to process the addresses in the input file and generate the output file and [Australian Reports](#). You can also view and regenerate the reports.
14. Use the [Manual Fix Address tab](#) to manually fix addresses which have not been allocated DPIDs, and regenerate and view reports.
15. Use the [Rapid Tab](#) at any time to search for addresses in the PAF.

Starting and Exiting

To start IQ Easy Post, select it from the Intech program menu or double-click the desktop icon. The [Input File tab](#) is displayed.

To exit IQ Easy Post, click the X button in the top right of the window or press Alt+F4.

Input File tab



Use this tab to specify the name and format of the text file containing addresses to be processed.

Click on Browse and select the input text

Click the **Browse** button and browse to the name of the input file. The [Input Text File Format screen](#) is then displayed to specify the file format. There is an option to accept the format which has been correctly guessed.

Input Fields

This panel displays fields in the input file which can be processed as Address Fields, or copied directly to the output file as Additional Output Fields. This list cannot be modified.

Address Fields

Select fields in this panel which will be processed to produce address lines, locality, state, postcode and barcode in the output file.

Additional Output Fields

Select fields in this panel which will be copied directly to the output file without modification.

Each field can only appear once in any list. Use the following actions to edit the Address and Additional Output fields.

- To remove fields from either list, select them then press the Del key, or click the left arrow button <, or drag and drop between lists.
- To add fields to either list, select them in the Input Fields list, then click the right arrow button >, or drag and drop between lists.

Quick Start

Click to process the addresses and go straight to the [Process tab](#), or select options in the other tabs.

Next

Click to go to the next tab required to complete address processing.

Input Text File Format screen

ID	Organisation	Title	Firstname	Surname	Address
1	Abercrombie Holdings	Mr	Aaron	Aarons	33
2	Abercrombie Enterprises	Mrs	Aaron	Abba	Le
3	Aberdare Pty Ltd	Miss	Aron	Abbey	Le
4	Aberdeen Holdings	Ms	Ari	Abbeywood	93
5	Aberfeldie Enterprises	Dr	Abbey	Abbotsbury	25
6	Aberfeldy Pty Ltd	Prof	Gail	Abbotsford	Un
7	Aberfoyle Holdings	Sir	Gayle	Abbotsham	10
9	Aberfoyle Enterprises	Lady	Abbey	Abels	Le

Use this screen to define the format of the input file specified in the [Input File tab](#). A preview of the input file is displayed at the bottom of the screen. It may also be useful to first view the contents of the file using a text file viewer or editor.

There are two variations of this screen depending on whether each field in the file is delimited by a character such as a tab or comma, or a fixed width.

Delimited

When a delimited file is opened, lines are automatically added at each delimiter (field). Manually edit the fields as required to define the size and name of each input field. All fields must be named, and names must be unique.

Check the option button then select other options:

- **First Row Contains Field Names** – check this option if the first row of the file contains the names of the fields. If not, the name of each field must be defined.
- **Field Delimiter** – select the single character that delimits the fields in each row:
, (comma) ; (semicolon) {tab} {space}

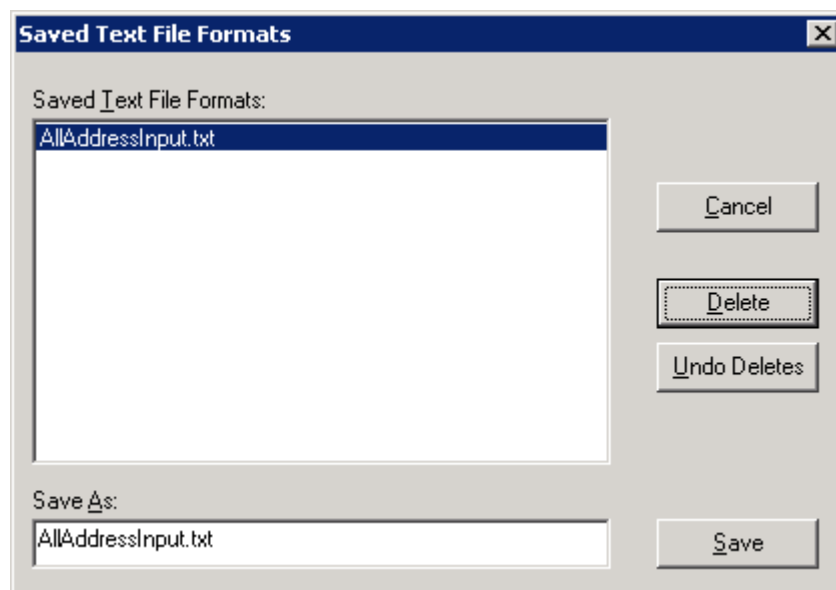
- **Text Qualifier** – (optional) select the character that surrounds text in each field:
{none} ' (single quote) " (double quote)
- **Max Field Size** – (optional) enter the maximum width of a field in numbers of characters.
- Use the **Next Field** and **Previous Field** buttons to scroll between fields.
- Click the **Load** or **Save** button to load or save field format definitions in [Saved Text File Formats](#).
- Click the **OK** button when all the fields have been defined.

Fixed Width

When a fixed-width file is opened, lines are automatically added at the assumed edge of each field. Manually edit the fields as required to define the size and name of each input field. All fields must be named, and names must be unique.

- Type the name of the field in the **Field Name** box, e.g. GivenNames
- Either click on the ruler to add a line between fields, or enter the width in number of characters in the **Field Size** box, e.g. 20
- Click an existing line between fields to remove it.
- Use the **Next Field** and **Previous Field** buttons to scroll between fields.
- Click the **Load** or **Save** button to load or save field format definitions in [Saved Text File Formats](#).
- Click the **OK** button when all the fields have been defined.

Saved Text File Formats



Use this dialog box to save or delete text format definitions created in the [Input Text File Format screen](#). These files should be given clear descriptive names.

- | | |
|--------------------|-------------------------------------|
| Open | Open a saved text file format. |
| Delete | Delete a saved text file format. |
| Undo Delete | Recover a deleted text file format. |
| Save | Enter a name and save the format. |

Options tab

Use this tab to specify the various options from three tabs:

- [File Options tab](#) – location and name of output files.
- [General Options tab](#) – AMAS address compliance level and sorting of the processed output.
- [Report Options tab](#) – options for the [Manifest Report](#) and [Address Matching Summary Report](#).

File Options tab

Use this tab to specify the text files into which processed and [Manual Fix Addresses](#) are placed. Files which do not exist are created, or are overwritten if they exist. Use an obvious name for the file.

Output file for Processed addresses

Click the **Browse** button and browse to the name of the output file. The [Output Text File Format screen](#) is then displayed to specify the file format.

Output file for Manually Fixed addresses

Click the **Browse** button and browse to the name of the file. This uses the same output format settings as the processed address file.

Report directory for addresses

Type the path of the directories into which the output files are placed. If the directory does not exist it is created when the addresses are processed.

Output Text File Format screen

Use this screen to define the format of the processed address output files specified in the [File Options tab](#). For **Fixed Width** format, check the option button. No other options can be selected.

For **Delimited** format, check the option button then select other options:

- **Field Delimiter** – select the single character that will delimit fields in each row:
, (comma) ; (semicolon) {tab} {space}
- **Text Qualifier** – (optional) select the character that will surround text in each field:
{none} ' (single quote) " (double quote)
- **First Row Contains Field Names** – (optional) check this option to include the names of the output fields at the top of the output file.

Examples

Fixed Width format:

```
Mr   Matthew James   Holm
Ms   Sheila          Cowper-Smith
```

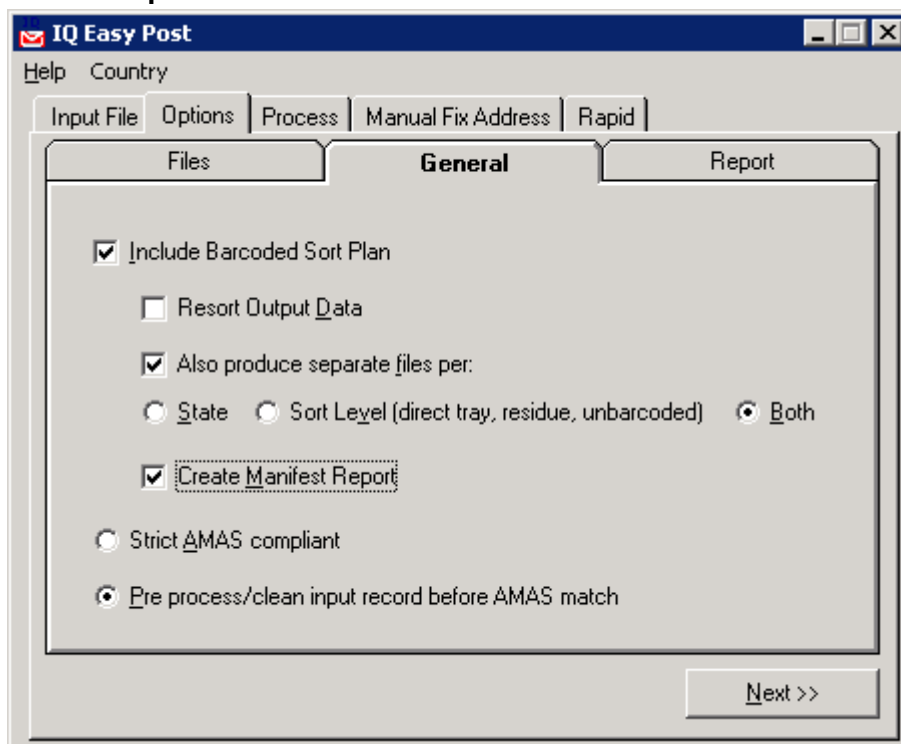
Delimited format, Field Delimiter comma, Text Qualifier {none}

```
Mr,Matthew James,Holm
Ms,Sheila,Cowper-Smith
```

Delimited format, Field Delimiter semi-colon, Text Qualifier double quotes:

```
"Mr";"Matthew James";"Holm"
"Ms";"Sheila";"Cowper-Smith"
```

General Options tab



Use this tab to specify AMAS address compliance level, and sorting of the processed output. There are three Sort Levels:

- **Barcode Direct Tray (BDT)** – If there are at least 300 barcoded letters (4kg) in a particular BSP, these letters can be put in direct trays, and benefit from reduced rates.
- **Barcode Residue (BR)** – These are barcoded letters of which there weren't enough to make a direct tray. If there are more than 2000 barcode residue letters, they will be divided into states.

- **Unbarcoded Residue (UR)** – These are unbarcoded letters. These may be lodged together with barcoded letters until July 2002, and only if they make up less than 10% of the lodgement.

Include Barcode Sort Plan

Select this option to include the BSP (Barcode Sort Plan) number in the output.

Resort Output Data

Check this option to sort the output file by Sort Level, then by BSP number.

Also produce separate files per:

Check this option to create separate files per state or Sort Level in the same directory as the output file. Files are named with a prefix before the output file name, e.g. `NSW_ProcessedOutput.csv`, `BDT_output.csv`, etc.

- **State** – create separate files per state with the state prefix, e.g. `NSW_`, `WA_` etc
- **Sort Level** – create separate files per sort level with the sort level prefix, i.e. `BDT_` (Barcode Direct Tray), `BR_` (Barcode Residue) or `UR_` (Unbarcoded residue)
- **Both** – create separate files per state and sort level, creating up to 24 separate files, e.g. `BDT_NSW_out.csv`, `BR_NSW_out.csv`. The prefix `ERR_UR` indicates unbarcoded letters without a state.

Note that there is no Sort Level field in the output file, the sort level is determined according to the number and weight of the letters per BSP.

Create Manifest Report

Check this option to create the [Manifest Report](#). Other options for this report must then be specified in the [Report Options tab](#).

Select one of the following options to specify AMAS Compliance.

Strict AMAS compliant

Use only AMAS-approved software in looking up addresses and appending barcodes.

Pre process/clean input records before AMAS match

Allow the IQ Easy Post “Address Enhancer” to pre-clean addresses, increasing their chance of getting a barcode. Addresses that don’t get a barcode appended when using AMAS-approved software are looked up in the PAF and pre-cleaned. The resulting address are then passed through the AMAS software to get a barcode. There is still a chance that the record will not get a barcode, even after the pre-clean.

Report Options tab

IQ Easy Post

Help Country

Input File Options Process Manual Fix Address Rapid

Files General Report

Manifest Report

State of Lodgement: NSW *

Number of letters per Tray: 300 * *required

Weight per letter in grams:

Customer Name: Bigmail Corp

Job Number: 101

Address Matching Process Summary Report

List Processor's Name: Mr Processor

Name of Address List: Monthly Mailout

List Owner/Manager: Ms List-Owner

Next >>

Use this tab to specify options for the [Manifest Report](#) and [Address Matching Summary Report](#). The Manifest Report options are only enabled if the **Create Manifest Report** option is selected on the [General Options tab](#).

Manifest Report

Enter the following options. The first two options are mandatory.

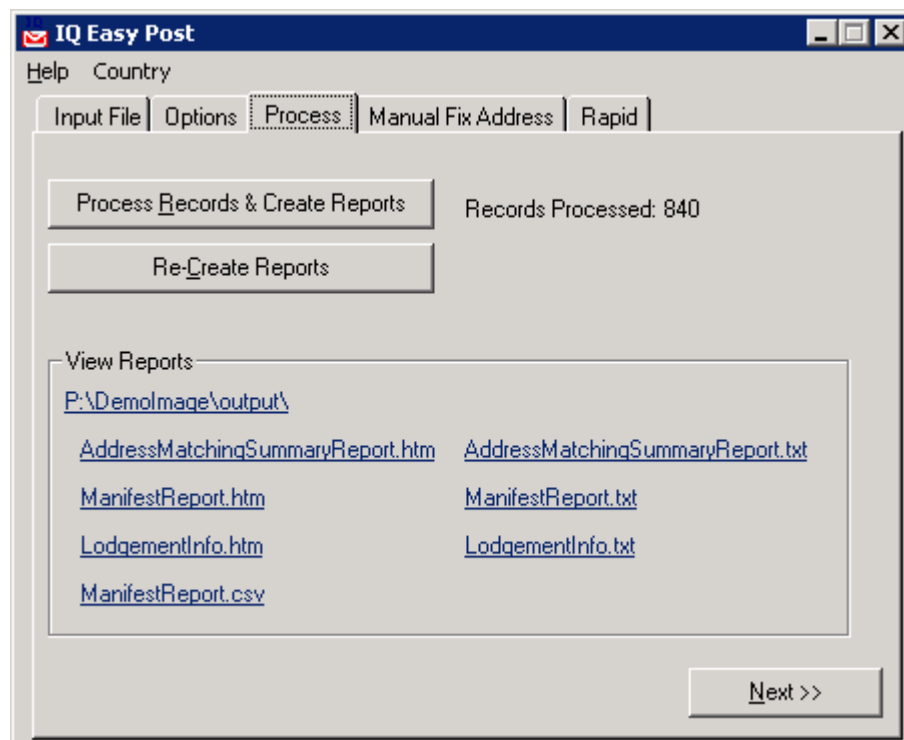
- **State of Lodgement** – the Manifest Report requires the state of lodgement, as same-state mail is given greater discounts.
- **Number of letters per Tray** – enter the maximum number of letters that can fit into one tray, which will determine the number of trays required, and is used to calculate the number of trays per group, e.g. per BSP
- **Weight per letter in grams** – (Optional) weight of each letter. This is used to provide the weight per group in the Manifest Report and to determine the minimum number of letters to make a direct tray: 300 or 4kg. If omitted, no weight information will appear on the report.
- **Customer Name** – (Optional) name of the mailout customer to appear on the top of the report.
- **Job Number** – (Optional) number of the job to appear on the top of the report.

Address Matching Process Summary Report

Enter the following options which will appear in the report as entered here:

- **List Processor's Name** – name of the person responsible for processing the list
- **Name of Address List** – name of the mailout list itself
- **List Owner/Manager** – name of person responsible for managing the list

Process tab



Use this tab to process the addresses in the input file to generate the output file, and regenerate and view the [Australian Reports](#). The re-create button and report links are unavailable if reports have not yet been created.

Process Records & Create Reports

Click this button to process the addresses and sort and create separate files if these options have been selected in the [General Options tab](#).

Re-Create Reports

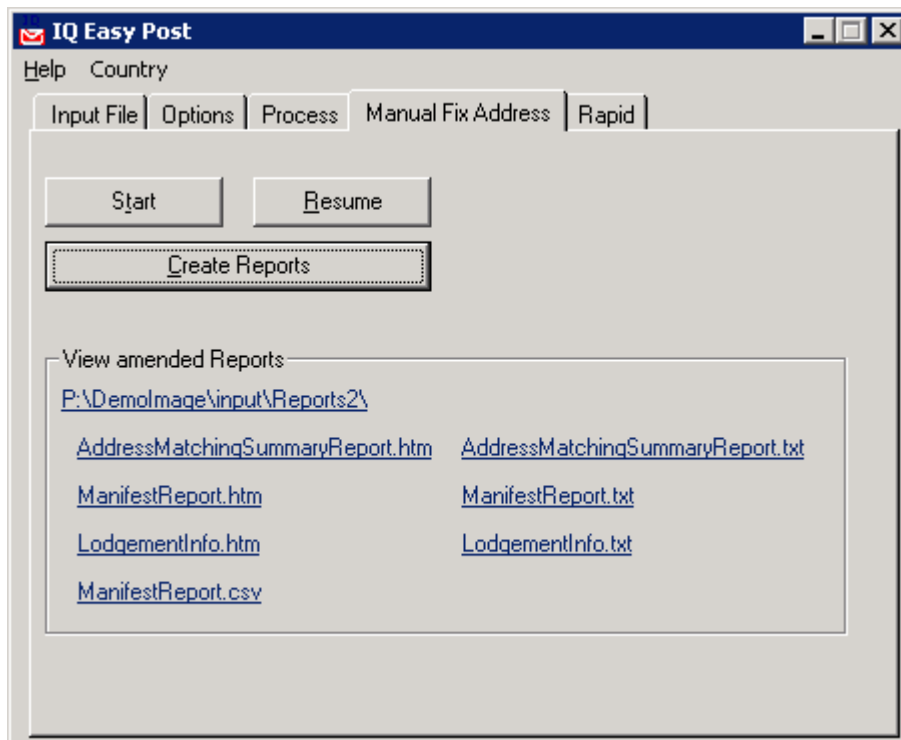
Click this button to re-create the reports and re-sort and re-create separate files if records have already been processed but the reporting, sorting and separate file options have changed.

View Reports

Click a link in this panel to view the directory or report. The viewing program depends on the file extension, for example:

Directory	Explorer
.htm	Internet Explorer
.txt	Notepad
.csv	Excel

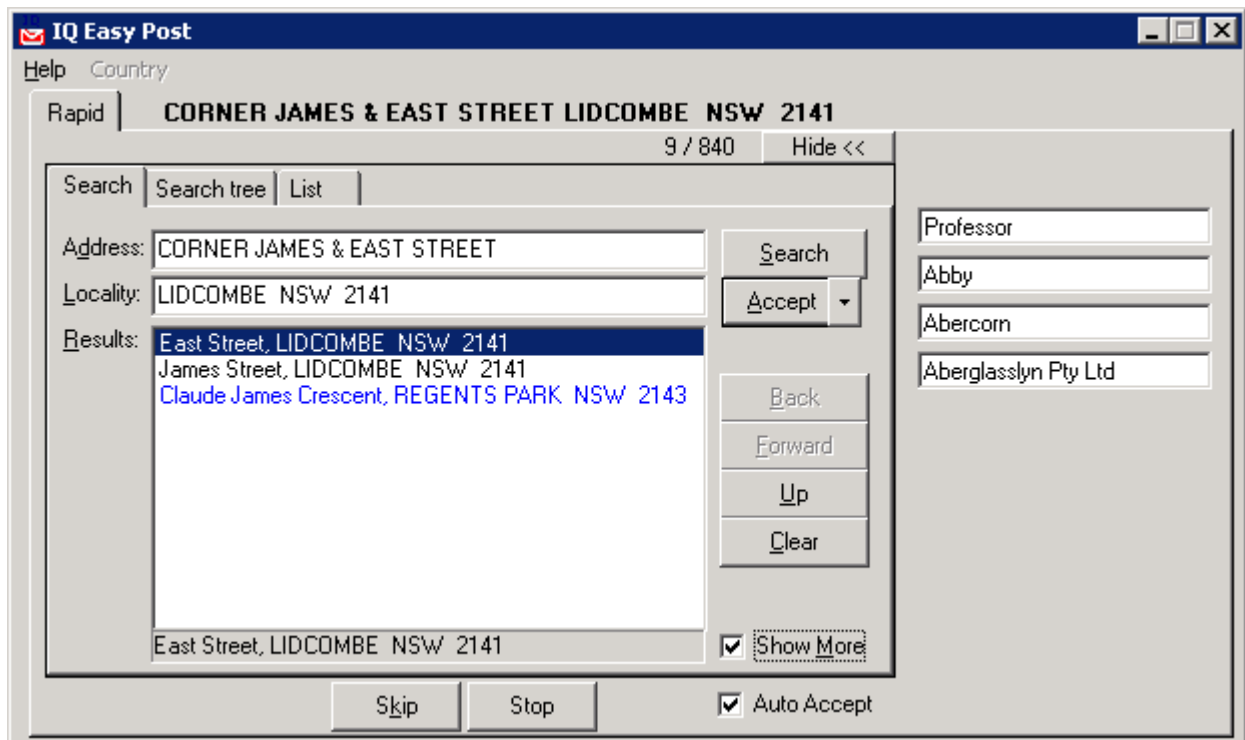
Manual Fix Address tab



Use this tab to [Manual Fix Address](#), and regenerate and view [Australian Reports](#) in the same way as the [Process tab](#). The Resume button, Create button and report links are unavailable if reports have not yet been recreated.

Start

Click to open a tab that is similar to the [Rapid Tab](#). This displays individual addresses that were not allocated DPIDs or barcodes during address processing, enabling each record to be manually corrected using [Manual Fix Address](#). The current and total number of records processed is displayed in the top right of the tab.



Search for the correct address and accept or skip it using the following buttons.

- Show / Hide** Click to show or hide up to ten Additional Output Fields selected in the [Input File tab](#) for this address. These can be used as extra information to help find the DPID and fix the address.
- Accept** Find the correct address then click to accept it as correct. This copies the record to the output file with the address fields, DPID and/or barcode updated, and goes to the next record.
- Skip** If unable to find the correct address, click to skip the current address and copy the record to the output file as is without DPID.
- Stop** Pause manual corrections, which can later be resumed at the same record by clicking the **Resume** button on the Manual Fix Address tab.

Resume

Click to open the [Rapid Tab](#) at the address being manual fixed when the **Stop** button was clicked. You can also exit Easy Post and continue manually fixing records if the processed data file has remained unchanged as follows:

1. Click Stop in the [Rapid Tab](#).
2. Exit Easy Post.
3. Start Easy Post when you want to resume manual fixing.
4. Click the Manual Fix Address tab. A prompt is displayed:
Output file: <name> already exists.
Click <YES> to replace it
Click <NO> to continue from where you left off.
5. Click No to open the [Rapid Tab](#) at the last record being checked.

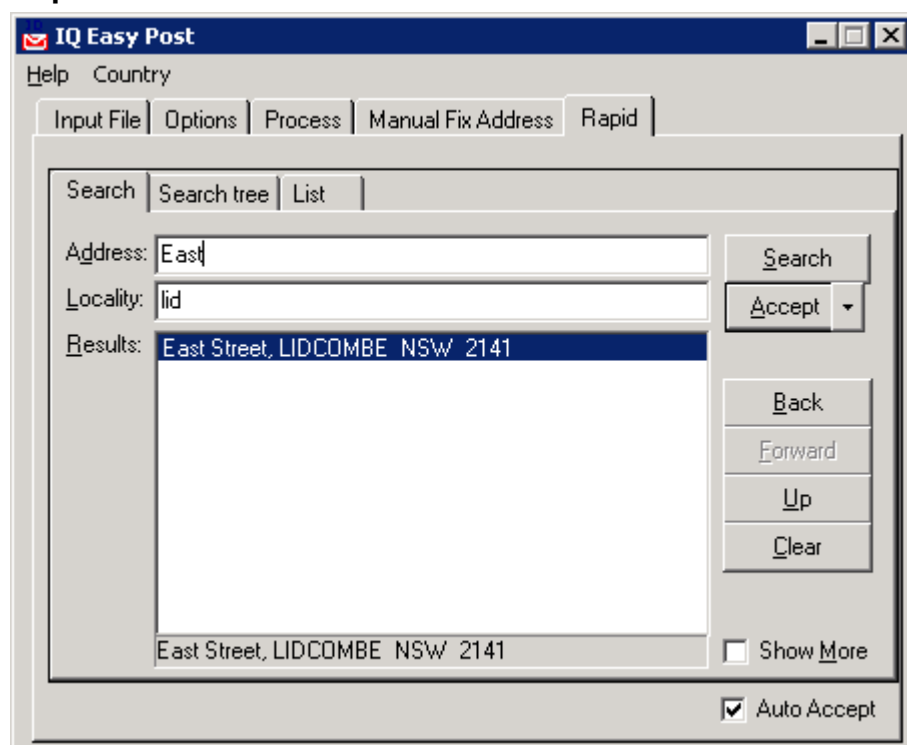
Create Reports

This button has the same function as in the [Process tab](#). Click to generate lodgement and summary reports after adjusting the records. These reports will indicate how many records were amended through the manual fix address process.

View amended Reports

This panel has the same function as in the [Process tab](#). Click a link in this panel to view the directory or report. The viewing program depends on the file extension.

Rapid Tab



Use this tab to search the Australia Post Postal Address File (PAF) using the [Rapid Address search functions](#), ensure that addresses are valid, and find DPIDs.

The **Show/Hide**, **Skip** and **Stop** buttons are only displayed in the [Manual Fix Address tab](#).

Rapid Address search functions

A summary of the Rapid Address search functions is provided below. Refer to the Rapid Address User Manual for complete details of searching and configuration settings.

Search tab

Find addresses by typing in some details of the street address in the Address line, and some details of the locality, postcode or state in the Locality line, and press Enter to display all possible matching addresses. Select an address or partial address from the list, then press Enter to narrow down the search until the full address is found

Search Tree tab

Find addresses by expanding and collapsing a hierarchical tree view of locality, street, property, level and unit number.

List tab

Find addresses by choosing the locality, then the street, then the property number, and finally a unit or level number if applicable. As you type, choices are displayed in the Results panel, which can then be selected, saving the need to type the full word.

Address line

Type partial or full details of the following address information: Street address, Lot number, Unit or level, Postal box, bag or address, Building name. Type any unit or level numbers in the address before the house number. Street spellings will be corrected.

Locality line

Type partial or full details of the following locality information: Locality name, Postcode, state abbreviation.

The locality name tolerates one letter spelling errors. You can type the first few letters of words in the locality, such as `Syd A` for `Sydney Airport`, according to the rules below:

- Each word beginning must be in the locality for it to be found, but you don't need to type all the word beginnings. For example, `syd` will display `Sydney`, `Sydenham`, `Sydney Markets`, `North Sydney` etc, but `n syd` will find `North Sydney` but not `Sydney`.
- The word beginnings must be in order, e.g. `sy m` will display `Sydney Markets`, but `m sy` will not.
- Abbreviations (N, S, E and W) are recognised and need not be in order, e.g. both `me n` and `n me` will display both `North Melbourne` and `Merah North`.
- State – Enter a state abbreviation to only find only addresses in the state, depending on the selected options.
- No locality or postcode – Enter either nothing or only a state to search all localities or all localities in the state.
- Locality or postcode – Enter a postcode or part of a locality name to search only addresses matching the locality.
- Locality and postcode – Enter both a postcode and some locality name information to search all localities matching either the name or the postcode, with a preference to those localities that match both.
- Exact matching – Enter all locality elements enclosed in square brackets to find an exact match, e.g. `[SYDNEY 2000 NSW]` will only find `SYDNEY NSW 2000`. Without brackets this will find all localities containing the word “SYDNEY” such as “`NORTH SYDNEY`”, as well as all localities with postcode 2000.

Popup List

A popup hint box listing addresses or localities matching any partial information entered into the address or locality line can be automatically displayed after a defined number of characters have been typed. Press Ctrl-Space to manually show the list at any time.

ID line

Displays ID, DPID and additional fields.

Results / Address panel

Displays found addresses.

Buttons

Use the buttons to search and navigate through addresses.

Search	Search for information entered in the Address line and Locality line.
Accept	Zoom to address details or accept address.
Back	Returns to previous search results. Same as pressing Esc key.
Forward	Goes forward through search results.

Up	Goes up through locality and address tree
Clear	Clears the address line, locality line and all found results, enabling another search.
Stop Search	Displayed during a long search. Click to cancel the search.

Show More checkbox

Check this box to display bordering or synonym localities, alternate street names and border synonyms matched to addresses as permitted under the 2004 AMAS rules. This can vastly increase the number of results returned for a partial address match.

Clear this box to display only addresses matching the search criteria

Auto Accept

Check to automatically accept but not display only one final found address.

Using IQ Easy Post (NZ)

Use Easy Post by filling in each tab in the order below. To select the next tab either click the tab name at the top of the screen, or the Next button at the bottom of the screen. If an item dependency is missed or not valid you will be unable to move to the next tab until valid options have been selected.

1. Start Easy Post, see [Starting and Exiting](#).
2. Select New Zealand from the Country menu.
3. Use the [Input File tab](#) to select the input data file and fields to process, then the [Input Text File Format screen](#) to define the format of the input fields. File formats can be loaded or saved using the [Saved Text File Formats](#) dialog box. Either click the Quick Start button to process addresses, or proceed to the other tabs.
4. Use the [Options tab](#) to specify details of the customer to appear on the [SOA](#), and define the location and name of the output files, then the [Output Text File Format screen](#) to define the format of the output fields.
5. Use the [Process tab](#) to process the addresses in the input file and generate the output file.
6. Use the [Process tab](#) to manually fix invalid addresses.
7. Use the [SOA Tab](#) to select the file on which the [SOA](#) is based, generate statistics from processed addresses, view the [Issued SOA](#), and generate the SOA.
8. Use the [Rapid tab](#) at any time to search for addresses in the PAF.

NZ Input File tab

The New Zealand tab is the same as the Australian [Input File tab](#).

NZ Options tab

Use this tab to specify details of the customer to appear on the [SOA](#), and the text files into which processed and [Manual Fix Addresses](#) are placed. Files which do not exist are created, or are overwritten if they exist. Use an obvious name for the file.

Advanced options

Check this option to [Manual Fix Addresses](#) that could not be automatically matched, and choose the file on which to base the SOA in the [SOA Tab](#).

Processed File

Click the **Browse** button and browse to the name of the output file. The [Output Text File Format screen](#) is then displayed to specify the file format.

Fixed File

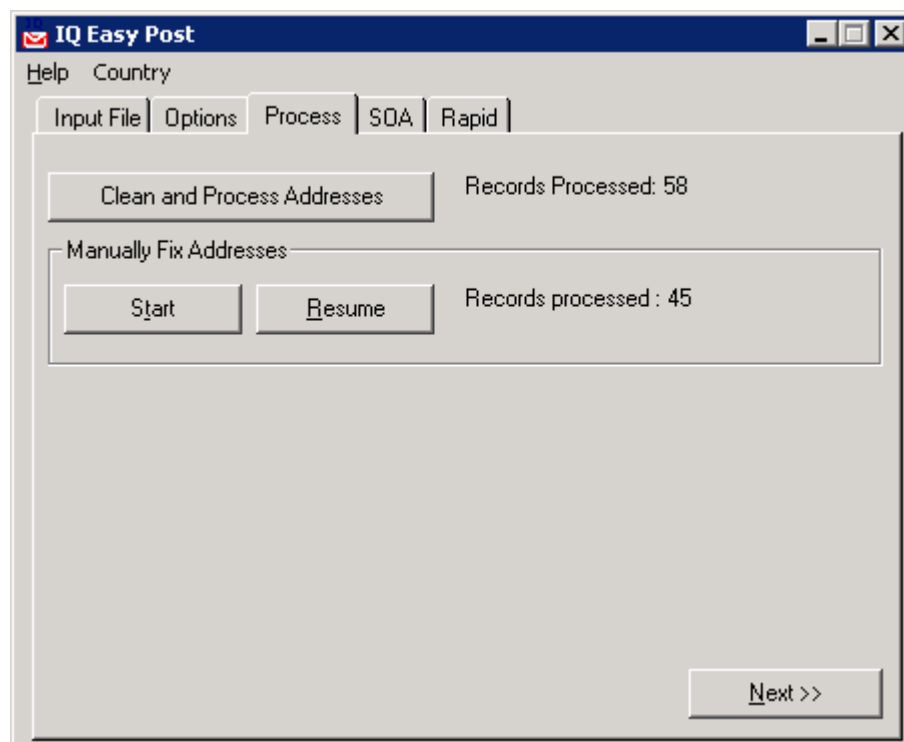
This option is only available if Advanced options is checked. Click the **Browse** button and browse to the name of the file used to store manually fixed addresses. This uses the same output format settings as the processed address file.

Customer information

Enter the details of the customer to appear on the [Issued SOA](#) (Statement of Accuracy) certificate. Click the Edit button to open a dialog box to enter the following fields.

Name	Customer name
NZP Number	Customer's New Zealand Post number
Address	Customer address

NZ Process tab



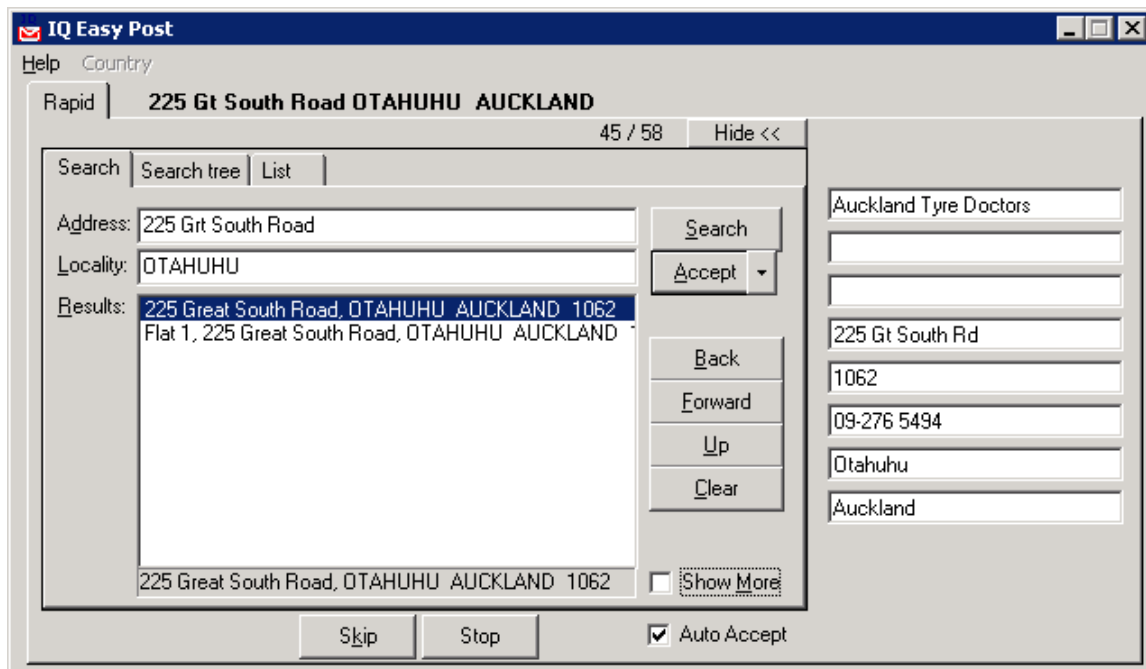
Use this tab to process the addresses in the input file to generate the output file. The Start and Resume button are unavailable if addresses have not yet been processed.

Clean and Process Addresses

Click this button to process the addresses and create the output files selected in the [Options tab](#).

Start

Click to open a tab that is similar to the [Rapid tab](#). This displays individual addresses that were not allocated AddressIds during address processing, enabling each record to be manually corrected using [Manual Fix Address](#). The current and total number of records processed is displayed in the top right of the tab.



Search for the correct address and accept or skip it using the following buttons.

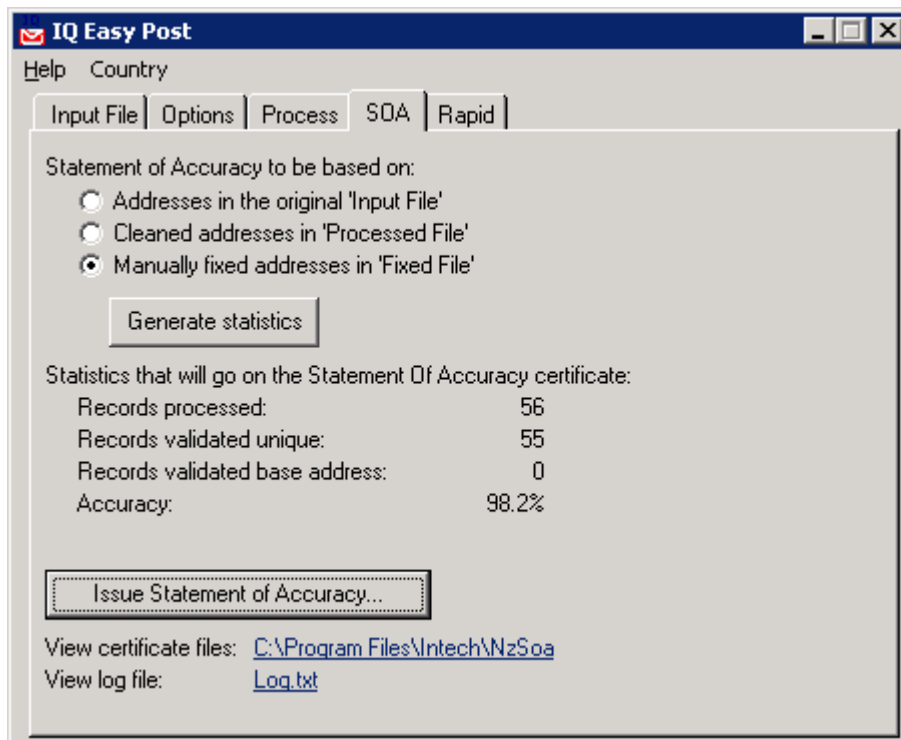
- Show / Hide** Click to show or hide up to ten Additional Output Fields selected in the [Input File tab](#) for this address. These can be used as extra information to help find the AddressId and fix the address.
- Accept** Find the correct address then click to accept it as correct. This copies the record to the output file with the address fields and AddressId updated, and goes to the next record.
- Skip** If unable to find the correct address, click to skip the current address and copy the record to the output file as is without AddressId.
- Stop** Pause manual corrections, which can later be resumed at the same record by clicking the **Resume** button on the [Process tab](#).

Resume

Click to open the [Process tab](#) at the address being manual fixed when the **Stop** button was clicked. You can also exit Easy Post and continue manually fixing records if the processed data file has remained unchanged as follows:

1. Click Stop in the [Process tab](#).
2. Exit Easy Post.
3. Start Easy Post when you want to resume manual fixing.
4. Click the Process tab. A prompt is displayed:
Output file: <name> already exists.
Click <YES> to replace it
Click <NO> to continue from where you left off.
5. Click No to open the [Process tab](#) at the last record being checked.

NZ SOA Tab



Use this tab to select the file on which the Statement of Accuracy is based, generate statistics from processed addresses, and view and generate the SOA.

Statement of Accuracy to be based on

Select the file on which the Statement of Accuracy is based, which is also the file used to print the addresses on the envelopes.

Input File	Raw address list specified in the Input File tab .
Processed File	Processed address list specified in the Options tab .
Fixed File	Manually fixed address list specified in the Options tab .

Generate statistics

Click to generate SOA statistics on the selected file.

Statistics that will go on the SOA certificate

This panel displays the number of processed and validated address records and percent accuracy.

Issue Statement of Accuracy

Click to display the SOA information in the [Issued SOA](#) panel before it is issued.

View certificate files	Click the link to open an Explorer window at the directory containing the text file of each issued SOA certificate, and the log file.
View log file	Click the link to open the text log file listing a summary of all SOA certificates issued.

Issued SOA

Statement Of Accuracy

SOA Issuer: **Intech Solutions Pty Ltd**
Software Name: **IQ Office**
Software Version: **5.4.54.1**
PAF Version Used: **PAF2_V2010Q2V01**
SOA Unique Reference: **INT10_ZQ3H4B03**

Customer Name: **Intech Solutions Pty Ltd**
Customer NZP No: **999**
Customer Address: **Level 7
35 Spring St
Bondi Junction
NSW 2022**

Customer Database: **NZAddressInput**

Date SOA Issued: **03/11/2010**
Date SOA Expires: **03/11/2011**

Number of address records -
Processed: **56**
Validated - Unique: **55**
Validated - Base Address: **0**
Total Validated: **55**
= Address Accuracy **98.2 %**

SOA Directory: <C:\Program Files\Intech\NzSoa>
Log File: <Log.txt>
Statement File: 0982_20111103_INT10_ZQ3H4B03.txt

This panel displays the IQ Easy Post software version, the PAF version, customer information entered in the [Options tab](#), and statistics generated from the [SOA Tab](#), which all go on the SOA certificate.

Edit

Click a button to edit the customer information. These fields are the same as those entered in the [Options tab](#).

Issue Statement

Click to issue and write the SOA to a text file, and update the log file.

- | | |
|----------------|---|
| SOA Directory | Click the link to open an Explorer window at the directory containing the text file of each issued SOA certificate, and the log file. |
| Log File | Click the link to open the text log file listing a summary of all SOA certificates issued. |
| Statement File | Click the link to view the issued SOA certificate as a text file. |

NZ Rapid tab

The screenshot shows the 'IQ Easy Post' application window with the 'Rapid' tab selected. The interface includes a menu bar with 'Help' and 'Country', and a tab bar with 'Input File', 'Options', 'Process', 'SOA', and 'Rapid'. Inside the 'Rapid' tab, there is a 'Search' section with 'Search tree' and 'List' sub-tabs. Below this, there are input fields for 'Address:' (containing 'great') and 'Locality:' (containing 'ota'). To the right of these fields are 'Search' and 'Accept' buttons. Below the input fields is a 'Results:' list box containing three entries: 'Great North Road, OTAMATEA WANGANUI 4501', 'Great North Road, OTAMATEA WANGANUI 4500', and 'Great South Road, OTAHUHU AUCKLAND 1062'. The third entry is selected. To the right of the list box are buttons for 'Back', 'Forward', 'Up', and 'Clear'. Below the list box is a 'Show More' checkbox. At the bottom right of the window is an 'Auto Accept' checkbox which is checked.

Use this tab to search the New Zealand Post Postal Address File (PAF) using the [Rapid Address search functions](#), ensure that addresses are valid, and find the AddressIDs.

The **Show/Hide**, **Skip** and **Stop** buttons are only displayed in the [Process tab](#).

IQ Rapid Address

Overview

IQ Rapid Address is a high performance, deterministic address matching application that enables rapid and efficient searching of Address Reference Files (ARF) to quickly and accurately capture, validate, and standardise addresses found from partial input data.

IQ Office supplies Address Reference Files in compressed and indexed format to optimise search speed, which may include the following depending on the IQ Office Edition and licence:

- The Australia Post Postal Address File (PAF), which contains all postal delivery addresses in Australia.
- The equivalent New Zealand Post PAF containing all New Zealand postal delivery addresses.
- The Australian Geocoded National Address File (G-NAF), the authoritative geocoded address index for Australian street addresses, containing state, suburb, street, number and coordinate reference or geocode.

Searching a Postal Address File enhances the integrity of customer data by returning an address that is both valid in content (accuracy) and format (standard), whilst reducing data entry costs. IQ Rapid Address has the following powerful address searching features:

- Matching on partial input of all address fields including street name, lot number, locality and postcode, minimising the number of keystrokes required to find a valid address. For example, `SP BON` finds five matching streets in the whole of Australia, `SPR BON` two, and `SPR BOND` or `SP BOND` only one, Spring St Bondi Junction, the Intech Solutions office address.
- Search on property names including schools, hospitals and other institutions.
- Search on all postal addresses including postal boxes, bags and delivery addresses.
- Automatic correction of misspelt address fields, e.g. `COLE` finds `COLES BAY`, `MOUNT COLE`, and `COLEAMBALLY` as well as `COLO` and `TOLL`, enabling difficult spellings to be quickly captured and validated
- Flexible locality name matching on multiple words and word orders, e.g. `E MA` finds `EAST MAITLAND` and `MALVERN EAST`
- Look-ahead returns all partial matches enabling choices to be rapidly narrowed down the final address. Automatic popup list shows addresses and localities matching partial input information.
- Optional display of border and synonym localities and alternate street names.
- Rapid navigation forward and backwards through found address results by button or keystroke using tree or single line view.
- Integration with other applications via the Windows clipboard, append or write to file, or keystrokes sent to another application. Background operation and pop-up on hotkey enables address data to be passed from another application to Rapid Address.
- Return of DPID, GroupDID or LocalityDID corresponding to address, and search by the same identifiers.

In addition to postal address validation, IQ Rapid Address can provide powerful and accurate geographic search and coding capabilities using the Australian Geocoded National Address File (G-NAF). A found address can return corresponding geographical information such as the latitude and longitude of the address, and Australian Bureau of Statistics (ABS) statistical (census) regions such as Mesh Block, Collection District or Statistical Local Area in which the address is located. Returned latitude and

longitude can be passed directly to a web-based mapping service via a link to a browser, enabling one-click visualisation and zooming to the found address.

The same geographical information can be used to search the G-NAF, for example, to find all addresses located with a specified radius of a given latitude and longitude, or in an ABS statistical region.

Using IQ Rapid Address

Starting and Exiting

Starting

To start IQ Rapid Address, select it from the Intech program menu, double-click the desktop icon, or run it from the command line with an optional parameter specifying an [Advanced Options](#) configuration. If no parameter is specified the default configuration is used.

To start IQ Rapid Address from the command line, type:

```
iRapid -cConfiguration
```

Where *Configuration* is the name of the configuration. For example, to start IQ Rapid Address with "PowerUser" as the current configuration:

```
iRapid -cPowerUser
```

Exiting

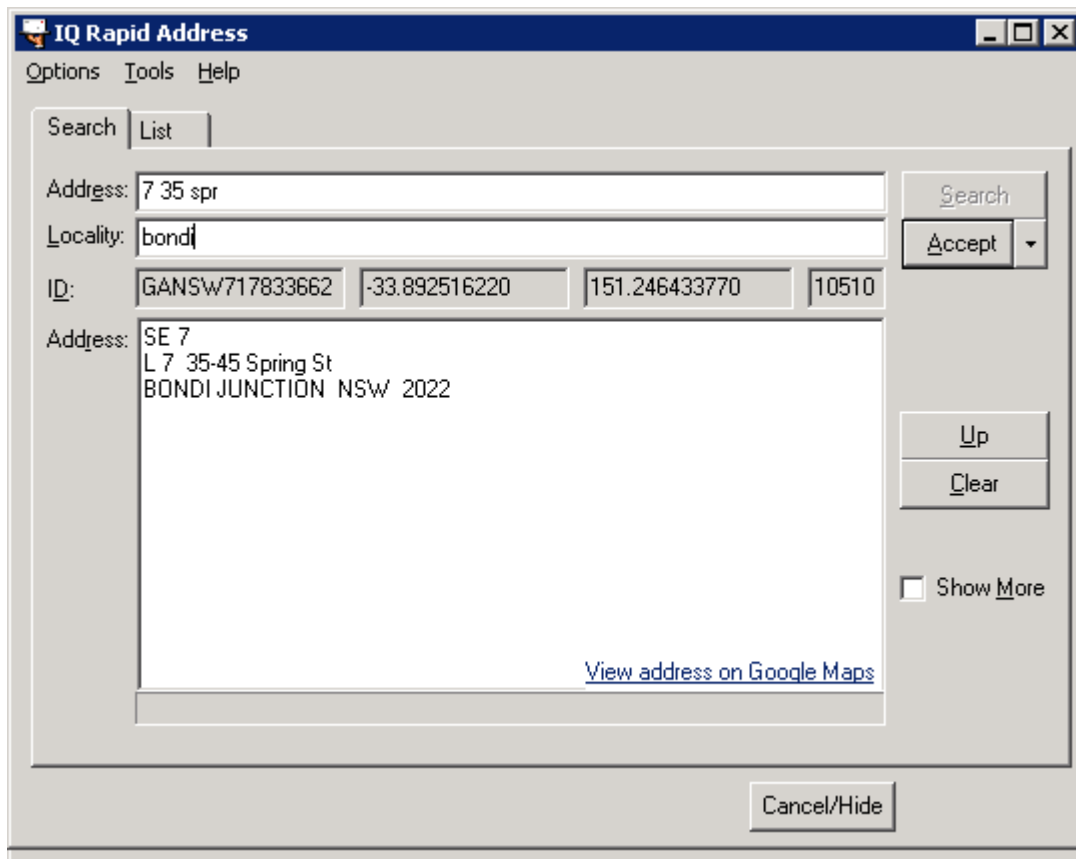
To exit IQ Rapid Address, click the X button in the top right of the window or press Alt+F4. It will not close but keep running in the background if the **Hide on Close** [Hiding options](#) has been specified.

Restoring

If the appropriate [Hiding options](#) have been specified, IQ Rapid Address keeps running when it is closed, with or without an icon in the taskbar.

To restore IQ Rapid Address, either press the defined [Application Hot Key](#), or double-click the Taskbar icon. Different hotkeys can be defined for different configurations defined using the [Advanced Options](#).

Rapid Address Window



The IQ Rapid Address window has the following features.

Feature	Use
Menus	Set General Options , Search Options and Advanced Options , and Selecting an ARF .
Search tab	Find addresses by the Search address method
List tab	Find addresses by the List address method
Address line	Enter street and postal address data.
Locality line	Enter locality, state and postcode data.
ID line	Displays ID, DPID and additional fields.
Popup List	Select list of partial matches from Address or Locality line.
Results panel	Displays found addresses.
Buttons	Search and navigate through found addresses.
Keys	Search and navigate through found addresses.

Menus

IQ Rapid Address window has the following menus.

Options

General	General Options for defining colours, formatting and hiding the window.
Search	Search Options for defining search behaviour.
Advanced	Advanced Options for defining hotkeys.

ARF For [Selecting an ARF](#) to be used. The options in this menu depend on the licensed ARFs, and configurations defined in [Advanced Options](#).

Tools

Find by AddressID For [Finding addresses by Address ID](#).

Find by GIF For [Finding addresses by GIF](#). This option is only available if an ARF with Geographical Information has been selected.

Help

Contents Displays the on-line Help at the Contents page.

About Displays application, PAF, licence and contact details.

Buttons

IQ Rapid Address has the following buttons.

Search Searches for information entered in the [Address line](#) and [Locality line](#).

Accept Zoom to address details or accept address (see details below).

Back Returns to previous search results. Same as pressing Esc key.

Forward Goes forward through search results.

Up Goes up through locality and address tree.

Clear Clears the address line, locality line and all found results, enabling another search.

Stop Search Displayed during a long search. Click to cancel the search.

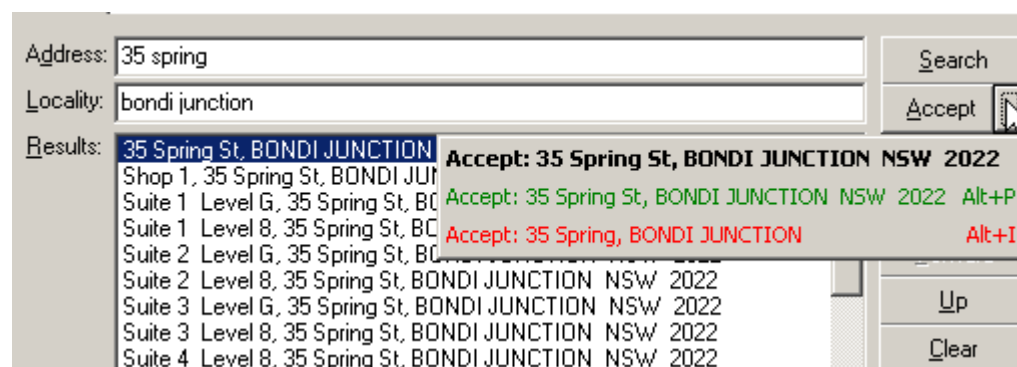
Cancel/Hide Hides the screen. To show the screen again, press the defined [Application Hot Key](#) if the [This application can be hidden](#) option has been selected.

Accept button

Click the Accept button to accept the highlighted full or partial address in the list and narrow down the search until a full address can be found. An incomplete address can be accepted if the relevant [General Options](#) have been selected.

Click the Accept button when a full address has been found to paste the address into another application, [Send Keystrokes to Window](#), [Copy address to Clipboard](#), or [Write address to File](#) if the relevant options have been set.

Click the arrow to the right of the Accept button to display a drop down list showing the default address that will be chosen, the partial address based on the chosen items, and the entered input address.



Keys

Use the following keystrokes in IQ Rapid Address.

Enter Search.

	Accept highlighted item
Esc	Clear last entered information or choice (return to previous search results)
	Clear the screen.
	Go up one level on the List tab
	Hide the screen (if Hide when pressing <Esc> on a clear screen option selected)
	Hide the Popup List
Ctrl-Space	Show the Popup List
Ctrl-Enter	Create new line when Full address found
Tab	Move between fields

Use the following keys to move through the list of found addresses in the Result panel while retaining the cursor in the Address or Locality line.

Up arrow	Previous item
Down arrow	Next item
PageUp	Previous page
PageDown	Next page
Ctrl-Home	First item in list
Ctrl-End	Last item in list

Use the standard Windows clipboard keystrokes to copy, cut and paste text to and from the Address or Locality lines.

Ctrl-C	Copy text
Ctrl-X	Cut text
Ctrl-V	Paste text

The following keystroke options can also be defined in Rapid Address:

- [Application Hot Key](#) which pops-up IQ Rapid Address from another application.
- [Send Keystrokes to Window](#) to automatically type a found and accepted address into another application.
- Shortcut keystrokes to accept partial selection or input, see [General Options](#).

Address line

Type partial or full details of the following address information in the Address line:

- Street address
- Lot number
- Unit or level
- Postal box, bag or address
- Building name

Locality line

Type partial or full details of the following locality information in the Locality line:

- Locality name, e.g. n syd, n pe, parra, mount ma

- Postcode, e.g. 2037, 2000
- State abbreviation without punctuation, e.g. NSW

Locality Name Matching

The locality name tolerates one letter spelling errors.

You can type the first few letters of words in the locality, such as Syd A for Sydney Airport, according to the rules below:

- Each word beginning must be in the locality for it to be found, but you don't need to type all the word beginnings. For example, syd will display Sydney, Sydenham, Sydney Markets, North Sydney etc, but n syd will find North Sydney but not Sydney.
- The word beginnings must be in order, e.g. sy m will display Sydney Markets, but m sy will not.
- Abbreviations (N, S, E and W) are recognised and need not be in order, e.g. both me n and n me will display both North Melbourne and Merah North.

State

Enter a state abbreviation to only find only addresses in the state, depending on the [Only search in State provided](#) options.

No locality or postcode

Enter either nothing or only a state to search all localities or all localities in the state.

Locality or postcode

Enter a postcode or part of a locality name to search only addresses matching the locality.

Locality and postcode

Enter both a postcode and some locality name information to search all localities matching either the name or the postcode, with a preference to those localities that match both.

Exact matching

Enter all locality elements enclosed in square brackets to find an exact match, e.g. [SYDNEY 2000 NSW] will only find SYDNEY NSW 2000. Without brackets this will find all localities containing the word "SYDNEY" such as "NORTH SYDNEY", as well as all localities with postcode 2000.

ID line

This line displays ID, DPID and additional fields if [Additional Field Options](#) have been defined for [Geographical Information Lookup](#). Hover the pointer over a field to display the field name.

Results panel

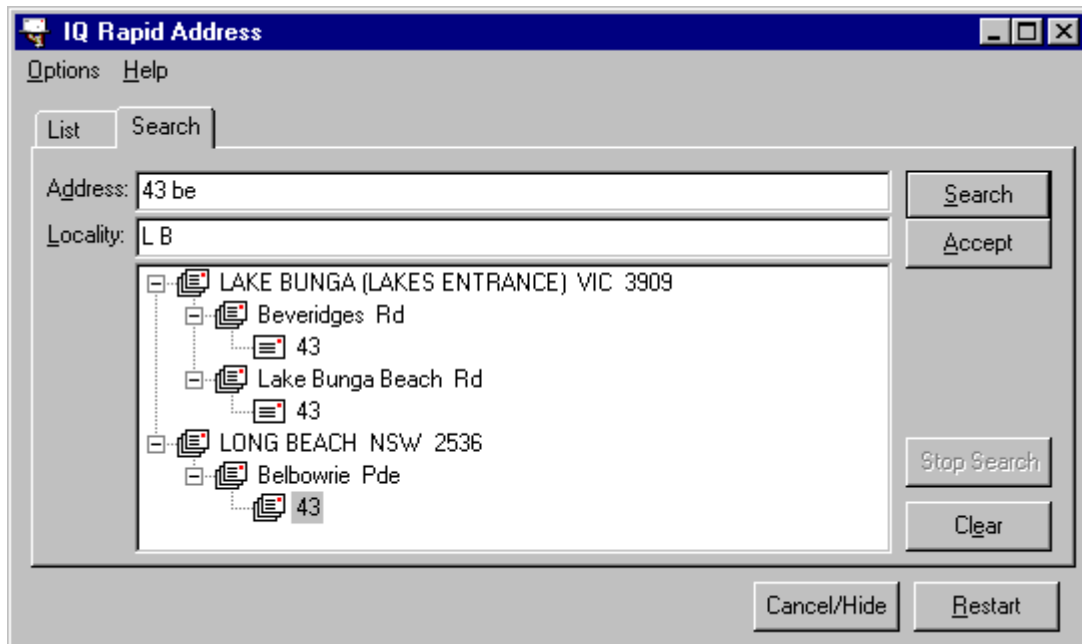
Found address data is displayed in the Results Tab in the centre of the Rapid Address window. Results are either displayed one per line in a text view, or as a tree structure in tree view, depending on the [Search options](#) that has been selected.

Text View

In this view, addresses are displayed one per line.

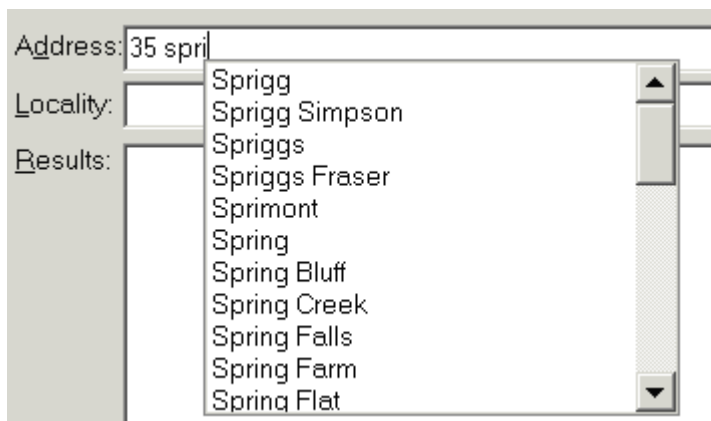
Tree View

In this view, addresses are displayed as a hierarchy or tree of locality, street, property, level and unit number.



Popup List

Depending on the selected [Popup Hint Options](#), a popup hint box listing addresses or localities matching any partial information entered into the address or locality line can be automatically displayed after a defined number of characters have been typed. Press Ctrl-Space to manually show the list at any time.



To select an item from the list and place it in the address or locality line, either click on it, or use the up and down arrow keys to highlight the item then press Enter.

To refine the popup list, continue to type additional characters, or press Esc key to hide the popup list from view until further characters have been typed.

Finding addresses

Use the following methods to search for addresses:

- Use the [Search address method](#) to find any partial address and locality information.
- Use the [List address method](#) to find the locality, then street, property number.
- Enter a specific Address ID for [Finding addresses by Address ID](#).
- Enter geographical information for [Finding addresses by GIF](#).

Auto Search

If the [Auto Search Options](#) is selected, a search is performed after the minimum numbers of characters has been typed.

Show More checkbox

☒ Show More

Check this box to display bordering or synonym localities, alternate street names and border synonyms matched to addresses as permitted under the AMAS rules. This can vastly increase the number of results returned for a partial address match. These additional matches are displayed in defined [Colours](#).

Clear this checkbox to display only addresses matching the search criteria.

Search address method

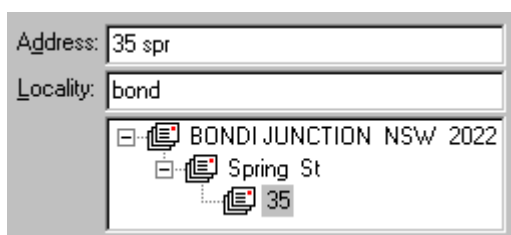
To use the Search method, type in some details of the street address in the [Address line](#), and some details of the locality, postcode or state in the [Locality line](#), and press Enter to display all possible matching addresses. Select an address or partial address from the list, using the [Keys](#) if required, then press Enter to narrow down the search until the full address is found.



Use one of the methods below according to the object of the search.

Street address

Type the house number and the first three or so letters of the street name in the Address line, and the first four or so letters of the locality or postcode in the locality line. For example, to search for 35 Spring St, BONDI JUNCTION, type 35 SPR in the address line, and BOND in the locality line.



Street name - uncertain spelling

Type the full street name, and the spelling will be automatically corrected, or use the [Popup List](#) to select the street name from a list.

Address:	bearwah
Locality:	dea
<div> DEAGON QLD 4017 Beerwah St </div>	

If street name contains more than one word, type the first few characters of each word.

Address:	a b p
Locality:	
<div> ARUNDEL QLD 4214 A B Patterson Dr </div>	

Street name - narrowing down a search

To limit the search to an exact street name type and suffix, enclose the address line in square brackets.

Address:	[SPENCE PL]
Locality:	ADEL
<div> ADELAIDE SA 5000 Spence Pl </div>	

Allotment numbers

To find a Lot number, type LOT followed by the allotment number

Address:	lot 3 alf
Locality:	kill
<div> Lot 3 Alfred Rd KILLABAKH NSW 2429 </div>	

Unit or Levels

Type any unit numbers or level numbers in the address before the house number in the address line. For example, to search for Level 7, 35 Spring St, type L7 35 SPR or LEVEL 7 35 SP in the address line.

Address:	L7 35 spr
Locality:	bond
<div> Level 7 35 Spring St BONDI JUNCTION NSW 2022 </div>	

To search for 5/30 Main Street, type 5/30 MAIN in the address line.

Address:	5/30 main
Locality:	G T
<div> 5/30 Main Rd GEORGE TOWN TAS 7253 </div>	

Postal address

Type the postal number in the address line, and either the postcode or the first four or so letters of the locality in the locality line. For example, to search for GPO Box 100, SYDNEY 2001, type 100 in the address line and 2001 in the locality line.

Address:	100
Locality:	2001
GPO BOX 100 SYDNEY NSW 2001	

Postal address - narrowing down a search

To narrow down the search, enter the postal type, e.g. LOCKED BAG

Address:	locked bag 20
Locality:	PARR
LOCKED BAG 20 PARRAMATTA NSW 2124	

Include any letter prefix or suffix in the postal number, with no space between the number and the suffix.

Address:	2a
Locality:	melb
GPO BOX 2A MELBOURNE VIC 3001	

Building name

The [Enhanced search](#) option must be checked to search by building names. If the search takes too long and times out, try again or increase the [Search Time Limit](#). Building name searches with postcodes are much faster. Australia Post is currently adding more building names to the PAF so some building names are not yet in the PAF.

To search for a building name, type all or part of the building name in quotes in the address line, and any known postcode or locality in the locality line. For example, to search for Sydney Grammar School, DARLINGHURST, type "GRAMMAR" in the address line and DARL in the locality line.

Address:	"grammar"
Locality:	DARL
Sydney Grammar School 10-12 College St DARLINGHURST NSW 2010	

To find all grammar schools, type "gram school" in the address line. Searching without the quotes will also find matching street names, e.g. GRAMMAR will find Grammar Dr, Grammar St, etc.

Building name - all hospitals in a locality

Type HOSPITAL in the address line, and the locality or postcode in the locality line. To find all hospitals named "District Hospital", type "dist hosp" in the address line.

Address: hospital

Locality: chatswood

- CHATSWOOD NSW 2067
 - Wyvern Ave
 - 10 (Hirondelle Hospital)
 - Victoria Ave
 - 256 (Chatswood District Hospital)
 - Hercules St
 - 36 (Royal North Shore Hospital)

Building name - unknown locality

If you don't know the locality, you can still search for a building name, though it may take longer than usual. If you know the state, you can specify it, and set the [Only search in State provided](#) option.

Address: melrose school

Locality:

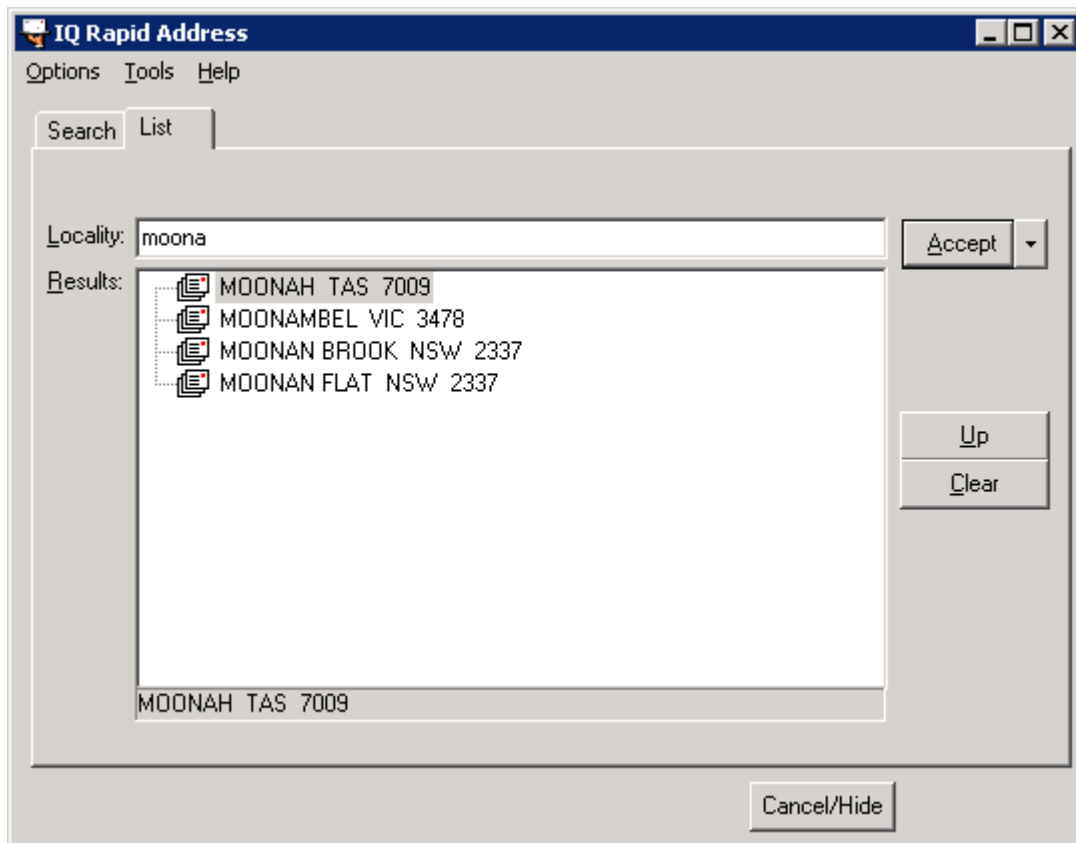
- ERMINGTON NSW 2115
 - Wharf Rd
 - 110 (Melrose Park Public School)
- ST MARYS SA 5042
 - Daws Rd
 - 139 (Melrose Park School)
- PEARCE ACT 2607
 - Marr St
 - 35 (Melrose High School)
- CHIFLEY ACT 2606
 - Eggleston Cres
 - 91 (Melrose Primary School)

List address method

To use the List method, choose the locality, then the street, then the property number, and finally a unit or level number if applicable. As you type, choices are displayed in the [Results panel](#), which can then be selected, saving the need to type the full word.

Entering the locality

Type the first four or so letters of the locality, or the full locality name, or the postcode in the [Locality line](#). To select a locality from the list, either double-click on it, or scroll to and press Enter. then select a locality from the displayed list and press Enter. A list of all streets in the locality is displayed.

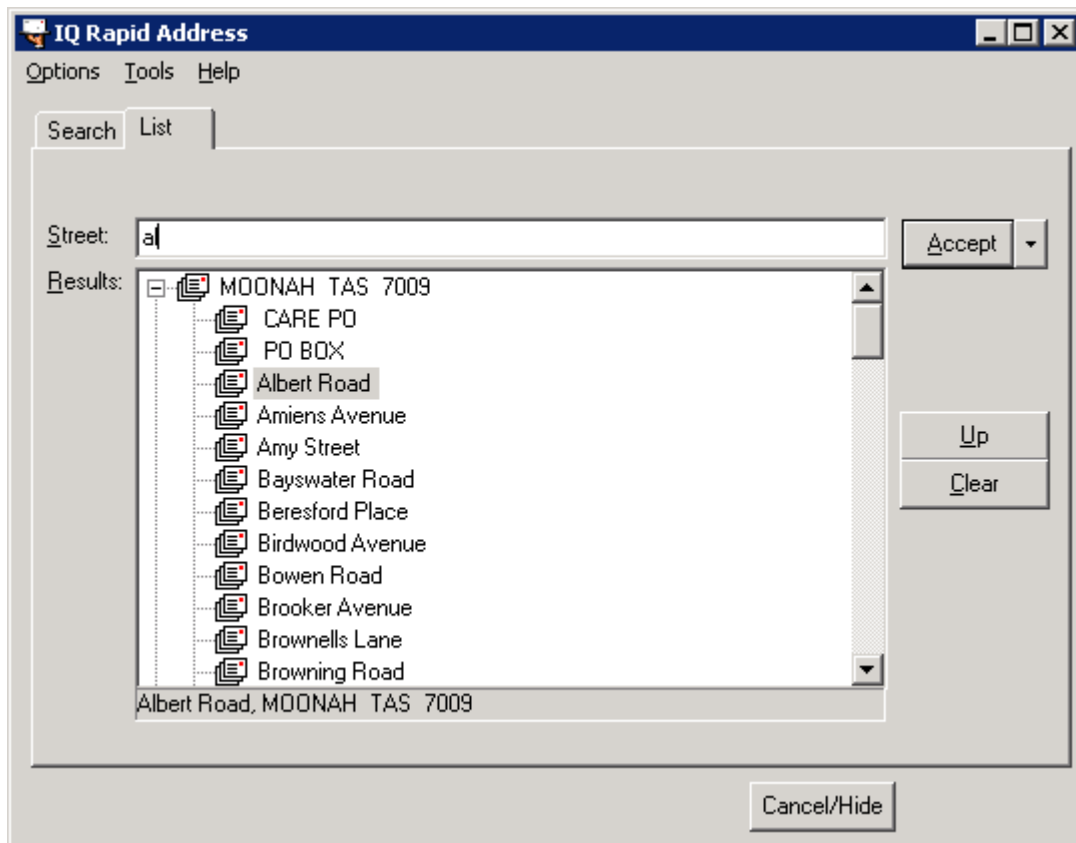


Synonym localities

Note that some localities are known by more than one name. One name is the official name, and the other names are recognised synonyms. When a locality is displayed under its synonym name, its official name will appear in parentheses. For example, the locality STIRLING SA 5152 is also called LONG GULLY. So when displayed under Long Gully, it will appear as LONG GULLY (STIRLING) SA 5152. If the postcode 5152 is entered, both will be displayed, although it is only one locality.

Entering the street

When the locality is chosen, a list of all streets in the locality is displayed. Type the first few letters of the street name to scroll to it in the list. To select a street from the list, either double-click on it, or scroll to it and press Enter. A list of all properties in the street is displayed.

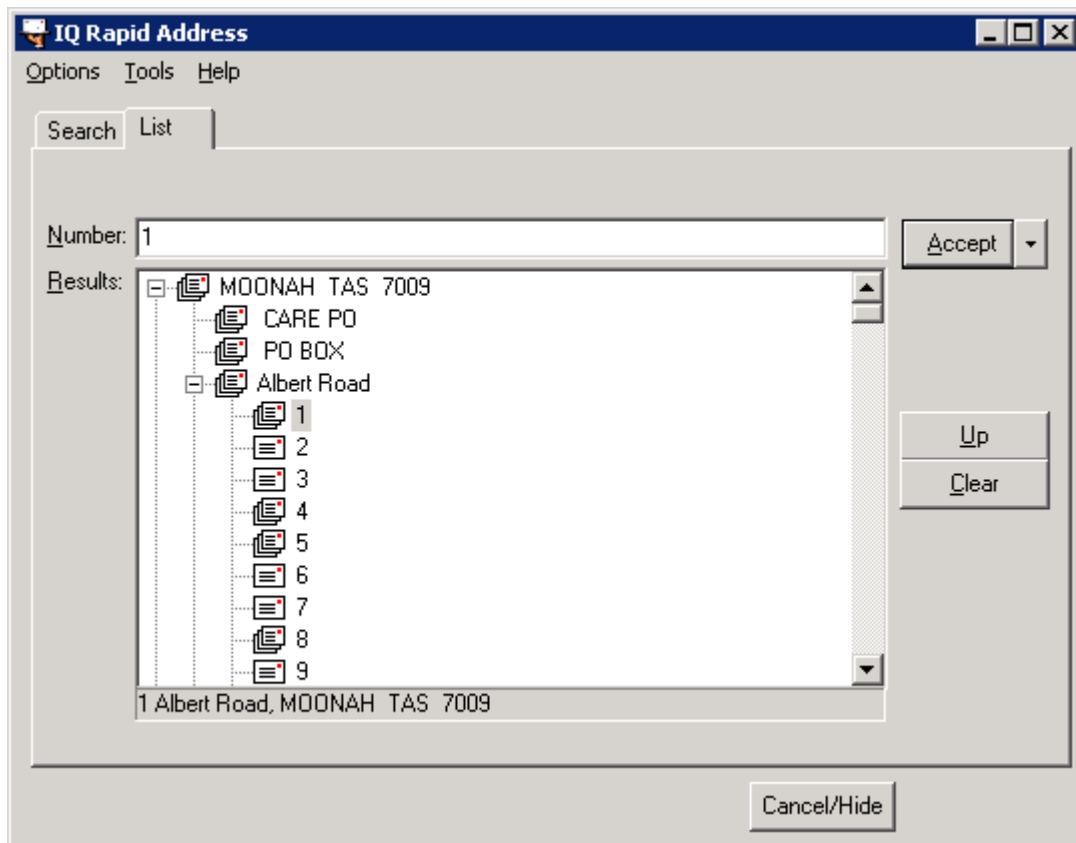


If the address is a postal address, not a street address, then select the appropriate postal type (e.g. PO BOX) from the list and press Enter.

Entering the property number

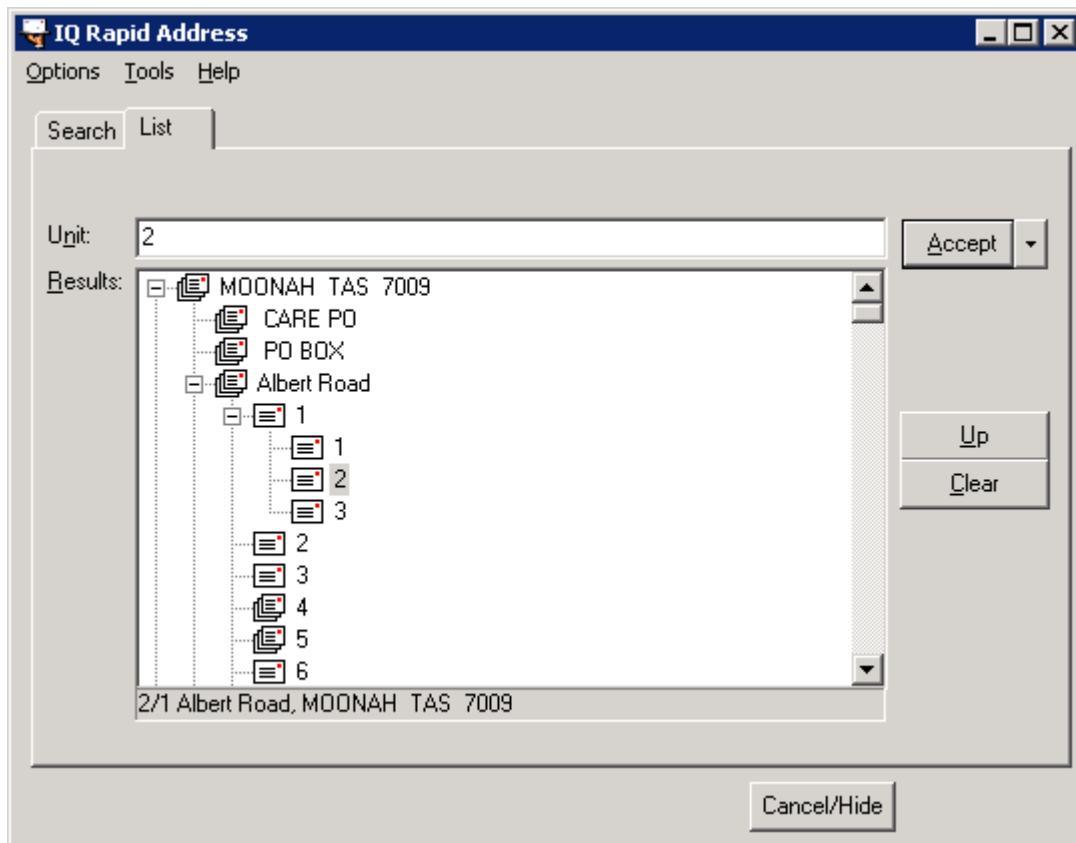
When the street is chosen, a list of properties in the street is displayed. Type the number of the property to scroll to it in the list. To select a property from the list, either double-click on it, or scroll to it and press Enter. A list of all units in the property is displayed.

If the address is not a street address, a list of postal numbers is displayed.



Entering the unit

When the property is chosen, a list of units in the property is displayed. Type the number of the unit to scroll to it in the list. To select a unit from the list, either double-click on it, or scroll to it and press Enter. The final address details are displayed.



If there are no units or levels at this address, or if it is a postal address, then this step will be automatically skipped.

If a list of units is displayed but your address does not have a unit, or the unit number is not in the list, reselect the address without the unit.

Note that there are some addresses in the PAF for which the address cannot be selected without a unit number, in AMAS terminology these are known as secondary points with no corresponding primary point.

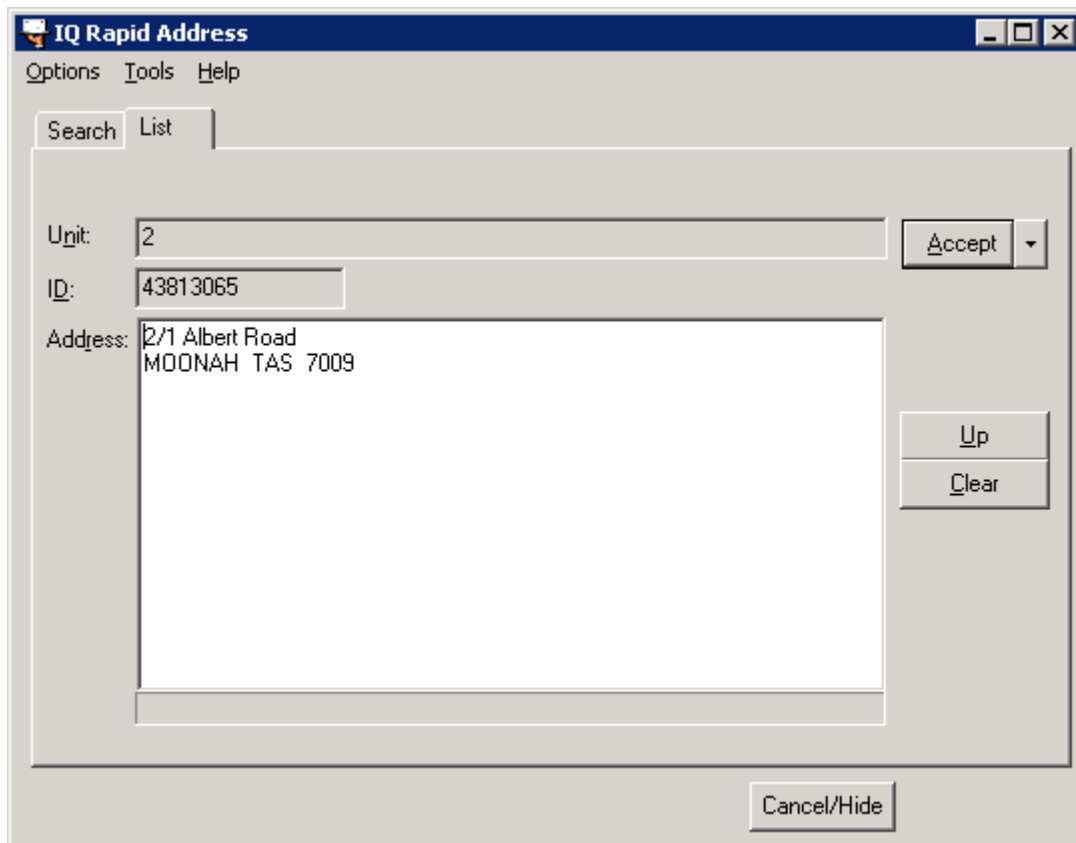
Secondary information

If the chosen address is a Primary Point, then the user will be warned with an appropriate message at the bottom of the window. The address box enables further unit or level information to be typed in. If there is no unit or level information, these can be left blank.

If the address is a Phantom Primary Point, then the status is displayed in red, indicating that unit or level information must be entered for the address to be deliverable.

Full address found

After all the full address has been selected, the address as it would appear on an envelope is displayed, along with the DPID, GroupDID or LocalityDID and barcode associated with the address.

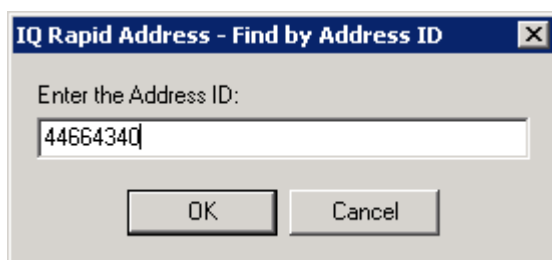


To enter other information with the address, click in the Address panel and press Ctrl-Enter, then type the additional information.

Click the Accept button when a full address has been found to paste the address into another application, [Send Keystrokes to Window](#), [Copy address to Clipboard](#), or [Write address to File](#) if the relevant options have been set.

Finding addresses by Address ID

To find an address by the unique ID or DPID, select **Find by AddressID** from the **Tools** menu and type the address ID in the dialog box, e.g. 44664340, then click the OK button. The found address is displayed, or an error message: No Address ID of "NNNNNN" found



Finding addresses by GIF

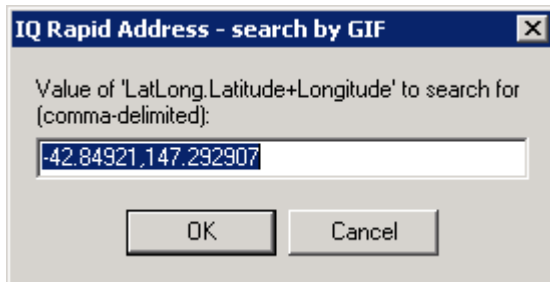
This option is only available if:

- An ARF with geographical information is in use (see [Selecting an ARF](#)).
- [Geographical Information Lookup](#) has been defined.
- [Additional Field Options](#) have been defined for the geographical information

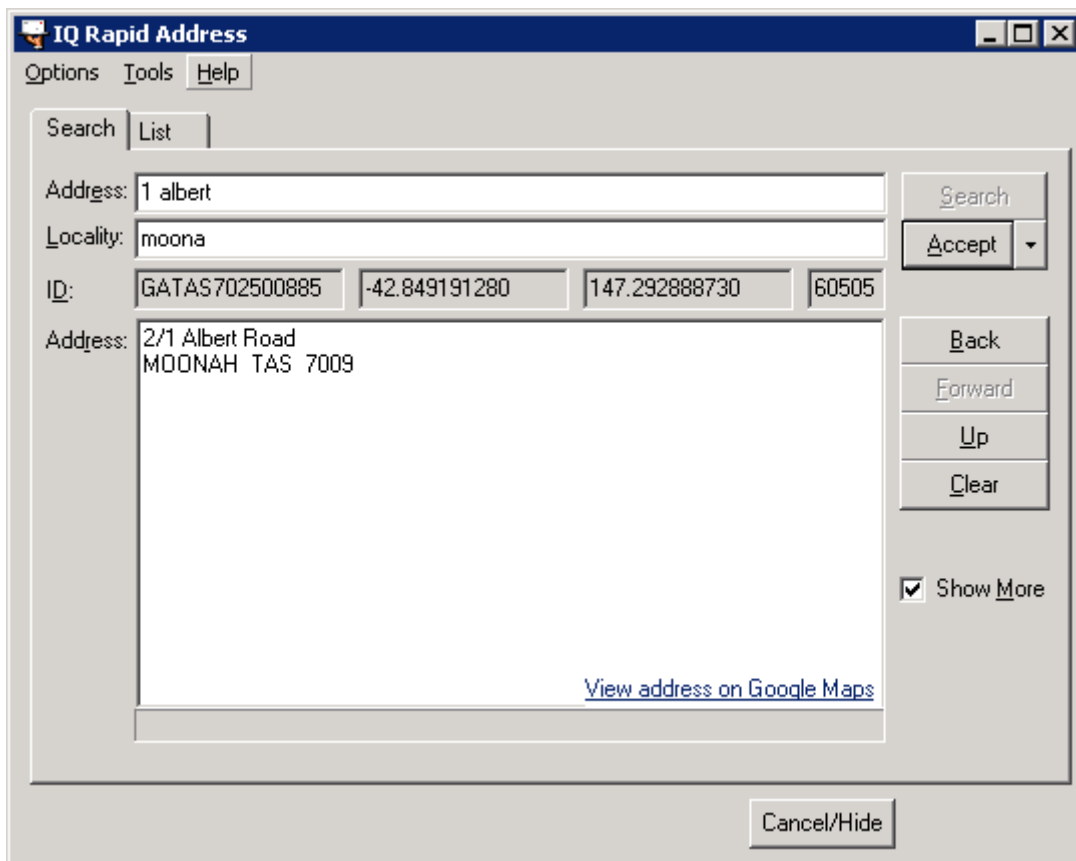
To find an address by geographical information:

1. Select **Find by GIF** from the **Tools** menu.

2. Select the geographical information fields from the sub-menu. These depend on the GIF in use, defined [Geographical Information Lookup](#) and [Additional Field Options](#).
3. Type the specified geographical information into the Search dialog box, for example, the Latitude and Longitude, e.g. -42.84921,147.292907, then click the OK button.



4. The Addresses found within the range defined in the [Geocode Search Options](#) are displayed in the Results panel, or an error message.
5. Select the required address to display the address including geographical information fields defined in the [Additional Field Options](#). In the example below the fields are Latitude, Longitude and SLA Code.



6. Click the **View address on Google Maps** link to open a browser at the geocoded location of the address.

Selecting an ARF

The address reference files which can be used to find address data depend on the Rapid Address licence. These may include the Australia Post Postal Address File (PAF), New Zealand Post PAF, and Australian Geocoded National Address File (GNAF).

To select an ARF, choose **ARF** from the **Options** menu, then choose the name of the ARF and any configuration defined in the Advanced Options [Configurations menu](#).

Passing address data between applications

IQ Rapid Address has powerful features which enable address data to be passed between it and other applications such as word processors, spreadsheets and databases.

For example, you might want to type an address in your database, word-processing or any other application, check the validity of this address, and perhaps also look up its DPID. In this case you can call IQ Rapid Address from the application to look up the address, then have IQ Rapid Address automatically put the found address into the application.

To set up IQ Rapid Address and the application:

1. Start IQ Rapid Address (select it from the Start menu).
2. Choose **Advanced** from the **Options** menu.
3. Define the [Application Hot Key](#) that will be pressed from the application to call IQ Rapid Address.
4. Select [Send Keystrokes to Window](#) and click **Format** to define the [Address Output Format](#) to be put into the application.
5. Switch to the application. When you are ready to put in the address, press the defined hot key to display IQ Rapid Address.
6. Search for the address and press **Accept** when found. The address will be pasted into your application.

How it works

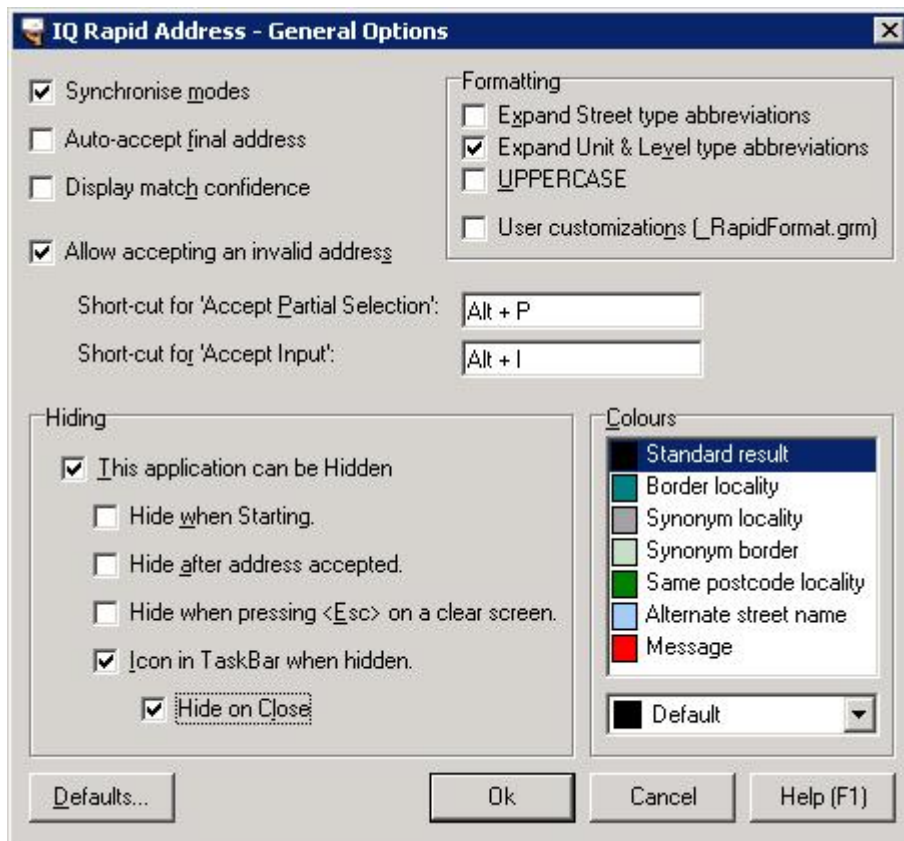
In this example the address is not actually pasted into your application, but the keystrokes are simulated in the selected application window. For example, typing "35 Happy Crescent" simulates the "3" key, then the "5", then a space, etc. Other keys such as "Enter" and "Tab" can be specified in the [Address Output Format](#) to enable the cursor to move between fields.

Because the keystrokes are simulated, this process will not work if a key is being held down. IQ Rapid Address will pause for up to 4 seconds if the Shift, Alt or Control keys are being pressed, before sending the keystrokes to the application.

Another alternative is to have specify that the formatted address is copied to the clipboard (see [Copy address to Clipboard](#)), then send the "Ctrl+V" key to the application to paste the address.

General Options

Select **General** from the **Options** menu to open the General Options window. These can be set when [Saving options](#) on a per configuration basis.



Synchronise modes

Check this option to copy found address information between the [List address method](#) tab and the [Search address method](#) tabs.

Auto-accept final address

Check this option not to display a found final address, but and return it as formatted in [Copy address to Clipboard](#), [Write address to File](#) or [Send Keystrokes to Window](#).

Clear this option to require the user to click the Accept button once the address had been displayed in its output format.

Display match confidence

Check this option to display a confidence score (0% - 100%) against each found address. This calculated against the each entered address component.

Allow accepting an invalid address option

Allow accepting an invalid address

Check this option to accept an address that has not been fully validated against the current ARF source.

Short-cut for 'Accept Partial Selection'

Short-cut for 'Accept Input'

To assign a shortcut key to accepting the partial selection or input, place the cursor in the key field and press the shortcut key combination, e.g. Alt + P, Ctrl + Shift + A. To clear the shortcut key, place the cursor in the key field and press Del or Backspace.

Formatting

Check these options to expand abbreviations and modify the output format of the addresses.

Expand Street type abbreviations

Check this option to expand the street type abbreviation to the full name, e.g. St to Street, Kings Hwy to Kings Highway, Hampton Gdns to Hampton Gardens.

Expand Unit & Level type abbreviations

Check this option to expand the Flat/Unit type abbreviation to the full name, e.g. Se 5 L 2 to Suite 5 Level 2.

UPPERCASE

Check this option to convert all output to upper case, e.g. Kings Highway to KINGS HIGHWAY

User Customizations (_RapidFormat.grm)

Check this option to use any customised output formats which have been defined in the special grammar file _RapidFormat.grm. Edit this file to change the displayed output of any address field prior to selection, for example, the street abbreviation of Avenue from AVE to AV.

Hiding options

Use these options to specify how IQ Rapid Address can be hidden. When hidden, it will not appear in the Taskbar unless the **Icon in TaskBar when hidden** option is checked.

To restore IQ Rapid Address back to view, press the Hot Key defined in the [Advanced Options](#), double-click the Taskbar icon, or start IQ Rapid Address again from the Windows start menu.

This application can be hidden

Check this option to hide IQ Rapid Address when the [Cancel/Hide](#) is clicked.

Hide when Starting

Check this option to start IQ Rapid Address in the hidden state.

Hide after address accepted

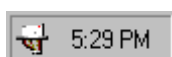
Check this option to hide IQ Rapid Address when an address is accepted by clicking the Accept button after an address with DPID is displayed.

Hide when pressing <Esc> on a clear screen

IQ Rapid Address normally clears the last entered information or choice made each time the Escape key is pressed. Check this option to hide IQ Rapid Address when the Esc key is pressed when the screen is empty.

Icon in TaskBar when hidden

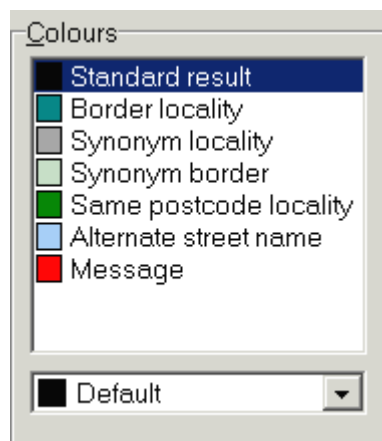
Check this option to display a small icon in the Taskbar when IQ Rapid Address is hidden. Double-click the icon to bring IQ Rapid Address back into view.



Hide on Close

Check this option to leave IQ Rapid Address running in the hidden state, not exit, when it is closed by clicking the X in the top right corner of the window.

Colours

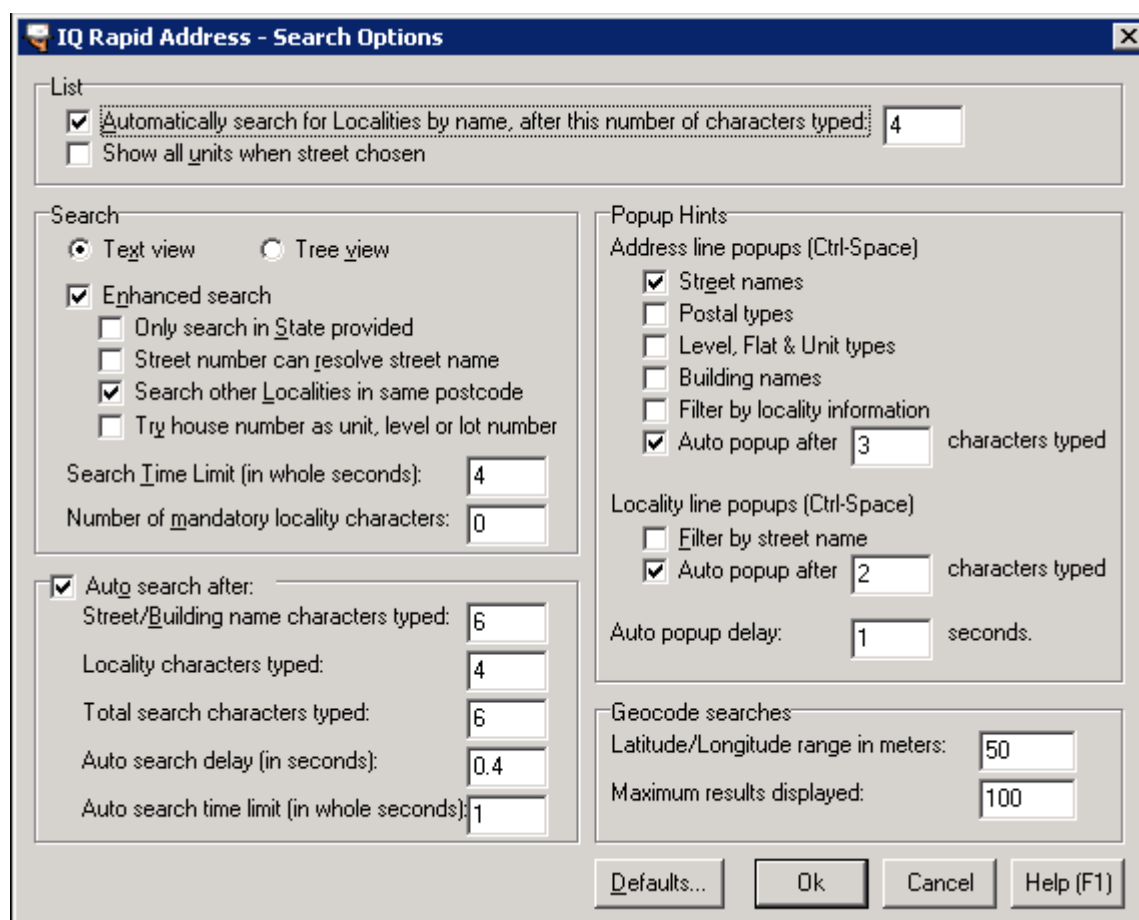


Use the Colours panel to define the colours of additional search results displayed if the [Show More checkbox](#) is checked. Click on an item in the list, then select the colour in which it will be displayed from the colour drop-down. The available items are Standard result (standard search result), Border locality, Synonym locality, Synonym border, Same postcode locality, Alternate street name and Message (a system message, e.g. <Too many matches Not all displayed>).

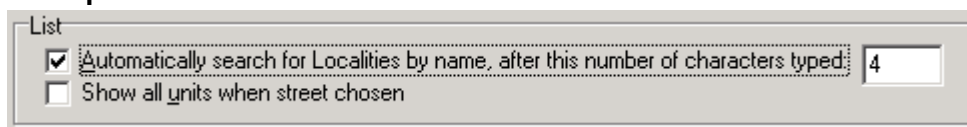
Search Options

Search options specify automatic search and popup behaviour, and can be set when [Saving options](#) on a per configuration basis.

Select **Search** from the **Options** menu to open the Search Options window.



List options



Automatically search for Localities by Name, after this number of Characters Typed

Enter the number of characters at the beginning of a locality which must be typed before an automatic search occurs when entering the name of a locality using the [List address method](#). To display localities before this number of characters has been typed, press Enter.

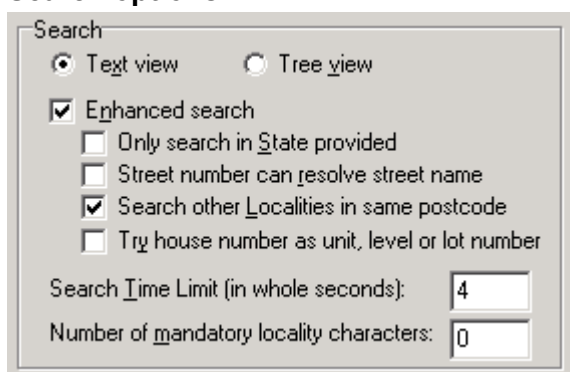
For example, if set to 4 and KEW is typed, nothing will be displayed until Enter is pressed, when KEW and KEWDALE will be displayed. If the fourth character KEWD is typed, "KEWDALE" will be displayed

Show all units when street chosen

Clear this option to display only street numbers when a street is chosen. This is the normal setting, as units/levels are only displayed after a street-number is chosen.

Check this option to display all the units/levels for each street number when a street is chosen. This is useful if the unit/level number is known but the street number is not.

Search options



Text View

Check this option to display found addresses in the [Results panel](#) one per line.

Tree View

Check this option to display found addresses in the [Results panel](#) in a tree structure.

Enhanced search

Check this option to enable searching enhanced beyond strict AMAS rules using the options below.

Clear this option to search according to AMAS rules, which provides a more limited search, but has less chance of finding an undesired address.

Only search in State provided

Check this option to limit the search to the selected state. This is useful if the street name and state is known, but the locality and postcode is uncertain.

Street number can resolve street name

Check this option to find a house number in two streets of the same name, only one of which has that house number.

Without this option, the street names must be clarified before the house numbers are found. For example, searching for 20 flin MELB will find Flinders Lane, Flinders Street and Flinders Court,

even though only Flinders Lane has a 20. If the option is selected, only 20 Flinders Lane MELBOURNE will be found.

Search Other Localities in Same Postcode

Check this option to broaden the search to include all localities in the found postcodes. For example, a search for n syd will find NORTH SYDNEY NSW 2060 and subsequently find all other localities with postcode 2060.

Try house number as unit, level or lot number

Check this option to use the house number entered as level, unit or lot number, for example, "35 Spring St" will try "Level 35", "Unit 35" and "Lot 35" as well as house number 35.

The level type or unit type can also be specified, e.g. Level 7 Spring Street

If there is a direct match to the house number, matches to the level, unit or lot number will only be displayed if the "Show More" option is checked.

Search Time Limit (in whole seconds)

Enter the timeout limit for user-initiated searches.

Number of mandatory locality characters

Enter the number of letters which must be typed in the Locality Line before the **Search** button is enabled. If a postcode is entered and no letters, the **Search** button will also be enabled. State abbreviation characters don't count.

Popup Hint Options

Auto popup is disabled if **Auto popup after [N]** is cleared in both the **Address Line popups** and **Locality Line popups** sections, but the user can still display popup hints by pressing Ctrl+Space.

Address line popups (Ctrl-Space)

Street Names

Check to include street names in the popup list.

Postal types

Check to include postal types such as PO BOX and Locked Bag in the popup list.

Level, Flat and Unit types

Check to include levels, flats and units in the popup list.

Building names

Check to include building names in the popup list.

Filter by locality information

Check to filter the popup list based on information currently on the locality line. For example, if Bondi Junction has already been entered as the locality, a partial address line of sp will only show Spring St in the popup list.

Auto popup after [N] characters

Enter the number of letters which must be typed in the Address line before the auto popup list is displayed. A minimum of 3 characters is recommended to increase the chance that the first page of the popup list displays the required address. The popup list can also be manually displayed by pressing Ctrl+Space.

Locality line popups (Ctrl-Space)

Filter by street name

Check to filter the popup list based on information currently entered on the address line. For example, if Spring St has already been entered as the address, a partial locality line of Bon will only show Bondi Junction in the popup list.

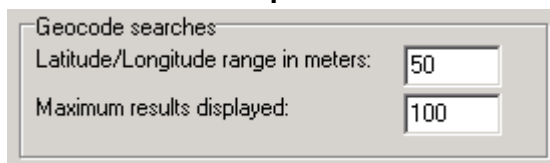
Auto popup after [N] characters

Enter the number of letters which must be typed in the Locality line before the auto popup list is displayed. A minimum of 3 characters is recommended to increase the chance that the first page of the popup list displays the required address. The popup list can also be manually displayed by pressing Ctrl+Space.

Auto popup delay

Enter the delay when typing subsequent characters before displaying the popup list. This will still take both **Auto popup after [N] characters** values into account.

Geocode Search Options



Geocode searches

Latitude/Longitude range in meters:

Maximum results displayed:

Use these options to specify the limits of searches by geographical information. These are only available when [Selecting an ARF](#) with Geographical Information.

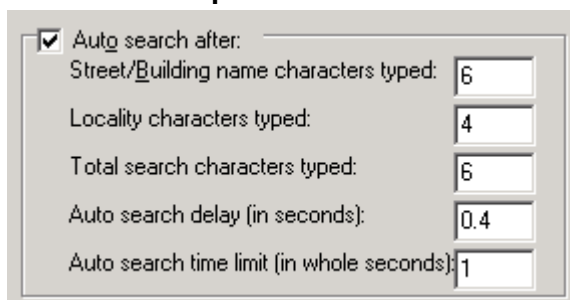
Latitude/Longitude range in meters

Enter the range in meters of the search when [Finding addresses by GIF](#).

Maximum results displayed

Enter the maximum number of results displayed when [Finding addresses by GIF](#). If more than this number of addresses is found a message is displayed at the bottom of the results: <Too many matches. Not all displayed>

Auto Search Options



☒ Auto search after:

Street/Building name characters typed:

Locality characters typed:

Total search characters typed:

Auto search delay (in seconds):

Auto search time limit (in whole seconds):

Auto search after:

Check this option to enable automatic searching. Clear to disable all auto-search options.

Street/Building name characters typed

Enter the number of letters in street or building names which must be typed in the Address line before matching address data is automatically found. The search will also use any locality information already entered.

Locality characters typed

Enter the number of letters in locality names which must be typed in the Locality line before matching address data is automatically found. The search will also use any address information already entered.

Total search characters typed

Enter the total number of letters in the address and locality lines which must be typed before matching address data is automatically found.

Auto search delay (in seconds)

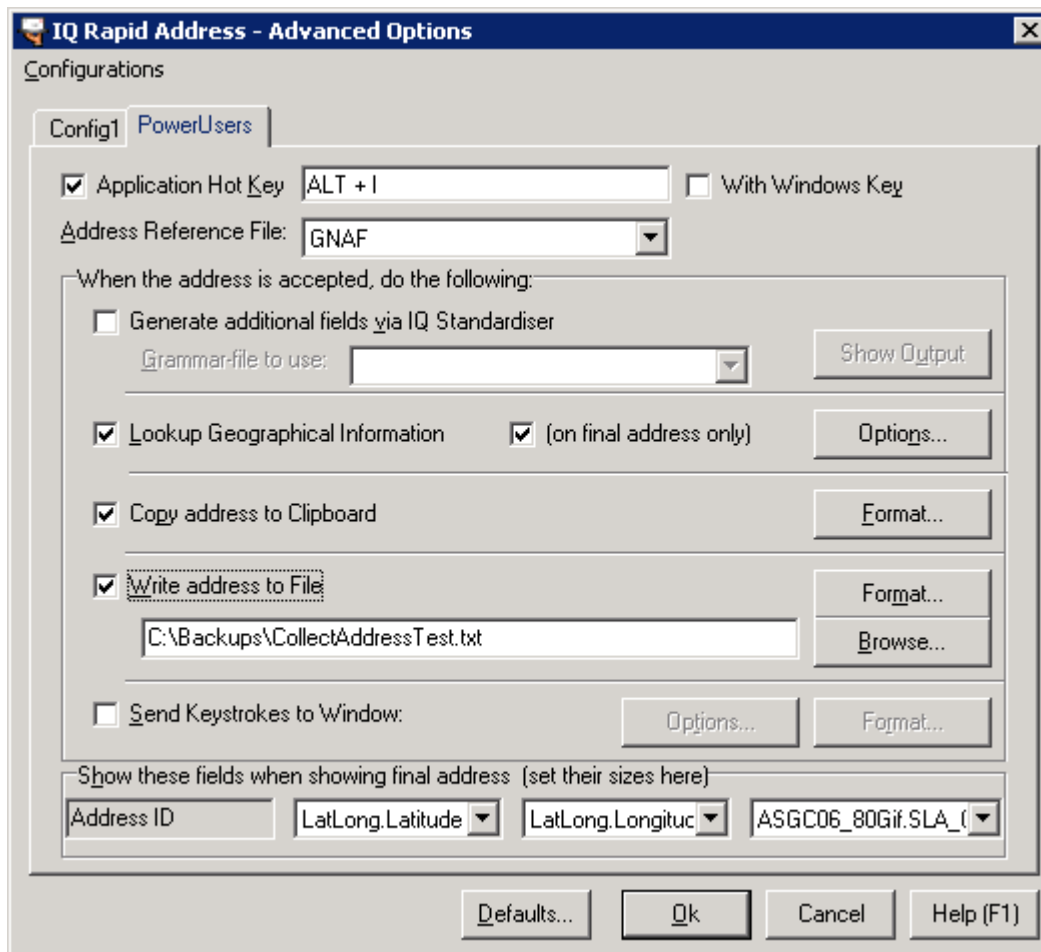
The value chosen here will determine how much of a delay is required in typing subsequent characters before the search begins. This will still take into account the chosen minimum characters setting for either address line or locality line respectively.

Auto search time limit (in whole seconds)

Set the time limit for searches that are initiated by the auto search process. This value should be set to a low value to prevent interrupting further entry of address data.

Advanced Options

Select **Advanced** from the **Options** menu to open the Advanced Options window.



Advanced options enable IQ Rapid Address to execute different actions when the address is found, depending on the hot key pressed to invoke IQ Rapid Address. Advanced options can be set when [Saving options](#) on a per-configuration basis.

Either use the default configuration, or define items and save as different configurations. Use the Advanced Options [Configurations menu](#) to add, copy, rename, select and delete configurations. Only one configuration can be current, but the current configuration can be changed by:

- Starting IQ Rapid Address with the Hot Key defined in a configuration.
- Starting IQ Rapid Address with a command line parameter, see [Starting and Exiting](#).
- Selecting **Set as Current** from the **Configurations** menu.

Advanced Options Configurations menu

Use the [Advanced Options Configurations](#) menu to manage configurations:

- | | |
|-----------------------|--|
| Add New | Click to add a new blank configuration tab which can then be edited. |
| Add New Copy | Select the Configuration tab to copy, then select this menu item to duplicate the configuration, which can then be edited. |
| Rename | Select the Configuration tab to rename, then select this menu item and type the new configuration name. |
| Set as Current | Select the Configuration tab to use, then select this menu item to use it as the current configuration. |
| Delete | Select the Configuration tab to delete, then select this menu item to delete the configuration. |

Application Hot Key




Check this option to define the keys that will pop-up IQ Rapid Address from another application if IQ Rapid Address is running, whether hidden or minimised. The hot key will not start IQ Rapid Address if it is not running.

The keys can be combination of one or more special keys such as Alt, Ctrl, or the Windows Key, and the standard letter, number, function (F1 – F12) or punctuation keys. Do not use key combinations already used by the application, such as Ctrl+C (copy) and Ctrl+V (paste), Win+I is recommended. If the combination is already in use a message is displayed "Cannot register HotKey", and the hot key will not be enabled.

Hot Key

Place the cursor in this field and press the shortcut key combination, e.g. Win+I

With Windows Key

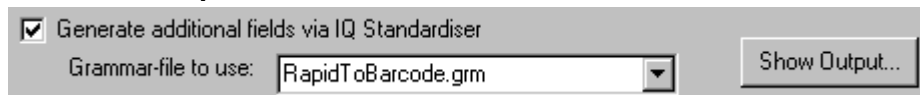
Check this option to use the "Windows" key , "Win+" will be displayed.

Address Reference File

Select the Address Reference File to use for this configuration from the drop-down list, e.g. PAF. If a specific ARF has been installed, select Other from the drop-down list and type the file name. The available ARFs in the list depend on the site licence.

Every named configuration and corresponding ARF creates a new entry in the Options | ARF menu which can be used when [Selecting an ARF](#), for example, PAF PowerUsers.

StanRt Lookup



Generate additional fields via IQ Standardiser

Check this option to pass found address data to a grammar file, which can return some other calculated or found fields associated with the address. This information can then be used to [Copy address to Clipboard](#), [Write address to File](#) or [Send Keystrokes to Window](#).

Grammar-file to use

Select, type, or browse for the name of the grammar file which will perform the calculation or lookup. This must be specifically designed to process the input address fields. Supplied Grammar Files are described in the [Grammar File Reference Manual](#).

Show Output

Click this button to display a list of the fields which the Grammar File will return.

Geographical Information Lookup

This option is only available when [Selecting an ARF](#) with Geographical Information if the GIF files are licensed.

Lookup Geographical Information

Check this option to search by and display geographical information.

(on final address only)

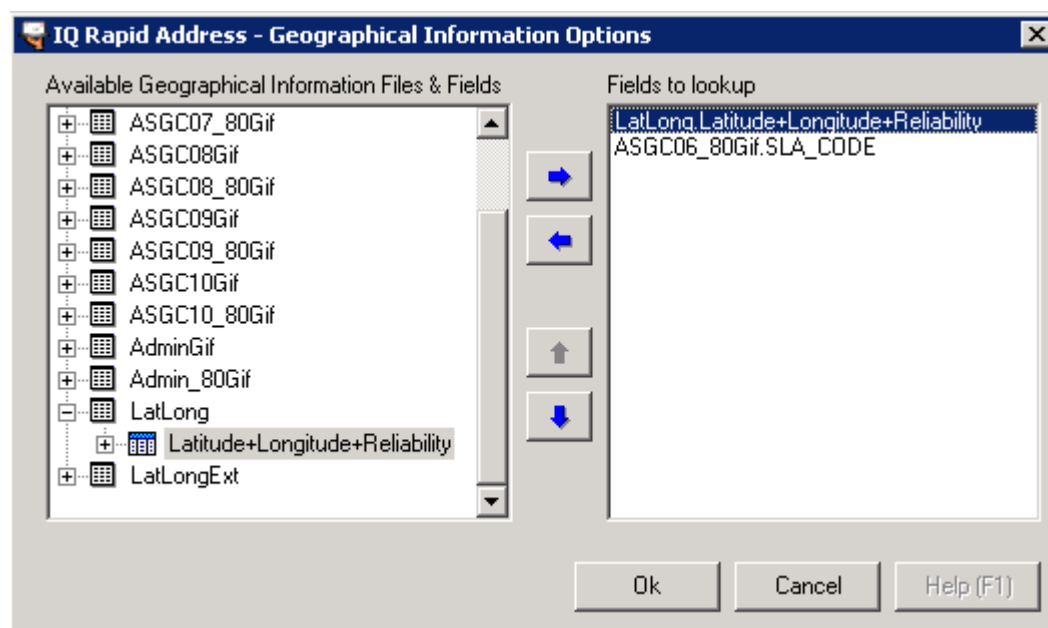
Check this option to retrieve geographical information only when an address has been selected. Otherwise geographical information is retrieved during the address search, which may be slower.

Options

Click the **Options** button to open the Geographical Information Options dialog, displaying a list of fields available in the Geographic Information Files (GIF). The fields looked up are available as additional output fields.

Use the arrow buttons to copy available fields on the left to fields to look up on the right. Each lookup field becomes available as a sub-menu of the Find by GIF option in the Tools menu, for [Finding addresses by GIF](#).

In the example below Latitude, Longitude and Reliability and SLA Code have been selected from the numerous available GIF sources.



Copy address to Clipboard



Check this option to copy the found and accepted address to the clipboard, from where it can be pasted into another file or application.

Click the **Format** button to define the [Address Output Format](#).

Write address to File

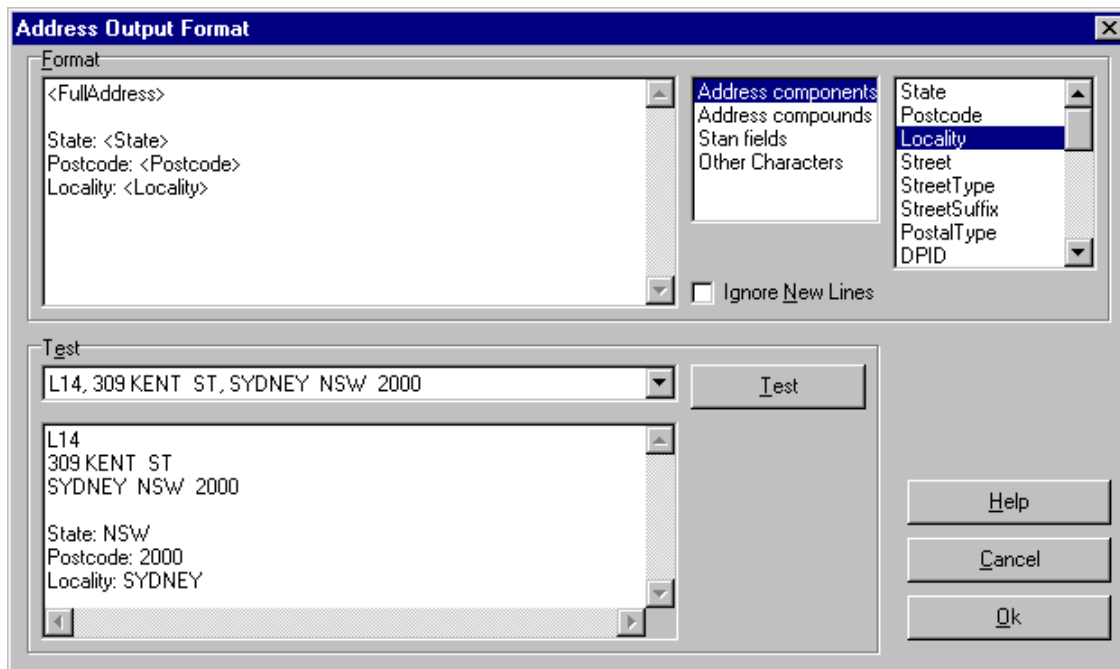


Check this option to append the found and accepted address to a file. Type the file name, or click the **Browse** button to select the file.

Click the **Format** button to define the [Address Output Format](#).

Address Output Format

Use this dialog box to define the format of the address copied, written or sent using [Copy address to Clipboard](#), [Write address to File](#) or [Send Keystrokes to Window](#).



Format panel

Use this panel to format the address, which is composed of address, character and keystroke tokens enclosed with "<" ">" pairs, and any other text.

- To add a token, position the cursor in the Format panel, then select a category from the Category list and double-click a token to add it to the format, e.g. <FullAddress><Cr>
- To delete a token, select it in the Format panel, then press the Del or Backspace key.
- To add any other text which will appear exactly as typed, position the cursor in the Format panel and type the text, e.g. Deliver to: To use the less than sign type "<<>".

Category list

The format tokens are divided into the following categories:

Address Components	The base components of the address.
Address Compounds	The address components combined together in different ways.
Stan Fields	The output fields of the of the StanRt Lookup
Geographical Fields	The fields made available by Lookup Geographical Information Lookup
Other Characters	(Not Send Keystrokes to Window) other characters that cannot be typed into the format box.
Special Key	(Send Keystrokes to Window only) keystrokes that involve special keys such as Alt and Ctrl.
Other Keys	(Send Keystrokes to Window only) other keystrokes not included in Special Key .

Ignore New Lines

Check this option to ignore new lines entered in the format. Use the <CrLf> token to insert a new line.

Test

Use this panel to test the defined address formatting. Select one of the addresses from the drop-down list, then click the **Test** button to display the formatted address.

Address Components

Individual address components are listed below.

State	2 or 3 character state abbreviation, e.g. NSW
Postcode	4-digit postcode, e.g. 2037
Locality	Locality name, e.g. "Pymont"
Street	Street name without Street Type e.g. "George"
Street Type	e.g. "AVE" in "15 Main Avenue"
Street Suffix	e.g. "S" in "15 Main Avenue South"
PostalType	e.g. "PO BOX"
DPID	8 digit Delivery Point ID
HouseNum1	e.g. "3" in "3-7 Railway Street"
HouseSuffix1	e.g. "B" in "3b Railway Street"
HouseNum2	e.g. "7" in "3-7 Railway Street"
HouseSuffix2	Not used
FlatType	e.g. "U" in "Unit 1 / 56 Main Road"
FlatNum	e.g. "1" in "Unit 1 / 56 Main Road"
LevelType	e.g. "L" in "Level 1"; "G" in "Ground Floor"
LevelNum	e.g. "1" in "Level 1"
Building1	Building name
Building2	Second part of building name
LotNum	e.g. "5" in "Lot 5 Main Street"
PostalNum	e.g. "123" in "P.O. Box 123Q"
PostalPrefix	e.g. "M" in "P.O. Box M123"
PostalSuffix	e.g. "Q" in "P.O. Box 123Q"
PrimaryPoint	"R" (real primary point), "P" (phantom primary point), or empty.
UserInfo	Information entered by the user to be included with the address
Synonym	Synonym locality name
SynonymFlag	"V" (valid synonym) , "U" (unacceptable synonym), or empty.
BorderName	Associated border name originally entered by user
BorderPostcode	Associated border postcode originally entered by user
AltStName	Associated alternate street name originally entered by user
AltStType	Associated alternate street type originally entered by user
AltStSuf	Associated alternate street suffix originally entered by user
LocalityDID	Locality Delivery identifier
GroupDID	Group Delivery identifier
PreAddress	Address data entered at the front of the validated address
PostAddress	Address data entered at the end of the validated address

BSP	Barcode Sort Plan number for this postcode.
NPSP	National PreSort Plan number for this postcode
Barcode37	37-bar barcode string generated from the DPID
StatusFlag	Code indicating the address match conditions
AmendFlags	Code indicating the modification of the entered address data

Address Compounds

Delimited	The individual address elements (from State to PrimaryPoint) in tab delimited form.
FullAddress	The address in a format ready for an envelope (2 or 3 lines)
OneLineFullAddress	The full address on one line.
FullAddress1	The first line of the full address.
FullAddress2	The second line of the full address.
FullAddress3	The third line (if there is one) of the full address.
Address	The address without the locality information (locality, state and postcode).
OneLineAddress	The above in one line
Address1	The first line of the address without locality information.
Address2	The second line (if there is one) of the address without locality information.
LotAddress	e.g. "LOT 5", blank if there is not lot number in the address.
HouseNumRange	e.g. "3-7" if there is a range in the address, else just the first house number with its suffix if it has one.
FlatSlash	The flat type and number followed by a slash (if there is a flat number).
ComboBoxState	The first letter of the state (with "NN" for N.T.). Useful to send keystrokes to an application that has a dropdown box for the state.

Stan Fields

These are the output fields of the IQ Standardiser Grammar File selected in [StanRt Lookup](#).

Geographical Fields

These are the output fields made available from [Geographical Information Lookup](#).

Special Key

Use this category to define special keys such as Alt and Ctrl. These can only be defined for [Send Keystrokes to Window](#). Use the [Other Keys](#) group to find the keystrokes that cannot be entered here.

To add a special key to the address format:

1. Position the cursor at the required location in the Format panel.
2. Place the cursor in the Keystroke field.
3. Press the shortcut key combination using special keys such as Ctrl, Alt, Shift, Function, CapsLock, Home and PgDown, and the character and number keys, for example:
<CTRL + SHIFT + V>, <alt + F8>, <shift + HOME>
4. Click the Insert Key button to add the keystrokes to the Format panel.

Other Keys

Use this category to define keys that cannot be entered using [Special Key](#). These can only be defined for [Send Keystrokes to Window](#).

To add a key, position the cursor in the Format panel, then select **Other keys** from the Category list and double-click the key to add it to the format, e.g. <Print>, <Escape>

Use Delay=100 to wait 100 milliseconds, or type another value such as <Delay=250> to wait another time in milliseconds.

Pressing a special key such as Ctrl, Alt or Shift in this category just presses the key. Use the [Special Key](#) category for these keys.

Other Characters

Use this category to define other characters that cannot be typed into the format box. These cannot be defined for [Send Keystrokes to Window](#).

To add a character, position the cursor in the Format panel, then select **Other characters** from the Category list and double-click the character to add it to the format, e.g. <Tab>, <CrLf>

To use another character, type the ASCII code of the character surrounded by the "<" and ">" symbols, e.g. <09> specifies the tab character.

Send Keystrokes to Window

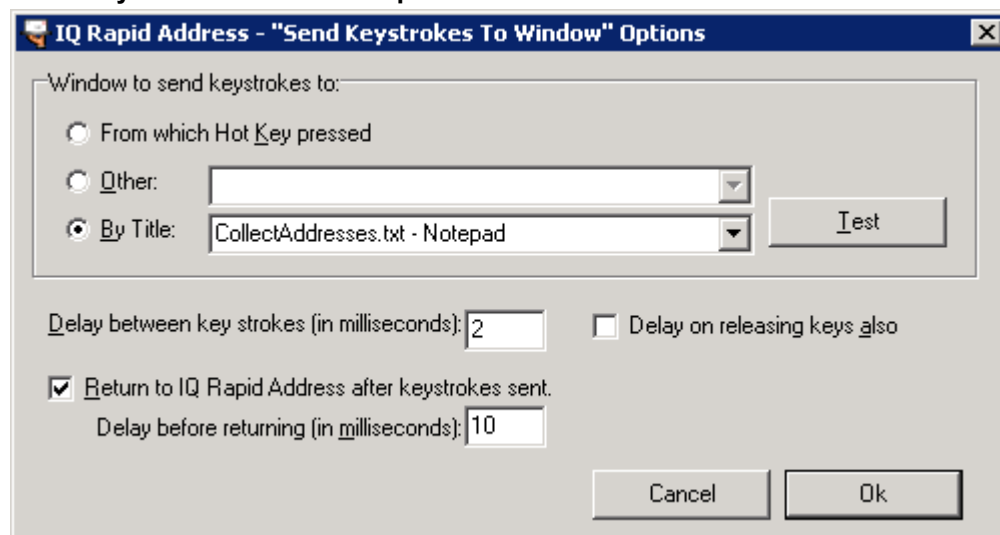


Check this option to automatically type the found and accepted address into another application.

Click the **Options** button to select the application the address is typed into.

Click the **Format** button to define the [Address Output Format](#).

Send Keystrokes to Window Options



Use this dialog box to choose the application the address is pasted into.

From which Hot Key pressed

This is the default. The address will be pasted into the window from which the hot key was pressed.

Other

Choose the window from the list of currently open windows. When this Window is closed it will no longer work, even if it is reopened.

By Title

Choose the title of the window from the list of currently open windows. If there are two windows with the same title, either might be chosen, and if the window changes name, it will no longer work. Use **Other** if you expect the window to change name.

Test

Click this button to test that the selected window is chosen.

Return to IQ Rapid Address after keystrokes sent

Choose this option to return to IQ Rapid Address after the address has been typed into the chosen window, or it will remain in the foreground.

This option has no effect if the [Hide after address accepted](#) option is selected.

Delay between key strokes (in milliseconds)

Use this option to set the delay between key strokes in milliseconds if the address is not being fully typed into the application, which may be because IQ Rapid Address is returning before the address has been typed. This value must be fine-tuned according to the application, system and hardware.

Delay on releasing keys also

Check this option to set the same delay between pressing and releasing the key strokes.

Test

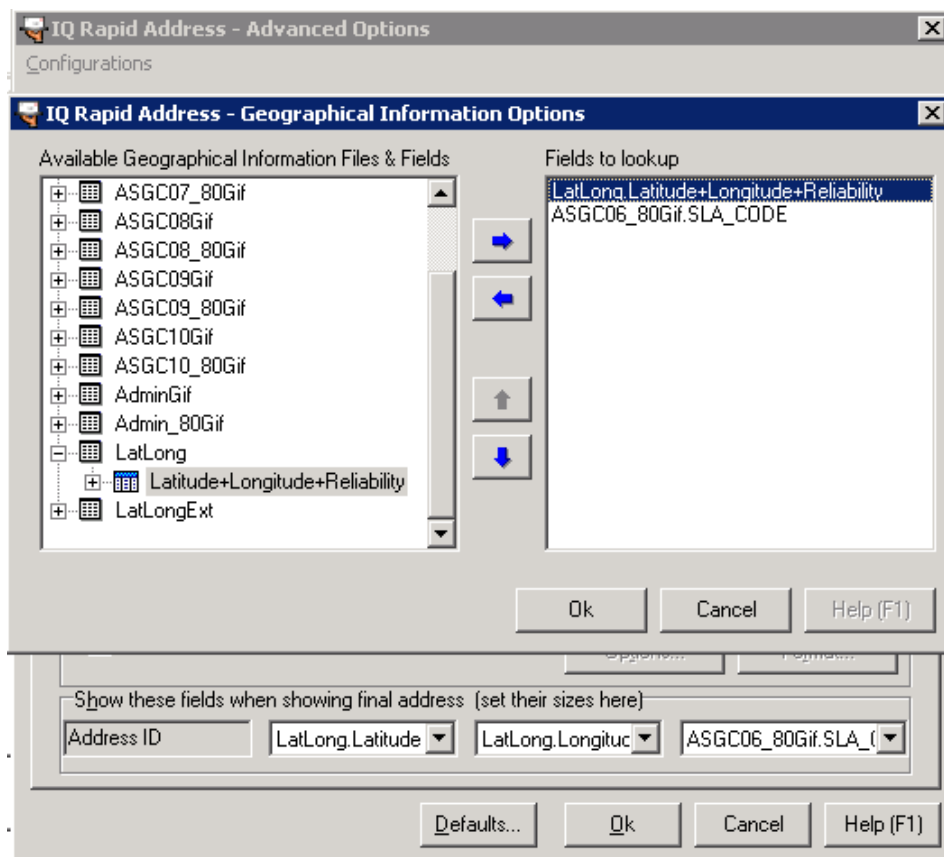
This will show the chosen window.

Additional Field Options

Show these fields when showing final address

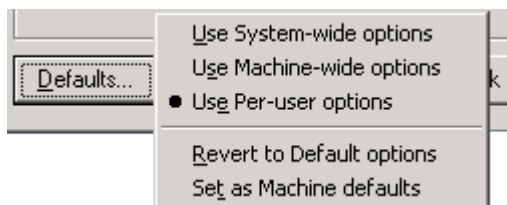
Use this section to define the additional fields which are displayed in the [ID line](#) when a final address is chosen. Up to three fields can be defined, but these can only be selected from the defined [Geographical Information Lookup](#) options.

In the example below the fields are Latitude, Longitude and SLA Code.



Saving options

Use the Defaults button on the [General Options](#), [Advanced Options](#) or [Search Options](#) page to define how IQ Rapid Address options are saved.



Use System-wide options

This uses the options previously set as the machine default on the Rapid Server.

To force all clients to use the server-wide options on the server, set the following registry value:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\iRapid]
"OptionScope"=dword:00070000
```

Use Machine-wide options

This uses the options previously set as the machine defaults and ensures that all users who log on to a particular machine have the same settings. When the Rapid Server is running locally this is the same as the system-wide options.

To force all clients to use the server-wide options on a specific machine, set the following registry value:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\iRapid]
"OptionScope"=dword:00000007
```

Use Per-User options

This enables a user to modify options and save these settings in their registry. When this option is chosen the current working configuration is used as the starting point for any user-specific changes.

Revert to Default options

This changes the settings back to the default configuration.

Set as Machine defaults

A user with administrative rights on the local PC can save per-user options and make it the machine default for any other users of the same PC. If this is done on the server this becomes the system-wide options.

IQ Rapid Address SDK

Overview

IQ Rapid Address is a suite of applications and library functions supplied with all IQ Office editions which provides rapid and efficient searching of Address Reference Files (ARF) to quickly and accurately capture, validate, and standardise addresses found from partial input data.

IQ Office supplies various ARFs in compressed and indexed format to optimise search speed. Depending on the IQ Office edition and licence these may include the Australia Post or New Zealand Postal Address File (PAF), which contain all postal delivery addresses in the country. Searching the PAF enhances the integrity of customer data by returning an address that is both valid in content (accuracy) and format (standard), whilst reducing data entry costs.

In addition to postal address validation, IQ Rapid Address can provide powerful and accurate geographic search and coding capabilities using the Australian Geocoded National Address File (G-NAF). A found address can return corresponding geographical information such as the latitude and longitude of the address, and Australian Bureau of Statistics (ABS) statistical (census) regions in which the address is located. Returned latitude and longitude can be passed directly to a web-based mapping service enabling one-click visualisation and zooming to the found address.

The IQ Rapid Address SDK is an extensive set of Information Quality Application Programming Interfaces and test code which enables these functions to be integrated into new and existing business systems.

About this manual

This manual provides a complete reference to:

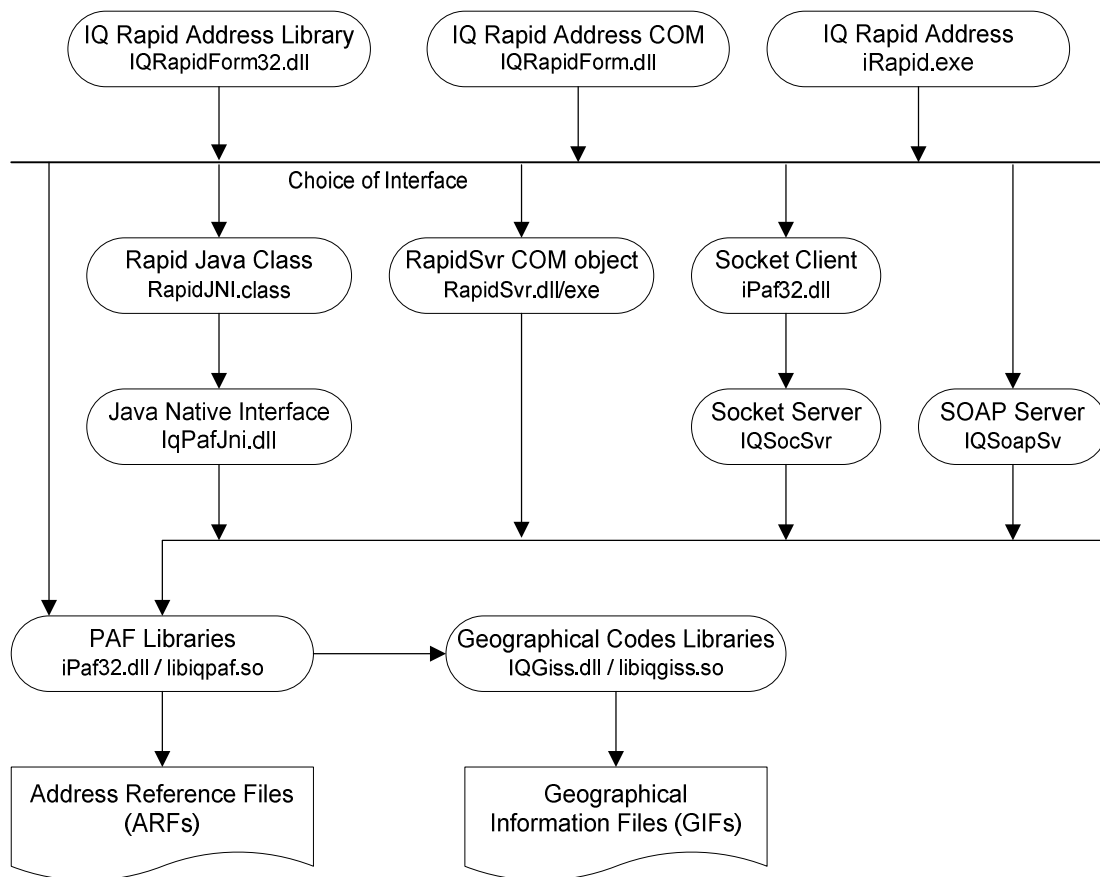
- The PAF Libraries which provide functions to search and access ARFs, available as a Windows 32 or 64 bit API DLL or C function library which can be integrated into other systems.
- The IQRapidForm32 Windows library and IQRapidForm COM object which provide the same screen as the IQ Rapid Address application.
- The RapidSvr DCOM / COM object which can run as a Windows service.
- Java, Socket and SOAP interfaces to the PAF Library functions.
- Address Reference Files.

Refer to the following manuals for details of other IQ Office components:

- The [Rapid Address](#), [Easy Post](#) and [Fix Address](#) User Manuals describe how to use these standalone applications.
- The [Grammar File Reference Manual](#) describes the various grammar files supplied with different IQ Office editions and data licences.
- The [IQ Office Configuration Manual](#) describes system configuration settings.
- The [IQ Office Component Setup Manual](#) describes how to edit configuration settings, and how to configure machines as a Rapid Address or Standardiser socket, SOAP or DCOM client or server, and install, remove or manage services.

Components

Rapid Address components are shown in the diagram and summarised below.



[iQRapidForm32](#)

A library, containing Windows 32-bit API function calls, providing the same screen as the IQ Rapid Address application.

[iQRapidForm](#)

A COM object that provides the same screen as the IQ Rapid Address application.

[IQ Rapid Address](#)

An application that allows entry of addresses using minimal keystrokes, validates the address, and appends the DPID, GroupDID or LocalityDID. It uses the RapisSvr COM library or the PAF Libraries.

[RapidJNI Java class](#)

Java class which provides the address searching functions of RapidSvr and the PAF Libraries via a Java Native Interface (JNI).

[RapidSvr](#)

A DCOM / COM server providing functions to search the PAF files. It can run as an Windows Service.

[PAF Libraries](#)

Library functions providing functions to search and access the PAF/ARF files, generate barcode code strings, via a Windows 32 and 64 bit API DLL and C-function libraries for several other operation systems.

[Socket Client](#)

The PAF Libraries can be configured to act as a socket client and redirect all calls to the Socket Server.

[Socket Server](#)

Provides PAF Library functions via a socket server.

[SOAP Server](#)

Provides some PAF Library functions via a SOAP / XML server.

[Address Reference Files](#)

(ARF) a compressed data source set that can be used by IQ Rapid. One such ARF is the [PAF files](#) (Postal Address File) provided by Australia Post, which contains all Australia deliverable addresses and associated DPID. It is supplied

with IQ Office in sorted and compressed format for rapid access. It is used by IQ Rapid and some grammar files.

RapidSvr

RapidSvr is a DCOM object library which can also run as an Windows Service, which provides access to search functions on the PAF/ARF file. These functions are also available as API calls via the [PAF Libraries](#).

RapidSvr Installation

This section describes steps required for a custom installation, or repair of a standard RapidSvr installation performed by the Setup program. Files required for RapidSvr are described below.

RapidSvr

- To use RapidSvr as an out-of-process server or an Windows Service, install `RapidSvr.exe` on the server machine, usually in the system directory, but at least on the system drive.
- To use RapidSvr as an in-process server, `RapidSvr.dll` is required.
- In both cases install the [PAF Libraries](#) by copying `iPaf32.dll` file into the System directory.

Grammar file

Place the file `_AddressLine.grm` in the directory specified in the [Grammar Directory](#) configuration setting.

PAF files

Place the compressed PAF files in the directory specified in the [PAF Directory](#) configuration setting.

Licence file

Place the licence file `iRapid.lic` in the directory specified in the [Home Directory](#) configuration setting.

Geographical Code files

Place any supplied GIF files in the directory specified in the [PAF Directory](#) configuration setting. Place the IQ Geographical Index Sub System file `iqgiss.dll` in the System directory to allow access to the geographical files.

RapidSvr Registration

This section describes how to register and unregister RapidSvr. StanRt can be registered as described below, or by using IQ Office Component Setup [Client Server Setup](#).

RapidSvr.exe

To register RapidSvr.exe as Windows service, type the following from the command prompt or from the Run menu:

```
RapidSvr /Service
```

To register RapidSvr.exe as a COM object but not as a service, type the following:

```
RapidSvr /RegServer
```

To unregister the COM object or the service type the following:

```
RapidSvr /UnregServer
```

If RapidSvr is called with another parameter, it will not end, and must be closed using Windows Task Manager.

RapidSvr.dll

To register `RapidSvr.dll` as an in-process COM object, type the following from the command prompt or from the Run menu:

```
Regsvr32 RapidSvr.dll
```

To unregister the COM object, type:

```
Regsvr32 /u RapidSvr.dll
```

Where appropriate, substitute the full path of `RapidSvr.dll`.

iPaf32.dll

No registration is required.

RapidSvr Errors

Errors occurring will be raised using the standard convention for COM errors. If an `OleException` error is raised, its message will begin with the [PAF Error codes](#) number.

Methods

`RapidSvr` contains just one object **Rapid** with a number of methods. The IQ Rapid Address user interface List tab uses the [GetAddress method](#), and the Search tab uses the [GetHintAddress method](#) and [GetPopupHint method](#).

GetAddress method

This method is used to get a list of one of the following:

- All localities
- All localities with a given postcode
- All localities in a given state (or NZ City)
- All locality names with a given beginning
- All streets and postal types (e.g. PO Box) in a given locality
- All house numbers in a given street
- All units and levels for a given house number
- Addresses matching given unit/level, for a given street address
- All postal numbers for a given postal type
- All street names with a given beginning
- All streets with a given name, and optionally a given street type beginning and/or locality name beginning.
- All streets in a given postcode
- The address having a given DPID or AddressId
- All addresses having a given Geographical code.

The first time `GetAddress` is called, the criteria of what to find are passed as the parameter, and the first element in the list is returned. Subsequent calls pass no parameter, and return subsequent elements in the list. When there are no more elements in the list, an [Empty record](#) (just delimiters) is returned.

Syntax

```
Object.GetAddress( RecIn )
```

Syntax description

Object	A Rapid object
RecIn	A string with the criteria of what to find. See Input record . This string must be in Australian address record format , or an error will be generated. If missing or empty, the next element (address) will be returned.

Return Value

A string with the element (address) found, in [Australian address record format](#).

See [Output record](#).

GetHintAddress method

This method is used to search the PAF for an address or partial address given some address information (a hint).

The first time GetHintAddress is called, the criteria of what to find are passed as parameters, and the first matching address found is returned. Subsequent calls pass no parameters, and return subsequent matching addresses. When there are no more matching addresses, an [Empty record](#) is returned.

The results returned will be in order, with the closest match first.

Input parameters

To see the results of different input parameters, try the “IQ Rapid Address” application.

The **Address** line and **Locality** line on the **Search** tab are passed directly to GetHintAddress.

Special return values

If the [UseFindStatus property](#) is set to **False**, then the following apply:

1. If there are too many addresses found (more than the [MaxAddressesReturned property](#)), then the first call to GetHintAddress will return an empty string, and subsequent calls will return some of the found addresses as normal.
2. If the search is taking too long and the [TimeLimit property](#) is reached, the first call to GetHintAddress will return an empty string, and subsequent calls will return an [Empty record](#).

Syntax

```
Object.GetHintAddress( LocalityLine, AddressLine )
```

Syntax description

Object	A Rapid object
LocalityLine	Optional. A string with information on the Locality to find. See Locality Line
AddressLine	Optional. A string with information on the Street and/or House Number, or Postal Address to find. See Address Line . If both LocalityLine and AddressLine are missing/empty, the next address found will be returned.

Return Value

A string with the address or partial address found in PAF [Australian address record format](#).

Locality Line

The first parameter (of the [GetHintAddress method](#) or the [GetAddressHint\(Id\) function](#)) is a string with some information on the locality.

It can contain any or all of the following:

- Locality/Suburb name
- Postcode

- State abbreviation (without the points, e.g. “NSW”)

Locality name matching

The locality name tolerates spelling mistakes of one letter. The first few letters of words in the locality can be passed, e.g. “Syd A” for Sydney Airport, according to the following rules:

1. For each word-beginning in the string, a word beginning with the same first few letters in the locality must be present for it to be found, but not all word-beginnings need to be present. For example, “syd” will find Sydney, Sydenham, Sydney Markets, North Sydney etc, but “n syd” will only find North Sydney, not Sydney.
2. The word beginnings must be in order, so “sy m” will find Sydney Markets, but “m sy” will not.
3. Abbreviations (N, S, E and W) are recognised and need not be in order, e.g. Both “me n” and “n me” will find both North Melbourne and Merah North.

State

If a state abbreviation is in this line and [SearchStateOnly property](#) is set to True, then only addresses in this state will be found.

No locality or postcode

If nothing or just a state is passed, then all localities in the state will be searched to match the address-line information.

If a postcode or part of a locality name is passed, then only addresses matching this locality information will be found.

Both locality and postcode

If both a postcode and some locality name information are passed, then all localities matching either the name or the postcode will be found, with preference to those localities that match both.

Exact matching

To find an exact match, ensure all locality elements are present, and enclose the line in brackets “[]”, for example:

[SYDNEY 2000 NSW] will only return SYDNEY NSW 2000

SYDNEY 2000 NSW will return all localities containing the word “SYDNEY” such as “NORTH SYDNEY”, as well as all localities with postcode 2000.

This can be useful when other address elements are passed via the address line field.

Partial exact matching

To find an exact match to the address elements entered (locality name, postcode and/or state), enclose this line in parentheses “{ }”

For example, SYDNEY will return all those localities that contain the word “SYDNEY” such as “NORTH SYDNEY”, whereas {SYDNEY} will only return localities called SYDNEY.

Address Line

The second parameter (of the [GetHintAddress method](#) the [GetAddressHint\(Id\) function](#)) is a string with some information on the street or postal part of the address.

If this parameter is missing or an empty string, only matching localities will be found according to criteria passed in the Locality Line.

Street address

The street name or first few letters can be passed, for example:

ge might find George Street

gorge might also find George Street, allowing some spelling tolerance
The house or lot number may precede the street name, and the street type may follow it, for example:

2 ge might find 2 George Street

Lot 5 ma rd might find Lot 5 Main Road

Unit or level information may be included if there is a street number and name, for example:

2/2 ge might find Unit 2, 2 George Street

Level 14 309 K might find Level 14, 309 Kent Street

Postal address

A postal number and or postal type may be passed, for example: “5”, “PO BOX 5”, “A100”, “GPO A100”, “locked bag 12b”, “po box”.

Exact matching

To find an exact match, enclose this line in brackets “[]”, for example:

circular quay might return both CIRCULAR QUAY and CIRCULAR QUAY EAST

[circular quay] will only return CIRCULAR QUAY

This can be useful when street numbers are also entered.

Exact partial matching

To find an exact match to the address elements entered (street name, type and suffix) enclose this line in parentheses “{ }”, for example

Bonny might return streets called Bonny, Benny, Bonn, Bonney, Bunny and Bonnydoon

{bonny} will only return streets called Bonny

No locality line

If no locality line is provided, and there is extra information on the address line, it will be taken to be locality information, e.g. “309 Kent st Syd” would take “Syd” as locality information.

If no locality line is provided, and a two-word (or more) street name is in the address line, with no street type, and there are no matching street names, then the last word(s) of the street name will be taken as a locality, e.g. “309 Kent Syd” would first look for streets matching “Kent Syd”, then for localities matching “Syd” with street matching “Kent”.

GetAddresses method

This method is equivalent to multiple calls to the [GetAddress method](#).

The GetAddress and GetAddresses methods are identical apart from the differences below.

	GetAddress	GetAddresses
Matches returned	First match on first call and subsequent matches on subsequent calls (with no parameter)	All the matches in one call
Field delimiter	Delimiter property	FieldDelimiter

Syntax

```
Object.GetAddresses( RecIn )
```

Syntax description

Object A Rapid object

RecIn A string with the criteria of what to find. See [Input record](#). This string must be in [Australian address record format](#), else an error will be generated.

Return Value

A string with all the elements (addresses) found.

The addresses will be delimited by the [RecordDelimiter](#) (semi-colon by default).

Each address will contain fields delimited by [FieldDelimiter](#) (comma by default), as specified in [Australian address record format](#).

GetHintAddresses method

This method is equivalent to multiple calls to the [GetHintAddress method](#).

Syntax

```
Object.GetHintAddresses( LocalityLine, AddressLine )
```

Syntax description

Object A Rapid object

LocalityLine Optional. A string with information on the Locality to find. See [Locality Line](#)

AddressLine Optional. A string with information on the Street and/or House Number, or Postal Address to find. See [Address Line](#)

Return Value

A string with all the matching addresses or partial addresses found.

The addresses will be delimited by the [RecordDelimiter](#) (semi-colon by default).

Each address will contain fields delimited by [FieldDelimiter](#) (comma by default), as specified in [Australian address record format](#).

GetHintAddress vs. GetHintAddresses

GetHintAddress will return the first match on the first call and subsequent matches on subsequent calls (will no parameter). GetHintAddresses will return all the matches in one call.

The individual fields returned by GetHintAddresses are delimited by [FieldDelimiter](#) ([Delimiter property](#) in GetHintAddress).

GetHintAddresses has no special return values for “too many matches” or “time limit up”.

Otherwise these two methods are identical.

GetPopupHint method

This method is used by IQ Rapid Address to provide the user with a popup list to help the user complete the address.

The first time GetPopupHint is called, the criteria of what to find are passed as parameters, and the first popup item is returned. Subsequent calls pass no parameters (or blank, blank, zero), and return subsequent popup items. When there are no more items on the popup list, an empty string is returned.

Syntax

```
Object.GetPopupHint( LocalityLine, AddressLine, Options)
```

Syntax description

Object A Rapid object

LocalityLine A string with Locality information (any, all or none of Locality Name, State and Postcode).

AddressLine	A string with address line information (e.g. Street address, building name, postal address).
Options	An integer specifying options, see list below.

Return Value

A string with the free form popup list item, or single item (from options above).

Option values

1	popup information for the locality line
2	popup information for the address line (mutually exclusive with 1 above)
4	don't list building names (used with option 2 above)
8	don't list postal types (used with option 2 above)
16	don't list level types (used with option 2 above)
32	don't list flat/unit types (used with option 2 above)
64	don't list street names (used with option 2 above)
256	(hex 100) list street types only
512	(hex 200) list street suffixes only
1024	(hex 400) list level types only
2048	(hex 800) list unit types only
4096	(hex 1000) list postal types only
8192	(hex 2000) list street names only
16384	(hex 4000) list locality names only
32768	(hex 8000) return abbreviated form also (used with options 256 to 4096)

Options 2 to 64 can be combined by adding them together (technically using bitwise-OR).

For options above 256 to 4096, place the type in the AddressLine parameter (eg. "Road"). Or place the beginning of a type followed by an asterisk (eg. "Ro*" may return Road & "Row"). If 32768 is added to the option, then the result will be in the form Abbreviation, Delimiter, Full (eg "RD,Road").

For option 16384, place the locality name in the LocalityLine parameter. Or place the beginning of the locality name, followed by an asterisk (eg "SYD*"). To limit the search to a specific state, use Locality, Delimiter, State (eg. "SYD*,NSW").

FormatAddress method

This method creates a formatted address from the address fields returned from many of the RapidSvr methods.

Syntax

```
Object.FormatAddress( Address, Options )
```

Syntax description

Object	A Rapid object
Address	A string with the address in Australian address record format (as returned from other methods in this object).
Options	Optional. An integer specifying formatting options, see list below.

Return Value

A string with the formatted address.

Option values

1	address on a single line
2	don't include locality information (locality name, state & postcode)
4	expand unit/level abbreviations
8	embed synonym name
16	expand street type & suffix abbreviations
32	upper case
128	unit and level info on separate line to street number and name (except for unit number followed by a slash, eg "7/35 Spring St")
256	don't include building name in address lines.
512	always have unit type. Eg. "7/35 Spring St" becomes "Unit 7 35 Spring St"
1024	don't include lot number if have street number
2048	custom format address using "_FormatAddress.grm"
4096	quote building name
16384	lines divided by CR (carriage return) and LF (line feed), instead of just a LF
32768	after expanding abbreviations (if options 4 and/or 16 are set) and changing case, the input is passed through the _RapidFormat.grm system grammar file.
65536	locality in Title Case. Ignored if option 2 or 32 set.

These options can be combined by adding them together (technically using bitwise-OR).

DpidToAddress method

This method will look up a DPID in the PAF, and return the address with this DPID.

Syntax

```
Object.DpidToAddress ( Dpid )
```

Syntax description

Object A Rapid object

Dpid A long integer specifying the Dpid to look up.

Return Value

A string with the address found in PAF [Australian address record format](#).

If there is no address with this Dpid, an [Empty record](#) will be returned.

Notes

For this function to work, the additional index file "Dpid.in1" or "AddrId.hin" must be in the PAF directory, else an error 672 will be returned.

DpidToBarcode method

Generates a [Barcodes](#) string.

Syntax

```
Object.DpidToBarcode ( Dpid, FormatCode, CustomerInfo, NCoding )
```

Syntax description

Object	A Rapid object
Dpid	A String specifying the DPID from which to create the barcode.
FormatCode	FCC String specifying the type of barcode: 11 – Standard customer barcode (37 bars, no customer information) 59 – Customer barcode 2 (52 bars including 16 bars of customer information) 62 – Customer barcode 3 (67 bars including 31 bars of customer information)
CustomerInfo	A String specifying the customer information to include in the barcode.
NCoding	Boolean specifying the Customer Information coding : False (zero) for C-coding or True (non-zero) for N-coding

Return Value

A String with barcode digits. See [Printing Barcodes](#).

PostcodeToBsp method

Returns the Barcode Sort Plan (BSP) number for the given postcode.

Syntax

```
Object.PostcodeToBsp ( Postcode )
```

Syntax description

Object	A Rapid object
Postcode	A 16-bit integer specifying the postcode to look up.

Return Value

16-bit integer	BSP number
Zero	No BSP number for this postcode

Notes

The additional [PAF files](#) Bsp.db must be present for this method to work, or zero will be returned.

PostcodeToNpsp method

Returns the Nation PreSort Plan (NPSP) number for the given postcode.

Syntax

```
Object.PostcodeToNpsp ( Postcode )
```

Syntax description

Object	A Rapid object
Postcode	A 16-bit integer specifying the postcode to look up it's NPSP number.

Return Value

16-bit integer	NPSP number
Zero	No NPSP number for this postcode

Notes

The additional [PAF files](#) Npsp.db must be present for this method to work, or zero will be returned.

ValidateFullAddress method

Parses a free form address, standardises it, looks it up in the PAF.

Syntax

`Object.ValidateFullAddress (InAddress, Options)`

Syntax description

Object	A Rapid object
InAddress	A string with the free-form address.
Options	A 32-bit integer with two options: 1 - validate using strict AMAS rules 2 - validate using more lenient rules

Return Value

A string with the standardised address in [Australian address record format](#). An empty string indicates an invalid option.

Notes

The address is standardised via [StanRt](#) or the [StanLibrary](#) using the `_ArfGeocode.grm` grammar file for an Australian ARF, and `_NzAddress.grm` for a New Zealand ARF.

ChangeARF method

Selects the Address Reference File (ARF) to use.

Syntax

`Object.ChangeARF (Database, Options)`

Syntax description

Object	A Rapid object
Database	A string with the name of the ARF with the correct ARF file naming .
Options	A 32-bit integer with options. See ARF loading options .

Notes

When an instance of the [RapidSvr](#) object is created, the default Address Reference File is opened, as specified in the [PAF Directory](#) setting. Call the **ChangeARF** method to open a different Address Reference File.

If the specified Address Reference File is already open, the *Options* parameter will be ignored.

Properties

FindStatus property

Returns an integer indicating the result of the previous call to the [GetHintAddress method](#) or the [GetHintAddresses method](#).

Return Value

- 0 OK, all results displayed
- 1 too many results, not all results displayed
- 2 time up, not all results displayed (see [TimeLimit property](#)).

Notes

This property will return an undetermined value if the [UseFindStatus property](#) is set to **False**.

ParsedInput property

Returns a String with the input of the previous call to the [GetHintAddress method](#), [GetHintAddresses method](#) or the [GetPopupHint method](#).

This input was in the *AddressLine* and *LocalityLine* parameters, and is returned here parsed into the standard [Australian address record format](#).

Properties property

Returns or sets a String with specific information about Rapid.

There are [Read-Only properties](#) and [Read/Write properties](#).

An error is generated if an attempt is made to set a property that is read-only, or that doesn't exist.

Syntax

```
PropertyValue = Object.Properties (PropertyName)
```

```
Object.Properties (PropertyName) = PropertyValue
```

Syntax description

Object	A Rapid object.
PropertyName	A String with the name of the required property. If no such property exists, nothing is returned.
PropertyValue	A String with the value of the property. This can also be a number represented by a string, e.g. "1"

Read-Only properties

ArfLibVersion	Returns the version of iPaf32 (or libiqpaf on unix).
ArfLibVersionAu	Returns the version of the IQArfAu DLL.
ArfLibVersionNz	Returns the version of the IQArfNz DLL.
ArfLibVersionUs	Returns the version of the IQArfUs DLL.
ARFs	Returns a semi-colon delimited list of Address Reference Files in the registry.
ClientCount	Returns the number of current clients.
Licence	Returns the licence information found on this computer for Rapid.
LicenceClients	Returns the client limit that appears on the licence file.
LicenceCompany	Returns the company that appears in the licence file.
LicenceComputerName	Returns the name of the computer that appears in the licence file.
LicenceConnections	Returns the connection limit that appears in the licence file.
LicenceCPUs	Returns the CPU limit that appears on the licence file.
LicenceDate	Returns the start date that appears in the licence file.
LicenceEdition	Returns the edition that appears in the licence file.
LicenceExpires	Returns the expiry date that appears in the licence file.
LicenceName	Returns the name that appears in the licence file.
LicenceRemote	Returns whether licence allows remote connections - "True" or "False".
LicenceValid	Returns whether there is a valid non-expired licence, either "True" or "False"
OutputFields	Returns a semi-colon delimited list of field-names that are returned by most of the functions, see Australian address record format .

PafEngineVersion	Returns the PafLibVersion and the machine name it's running on ., e.g. 5.6.244.1 on Computer1
PafLibVersion	Returns the version of the country specific DLL of currently open ARF (or that of IQArfAu if no ARF opened) e.g. 5.6.244.1
PafVersionInfo	Returns information on the PAF/ARF version, e.g. PAF V2007.4, Expires 12/2007 If an ARF is not currently open, the version of the default ARF will be returned followed by a full stop.
Properties	Returns a semi-colon delimited list of supported properties. Alternatively, use <code>PropertyName<index></code>
RapidLicence	Returns whether there is a licence allowing Rapid to function, either "True" or "False"
SortPlanDate	Returns the date from which the sort plans are valid (e.g. "09/2007").

The following properties are only available if the PAF/ARF is open:

GeographicalCodes2	Returns a comma delimited list of geographical codes. This is an alternate format to the GeographicalCodes property
GIFIndexes	Returns a semi-colon delimited list of Geographical Information File indexes available with the currently opened ARF. Each file may have one or more index. Indexes are returned in the form <code>File.IndexName</code> .
GIFFields	Returns a semi-colon delimited list of Geographical Information fields available with the currently opened ARF. Fields are returned in the form <code>File.FieldName</code>
GIFs	Returns a semi-colon delimited list of Geographical Information Files available with the currently opened ARF.
LastResult	Returns the previous result of a call to <code>GetAddress</code> or <code>GetHintAddress</code> functions. If the function returned a "Buffer too small" error, the result can be retrieved via this property. See the below <code>LastResultLength</code> property.
LastResultLength	Returns the length that the above <code>LastResult</code> property will return. Use a buffer size one more than this length when querying the above <code>LastResult</code> property.
LicenceARFs	Returns a hexadecimal number indicating which ARFs are licenced.
LicenceGIFs	Returns a hexadecimal number indicating which GIFs are licenced.
PafCycle	Returns the cycle number of the PAF version.
PafDatabaseName	Returns the name of the ARF (as in the version file).
PafExpiryMonth	Returns the month at the end of which the PAF version expires, i.e. "1"- "12"
PafExpiryYear	Returns the 4-digit year in which the PAF version expires.
PafFullExpiry	Returns the date, if any, that the PAF fully expires.
PafRelease	Returns the release number of the PAF version.
PafVersionStatus	Returns information about the PAF version file: "Valid", "Invalid", "Expired" or "Not Found"

For each of the properties below, include the colon, and replace the variable name and angle brackets (e.g. `BSP:<postcode>`) with the variable value (e.g. `BSP:3039`).

ArfStatus:<ARF>	Returns the version status of the given Address Reference File. This is the same as the <code>PafVersionStatus</code> property, except for a specified ARF.
-----------------	---

ArfVersion:<ARF>	Returns the version of the given Address Reference File. This is the same as the PafVersionInfo property without the expiry date, and for a specified ARF.
Barcode:<dpid>	Returns the 37-digit barcode string for the given 8-digit DPID.
Barcode2:<input>	Returns a digit barcode string. The input is 8-digit dpid, 2-digit format control ("11", "59", "62",...), "C" or "N", customer-info.
Barcode3:<input>	Equivalent to Barcode2 above, except returns blank if customer info too long or in wrong format.
BSP:<postcode>	Returns the Barcode Sort Plan number for the given postcode.
GIFFields:<file>	Returns a comma-delimited list of fields that exist in the given Geographical Information File.
GIFProperty:<file>:<property>	Returns the property for the given Geographical Information File. See help on iqGiss for valid properties.
iRapid:<property>	Returns the given property used in iRapid.exe (internal use only).
NPSP:<postcode>	Returns the National PreSort Plan number for the given postcode.
PropertyName<index>	Returns the name of the property given by the index. Index is a number from 1 to the number of currently supported properties. This is an alternative to reading the "Properties" property. If there is an open ARF, then the list of supported properties is longer.

Read/Write properties

The following read/write properties are available via the [Properties property](#) or the [SetProperties\(Id\) function](#) and [GetProperties\(Id\) function](#).

- [AbbreviateLevel property](#)
- [AbbreviateStreet property](#)
- [AddressFilter property](#)
- [AllowWithinRangeMatch property](#)
- [ComplianceType property](#)
- [ConvertHouseNumRange property](#)
- [Delimiter property](#)
- [EmbedSynonym property](#)
- [ExpandAbbreviations property](#)
- [ExpandGifLists property](#)
- [ExpandLevelType](#)
- [ExpandStreetSuffix](#)
- [ExpandStreetType](#)
- [ExpandUnitType](#)
- [FieldDelimiter](#)
- [GeographicalCodes property](#)

- [GetSearchStatisticsTimeLimit property](#)
- [GifHouseNumberRange property](#)
- [HouseNumResolveLocality property](#)
- [LatLongSearchRange property](#)
- [ListAllPoints property](#)
- [ListAlternateStreetNames property](#)
- [ListDeeperIfOnly1Found property](#)
- [ListDeeperOnBestMatch](#)
- [ListDeliveryIDParent property](#)
- [ListSuffixesSeparate property](#)
- [ListSynonyms property](#)
- [LoneNumberIsPostal property](#)
- [MaxAddressesReturned property](#)
- [OnlyReturnGeocodes property](#)
- [OutputType property](#)
- [PolygonLongLatOrder property](#)
- [PolygonSearchFilter property](#)
- [PutDidInDpid property](#)
- [RapidCodeDump property](#)
- [RecordDelimiter](#)
- [ResolveGroupAmbiguity property](#)
- [ReturnFormattedAddress property](#)
- [ReturnFormattedAddressOptions property](#)
- [SearchBorderLocalities property](#)
- [SearchHouseNumAsOther property](#)
- [SearchPostcodes property](#)
- [SearchStateOnly property](#)
- [SwitchAlternateStreetPosition property](#)
- [SwitchSynonymLocalityPosition property](#)
- [TcpHost property](#)
- [TcpPort property](#)
- [TimeLimit property](#)
- [TitleCaseFields](#)
- [UseFindStatus property](#)
- [UseSockets property](#)
- [ValidateAddressOptions property](#)

AbbreviateLevel property

This property affects the [Australian address record format](#) returned from many of the functions.

It can have the following values:

True The Flat/Unit Type & Level fields are abbreviated. This applies if the ARF contains unabbreviated values with a table of abbreviations (ARF.txt).

False Default unless set in [Default property values](#).

This property is ignored if [ExpandAbbreviations property](#) is True.

[Note: **AbbreviateLevel** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

AbbreviateStreet property

This property affects the [Australian address record format](#) returned from many of the functions.

It can have the following values:

True The Street type (eg STREET) and suffix (eg EAST) fields are abbreviated. This applies if the ARF contains unabbreviated values with a table of abbreviations (ARF.txt).

False Default unless set in [Default property values](#).

This property is ignored if [ExpandAbbreviations property](#) is True.

[Note: **AbbreviateStreet** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

AddressFilter property

This property filters the results returned from address searching functions.

It can have the following values:

Postal Searches will only return addresses with postal types, eg. PO Boxes.

Streets Searches will only return addresses with streets, no PO Boxes.

Blank/not set Searches are not filtered.

This property affects the GetAddress functions, the GetHintAddress functions and ValidateFullAddress. So it will affect IQ Rapid Address and the SOAP functions GetAddresses, GetAddressesHint, ValidateFullAddress and ValidateAddressWithOptions

[Note: **AddressFilter** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

AllowWithinRangeMatch property

This property affects the [GetDpidNonAmas\(Id\) function](#).

It can have the following values:

True An input with a single house number can match to a range address in the ARF if the single house number is within the range and on the same side of the street. For example, 7 will match to “3-9”. This is the default, unless set in [Default property values](#).

False An input with a single house number can only match to a range address in the ARF if the single number is the first or last number in the range. For example, 7 can match to “7-9” or “3-7” but not “3-9”.

[Note: **AllowWithinRangeMatch** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ConvertHouseNumRange property

This property affects the [GetDpidNonAmas\(Id\) function](#).

If the free formatted input contains a street range (e.g. 7-35 Spring Street), and the difference between these two numbers is greater than **ConvertHouseNumRange**, and such a range doesn't exist, then a unit number / street number combination will also be looked for (7/35 Spring Street in our example).

[Note: **ConvertHouseNumRange** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ComplianceType property

This property affects the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#).

It can have the following values:

AMAS	Searching will be done strictly according to Australia Post AMAS rules. Automatically sets the following properties to false: SearchBorderLocalities property , ResolveGroupAmbiguity property , SearchPostcodes property & SearchStateOnly property .
None	There will be a few improvements in the searching facilities, which are not AMAS compliant. Default unless set in Default property values .

[Note: **ComplianceType** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

Delimiter property

The character that is used to delimit the fields in the PAF [Australian address record format](#).

By default this is the tab character (ASCII 9) unless set in [Default property values](#).

Notes

The delimiter of the output of the [GetAddresses method](#) and the [GetHintAddresses method](#) is governed by the [FieldDelimiter](#) and [RecordDelimiter](#) properties.

The delimiter of the output of the [Direct reading functions](#) is the tab character.

The delimiter of the input to the [FormatAddress\(Id\) function](#) is the tab character.

[Note: **Delimiter** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

EmbedSynonym property

This property is only kept for backwards compatibility and affects how a [Synonym Localities](#) are represented in the returned [Australian address record format](#).

It can have the following values:

True	The synonym locality name will appear in the <i>Locality</i> field, with the real locality in parenthesis. This value is only kept for backward compatibility.
False	(default) the synonym locality name will appear in the <i>SynonymLocality</i> field, with the real locality in the <i>Locality</i> field.

[Note: **EmbedSynonym** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ExpandAbbreviations property

This property affects the [Australian address record format](#) returned from many of the functions

It can have the following values:

True	The Street Type, Street Suffix, Flat/Unit Type & Level field abbreviations are expanded to their full names. For example, “STREET” may be returned in the Street Type field instead of “ST”.
False	Default unless set in Default property values .

Notes

This property is the equivalent of a combination of the [ExpandStreetType](#), [ExpandStreetSuffix](#), [ExpandLevelType](#) & [ExpandUnitType](#) properties.

[Note: **ExpandAbbreviations** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ExpandGifLists property

This property affects the records returned by the [GetNextAddressInPolygon function](#). It only applies if at least one of the geographical (GIF) fields contains a list and must be set before calling [BeginPolygonSearch function](#).

The addresses returned by the Polygon Search function include other geographical data as specified in the [GeographicalCodes property](#). This geographical data may contain a field with a list of values, such as a list of phone numbers.

It can have the following values:

True	Each call to GetNextAddressInPolygon will return another item in the list (another phone number) with the same address. So if there are 5 items in the list, the address will be returned 5 times, each time with a different item in the list. Subsequent calls will then return subsequent addresses.
False	Default unless set in Default property values . Each call to GetNextAddressInPolygon will return one address. The full list (for example of phone numbers) will be returned with this address.

[Note: **ExpandGifLists** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ExpandLevelType property

This property affects the [Australian address record format](#) returned from many of the functions

It can have the following values:

True	The Level type is expanded. Eg “Ground Floor” instead of “G”, or “Level” instead of “L”.
False	Default unless set in Default property values .

[Note: **ExpandLevelType** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ExpandStreetSuffix property

This property affects the [Australian address record format](#) returned from many of the functions

It can have the following values:

True	The street suffix is returned expanded. Eg “North” instead of “N”.
False	Default unless set in Default property values .

[Note: **ExpandStreetSuffix** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ExpandStreetType property

This property affects the [Australian address record format](#) returned from many of the functions

It can have the following values:

True The street type is returned expanded. Eg “Road” instead of “RD”.

False Default unless set in [Default property values](#).

[Note: **ExpandStreetType** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ExpandUnitType property

This property affects the [Australian address record format](#) returned from many of the functions

It can have the following values:

True The unit/flat type is returned expanded. Eg “Unit” instead of “U”, or “Apartment” instead of “APT”.

False Default unless set in [Default property values](#).

[Note: **ExpandUnitType** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

FieldDelimiter property

The character that is used to delimit the fields in the PAF [Australian address record format](#) for the output of the [GetAddresses method](#) and the [GetHintAddresses method](#).

By default this is the comma (“,”), unless set in [Default property values](#).

[Note: **FieldDelimiter** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

GeographicalCodes property

Use this property to set which (if any) geographical codes are to be looked up. These codes will be appended to the address returned from many of the PAF library functions.

This property is comma-delimited list of codes in the format *File.Code*, where *File* is the name of the Geographical Information File (without the .iqg, _L.iqg or _G.iqg extension) and *Code* is the name of the field in the file.

Setting this property is an alternate to calling the [SetGeographicalCodesId function](#).

Some code-fields in a file form a group. In this case, all the code-fields in the group need to be specified, with a plus sign (+) separating them. For example if a file “LatLong” has a group of 3 code-fields Latitude, Longitude and Reliability, then set this property to “LatLong.Latitude+Longitude+Reliability”.

Even if the specified Geographical Codes don’t exist, no error will be returned. However, obviously the codes won’t be returned with the address.

When reading this property, it will simply return the value used in setting it.

GeographicalCodes2 is an alternate Read-Only property, which will return the list of code-fields. However, the file name will not be included and field groups will be broken up. So in the example above, reading the GeographicalCodes property will return “LatLong.Latitude+Longitude+Reliability”, and reading the GeographicalCodes2 property will return “Latitude,Longitude,Reliability”.

[Note: **GeographicalCodes** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

GetSearchStatisticsTimeLimit property

The maximum number of seconds the [GetSearchStatistics function](#) will take. If the time limit is reached before the function has finished, the function will return a “Search time out” error. If this property is set to zero, there will be no time limit.

The default value is 60 (seconds), unless set in [Default property values](#).

[Note: **GetSearchStatisticsTimeLimit** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

GifHouseNumberRange property

This property affects the [GetDpid\(Id\) function](#) and [GetDpidNonAmas\(Id\) function](#).

If the Geocode for a given address doesn’t exist, and it does exist for neighbouring addresses, then the Geocode of neighbouring addresses may be used. Neighbouring addresses will be searched for up to GifHouseNumberRange numbers from the given address.

For example, if the input address was 35 Spring Street, and GifHouseNumberRange is 10, then the neighbouring addresses that will be searched are 37, 39, 41, 43 and 45. Similarly the other way – 33, 31, 29, 27 and 25.

[Note: **GifHouseNumberRange** is the property name to use in the [Properties property](#)]

HouseNumResolveLocality property

This property affects the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#).

It can have the following values:

True	If there is more than one matching street in different localities, and at least one has the given house number, then only those streets with the given house number will be returned. This is the default, unless set in Default property values . For example, a search for “35 Liverpool St, Sydney” may return “35 Liverpool St Paddington” using synonym localities. Even though there is a Liverpool Street in Sydney, but there is no number 35. So the house number 35 changed/chose the locality to return.
False	The input house number will not cause a match to a synonym or border locality if the street exists in the given locality. Instead, all matching streets will be returned.

This property is only applicable if the [ResolveGroupAmbiguity property](#) is False.

[Note: **HouseNumResolveLocality** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

LatLongSearchRange property

The number of meters tolerance when searching by Latitude and Longitude.

In other words, all addresses within this number of meters from the given Latitude and Longitude will be returned.

A Latitude/Longitude search can be performed using the [GetAddress\(Id\) function](#) or [GetAddress method](#).

[Note: **LatLongSearchRange** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ListAllPoints property

This property affects the listing of addresses in a particular street.

It can have the following values:

True	All addresses in this street will be returned, including all unit/level numbers for each address.
False	Only the Primary Points (individual house/street addresses with no unit or level numbers) in this street will be returned. To list the units of levels for a particular address, another call to the search or listing function needs to be made, specifying the house/street number. Default unless set in Default property values .

See also [ListSuffixesSeparate property](#).

[Note: **ListAllPoints** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ListAlternateStreetNames property

This property determines whether alternate street names will be listed, when listing streets in a locality.

It can have the following values:

True	Default unless set in Default property values .
False	Alternate street names are not listed.

[Note: **ListAlternateStreetNames** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)].

ListDeeperIfOnly1Found property

This property affects the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#).

It can have the following values:

True	If the result of the search is only one locality, returns all streets and postal types in the locality; if the result is only one street, returns all the houses in the street; if the result is only one house/street number, returns all units and levels for this street number (if applicable). Default unless set in Default property values .
False	To list the sub-addresses of the partial address found, call the GetAddress method , GetAddresses method , or GetAddress(Id) function .

[Note: **ListDeeperIfOnly1Found** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ListDeeperOnBestMatch property

This property affects the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#). This property extends the [ListDeeperIfOnly1Found property](#), and so only applies if the latter is also true.

It can have the following values:

True	If the result of the search is a list of localities, with one locality matching exactly, then “list deeper” in this locality – return all streets and postal types in this locality. Similarly, if the result of the search is a list of streets with or without house numbers, and one street matches much better than the others, then “list deeper” in this match.
False	Default unless set in Default property values .

[Note: **ListDeeperOnBestMatch** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ListDeliveryIDParent property

This property affects the listing of Groups in a given Locality where the Locality has a Delivery ID.

It can have the following values:

- | | |
|-------|---|
| True | The Locality (on its own without any Group information) will be listed at the beginning of the list of Groups. Similarly when listing Points of a given Group, where the Group has a Delivery ID. Default unless set in Default property values . |
| False | The Group (on its own without any Point information) will be listed at the beginning of the list of Points. |

[Note: **ListDeliveryIDParent** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)].

ListSuffixesSeparate property

This property affects the listing of addresses in a particular street. See also [ListAllPoints property](#).

It can have the following values:

- | | |
|-------|---|
| True | House/street numbers with a suffix will be listed separately, e.g. 3, 3A and 3B. Default unless set in Default property values . |
| False | Only the house/street number will be listed (3 in the above example). To list the addresses with the suffixes, call the search or listing function, specifying the house/street number. |

[Note: **ListSuffixesSeparate** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)].

ListSynonyms property

This property determines whether [Synonym Localities](#) will be listed, when listing addresses by postcode.

It can have the following values:

- | | |
|-------|--|
| True | Synonym localities will be listed. Default unless set in Default property values . |
| False | Synonym localities will not be listed. |

[Note: **ListSynonyms** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)].

LoneNumberIsPostal property

This property affects the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#).

It can have the following values:

- | | |
|-------|---|
| True | If the address line contains only a number, it will be understood as a postal (PO Box) number. This is the default, unless set in Default property values . |
| False | A lone number on the address line is ignored. |

[Note: **LoneNumberIsPostal** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)].

MaxAddressesReturned property

This is the maximum number of addresses returned by one of the PAF searching functions ([GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#)).

The default value is 100, unless set in [Default property values](#).

[Note: **MaxAddressesReturned** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)].

OnlyReturnGeocodes property

This property affects the records returned by the [GetNextAddressInPolygon function](#) and must be set before calling [BeginPolygonSearch function](#). The addresses returned by the Polygon Search function include other geographical data as specified in the [GeographicalCodes property](#).

It can have the following values:

True	Each call to GetNextAddressInPolygon function will return the geographical information corresponding to the address but without the address.
False	Each call to GetNextAddressInPolygon function will return an address with the corresponding geographical information. Default unless set in Default property values .

[Note: **OnlyReturnGeocodes** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

OutputType property

This property determines the format of the returned addresses.

It can have the following values:

XML	Addresses are returned in XML format.
Delimited	Addresses are returned in tab-delimited format (see Australian address record format). Default unless set in Default property values .

[Note: **OutputType** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

PolygonLongLatOrder property

This property affects the [BeginPolygonSearch function](#) and the [GetSearchStatistics function](#).

It can have the following values:

True	The polygon coordinates are to be given as Longitude, Latitude.
False	The polygon coordinates are to be given as Latitude, Longitude. Default unless set in Default property values .

[Note: **PolygonLongLatOrder** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

PolygonSearchFilter property

Setting this property filters the addresses returned by [GetNextAddressInPolygon function](#). Addresses can be filtered by values in the geographical code fields.

Example

The examples below show values for the **PolygonSearchFilter** property for two geographical information files (GIF), one named PhoneGif with a field PhoneTypes, and another named LatLong with a field Reliability

```
PhoneGif.PhoneTypes = "M"  
(PhoneTypes="M") OR (PhoneTypes='L')  
(Reliability < 4) AND ((PhoneTypes="M") OR (PhoneTypes="L"))  
PhoneTypes <> "F"  
(PhoneTypes="M") OR (PhoneTypes="L") OR (PhoneTypes="")
```

Notes

The [GeographicalCodes property](#) must be set before setting **PolygonSearchFilter**.

Setting **PolygonSearchFilter** may return an error if it cannot be correctly parsed.

Field names and comparisons are case-insensitive.

Either single-quotes (') or double-quotes (") may be used.

Field names can be qualified (e.g. PhoneGif.PhoneTypes) or unqualified (e.g. PhoneTypes). However, if there are two fields with the same names but in different GIF files, then they need to be qualified with the file name.

Parenthesis are required when using OR or AND to specify operator precedence.

If the [ExpandGifLists property](#) is true, then the individual list item will be used. If it is false, then the full field will be used.

[Note: **PolygonSearchFilter** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

PutDidInDpid property

This property determines whether the DPID field will be populated with the Group Delivery ID or Locality Delivery ID in the case where there is no DPID.

It can have the following values:

- | | |
|-------|---|
| True | DPID field is populated with Group Delivery ID or Locality Delivery ID. |
| False | DPID field is not populated with Group Delivery ID or Locality Delivery ID. Default unless set in Default property values . |

[Note: **PutDidInDpid** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)].

RapidCodeDump property

This property affects the [GetAddress method](#), [GetAddresses method](#), [GetAddress\(Id\) function](#), [GetHintAddress method](#), [GetHintAddresses method](#) and [GetAddressHint\(Id\) function](#).

It can have the following values:

- | | |
|-------|--|
| True | Geocode dumping is performed to return geographical information. This means that adjacent addresses or street or locality level addresses are looked at to provide a geocode for the address if it doesn't have one. This is the default unless set in Default property values . |
| False | Geocode dumping is not done. |

[Note: **RapidCodeDump** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

RecordDelimiter property

This property determines the character used to delimit the records for the output of the [GetAddresses method](#) and the [GetHintAddresses method](#).

By default this is the tab character (ASCII 9), unless set in [Default property values](#).

[Note: **RecordDelimiter** is the property name to use in the [Properties property](#)]

ResolveGroupAmbiguity property

This property affects the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#).

It can have the following values:

- True If there is more than one street of the same name in a locality (each with a different street type), then the street/house number will be used to select which street. Default unless set in [Default property values](#).
- For example, a search for “717 Flinders” and “Melbourne” will return “717 Flinders Street”.
- False Street/house number not used to select street.
- For example, a search for “717 Flinders” and “Melbourne” will not return “717 Flinders Street”. This is because there is also a “Flinders Lane” and a “Flinders Court” in Melbourne, even though there is no number 717 in either of these two. Instead, it will return “Flinders Street”, “Flinders Lane” & “Flinders Court”, forcing the user to choose which street.

See also the [HouseNumResolveLocality property](#).

[Note: **ResolveGroupAmbiguity** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ReturnFormattedAddress property

This property determines whether formatted address lines are returned in fields 40 and 41 reserved for this output in the [Australian address record format](#) returned by the following: [DpidToAddress\(Id\) function](#), the [GetAddress\(Id\) function](#), the [GetAddressHint\(Id\) function](#), the [GetDpid\(Id\) function](#), the [GetDpidNonAmas\(Id\) function](#), the [GetNextAddress\(Id\) function](#), the [GetNextAddressHint\(Id\) function](#), the [GetParsedInput\(Id\) function](#), the [DpidToAddress method](#), the [GetAddress method](#), the [GetAddresses method](#), the [GetHintAddress method](#), the [GetHintAddresses method](#) and the [ParsedInput property](#).

It can have the following values:

- True Fields 40 and 41 of [Australian address record format](#) contain formatted output.
- False Fields 40 and 41 not populated. Default unless set in [Default property values](#).

[Note: **ReturnFormattedAddress** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ReturnFormattedAddressOptions property

This property affects the format of the address returned by many of the functions. It only applies if the [ReturnFormattedAddress property](#) is True.

Its value is that of the option parameter in the [FormatAddress\(Id\) function](#). The option 2 (don't include locality info) is automatically set, and options 8(embed synonym) and 16384 (use CrLf) are ignored.

Setting this property to blank will use the default formatting options.

[Note: **ReturnFormattedAddressOptions** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

SearchBorderLocalities property

This property affects the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#). The list of bordering localities used is provided by Australia Post, and is not exhaustive.

It can have the following values:

- True Search is broadened to include localities bordering those initially found.
- False Default unless set in [Default property values](#).

[Note: **SearchBorderLocalities** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

SearchHouseNumAsOther property

This property affects the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#).

It can have the following values:

True	Provided House Number will also be considered as a Lot Number, Unit/Flat Number and Level Number. For example, “3 Main St” might return “3 Main St”, “3/100 Main St”, “Suite 3, 90 Main St”, “Suite 5 Level 3, 90 Main St” and/or “Lot 3 Main St”.
False	Default unless set in Default property values .

Notes

[Note: **SearchHouseNumAsOther** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

SearchPostcodes property

This property affects the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#).

It can have the following values:

True	Search is broadened to include all localities in postcodes found. For example, a search for n syd will find NORTH SYDNEY NSW 2060. Then all other localities with postcode 2060 will be also be found.
False	Default unless set in Default property values .

[Note: **SearchPostcodes** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

SearchStateOnly property

This property affects the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#).

It can have the following values:

True	Search will be limited to the state provided in the locality line. This can be useful if the user knows the street name and state, but is unsure of the locality and postcode.
False	Default unless set in Default property values .

[Note: **SearchStateOnly** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

SwitchAlternateStreetPosition property

This property affects the position of the alternate street name in the [Australian address record format](#) returned from and passed to many of the functions.

It can have the following values:

True	The alternate street name, type and suffix are in field positions 4, 5 & 6. The official street name, type and suffix are in field positions 29, 30 & 31. The alternate street name will be returned by the FormatAddress(Id) function and FormatAddress method .
False	The official street name, type and suffix are in field positions 4, 5 & 6. The alternate street name, type and suffix are in field positions 29, 30 & 31. This is the default unless set in Default property values .

[Note: **SwitchAlternateStreetPosition** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

SwitchSynonymLocalityPosition property

This property affects the position of the synonym locality in the [Australian address record format](#) returned from and passed to many of the functions.

It can have the following values:

True	The synonym locality name is in field position 3, and the official locality name in field position 25.
False	The official locality name is in field position 3, and the synonym locality name in field position 25. This is the default unless set in Default property values .

[Note: **SwitchSynonymLocalityPosition** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

TcpHost property

The name of the remote socket server, either an IP address or the domain/host name. See [Socket Client](#).

If setting this property, set it and the [TcpPort property](#) and [UseSockets property](#) before calling any other function.

This property applies to the PAF library as a whole, not to only a particular Id/handle.

[Note: **TcpHost** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

TcpPort property

The port number of the remote socket server. See [Socket Client](#).

If setting this property, set it and the [TcpHost property](#) and [UseSockets property](#) before calling any other function.

This property will only work when using the [PAF Libraries](#) directly. Do not use it via the [RapidSvr](#) COM library.

This property applies to the PAF library as a whole, not to only a particular Id/handle.

[Note: **TcpPort** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

TimeLimit property

The time limit is the maximum number of seconds RapidSvr or the PAF library functions will spend searching for an address.

This time limit only applies when using the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#), and where no postcode and no or little locality information is provided. In other cases, the search is not likely to take a long time.

The default value is 4, unless set in [Default property values](#).

[Note: **TimeLimit** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

TitleCaseFields property

This property affects the [Australian address record format](#) returned from many of the functions.

It can have the following values:

True	In the individual fields returned, the first letter of each word is in uppercase, and the remaining letters in lowercase. This does not apply to the formatted address lines returned.
False	The individual fields returned are in uppercase. This is the default.

[Note: **TitleCaseFieds** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

UseFindStatus property

This property affects the [GetHintAddress method](#), [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#). It determines how Rapid indicates when not all results are returned (there are too many, or the [TimeLimit property](#) was reached).

It can have the following values:

True	After a call to one of the above methods/functions, the FindStatus property (or the GetFindStatus(Id) function) will indicate whether all results are returned. Default unless set in Default property values .
False	Value only kept for backward compatibility. If not all results are going to be returned, the first call to the GetHintAddress method (or the GetAddressHint(Id) function) will return an empty string. If there are too many results, subsequent calls to the GetHintAddress method (or the GetNextAddressHint(Id) function) will return the found addresses as normal. If the time limit was reached, the second call to GetHintAddress method (or the first call to the GetNextAddressHint function) will return an Empty record .

Notes

[Note: **UseFindStatus** is the property name to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

UseSockets property

This property determines whether the calls made to functions in the [PAF Libraries](#) are redirected to a socket server. See [Socket Client](#).

It can have the following values:

True	Calls are redirected to a socket server.
False	Calls are not redirected to a socket server.

If setting this property, set it and the [TcpHost property](#) and [TcpPort property](#) before calling any other function.

This property will only work when using the [PAF Libraries](#) directly. Do not use it via the [RapidSvr](#) COM library.

This property applies to the PAF library as a whole, not to only a particular Id/handle.

[Note: **UseSockets** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

ValidateAddressOptions property

This property allows setting options for the [ValidateFullAddress method](#) or [ValidateFullAddress\(Id\) function](#).

Currently the only values supported are “CorrectNzSuburb=True” and “CorrectSynonym=True”.

[Note: **ValidateAddressOptions** is the name of the property to use in the [Properties property](#) or [SetProperties\(Id\) function](#)]

PAF Libraries

The PAF Libraries provide access to an Address Reference File, such as the Australian Postal Address Files (the PAF).

They include functions to:

1. Correct an address and allocate it an Address Id (such as the PAF's DPID).
2. Generate barcode strings from the DPID.
3. Search the Address Reference File.
4. Gain direct access to all the addresses in the Address Reference File.

Win32 DLL

The Windows 32-bit DLL version of this library is in the file `iPaf32.dll`.

It is used by several of the supplied grammar files, by [RapidSvr](#), and can be used by [IQ Rapid Address](#) as an alternative to [RapidSvr](#).

The functions in this DLL use the **stdcall** calling convention, whereby the parameters are passed from right to left, and the called function is responsible for removing the parameters off the stack.

C Libraries

A C-Library is available for several other platforms.

A header file "PafLib.h" is provided with this SDK containing the function prototypes.

PAF Libraries Installation

The PAF Libraries `iPaf32.dll` are installed into the System directory during [RapidSvr Installation](#).

Function summary

Note that most functions have in two versions, with and without the "Id". See [ID functions](#).

Note that on platforms other than Windows, all the functions have a prefix of "paf" not shown here in the documentation.

Opening and closing

[OpenPaf function](#)

[OpenPafId function](#)

[OpenPafMem function](#)

[OpenPafMemId function](#)

[OpenArfId function](#)

[ClosePaf function](#)

[ClosePafId function](#)

[ForceCloseAll function](#)

Barcodes & sort plans

[DpidToBarcode function](#)

[BuildBarcode_ function](#)

[CustInfoBarcode function](#)

[GenerateBarcode function](#)

[PostcodeToBsp function](#)

[PostcodeToNpsp function](#)

Direct reading

[ReadLocalityRecord\(Id\) function](#)

[ReadGroupRecord\(Id\) function](#)

[ReadPointRecord\(Id\) function](#)

[ReadSynonymRecord\(Id\) function](#)

[ReadBorderRecord\(Id\) function](#)

[GetLocalityRecordCount\(Id\) function](#)

[GetGroupRecordCount\(Id\) function](#)

[GetPointRecordCount\(Id\) function](#)

[GetSynonymRecordCount\(Id\) function](#)

Searching

[GetAddress\(Id\) function](#)

[GetNextAddress\(Id\) function](#)

[GetAddressHint\(Id\) function](#)

[GetNextAddressHint\(Id\) function](#)

[GetPopupHint\(Id\) function](#)

[GetNextPopupHint\(Id\) function](#)

[GetFindStatus\(Id\) function](#)

[GetParsedInput\(Id\) function](#)

[DpidToAddress\(Id\) function](#)

[GetBorderPostcodes\(Id\) function](#)

[BeginPolygonSearch function](#)

[GetNextAddressInPolygon function](#)

[GetSearchStatistics function](#)

Address correction

[GetDpid\(Id\) function](#)

[GetDpidNonAmas\(Id\) function](#)

[FixLastLine function](#) / FixLastLineId

[ValidateFullAddress\(Id\) function](#)

Miscellaneous

[GetErrorDescription function](#)

[GetProperties\(Id\) function](#)

[SetProperties\(Id\) function](#)

[FormatAddress\(Id\) function](#)

[FormatAddressS function](#)

[SetGeographicalCodesId function](#)

[CallPafFromStan function](#)

ID functions

Most of the functions in the [PAF Libraries](#) have two forms, one ending with “Id” and one without this suffix. We recommend using the newer “Id” functions.

“Id” functions

The “Id” functions allow specifying which Address Reference File (ARF/PAF) to use (see [OpenArfId function](#)). They also allow the retrieval of additional data from a Geographical Information File (see [SetGeographicalCodesId function](#)).

When an Address Reference File is opened, the caller is given an Id or a handle. This Id is then passed to each subsequent function.

In a multi-threaded calling application, each thread should get its own Id (by calling [OpenArfId function](#), [OpenPafId function](#) or [OpenPafMemId function](#)).

The properties set for each Id only apply to that Id.

Non-“Id” functions

The functions without the “Id” suffix use the global Address Reference File specified in the see [PAF Directory](#) registry entry.

In a multi-threaded calling application, each thread will internally be allocated its own Id or handle.

Direct reading functions

The [PAF Libraries](#) provide direct access to the Address Reference Files ([PAF files](#)). Files are accessed via their record number, from 1 to the number of records.

It is an Australia Post requirement that DPIDs are only provided via AMAS approved software. Hence, the DPID is not provided via these functions.

The Address Reference Files must be opened before reading, so the [OpenPaf function](#) or [OpenPafMem function](#) must have been called before calling any of the **ReadFileRecord** or **GetFileRecordCount** functions.

Similarly, the [OpenPafId function](#), [OpenPafMemId function](#) or [OpenArfId function](#) must have been called before calling any of the **ReadFileRecordId** or **GetFileRecordCountId** functions.

See also [ReadLocalityRecord\(Id\) function](#), [ReadGroupRecord\(Id\) function](#), [ReadPointRecord\(Id\) function](#), [ReadSynonymRecord\(Id\) function](#), [ReadBorderRecord\(Id\) function](#), [GetLocalityRecordCount\(Id\) function](#), [GetGroupRecordCount\(Id\) function](#), [GetPointRecordCount\(Id\) function](#), [GetSynonymRecordCount\(Id\) function](#).

Direct reading constants

```
MAX_LOCALITY_RECORD = 55
MAX_GROUP_RECORD    = 80
MAX_POINT_RECORD     = 128
MAX_SYNONYM_RECORD  = 55
```

Function reference

BeginPolygonSearch function

Lists addresses within a given polygon.

Function Definitions

```
int BeginPolygonSearch (
    int id,
    const char *      sPolygon,
```

```
const char *    sStart
);
```

Parameters

id	The Id or handle received when opening the Address Reference File (via the OpenArfId function , the OpenPafId function or the OpenPafMemId function).
sPolygon	Pointer to a null-terminated string that specifies the polygon. This must be a comma-delimited list of latitude/longitude pairs, one pair for each point of the polygon. The latitude is also separated from the longitude by a comma.
sStart	Pointer to a null-terminated string that specifies the starting point. This string must have the format Latitude Comma Longitude. This may be an empty string, in which case the starting point will be the centre of the polygon.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Notes

The Address Reference File (ARF) requires a geocode file (GIF) with Latitude & Longitude. This needs to be set via the [GeographicalCodes property](#).

To retrieve the addresses, repeatedly call the [GetNextAddressInPolygon function](#) until no more results are returned (an empty string is returned).

Addresses closer to the starting point will be returned first.

The points of the polygon must be specified in order. The last point is joined to the first point to draw the closing line of the polygon.

These properties specifically affect the polygon search: [ExpandGifLists property](#), [OnlyReturnGeocodes property](#) & [PolygonSearchFilter property](#).

See also [GetSearchStatistics function](#).

BuildBarcode_ function

This generic function creates a barcode (see [Barcodes](#)).

Syntax

```
int BuildBarcode_(
    const char *    sFcc,
    const char *    sSortingCode,
    const char *    sCustInfo,
    int bCustNCoded,
    char *          sBarcode,
    int *           iBarcodeLen
);
```

Parameters

sFcc	Pointer to a null-terminated string specifying the FCC, value one of “11”, “87”, “45”, “92”, “59”, “62” or “44”.
sSortingCode	8 digit pointer to a null terminated string that specifies the DPID.
sCustInfo	Pointer to a null terminated string that specifies the additional customer information to be included in the barcode.

bCustNCoded	Specifies the Customer Information coding : zero for C-coding or non-zero for N-coding.
sBarcode	Pointer to the buffer to receive the barcode string. This buffer must be at least 38, 53 or 68 characters in length, depending on sFcc.
iBarcodeLen	Pointer to receive the length of the barcode. This will be set to one of 37, 52 or 67.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns Barcode Function Error Codes .

CallPafFromStan function

This function is an interface to some of the other functions in this library. It makes these other functions available to a grammar file, by providing a function with the function signature required by Grammar files.

Syntax

```
int CallPafFromStan (
    char * sInOut,
    int    maxlen
);
```

Parameters

sInOut	Pointer to a null-terminated string containing the input (see below for acceptable inputs). It also serves as a buffer to receive the output.
maxlen	The size of the above buffer, excluding the null-terminator.

Acceptable Inputs

```
OpenArf;Database=sDatabase;
```

Calls the [OpenArfId function](#), passing it the *sDatabase* parameter and an *iOption* parameter of zero. If OpenArfId is successful, the *id* is placed in the *sInOut* buffer, otherwise “Error:” followed by an error message is placed in this buffer. In either case, CallPafFromStan will return zero.

```
CloseArf;Id=id;
```

Calls the [ClosePafId function](#), passing it the *id* parameter. **CallPafFromStan** returns the return value of **ClosePafId**. The *sInOut* buffer is cleared.

```
LookupAddress;Id=id;address
```

Calls the **GetDpidId** function (see [GetDpid\(Id\) function](#)), passing it the *id* parameter and the *address* as the *s* parameter. Include `Lenient=True;` before *address* to call **GetDpidNonAmasId** instead of **GetDpidId**.

```
DpidToAddress;Dpid=iDpid;
```

Calls **DpidToAddress** (see [DpidToAddress\(Id\) function](#)) with the *iDpid* parameter.

```
SetGeographicalCodes;Id=id;codes
```

Calls the [SetGeographicalCodesId function](#) with the given *id* and *codes*.

```
LookupByAddressId;Id=id;addressID
```

Calls the [GetAddress\(Id\) function](#) to lookup an address by its ID. The found address, if any, is returned. This is an alternate to DpidToAddress (above), allowing for a non-numeric and/or longer addressID.

LookupByGeocode; Id=*id*; geocode(*s*)

Calls the [GetAddress\(Id\) function](#) to lookup an address by a geocode. If there is only one address found then it is returned. If more than one address is found then *sInOut* is set to empty.

FormatAddress; Id=*id*; Options=*options*; addressFields

Calls the [FormatAddress\(Id\) function](#).

ClosePaf function

Closes the PAF files.

Syntax

```
int ClosePaf ();
```

Return Value

Zero Function succeeded.

Non-zero Function failed, returns [PAF Error codes](#).

Notes

When the PAF files are no longer needed, call **ClosePaf** to release allocated resources.

For every successful call to the [OpenPaf function](#) or the [OpenPafMem function](#), **ClosePaf** must be called.

ClosePafId function

Closes the Address Reference File.

Syntax

```
int ClosePafId (  
    int id  
) ;
```

Parameters

id The Id or handle received when opening the Address Reference File (via the [OpenArId function](#), the [OpenPafId function](#) or the [OpenPafMemId function](#)).

Return Value

Zero Function succeeded.

Non-zero Function failed, returns [PAF Error codes](#).

Notes

When the Address Reference File is longer needed, call **ClosePafId** to release allocated resources.

ClosePafId must be called for every successful call to the [OpenArId function](#), [OpenPafId function](#) or [OpenPafMemId function](#).

After calling **ClosePafId**, the Id is no longer valid.

If an Address Reference File has been opened more than once, it will only be closed when all handles (Ids) to that Address Reference File have been closed. However, the closing of each handle will release some resources allocated with that handle.

CustInfoBarcode function

This function creates one of the three types of customer barcode (see [Barcodes](#)). It conforms to the format of the functions that StanRt grammar files can call.

Syntax

```
int CustInfoBarcode(
    char * s,
    int    maxlen
);
```

The first parameter is used as both the input and output:

Input

The first 8 characters of the input are the DPID.

For a Standard Customer Barcode (FCC=11), nothing more needs to be in the input.

For a barcode with customer information, the next 2 characters specify the Format Control Code (“59” for Customer barcode 2, or “62” for Customer barcode 3).

The next character (“C” or “N”) specifies the [Customer Information coding](#) used for customer information.

The rest of the input is the customer information.

Example

1234567852Cabdcde

Will create a 52-digit barcode Customer Barcode 2 (FCC = 59) for a DPID of 12345678, with customer information of “abcde”, C-encoded.

Output

The output will be the barcode character sequence.

Maxlen is the maximum number of characters (besides the null-terminator) that will be returned to the output string.

Return Value

Zero Function succeeded.

Non-zero Function failed, returns [Barcode Function Error Codes](#)

DpidToAddress(Id) function

This function will look up the Address Identifier in the Address Reference File, and return the address with the given Address Identifier.

For the PAF, the Address Identifier is the DPID.

Syntax

```
int DpidToAddress(
    long    iDpid,
    char * sResult,
    int     maxlen
);

int DpidToAddressId(
    int     id,
    long    iDpid,
    char * sResult,
    int     maxlen
);
```

Parameters

id The Id or handle received when opening the Address Reference File (via the [OpenArId function](#), the [OpenPafId function](#) or the [OpenPafMemId function](#)).

iDpid	The DPID or Address Identifier.
sResult	Pointer to the buffer to receive the found address. This is a delimited string of address fields, as described in the Australian address record format section. If there is no address with this DPID, this will be an Empty record .
maxlen	The size of the above buffer.

Return Value

Zero	Function succeeded.
Non-zero	Function failed.

Notes

The Address Reference Files need to be open to call **DpidToAddress** (see the [OpenPaf function](#)).

For this function to work, the additional index file “Dpid.in1” or “AddrId.hin” must be in the Address Reference File directory, else an error 672 will be returned.

To search for a non-numeric AddressId, use the [GetAddress\(Id\) function](#), passing it the AddressId only. **GetAddress(Id)** will also work for a numeric AddressId.

DpidToBarcode function

This function creates the standard customer barcode – with no customer information (see [Barcodes](#)).

It conforms to the format of the functions that StanRt grammar files can call.

Syntax

```
int DpidToBarcode(
    char * s,
    int    maxlen
);
```

The first parameter is used as both the input and output.

The input is the 8-digit dpid.

The output is the 37-digit barcode sequence.

Maxlen is the maximum number of characters that will be returned to the output string. Should be at least 37 for the function to work. The size of the buffer *s* must be at least *Maxlen* + 1.

Return Value

Zero	Function succeeded.
Non-zero	Returns Barcode Function Error Codes

FixLastLine function

This function corrects the locality information of an address. It is used by the [Dpid3Pass](#) grammar file.

Syntax

```
int FixLastLine (
    char * s,
    int    maxlen
);

int FixLastLineId (
    int    id,
    char * s,
    int    maxlen
);
```

Parameters

id	The Id or handle received when opening the Address Reference File.
s	Pointer to a null-terminated string that specifies the address elements. It also serves as a buffer in which the result is placed.
maxlen	The size of the above buffer, excluding the null-terminator.

Return Value

Zero	Function succeeded.
Non-zero	Function failed.

Input and output format

The input and output format of the *s* parameter of this function is a tab delimited string with the following fields: State, Postcode, Locality Name, Street Name, Street Type, Street Suffix, Postal Type, Correction flag. If desired, only the first 3 fields can be passed to this function. Give the Correction flag a value of "1" if the locality is to determine the state over the postcode (e.g. "Sydney 3000" will put a state of NSW), otherwise leave this flag blank or "0". The street information is sometimes used in determining the correct locality.

Notes

The PAF files need to be open to call **FixLastLine** (see the [OpenPaf function](#)).

ForceCloseAll function

Closes all Address Reference Files.

Syntax

```
int ForceCloseAll ();
```

Return Value

Zero	Always returns zero.
------	----------------------

Notes

The use of this function is not recommended, as it will close all opened Address Reference Files and invalidate all handles/Ids.

FormatAddress(Id) function

This function creates a formatted address from the address fields returned from many of the PAF Library functions. See also [FormatAddressS function](#).

Syntax

```
int FormatAddress (  
    const char *    sIn,  
    char *          sOut,  
    long            iOptions  
);  
  
int FormatAddressId (  
    int id,  
    const char *    sIn,  
    char *          sOut,  
    long            iOptions,  
    int maxlen  
);
```

Parameters

<code>id</code>	The Id or handle received when opening the Address Reference File
<code>sIn</code>	Pointer to a null-terminated string containing the address fields in tab-delimited Australian address record format .
<code>sOut</code>	Pointer to the buffer to receive the formatted address.
<code>iOptions</code>	An integer specifying the formatting options (see Option Values below).
<code>maxlen</code>	The length in characters of the <code>sOut</code> buffer.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Option values

1	address on a single line
2	don't include locality information (locality name, state & postcode)
4	expand unit/level abbreviations
8	embed synonym name
16	expand street type & suffix abbreviations
32	upper case
128	unit and level info on separate line to street number and name (except for unit number followed by a slash, eg "7/35 spring St")
256	don't include building name in address lines.
512	always have unit type. Eg. "7/35 Spring St" becomes "Unit 7 35 Spring St"
1024	don't include lot number if have street number
2048	custom format address using "_FormatAddress.grm"
4096	quote building name
16384	lines divided by CR (carriage return) and LF (line feed), instead of just a LF
32768	after expanding abbreviations (if options 4 and/or 16 are set) and changing case, the input is passed through the RapidFormat.grm system grammar file.
65536	locality in Title Case. Ignored if option 2 or 32 set.

These options can be combined by adding them together (technically using bitwise-OR).

Notes

The option 32768 is only available to **FormatAddressId**.

When using **FormatAddressId**, *sIn* must be delimited by [Delimiter property](#). When using **FormatAddress**, *sIn* must be delimited by the tab character.

FormatAddressS function

This function creates a formatted address from the address fields returned from many of the PAF Library functions.

It is equivalent to the [FormatAddress\(Id\) function](#), except that conforms to the format of the functions that StanRt grammar files can call, and (hence) options cannot be specified.

Syntax

```
int FormatAddress (
```



```
char * s,
int    maxlen
);
```

Parameters

s Pointer to a null-terminated string containing the address fields in tab-delimited [Australian address record format](#). It also serves as a buffer to receive the formatted address.

maxlen The size of the above buffer, excluding the null-terminator.

Return Value

Zero Function succeeded.

Non-zero Function failed, returns [PAF Error codes](#).

GenerateBarcode function

This function creates one of the three customer [Barcodes](#). It conforms to the format of the functions that StanRt grammar files can call.

Syntax

```
int GenerateBarcode(
    char *    s,
    int maxlen
);
```

The first parameter is used as both the input and output:

Input

The first 8 characters of the input specify the DPID.

The next 2 characters specify the Format Control Code (“11” for standard customer barcode, “59” for Customer barcode 2, or “62” for Customer barcode 3).

The next character (“C” or “N”) specifies the [Customer Information coding](#) used for customer information. This must be specified, even if there is no customer information and the standard customer barcode “11” is used.

The remaining characters specify the customer information.

Example

1234567859Cabdcde

Creates a 52-digit barcode Customer Barcode 2 (FCC = 59) for a DPID of 12345678, with customer information of “abcde”, C-encoded.

Output

The output will be the barcode character sequence.

Maxlen is the maximum number of characters (excluding the null-terminator) that will be returned to the output string.

Return Value

Zero 0

The return value is always zero. This is useful when called by a StanRt grammar file, as standardising is aborted on errors.

The output is a zero length string as a result of one of any errors such as the following:.

Errors include:

Barcode longer than maxlen
 Invalid dpid (not 8 digits)
 Invalid FCC (not "11", "59" or "62")
 Invalid coding (not "C" or "N")
 User information too long to fit into barcode
 User information includes characters that cannot be encoded.

GetAddress(Id) function

List addresses from the Address Reference File based on some criteria.

In particular, it can list one of the following:

- All localities
- All localities with a given postcode
- All localities in a given state (or NZ City)
- All locality names with a given beginning
- All streets and postal types (e.g. PO Box) in a given locality
- All house numbers in a given street
- All units and levels for a given house number
- Addresses matching given unit/level, for a given street address
- All postal numbers for a given postal type
- All street names with a given beginning
- All streets with a given name, and optionally a given street type beginning and/or locality name beginning.
- All streets in a given postcode
- The address having a given DPID or AddressId.
- All addresses having a given Geographical code.

Function Definitions

```
int GetAddress (
    const char *    sInput,
    char *          sOutput
);
```

```
int GetAddressId (
    int id,
    const char *    sInput,
    char *          sOutput,
    int maxlen
);
```

Parameters

id	The Id or handle received when opening the Address Reference File (via the OpenArfId function , the OpenPafId function or the OpenPafMemId function).
sInput	Pointer to a null-terminated string that specifies the address criteria to find. This is a delimited string of address fields, as described in Input record .
sOutput	Pointer to the buffer to receive the first found address. This is a delimited string of address fields, as described in Output record . If there are no matching addresses, this will be an Empty record .

maxlen The size in characters of the sOutput buffer.

Return Value

Zero Function succeeded.

Non-zero Function failed, returns [PAF Error codes](#).

Notes

The Address Reference File needs to be open (via the [OpenPaf function](#) or [OpenPafMem function](#)) before calling **GetAddress**.

Call the [GetNextAddress\(Id\) function](#) to retrieve subsequent addresses from the list.

The **List** tab on the **IQ Rapid Address** application uses this function.

[RapidSvr](#) provides a similar function – the [GetAddress method](#).

GetAddressHint(Id) function

Search the Address Reference File for addresses or partial addresses based on some criteria.

Function Definitions

```
int GetAddressHint (
    const char *    sLocalityLine,
    const char *    sAddressLine,
    char *          sOutput
);
```

```
int GetAddressHintId (
    int id,
    const char *    sLocalityLine,
    const char *    sAddressLine,
    char *          sOutput,
    int maxlen
);
```

Parameters

id The Id or handle received when opening the Address Reference File (via the [OpenArId function](#), the [OpenPafId function](#) or the [OpenPafMemId function](#)).

sLocalityLine Pointer to a null-terminated string that specifies some locality information (see [Locality Line](#)).

sAddressLine Pointer to a null-terminated string that specifies some address information. (see [Address Line](#)).

sOutput Pointer to the buffer to receive the first found address. This is a delimited string of address fields, as described in [Australian address record format](#). If there are no matching addresses, this will be an [Empty record](#).

maxlen The size in characters of the sOutput buffer.

Return Value

Zero Function succeeded.

Non-zero Function failed, returns [PAF Error codes](#).

Input parameters

To see the results of different input parameters, try the “IQ Rapid Address” application.

The **Address** line and **Locality** line on the **Search** tab are passed directly to **GetAddressHint(Id)**.

Notes

The Address Reference File needs to be open (via the [OpenPaf function](#) or [OpenPafMem function](#)) before calling **GetAddressHint**.

Call the [GetNextAddressHint\(Id\) function](#) to retrieve subsequent found addresses.

The **Search** tab on the **IQ Rapid Address** application uses this function.

[RapidSvr](#) provides a similar function – the [GetHintAddress method](#).

GetDpid(Id) function

This function is used by grammar files (such as DpidGeocode.grm) to look up an address against the PAF (or other Address Reference File), and allocate it a DPID (or Address Identifier).

Syntax

```
int GetDpid(  
    char * s,  
    int    maxlen  
);  
  
int GetDpidId(  
    int    id,  
    char * s,  
    int    maxlen  
);
```

Parameters

id	The Id or handle received when opening the Address Reference File
.s	Pointer to a null-terminated string that specifies the address criteria to look up (see GetDpid Input format . It also serves as a buffer in which the result is placed (in Australian address record format).
maxlen	The size of the above buffer, excluding the null-terminator.

Return Value

Zero	Function succeeded.
Non-zero	Function failed.

Notes

The PAF files need to be open to call **GetDpid** (see the [OpenPaf function](#)).

This function has been designed to be used only in conjunction with the DpidGeocode grammar file.

See also the [GetDpidNonAmas\(Id\) function](#).

GetDpid Input format

The input to the [GetDpid\(Id\) function](#) and the [GetDpidNonAmas\(Id\) function](#) is a tab-delimited string of address fields as described in [Australian address record format](#) with the exception that the last four fields are replaced as listed and described below.

Existing	Replacement
Primary Point Flag	ModifyFlag
Synonym Locality Flag	Locality2
Synonym Locality	StreetType2
Final Address Flag	FullStreetName

ModifyFlag

ModifyFlag determines which components of the input address are left as is, and which components are corrected:

- 2 for a postal address of type MS, RMB, RMS or RSD, keep the street address information from the input address, as this is often useful in delivering the mail. Otherwise the street address information is not returned.
- 4 keep input locality name if it is a valid synonym of the found address. Otherwise the synonym is replaced with the real locality name.
- 8 keep input locality name if it exists in the postcode of the found address. Otherwise the locality name is corrected.
- 32 keep input (alternate) street name. Otherwise the input street name is replaced with the street name as it appears in the PAF.
- 64 fix last line if cannot file DPID.
- 1024 don't change state if postcode confirms state.

These values may be logically OR'ed together.

Locality2

If two localities appear in the input address, place the second one here.

StreetType2

If the input contains apparently two street types (e.g. "High Street Road") then pass "High Street" in the StreetName field, "RD" in the StreetType field and "ST" in the StreetType2 field. This will check for a match with "High Street RD", "High RD" and "High ST".

FullStreetName

Pass the full street name (street name type and suffix) in this field. This will check for a match with streets whose name includes a street type. For example, if the input contains "4 Castle Hill", pass "Castle" in StreetName, "HILL" in StreetType and "Castle Hill" in FullStreetName. This will check for a match with "Castle Hill" as a street name (for example in "Castle Hill RD").

GetDpidNonAmas(Id) function

This function is used by the grammar files (such as AddressEnhancer and indirectly Dpid3Pass) to look up an address against the PAF (or other Address Reference File) and correct it.

It is equivalent to the [GetDpid\(Id\) function](#) except it doesn't abide by all AMAS rules. This results in more often finding a match to an address in the PAF. However, one is less sure that this is the desired address.

For example, GetDpid will not match to an address if both the state and postcode are missing; GetDpidNonAmas will.

GetErrorDescription function

This function provides a description of an error from the error number which most functions in the PAF Libraries return on error.

Sometimes, information specific to the last occurring error will be included in the error description. After calling this function, the information about the last occurring error will be removed.

Syntax

```
int GetErrorDescription (  
    char * sDescription,  
    int    iError,
```

```
int    maxlen
);
```

Parameters

sDescription Pointer to the buffer to receive the description of the error.

iError The error number returned from a call to another PAF Library function.

maxlen The length in characters of the above buffer, excluding the null-terminator.

Return Value

Zero Function succeeded. Always zero.

If the error number is not recognised, then a zero-length string will be placed in the buffer.

GetFindStatus(Id) function

Returns the status of the previous call to the [GetAddressHint\(Id\) function](#).

Syntax

```
int Get_FindStatus ();
int GetFindStatusId (int id);
```

Parameters

id The Id or handle received when opening the Address Reference File.

Return Value

- 0 OK, all results displayed
- 1 Exceeded [MaxAddressesReturned property](#), not all results displayed
- 2 Exceeded [TimeLimit property](#), not all results displayed

Notes

GetFindStatus(Id) will return an undetermined value if the [UseFindStatus property](#) is **False**.

GetGroupRecordCount(Id) function

These functions return the number of records in the Group file of the Address Reference File.

Syntax

```
int GetGroupRecordCount ();
int GetGroupRecordCountId (int id);
```

Parameters

id The Id or handle received when opening the Address Reference File.

Return Value

- Zero Function failed.
- Non-zero Function succeeded.

Notes

See [Direct reading functions](#).

GetLocalityRecordCount(Id) function

These functions return the number of records in the Locality file of the Address Reference File.

Syntax

```
int GetLocalityRecordCount ();
```

```
int GetLocalityRecordCountId (int id);
```

Parameters

id The Id or handle received when opening the Address Reference File.

Return Value

Zero Function failed.

Non-zero Function succeeded.

Notes

See [Direct reading functions](#).

GetNextAddress(Id) function

Retrieves subsequent addresses after a call to the [GetAddress\(Id\) function](#). Each time **GetNextAddress(Id)** is called, the next address is retrieved.

Function Definitions

```
int GetNextAddress (
    char *    sOutput
);

int GetNextAddressId (
    int id,
    char *    sOutput,
    int maxlen
);
```

Parameters

id The Id or handle received when opening the Address Reference File (via the [OpenArId function](#), the [OpenPafId function](#) or the [OpenPafMemId function](#)).

sOutput Pointer to the buffer to receive the address. This is a delimited string of address fields, as described in [Output record](#). If there are no more matching addresses, this will be an [Empty record](#).

maxlen The size in characters of the sOutput buffer.

Return Value

Zero Function succeeded.

Non-zero Function failed, returns [PAF Error codes](#).

GetNextAddressHint(Id) function

Retrieves subsequent found address after a call to [GetAddressHint\(Id\) function](#). Each time **GetNextAddressHint(Id)** is called, the next found address is retrieved.

Function Definitions

```
int GetNextAddressHint (
    char *    sOutput
);

int GetNextAddressHintId (
    int id,
    char *    sOutput
    int maxlen
);
```

Parameters

id	The Id or handle received when opening the Address Reference File (via the OpenArfId function , the OpenPafId function or the OpenPafMemId function)
.sOutput	Pointer to the buffer to receive the address. This is a delimited string of address fields, as described in Australian address record format . If there are no more matching addresses, this will be an Empty record .
maxlen	The size in characters of the sOutput buffer.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

GetNextAddressInPolygon function

Retrieve the next address in the polygon specified in the [BeginPolygonSearch function](#).

Function Definitions

```
int GetNextAddressInPolygon (  
    int id,  
    char *    sOutput,  
    int maxlen  
);
```

Parameters

id	The Id or handle received when opening the Address Reference File (via the OpenArfId function , the OpenPafId function or the OpenPafMemId function)
.sOutput	Pointer to the buffer to receive the address. This is a delimited string of address fields, as described in Australian address record format . If there are no more addresses, this will be an empty string.
maxlen	The size in characters of the sOutput buffer (excluding the null-terminator - till version 5.4.164).

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

GetNextPopupHint(Id) function

Retrieves subsequent popup strings after a call to [GetPopupHint\(Id\) function](#). Each time [GetNextPopupHint\(Id\)](#) is called, the next popup string is retrieved.

Function Definitions

```
int GetNextPopupHint (  
    char *    sOutput,  
    int maxlen  
);  
  
int GetNextPopupHintId (  
    int id,  
    char *    sOutput,  
    int maxlen  
);
```

Parameters

id	The Id or handle received when opening the Address Reference File (via the OpenArfId function , the OpenPafId function or the OpenPafMemId function)
.sOutput	Pointer to the buffer to receive the popup string. If there are no more results, this will be an empty string.
maxlen	The size in characters of the sOutput buffer.

GetParsedInput(Id) function

Retrieves the input of the previous call to the [GetAddressHint\(Id\) function](#) or [GetPopupHint\(Id\) function](#).

This input was in the *sAddressLine* and *sLocalityLine* parameters, and is returned here parsed into the standard [Australian address record format](#).

Syntax

```
int Get_ParsedInput (
    char *      s
);

int GetParsedInputId (
    int id,
    char *      s,
    int maxlen
);
```

Parameters

id	The Id or handle received when opening the Address Reference File (via the OpenArfId function , the OpenPafId function or the OpenPafMemId function)
.s	Pointer to the buffer to receive the parsed input.
maxlen	The size in characters of the s buffer.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

GetPointRecordCount(Id) function

These functions return the number of records in the Point file of the Address Reference File.

Syntax

```
int GetPointRecordCountId ();
int GetPointRecordCountId (int id);
```

Parameters

id	The Id or handle received when opening the Address Reference File.
----	--

Return Value

Zero	Function succeeded.
Non-zero	Function failed.

Notes

See [Direct reading functions](#).

GetPopupHint(Id) function

This method is used by IQ Rapid Address to provide the user with a popup list to help the user complete the address.

Function Definitions

```
int GetPopupHint (
    const char *    sLocalityLine,
    const char *    sAddressLine,
    int iOptions,
    char *    sOutput,
    int maxlen
);

int GetPopupHintId (
    int id,
    const char *    sLocalityLine,
    const char *    sAddressLine,
    int iOptions,
    char *    sOutput,
    int maxlen
);
```

Parameters

id	The Id or handle received when opening the Address Reference File (via the OpenArfId function , the OpenPafId function or the OpenPafMemId function)
sLocalityLine	Pointer to a null-terminated string that specifies some locality line information.
sAddressLine	Pointer to a null-terminated string that specifies some address line information.
iOptions	An integer specifying options (see below)
sOutput	Pointer to the buffer to receive the popup string. If there are no results, this will be an empty string.
maxlen	The size in characters of the sOutput buffer.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Option values

- 1 popup information for the locality line
- 2 popup information for the address line (mutually exclusive with 1 above)
- 4 don't list building names (used with option 2 above)
- 8 don't list postal types (used with option 2 above)
- 16 don't list level types (used with option 2 above)
- 32 don't list flat/unit types (used with option 2 above)
- 64 don't list street names (used with option 2 above)

These options can be combined by adding them together (technically using bitwise-OR).

Notes

The Address Reference File needs to be open (via the [OpenPaf function](#) or [OpenPafMem function](#)) before calling **GetPopupHint**.

Call the [GetNextPopupHint\(Id\) function](#) to retrieve subsequent found popup strings.

[RapidSvr](#) provides a similar function – the [GetPopupHint method](#).

GetProperties(Id) function

Retrieves specific information about the PAF Libraries and its current status.

Syntax

```
int GetProperties (
    const char *    sName,
    char *          sValue,
    int maxlen
);

int GetPropertiesId (
    int id,
    const char *    sName,
    char *          sValue,
    int maxlen
);
```

Parameters

id	The Id or handle received when opening the Address Reference File
.sName	Pointer to a null-terminated string that specifies the name of the property. (See Read-Only properties and Read/Write properties)
sValue	Pointer to the buffer to receive the property value.
maxlen	The length in characters of the above buffer.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Notes

When using **GetProperties** in a multi-threaded application, some properties will be specific to the calling thread. In other words, the property value may be different depending on which thread calls the function.

When using **GetPropertiesId**, some properties will be specific to the passed Id. In other words, the property value may be different depending on which Id is passed to the function.

The *id* parameter is ignored for the [UseSockets property](#), [TcpHost property](#) and [TcpPort property](#). So one doesn't need to open the Address Reference File to read these properties.

GetSearchStatistics function

Retrieves statistics about the results that would be returned from a given search function.

Syntax

```
int GetSearchStatistics (
    int id,
    const char *    sSearch,
    const char *    sStatistic,
    const char *    sParam1,
    const char *    sParam2,
    char *          sOutput,
    int maxlen
);
```

Parameters

id	The Id or handle received when opening the Address Reference File.
----	--

<code>sSearch</code>	The name of the search function.
<code>sStatistic</code>	The name of the statistic(s) to retrieve.
<code>sParam1</code>	A parameter that would be parsed to the search function.
<code>sParam2</code>	A parameter that would be parsed to the search function.
<code>sOutput</code>	Pointer to the buffer to receive the statistic(s).
<code>maxlen</code>	The length in characters of the above buffer (excluding the null-terminator - till version 5.4.164).

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Notes

The [GetSearchStatisticsTimeLimit property](#) may be set to limit the time taken by this function.

The parameters (`sParam1` & `sParam2`), the requested statistic (`sStatistic`) and the output depends on the search function (`sSearch`) as described below. Currently the only search function supported is [Polygon](#).

Polygon search function

To use the [GetSearchStatistics function](#) for the Polygon search function to return the number of values that would be returned for a given search using the [BeginPolygonSearch function](#) and [GetNextAddressInPolygon function](#), set the `sSearch` parameter to “Polygon”.

Parameters

The `sStatistic` parameter can be one or more of the following statistics to retrieve separated by a space:

Count	Returns the number of addresses that would be returned by the polygon search.
Count(<i>GifFile.GifField</i>)	
Count(<i>GifField</i>)	Returns the number of non-null occurrences of the given GIF field that would be returned.
CountList(<i>GifFile.GifField</i>)	
CountList(<i>GifField</i>)	Returns the sum of the number of individual items in the given GIF field. This only applies if the GIF Field is a list field, otherwise “CountList(…)” will return the same as “Count(…)”.

Set `sParam1` to the value of the `sPolygon` parameter of `BeginPolygonSearch`.

Set `sParam2` to the value of the `sStart` parameter of `BeginPolygonSearch`.

Examples

`sStatistic` has a value of “Count”

`sOutput` parameter returns the number of addresses that would be returned from the polygon search

`sStatistic` has a value of “Count(Field1)”

The number of addresses with a non-null GIF field Field1 will be returned.

`sStatistic` has a value of “Count Count(Field1)”

The above two counts will be returned, separated by a space.

Properties

The [ExpandGifLists property](#) and [PolygonSearchFilter property](#) affect the results of this function.

The [SetGeographicalCodesId function](#) needs to be called (or the [GeographicalCodes property](#) set) with the GIF containing the Latitude & longitudes, as well as other GIF fields for which statistics are required.

GetSynonymRecordCount(Id) function

These functions return the number of records in the Synonym file of the Address Reference File.

Syntax

```
int GetSynonymRecordCount ();  
int GetSynonymRecordCountId (int id);
```

Parameters

id The Id or handle received when opening the Address Reference File.

Return Value

Zero Function failed.
Non-zero Function succeeded.

Notes

See [Direct reading functions](#).

OpenArfId function

Opens the Address Reference File (ARF / PAF).

Syntax

```
int OpenArfId (  
    int *          id,  
    int            iOption,  
    const char *sDatabase  
);
```

Parameters

id Pointer to the variable to receive the Id or handle. This Id is only valid if the function succeeds.

iOption This determines whether the Address Reference File is loaded into memory. It can be one of 0 (Registry), 1 (Disk), 2 (Memory) or 3 (Mapping), see [ARF loading options](#). It will be ignored if the Address Reference File is already open.

sDatabase Pointer to a null-terminated string that specifies which Address Reference File to open. See [ARF file naming](#).

Return Value

Zero Function succeeded.
Non-zero Function failed, returns [PAF Error codes](#).

Notes

The Id or handle received from this function can be used in other functions in the PAF library.

For each successful call to `OpenArfId`, call the [ClosePafId function](#) to release the handle.

If `OpenArfId` is called more than once with the same database name, then the database/ARF itself will be opened only once, but a new handle will be provided for each call.

It is not safe to share the handle between simultaneously executing threads.

This function is same as the [OpenPafMemId function](#) with the extra *sDatabase* parameter.

OpenPaf function

Opens the Address Reference File (ARF / PAF).

Syntax

```
int OpenPaf ();
```

Return Value

Zero Function succeeded.

Non-zero Function failed, returns [PAF Error codes](#).

Notes

Call this function or one of the other open functions [OpenPafMem function](#), [OpenPafId function](#), [OpenPafMemId function](#), [OpenArfId function](#) before calling other functions to access the PAF.

For every successful call to `OpenPaf`, the [ClosePaf function](#) must also be called.

To control how the PAF files are loaded into memory, use the [OpenPafMem function](#). `OpenPaf` is equivalent to `OpenPafMem(0)`.

OpenPafId function

Opens the Address Reference File (ARF / PAF).

Syntax

```
int OpenPafId (  
    int * id  
);
```

Parameters

`id` Pointer to the variable to receive the Id or handle. This Id is only valid if the function succeeds.

Return Value

Zero Function succeeded.

Non-zero Function failed, returns [PAF Error codes](#).

Notes

The Id or handle received from this function can be used in other functions in the PAF library.

For each successful call to `OpenPafId`, call the [ClosePafId function](#) to release the handle.

If `OpenPafId` is called more than once, then the Address Reference File will be opened only once, but a new handle will be provided for each call.

It is not safe to share the handle between simultaneously executing threads.

This function is same as the [OpenPafId function](#) with the extra *iOption* parameter.

To specify whether the Address Reference File is to be loaded or mapped into memory, use the [OpenPafMemId function](#).

To specify which Address Reference File to use, use the [OpenArfId function](#).

`OpenPafId(id)` is equivalent to the following functions:

```
OpenPafMemId(id, 0)
```

```
OpenArfId(id, 0, "") .
```

OpenPafMem function

Opens the PAF files.

Syntax

```
int OpenPafMem (  
    int    iOption  
);
```

Parameters

<code>iOption</code>	This determines whether the PAF files are pre-loaded into memory when the PAF is opened. It can be one of 0 (Registry), 1 (Disk), 2 (Memory) or 3 (Mapping), see ARF loading options .
----------------------	--

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Notes

Call this function or one of the other open functions [OpenPaf function](#), [OpenPafId function](#), [OpenPafMemId function](#), [OpenArfId function](#) before calling other functions to access the PAF.

For every successful call to `OpenPafMem`, the [ClosePaf function](#) must be called.

This function is equivalent to the [OpenPaf function](#), except for the extra parameter.

If the PAF has already been opened, then the `iOption` parameter will be ignored.

OpenPafMemId function

Opens the Address Reference File (ARF / PAF).

Syntax

```
int OpenPafMemId (  
    int *   id,  
    int    iOption  
);
```

Parameters

<code>id</code>	Pointer to the variable to receive the Id or handle. This Id is only valid if the function succeeds.
<code>iOption</code>	This determines whether the Address Reference File is loaded into memory. It can be one of 0 (Registry), 1 (Disk), 2 (Memory) or 3 (Mapping), see ARF loading options .

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Notes

The Id or handle received from this function can be used in other functions in the PAF library.

For each successful call to `OpenPafMemId`, call the [ClosePafId function](#) to release the handle.

If `OpenPafMemId` is called more than once, then the Address Reference File will be opened only once, but a new handle will be provided for each call.

The `iOption` parameter will be ignored if the Address Reference File is already open.

It is not safe to share the handle between simultaneously executing threads.

This function is same as the [OpenPafId function](#) with the extra *iOption* parameter.

To specify which Address Reference File to use, use the [OpenArfId function](#).

`OpenPafMemId(id,iOption)` is equivalent to `OpenArfId(id,iOption,"")`

PostcodeToBsp function

This function will look up the BSP (Barcode Sort Plan) number for a particular postcode.

Syntax

```
int PostcodeToBsp (  
    char * s,  
    int    maxlen  
);
```

Parameters

<code>s</code>	Pointer to a null-terminated string that contains the postcode. It also serves as a buffer in which the result BSP is placed.
<code>maxlen</code>	The size of the above buffer, excluding the null-terminator.

Return Value

Zero	Function succeeded.
Blank	No BSP number for this postcode, or another error occurred.

Notes

The PAF files must be opened using [OpenPaf function](#) or [OpenPafMem function](#) before calling this function.

The additional [PAF files](#) `Bsp.db` must be present for this function to work.

An alternative to this function is to call the [GetProperties\(Id\) function](#) with **BSP:Postcode** (see [Read-Only properties](#)). In this case, **OpenPaf** or **OpenPafMem** don't need to have been called. However, one of [OpenArfId function](#), [OpenPafId function](#) or [OpenPafMemId function](#) do need to have been called.

PostcodeToNpsp function

This function will look up the NPSP (Nation PreSort Plan) number for a particular postcode.

Syntax

```
int PostcodeToNpsp (  
    char * s,  
    int    maxlen  
);
```

Parameters

<code>s</code>	Pointer to a null-terminated string that contains the postcode. It also serves as a buffer in which the result NPSP is placed.
<code>maxlen</code>	The size of the above buffer, excluding the null-terminator.

Return Value

Zero	Function succeeded.
Blank	No BSP number for this postcode, or another error occurred.

Notes

The PAF files must be opened using [OpenPaf function](#) or [OpenPafMem function](#) before calling this function.

The additional [PAF files](#) `Npsp.db` must be present for this function to work.

An alternative to this function is to call the [GetProperties\(Id\) function](#) with **NPSP:Postcode** (see [Read-Only properties](#)) In this case, **OpenPaf** or **OpenPafMem** don't need to have been called. However, one of [OpenArfId function](#), [OpenPafId function](#) or [OpenPafMemId function](#) do need to have been called.

ReadGroupRecord(Id) function

```
int ReadGroupRecord (
    int    iRecNo,
    char * sRec,
    int *   iFromPointRecNo
    int *   iToPointRecNo
);

int ReadGroupRecordId (
    int    id,
    int    iRecNo,
    char * sRec,
    int *   iFromPointRecNo
    int *   iToPointRecNo
);
```

Parameters

<code>id</code>	The Id or handle received when opening the Address Reference File.
<code>iRecNo</code>	The record number of the group – from 1 to <code>GroupRecordCount(Id)</code> .
<code>sRec</code>	Pointer to the buffer to receive the group record. This buffer needs to be at least MAX_GROUP_RECORD + 1 characters in size.
<code>iFromPointRecNo</code>	Pointer to the variable to receive the record number of the first point in this group.
<code>iToPointRecNo</code>	Pointer to the variable to receive the one more than the record number of the last point in this group.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Group record format

The group record placed in the `sRec` buffer is a tab delimited null-terminated string with these fields:

1. Street name
2. Street type (see [Street type abbreviations](#))
3. Street suffix (see [Street suffix abbreviations](#))
4. Postal delivery type (see [Postal delivery type abbreviations](#))

Notes

See [Direct reading functions](#).

ReadLocalityRecord(Id) function

```
int ReadLocalityRecord (
    int    iRecNo,
    char * sRec,
    int *   iFromGroupRecNo
```

```

    int * iToGroupRecNo
);

int ReadLocalityRecordId (
    int id,
    int iRecNo,
    char * sRec,
    int * iFromGroupRecNo
    int * iToGroupRecNo
);

```

Parameters

id	The Id or handle received when opening the Address Reference File.
iRecNo	The record number of the locality – from 1 to LocalityRecordCount(Id).
sRec	Pointer to the buffer to receive the locality record. This buffer needs to be at least MAX_LOCALITY_RECORD + 1 characters in size.
iFromGroupRecNo	Pointer to the variable to receive the record number of the first group in this locality.
iToGroupRecNo	Pointer to the variable to receive the one more than the record number of the last group in this locality.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Locality record format

The locality record placed in the *sRec* buffer is a tab delimited null-terminated string with these fields:

1. State (see [State Abbreviations](#))
2. Postcode
3. Locality name

Notes

See [Direct reading functions](#).

ReadPointRecord(Id) function

```

int ReadPointRecord (
    int iRecNo,
    char * sRec,
);

int ReadPointRecordId (
    int id,
    int iRecNo,
    char * sRec,
);

```

Parameters

id	The Id or handle received when opening the Address Reference File.
iRecNo	The record number of the point – from 1 to PointRecordCount.

sRec Pointer to the buffer to receive the group record. This buffer needs to be at least [MAX_POINT_RECORD](#) + 1 characters in size.

Return Value

Zero Function succeeded.

Non-zero Function failed, returns [PAF Error codes](#).

Point record format

The point record placed in the *sRec* buffer is a tab delimited null-terminated string with these fields:

- 1 DPID
- 2 House number 1
- 3 House number suffix 1
- 4 House number 2
- 5 House number suffix 2
- 6 Flat/Unit type (see [Flat/Unit type abbreviations](#))
- 7 Flat/Unit number
- 8 Floor/Level type (see [Floor/Level type abbreviations](#))
- 9 Floor/Level number
- 10 Building/Property name 1
- 11 Building/Property name 2
- 12 Lot number
- 13 Postal delivery number
- 14 Postal delivery number prefix
- 15 Postal delivery number suffix
- 16 Primary delivery point indicator (“R” real deliverable address, or “P” phantom address requiring some flat or level information)

Notes

See [Direct reading functions](#).

ReadSynonymRecord(Id) function

```
int ReadSynonymRecord (
    int    iRecNo,
    char * sSynonymRec,
    int *  iLocalityRecNo
);

int ReadSynonymRecordId (
    int    id,
    int    iRecNo,
    char * sSynonymRec,
    int *  iLocalityRecNo
);
```

Parameters

id The Id or handle received when opening the Address Reference File.

<code>iRecNo</code>	The record number of the synonym – from 1 to <code>SynonymRecordCount</code> .
<code>sSynonymRec</code>	Pointer to the buffer to receive the synonym record. This buffer needs to be at least MAX_SYNONYM_RECORD + 1 characters in size.
<code>iLocalityRecNo</code>	Pointer to the variable to receive the record number of the locality for which this is a synonym.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Synonym record format

The synonym record placed in the `sSynonymRec` buffer is a tab delimited null-terminated string with these fields:

1. Synonym name
2. Postcode
3. Valid Flag (“V” for valid or “U” for unacceptable)
4. Blank.

Notes

The Valid Flag indicates whether it is acceptable to use the synonym name on the mailing address in place of the real locality name.

See [Direct reading functions](#).

ReadBorderRecord(Id) function

```
int ReadBorderRecord (
    int    iLocalityRecNo,
    int *  iBorderRecNoArray,
    int *  iArraySize
);

int ReadBorderRecordId (
    int    id,
    int    iLocalityRecNo,
    int *  iBorderRecNoArray,
    int *  iArraySize
);
```

Parameters

<code>id</code>	The Id or handle received when opening the Address Reference File.
<code>iLocalityNo</code>	The record number of the Locality of which to find its border localities – from 1 to <code>LocalityRecordCount</code> .
<code>iBorderRecNoArray</code>	Pointer to the buffer to receive the border locality record numbers. The size of this buffer must be specified in <code>iArraySize</code> .
<code>iArraySize</code>	Pointer to the variable with the size of the above array. This is an input and an output parameter, see notes below.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Notes

The size of the `iBorderRecNoArray` buffer must be placed into `iArraySize` before calling the function. After calling the function, the number of border record numbers will be placed into `iArraySize`. If the buffer was big enough, this will be the number of border record numbers written to the buffer (the rest of the buffer is undefined). If the buffer was not big enough, then this will be the size of the buffer required to fit all the border record numbers.

The border locality records may be read using the [ReadLocalityRecord\(Id\) function](#), passing it the border record numbers returned from this function.

See also [Direct reading functions](#).

SetGeographicalCodesId function

Specifies which geographical codes are to be looked up.

This will apply to the [GetDpid\(Id\) function](#), [GetDpidNonAmas\(Id\) function](#), [GetAddressHint\(Id\) function](#), [GetNextAddressHint\(Id\) function](#), [GetAddress\(Id\) function](#) and [GetNextAddress\(Id\) function](#).

The PAF library will call the iqqiss library to look up the specified Geographical codes. These codes are then returned in positions 43 to 52 in the [Australian address record format](#) returned by any of the above listed functions.

Syntax

```
int SetGeographicalCodesId (  
    int      id,  
    const char * s  
);
```

Parameters

<code>id</code>	The Id or handle received when opening the Address Reference File.
<code>s</code>	Pointer to a null-terminated string containing the list of codes. This list of codes is to be comma delimited, with each code in the format <code>File.Code</code> , where <code>File</code> is the name of the Geographical Information File (without the <code>.iqg</code> , <code>_L.iqg</code> or <code>_G.iqg</code> extension) and <code>Code</code> is the name of the field in the file.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Notes

If the Geographical Information File doesn't exist, or if the Code doesn't exist in the File, then it will not be added to the list, but no error will be returned.

Alternatively, the geographical codes may be set via the [GeographicalCodes property](#).

SetProperties(Id) function

This function can be used to specify the functioning of the PAF Libraries.

Syntax

```
int SetProperties (  
    const char *   sName,  
    const char *   sValue  
);  
  
int SetPropertiesId (  
    int id,
```

```

const char *    sName,
const char *    sValue
);

```

Parameters

<code>id</code>	The Id or handle received when opening the Address Reference File.
<code>sName</code>	Pointer to a null-terminated string that specifies the name of the property. (See Read/Write properties)
<code>sValue</code>	Pointer to a null-terminated string that specifies the value of the property.

Return Value

Zero	Function succeeded.
Non-zero	Function failed, returns PAF Error codes .

Notes

When using `GetProperties` in a multi-threaded application, each thread has its own set of properties. Setting the properties from one calling thread will not set these properties for other threads.

When using `GetPropertiesId`, each Id has its own set of properties. Setting the properties for one Id will not set these properties for other Id's.

The `id` parameter is ignored for the [UseSockets property](#), [TcpHost property](#) and [TcpPort property](#). Do not set these three properties after the Address Reference File has already been opened.

ValidateFullAddress(Id) function

This function parses a free-form address, standardising it and looking it up in the Address Reference File.

Syntax

```

int ValidateFullAddress (
    const char * sIn,
    char        * sOut,
    int          maxlen,
    int          iOptions
    char        * sError,
);

int ValidateFullAddressId (
    int          id,
    const char * sIn,
    char        * sOut,
    int          maxlen,
    int          iOptions
    char        * sError,
);

```

Parameters

<code>id</code>	The Id or handle received when opening the Address Reference File.
<code>sIn</code>	Pointer to a null-terminated string that specifies the free-form address to parse.
<code>sOut</code>	Pointer to the buffer to receive the standardised address. This is a tab-delimited string of address fields, as described in Australian address record format . This buffer needs be at least 1110 bytes in size.
<code>maxlen</code>	The size of the above buffer.

<code>iOptions</code>	1 for validation using struct AMAS rules. 2 for validation using more lenient rules.
<code>sError</code>	Pointer to the buffer to receive any error information. This parameter may be null if no error information is desired. If this parameter is not null, the buffer should be at least 256 characters in length.

Return Value

Zero	Function succeeded.
Non-zero	Function failed.

Notes

Addresses are standardised via [StanRt](#) or the [StanLibrary](#) using the system grammar files `_ArfGeocode.grm` for Australian addresses, or `_NzAddress.grm` for New Zealand addresses.

ValidatePartialAddressId

This function has been designed to be used only in conjunction with the `ValidatePartialAddress` grammar file to validate partial address information against an Address Reference File.

Syntax

```
int ValidatePartialAddressId(
    int    id,
    char * s,
    int    maxlen
);
```

Parameters

<code>id</code>	The Id or handle received when opening the Address Reference File.
<code>s</code>	Pointer to a null-terminated string that specifies the address criteria to look up. It also serves as a buffer in which the result is placed.
<code>maxlen</code>	The size of the above buffer, excluding the null-terminator.

Return Value

Zero	Function succeeded.
Non-zero	Function failed.

Notes

An ARF must be opened using [OpenArfId function](#) before this function is called.

PAF Error codes

530	Expecting more fields
539	Out of memory
540	Invalid code.cpr file
541	Cannot open code.cpr file
542	Error in reading code.cpr file
543	System error or corrupt code.cpr file
601	Buffer too small to contain result
602	Too many localities found
604	Too many streets (groups) found

606 Too many addresses (points) found
609 Address matching exception.
619 Attempt to set a read only property
620 Attempt to set a non-existent property
621 Attempt to set a property with an invalid value

622 Invalid point file record number
624 File name too long
626 Check sum error in version file (PAF.ver)
627 Record counts in version file (PAF.ver) don't match those in Address Reference File
628 Cannot open or read version file (PAF.ver)
629 Not licenced to use specific Addresses Reference File
630 Buffer too small for XML output
633 Address reference file (ARF/PAF) not open
637 Cannot find or open Address reference file (ARF/PAF)
638 Border.cdb file corrupt or incompatible
639 PAF version file error

640 Licence file corrupt
641 Invalid licence
642 Licence expired
644 Licence invalid for this computer
645 Licence file not found
649 Cannot open registry or configuration file
650 Missing licence registry key
651 Cannot open registry or configuration file
652 Missing ARF/PAF registry key
653 ARF/PAF directory too long
654 Too many Address Reference Files open

660 Invalid memory size in retrieving ARF from shared memory
661 Missing header in retrieving ARF from shared memory
662 Missing trailer in retrieving ARF from shared memory
663 No Alternate Street file
664 No border localities file
665 No synonym localities file
666 ARF file missing

- 667 Size of ARF file not as expected
- 668 ARF version file read error
- 669 ARF file open or read error
- 670 Out of shared memory error (CICS GETMAIN)
- 671 ARF in memory corrupt
- 672 No index on AddressId or DPID (AddrId.hin or Dpid.in1 file)

- 802 Error in opening index file (*.in?)
- 803 Error in reading index file
- 804 Index buffer too small
- 805 Record not found in index
- 807 Out of memory
- 808 Writing to index file error
- 809 Internal error in index file

- 810 Cannot find or open file
- 811 Invalid record number
- 812 Internal error
- 814 Out of memory
- 815 File read error
- 816 Error in initialising mutex
- 817 Error in locking mutex
- 818 Error in unlocking mutex

- 820 Error in initialising mutex
- 821 Error in locking mutex
- 822 Error in unlocking mutex
- 825 Index file seeking error
- 826 Index file reading error

- 904 Buffer too small to contain barcode
- 905 iqpf / iPaf32 library not initialized
- 906 Error in getting thread local storage data
- 907 Error in getting thread local storage data
- 908 Error in allocating thread local storage data
- 909 iPaf32 threw an acception
- 910 Out of memory

- 911 Invalid request for barcode
- 912 Barcoding error: Invalid Format Control field
- 913 Barcoding error: Invalid Sorting Code field
- 914 Barcoding error: Invalid customer information field
- 915 Barcoding error: Unable to encode data
- 916 Barcoding error: Invalid barcode range
- 917 Error in initialising mutex
- 918 Error in locking mutex
- 919 Error in unlocking mutex
- 920 Invalid PAF id/handle. Perhaps ARF not opened.
- 921 Buffer too small to contain result
- 922 Client limit reached
- 923 Functionality not yet implemented
- 924 Use of [Direct reading functions](#) not authorised

- 990 Error in loading library.
- 991 Error in loading functions from library.
- 992 Library not loaded.

- 1300 Registry value too big
- 1301 Error in opening configuration file
- 1302 Key doesn't exist in registry/configuration file
- 1303 Error in opening registry
- 1304 Out of memory

- 1501 COM library initialization error
- 1502 Creating StanRt or Manager COM object error
- 1503 Error in loading Stan32.dll / libiqstan
- 1504 Entry point not found in Stan32 / libiqstan
- 1505 No StanRt object or Stan32/libiqstan library function
- 1506 Stan library Out of memory error
- 1507 Stan library Open grammar error
- 1508 Stan library Standardization error
- 1509 Stan library Close grammar error
- 1510 Stan library Grammar not open
- 1511 Stan library Bstring conversion error
- 1512 Stan library put properties error

1513 Stan library Get description error
1514 Stan library Get grammar name error
1515 Stan library Get output delimiter error
1516 Stan library Put output delimiter error
1517 Stan library Get output qualifier error
1518 Stan library Put output qualifier error
1519 Stan library Get output field count error
1520 Stan library Get output field description error
1521 Stan library Get output field names error
1522 Stan library Get output field sizes error
1523 Stan library Get output format type error
1524 Stan library Put output format type error
1525 Stan library Get suggested divider error
1526 Stan library Dont keep loaded error
1527 Stan library Com not initialised error
1528 Stan library Get default directory error
1529 Stan library Get default processes error
1530 Stan library Get licence error
1531 Stan library Buffer too small error
1532 Stan library Get properties error
1533 Stan library Keep loaded error
1534 Stan library Put default directory error
1535 Stan library Update error

4002 Error in opening a file map
4003 Error in opening a file for mapping
4004 Error in creating a file mapping
4005 Error in mapping a file
4006 File mapping not supported
4007 Error in getting file information for mapping
4008 Error in creating the DACL (Discretionary Access Control List)
4009 Error in initialising Security Descriptor
4010 Error in setting the Security Description

4020 File seeking error
4021 File reading error
4022 Invalid position to read file

4023 Attempting to read a closed file
4024 Internal error in file reading
4025 Error in opening a file
4027 Reading pass the end of a file

4030 Error in initialising mutex
4031 Error in locking mutex
4032 Error in unlocking mutex

4200 Socket call error
4201 Set socket option error
4202 Socket bind error
4203 Socket listen error
4204 Socket connect error
4205 Unknown socket host
4206 Unknown socket service

4300 Out of memory
4301 Invalid GIF file
4302 Unsupported GIF file
4304 Invalid GIF record number
4305 Invalid GIF field number
4306 Unsupported GIF field type
4307 GIF filename too long
4309 No fields in GIF file
4310 System error in reading GIF file
4311-25 Invalid GIF file
4330 GIF value too big
4341 GIF file corrupt

4400 Out of memory
4401-3 Mutex error
4404 Too many open GIF files
4405 No such GIF property
4406 No such GIF field
4407 Unknown aggregate function
4408 Invalid GISS Id

4409 Too many open .hin indexes
4410 Requested .hin index file doesn't exist
4411 Invalid search value
4412 Search cannot open GIF file

4470 Error in loading iqGiss library
4471 Entry points not found in iqGiss library

4500 Unrecognised socket request
4501 Socket "accept" function failed
4502 Socket request too long

4550 Socket read error
4551 Socket send error
4552 Error in initialising mutex
4553 Error in locking mutex
4554 Error in unlocking mutex
4555 Socket response longer than buffer

4600 Out of memory
4603 Error in opening .tin file
4604 Error in reading .tin file
4611 Invalid .tin file
4612 Invalid .tin file

4700 Out of memory
4701 CICS READQ TS error
4702 CICS WRITEQ TS error
4703 CICS DELETEQ TS error
4704 File read error
4705 File read error
4706 File read error
4707 Functionality not supported

4800 Out of memory
4806-7 New version of .hin index file not yet supported
4810 System error. Expecting different type of .hin index file.

4811-8 Invalid .hin index file
4821-2 System error or corrupt .hin index file
4823 System error.
4826 New version of .hin index file not yet supported

4900 Polygon filter error - out of memory
4901 Polygon filter error - missing closing parenthesis
4902 Polygon filter error - unknown operator
4903 Polygon filter error - unknown operand
4904 Polygon filter error - incompatible operand
4905 Polygon filter error - unknown field name
4906 Polygon filter error - ambiguous field name
4907 Polygon filter error - syntax error

6400 Paf library not initialised
6401 TlsGetValue Error
6402 TlsSetValue / pthread_setspecific error
6403 TlsAlloc / pthread_key_create error
6404 Out of Memory
6405 Invalid Id passed to function
6406 Buffer too small
6407 Client limit reached
6408 Functionality not implemented
6409 Action timed out
6410 System error
6411 System error
6412 Id already in use
6413 PAF files not open
6414 System error
6415 Barcoding error

6493 Attempting to open the default ARF, and it is not Australian.
6494 System error
6495 File open error
6496 Out of memory
6497 Unknown country
6498 Cannot open registry or configuration file

6499 ARF not installed

Barcode Function Error Codes

The PAF library barcoding functions can return the following error codes:

- 904 Barcode too long. The value of maxlen is greater than the length of the barcode.
- 911 Invalid input. For the CustInfoBarcode function, the input does not conform to what's expected.
- 912 Invalid FCC (expecting one of 11, 87, 45, 92, 59, 62, and 44).
- 913 Invalid dpid (expecting 8-digits)
- 914 Invalid [Customer Information coding](#) (for N-coding, only expecting numbers. For C-coding, expecting letters, numbers the space or hash characters)
- 915 Internal error – unable to encode data
- 916 Internal error – invalid barcode range

Socket Client

The [PAF Libraries](#) can be configured to act as a socket client. All calls to their functions are then redirected to the [Socket Server](#).

To do this, set the [TcpHost and TcpPort setting](#) to point to the socket server.

Alternatively, set the [TcpHost property](#), [TcpPort property](#) and [UseSockets property](#) before calling any other functions in the PAF Library.

Socket Server

The Socket Server provides the address searching functions of the [PAF Libraries](#).

Socket Server Installation

The Socket Server usually runs as Service on Windows (IQSocSvr.exe) and a daemon on UNIX (iqsocsvr).

Windows

Install the [PAF Libraries](#) (iPaf32.dll) in the system directory.

To install the socket server as a Service (listening on port 4000, for example), type:

```
IQSocSvr.exe -i 4000
```

Start the service using Windows Services Manager or Service Controller (sc.exe).

Use -h to see a full list of options:

```
C:\>IQSocSvr.exe -h
```

```
iqsocsvr [-h | -d | -D | -s ] [-lFile] [hostname] port
iqsocsvr -i [ [-cFile] [hostname] port]
iqsocsvr -u
```

```
-h    Show this help message
-d    Debug. Show data sent and received via sockets
-D    Debug connections only
-s    Silent. No writing to terminal
-l    Log errors to file specified
-i    Install as a Window's service
-u    Uninstall Window's service
```

To change the port, stop the service, uninstall it, and reinstall it.

UNIX

Install the [PAF Libraries](#) (`libiqpaf.so`) in the Library Path.

To listen on port 4000 for example, type:

```
iqsocsvr 4000
```

To run as a daemon, type:

```
iqsocsvr -a 4000
```

To see a full list of options, type without any parameters:

```
iqsocsvr
```

Calling the socket server

This section describes how to write a client to call the socket server to access the functions in the [PAF Libraries](#).

Conventions

- Each message sent to the socket server must be terminated with a New-Line character (ASCII 10) or a Carriage-Return character followed by a New-Line character (ASCII 13 then ASCII 10).
- Each message returned from the socket server will be terminated by a New-Line character.
- In each message (both sent and received), the tab character (ASCII 9) is used to divide parameters.
- Messages are case sensitive.
- Send a `quit` message to the server to close the connection.
- Most messages returned from the socket server will begin with an error code. This is the same as the return value of the corresponding Library function.
- If the call was successful, there might be return parameters after the zero error code.
- If the call was unsuccessful, there might be an error message after the error code.

Function Summary

An equivalent message can be sent to the socket server to call most of the functions in the PAF Library. The table below lists each message sent, and the result of the function returned by the socket server. Parameters in the table are shown separated by a comma and a space, but this is a tab in the messages. The RV (Return Value) is the value returned by the equivalent library function, usually an error code or zero for success.

See each topic in the list for a description of the function and its parameters.

Function Called	Message Sent	Message Returned
BuildBarcode function	pafBuildBarcode_, sFcc, sSortingCode, sCustInfo, bCustNCoded	RV, sBarcode
ClosePafId function	pafClosePafId, id	RV
CustInfoBarcode function	pafCustInfoBarcode, sIn	RV, sOut
DpidToAddress(Id) function	pafDpidToAddressId, id, iDpid	RV, sResult
DpidToBarcode function	pafDpidToBarcode, Dpid	RV, Barcode
FixLastLine function	pafFixLastLineId, id, sInput	RV, sOutput
FormatAddress(Id) function	pafFormatAddress, iOptions, sIn	RV, sOut

	paFormatAddressId, id, iOptions, sIn	RV, sOut
FormatAddressS function	paFormatAddressS, sIn	RV, sOut
GenerateBarcode function	paGenerateBarcode, sIn	RV, sOut
GetAddress(Id) function	paGetAddressId, id, sInput	RV, sOutput
GetAddressHint(Id) function	paGetAddressHintId, id, sLocalityLine, sAddressLine	RV, sOutput
GetBorderPostcodes(Id) function	paGetBorderPostcodesId, id, iPostcode	RV, sOutput
GetDpid(Id) function	paGetDpidId, id, sInput	RV, sOutput
GetDpidNonAmas(Id) function	paGetDpidNonAmasId, id, sInput	RV, sOutput
GetErrorDescription function	paGetErrorDescription, iError	RV, sDescription
GetFindStatus(Id) function	paGetFindStatusId, id	RV
GetGroupRecordCount(Id) function	paGetGroupRecordCount	RV
	paGetGroupRecordCountId, id	RV
GetLocalityRecordCount(Id) function	paGetLocalityRecordCount	RV
	paGetLocalityRecordCountId, id	RV
GetNextAddress(Id) function	paGetNextAddressId, id	RV, sOutput
GetNextAddressHint(Id) function	paGetNextAddressHintId, id	RV, sOutput
GetNextPopupHint(Id) function	paGetNextPopupHintId, id	RV, sOutput
GetParsedInput(Id) function	paGetParsedInputId, id	RV, sOutput
GetPointRecordCount(Id) function	paGetPointRecordCount	RV
	paGetPointRecordCountId, id	RV
GetPopupHint(Id) function	paGetPopupHintId, id, sLocalityLine, sAddressLine, iOptions	RV, sOutput
GetProperties(Id) function	paGetPropertiesId, id, sName	RV, sValue
GetSynonymRecordCount(Id) function	paGetSynonymRecordCount	RV
	paGetSynonymRecordCountId, id	RV
OpenArId function	paOpenArId, iOption, sDatabase	RV, Id
OpenPaId function	paOpenPaId	RV, Id
OpenPafMemId function	paOpenPafMemId, iOption	RV, Id
PostcodeToBsp function	paPostcodeToBsp, Postcode	RV, Bsp
PostcodeToNpsp function	paPostcodeToNpsp, Postcode	RV, Npsp
ReadBorderRecord(Id) function	paReadBorderRecord, iLocalityRecNo	RV, iBorderRecNo, iBorderRecNo, ...
	paReadBorderRecordId, id, iLocalityRecNo	RV, iBorderRecNo, iBorderRecNo, ...
ReadGroupRecord(Id) function	paReadGroupRecord, iRecNo	RV, iFromPointRecNo,

		iToPointRecNo, sRec
	pafReadGroupRecordId, id, iRecNo	RV, iFromPointRecNo, iToPointRecNo, sRec
ReadLocalityRecord(Id) function	pafReadLocalityRecord, iRecNo	RV, iFromGroupRecNo, iToGroupRecNo, sRec
	pafReadLocalityRecordId, id, iRecNo	RV, iFromGroupRecNo, iToGroupRecNo, sRec
ReadPointRecord(Id) function	pafReadPointRecord, iRecNo	RV, sRec
	pafReadPointRecordId, id, iRecNo	RV, sRec
ReadSynonymRecord(Id) function	pafReadSynonymRecord, iRecNo	RV, iLocalityRecNo, sSynonymRec
	pafReadSynonymRecordId, id, iRecNo	RV, iLocalityRecNo, sSynonymRec
SetProperties(Id) function	pafSetPropertiesId, id, sName, sValue	RV
ValidateFullAddress(Id) function	pafValidateFullAddressId, id, iOptions, sIn	RV, sOutput
		RV, sError

Notes

FormatAddress will not include new lines in the output, as this would break the socket server's convention.

Example

The following example will open the Address Reference File, and get its version and close it. The id returned here is 1, but it could be another value.

```

Send:      pafOpenPafId
Receive:   0<tab>1
Send:      pafGetPropertiesId<tab>1<tab>PafVersionInfo
Receive:   0<tab>PAF V2007.4
Send:      pafClosePafId<tab>1
Receive:   0

```

SOAP Server

The SOAP/XML Server provides some of the address searching functions provided by the [PAF Libraries](#). It also provides access to the Stan Libraries.

SOAP Server Installation

The SOAP Server handles its own socket connections and HTTP requests, so does not require a Web Server. However, it must be configured to use a different port to any existing Web Servers already running on the machine. Check which ports are already in use by any existing web server (e.g. 80) before starting the SOAP Server.

The SOAP Server usually runs as Service on Windows (IQSoapSv.exe) and a daemon on UNIX (iqsoapsv). However, it can also run as a stand-alone executable, usually for debugging.

Windows

To install the SOAP server as a Windows service (listening on port 82, for example), type:

```
IQSoapSv.exe -i 82
```

Start the service using Windows Services Manager or Service Controller (`sc.exe`).

Use `-h` to see a full list of options:

```
C:\>IQSoapSv.exe -h
```

```
IQSoapSv [-d | -s ] [-llogfile] port
IQSoapSv -i [port]
IQSoapSv [-h | -u | -v]
```

```
-d    Debug. Show data sent and received
-s    Silent. No writing to terminal
-l    Log errors to file specified
-i    Install as a Window's service
-h    Show this help message
-u    Uninstall Window's service
-v    Show version number
```

To change the port, stop the service, uninstall it, and reinstall it.

UNIX

To listen on port 82 for example, type:

```
iqsoapsv 82
```

To run as a daemon, type:

```
iqsoapsv -a 82
```

To see a full list of options, type:

```
iqsoapsv -h
```

Calling the SOAP Server

This section describes how to make requests to the SOAP Server to access the [PAF Libraries](#) functions. SOAP functions are defined in `iqoffice.wsdl`

The interface to the SOAP server is different from the other Java, sockets, and COM interfaces, because SOAP is a connectionless interface. The full request must be sent to the SOAP server in one call, and the full result retrieved by the call.

It may also be necessary to provide a UserId and Password as part of the optional array properties if the SOAP Server has been configured to require them. If the UserId or Password combination is incorrect the Authentication Module being used will return an error code and possibly an error string. Further information on the authentication module can be found in the [IQ Component Setup Advanced Registry settings](#).

Function Reference

BuildBarcode

This SOAP function creates barcodes, similar to the [BuildBarcode_function](#), (see [Barcodes](#)).

Input / Request Parameters

`FormatControl` Format Control Code, `sFcc` parameter in the [BuildBarcode_function](#)

`Dpid` DPID, `sSortingCode` parameter in the [BuildBarcode_function](#)

CustomerInfo	Additional customer information to put into barcode, sCustInfo parameter in the BuildBarcode_function
Ncoding	Type of Customer Information coding , bCustNCoded parameter in the BuildBarcode_function
Optional	Array of properties to set

Example Request

```
POST / HTTP/1.1
Host: soapservermachine.yourcompany.com
Content-Type: text/xml; charset=utf-8
Content-Length: 639
Connection: close
SOAPAction: "http://intechsolutions.com.au/soap/"
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://intechsolutions.com.au/soap/">
  <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns1:BuildBarcode>
      <FormatControl>11</FormatControl>
      <Dpid>30000000</Dpid>
      <CustomerInfo></CustomerInfo>
      <NCoding>false</NCoding>
    </ns1:BuildBarcode>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Output / Response

Result	The found address, sOut parameter in the FormatAddress(Id) function
ErrorCode	Error code number, zero for success.
Optional	Error string.

Example Response

```
HTTP/1.1 200 OK
Server: IQSoapSv/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: 555
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://intechsolutions.com.au/soap/">
  <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns1:StringResult>
      <Result>13010110000000000000000310100020211113</Result>
      <ErrorCode>0</ErrorCode>
    </ns1:StringResult>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

DpidToAddress

This SOAP function is similar to the [DpidToAddress\(Id\) function](#).

Input / Request

The input/request to this function has these parameters:

1. The name of the Address Reference File to use – the *sDatabase* parameter to the [OpenArfId function](#).
2. The DPID or Address Identifier – the *iDpid* parameter in the **DpidToAddress(Id) function**.
3. An optional array of properties to set (see the [SetProperties\(Id\) function](#)).

Output / Response

The output/response from this function has these parts:

1. The found address – the *sResult* parameter in the **DpidToAddress(Id) function**.
2. An error code number, zero for success.
3. Possibly an error string.

Notes

The found address will be delimited address fields. The delimiter may be set in the properties array.

FormatAddress

This SOAP function is similar to the [FormatAddress\(Id\) function](#).

Input / Request

The input/request to this function has these parameters:

1. The name of the Address Reference File to use – the *sDatabase* parameter in the [OpenArfId function](#).
2. The address to format – the *sIn* parameter in the **FormatAddress(Id) function**. This is the address returned from other SOAP server functions.
3. The options parameter – *iOptions* in the **FormatAddress(Id) function**.
4. An optional array of properties to set (see the [SetProperties\(Id\) function](#)).

Output / Response

The output/response from this function has these parts:

1. The found address – the *sOut* parameter in the **FormatAddress(Id) function**.
2. An error code number, zero for success.
3. Possibly an error string.

GetAddresses

This SOAP function is similar to the [GetAddress\(Id\) function](#).

Input / Request Parameters

Database	name of the ARF to use, <i>sDatabase</i> parameter to the OpenArfId function
InputFields	criteria of which addresses to find – the <i>sInput</i> parameter to the GetAddress(Id) function
StartFrom	number of records from which to start, see Notes below
Optional	array of properties to set (see SetProperties(Id) function).

Example Request

This sample request retrieves localities in NSW with postcode of 2022, defining that at most 50 addresses are to be returned, delimited by a comma .

```
POST / HTTP/1.1
Host: soapservermachine.yourcompany.com
Content-Type: text/xml; charset=utf-8
Content-Length: 876
Connection: close
SOAPAction: "http://intechsolutions.com.au/soap/"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://intechsolutions.com.au/soap/">
  <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns1:GetAddresses>
      <Database></Database>
      <InputFields>NSW,2022</InputFields>
      <StartFrom>0</StartFrom>
      <Properties xsi:type="SOAP-ENC:Array" SOAP-
ENC:arrayType="ns1:NameValuePair[2]">
        <item>
          <Name>MaxAddressesReturned</Name>
          <Value>50</Value>
        </item>
        <item>
          <Name>Delimiter</Name>
          <Value>,</Value>
        </item>
      </Properties>
    </ns1:GetAddresses>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Output / Response

Addresses	an array of found addresses
ErrorCode	zero for success, or an error code number and optional error string
FindStatus	the find status code as returned from GetFindStatus(Id) function
Optional	error string

Example Response

```
HTTP/1.1 200 OK
Server: IQSoapSv/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: 842
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://intechsolutions.com.au/soap/">
```



```

        <Name>GeographicalCodes</Name>
        <Value>LatLong.Latitude+Longitude+Reliability</Value>
    </item>
</item>
    <Name>Delimiter</Name>
    <Value>|</Value>
</item>
</Properties>
</nsl:GetAddressesHint>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Output / Response

Addresses	an array of found addresses
ErrorCode	zero for success, or an error code number and optional error string
FindStatus	the find status code as returned from GetFindStatus(Id) function
Optional	error string

Example Response

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:nsl="http://intechsolutions.com.au/soap/">
  <SOAP-ENV:Body SOAP-
    ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <nsl:AddressesResult>
      <Addresses xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:string[2]">

        <item>NSW|CROOKWELL|SPRING|ST|U|BINDA|16064082|525
        647|47%|Spring St|-34.457680823|149.470599309|4</item>
        <item>NSW|BONDI
        JUNCTION|SPRING|ST|16064059|525646|81%|Sprin
        g St|-33.892779450|151.247113833|4</item>
      </Addresses>
      <ErrorCode>0</ErrorCode>
      <FindStatus>0</FindStatus>
      <ErrorString></ErrorString>
    </nsl:AddressesResult>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Notes

If the error code is non-zero, there may be more information in the error string returned.

GetProperties

This SOAP function is the equivalent of the [GetProperties\(Id\) function](#).

Input / Request Parameters

Database	(optional) name of the ARF to use, sDatabase parameter to the OpenArfId function
Name	the name of the property to retrieve, sName parameter to the GetProperties(Id) function
Optional	array of properties to set (see SetProperties(Id) function).

Example Request


```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://intechsolutions.com.au/soap/">
  <SOAP-ENV:Body SOAP-
    ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns1:GetProperties>
      <Name>ARFs</Name>
    </ns1:GetProperties>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Output / Response

Found address the `sResult` parameter in the [GetProperties\(Id\) function](#)

ErrorCode zero for success, or an error code number and optional error string

Optional error string

Example Response

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://intechsolutions.com.au/soap/">
  <SOAP-ENV:Body SOAP-
    ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns1:StringResult>
      <Result>GNAF;NZPaf;PAF</Result>
      <ErrorString></ErrorString>
      <ErrorCode>0</ErrorCode>
    </ns1:StringResult>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

ValidateFullAddress

This SOAP function is similar to the [ValidateFullAddress\(Id\) function](#).

Input / Request Parameters

Database (optional) name of the ARF to use, `sDatabase` parameter to the [OpenArfId function](#)

Address free format address to validate, `sIn` parameter the [ValidateFullAddress\(Id\) function](#)

Options the *iOptions* in the [ValidateFullAddress\(Id\) function](#)

Optional array of properties to set (see [SetProperties\(Id\) function](#)).

Output / Response

Found address the `sResult` parameter in the [ValidateFullAddress\(Id\) function](#)

ErrorCode zero for success, or an error code number and optional error string

Optional error string

Notes

If the error code is non-zero, there may be more information in the error string returned.

ValidateAddressWithOptions

This SOAP function is similar to a combination of the [ValidateFullAddress\(Id\) function](#) and the [GetAddressHint\(Id\) function](#).

Input / Request Parameters

Database	The name of the Address Reference File, or blank for the default, the sDatabase parameter in the OpenArfld function .
LocalityLine	The locality information of the address to validate (locality/suburb, state and/or postcode). May be blank. The sLocalityLine parameter in the GetAddressHint(Id) function .
AddressLine	The address to validate, without locality information. May be blank. The sAddressLine parameter in the GetAddressHint(Id) function . These two parameters become the sIn parameter in the ValidateFullAddress(Id) function
Options	1 (strict validation) or 2 (lenient validation). The iOptions parameter in the ValidateFullAddress(Id) function
GenerateAddress	Whether matching addresses should be returned in the Addresses parameter, one of Always, Never or IfNeeded (if the address in AddressLine and LocalityLine couldn't be matched to a unique address). This parameter is ignored if DrillDown is set.
DrillDown	Pass an address in Australian address record format to retrieve sub-addresses of this addresses in the Addresses parameter.
StartFrom	Which sub-address of DrillDown to start from. On the first call, set this to zero. See Notes below.
Properties	An array of properties to set.

Example Request

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://intechsolutions.com.au/soap">
  <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    <ns1:ValidateAddressWithOptions>
      <Database>PAF</Database>
      <LocalityLine>Bondi Junction NSW 2022 </LocalityLine>
      <AddressLine>Level 7, 35 Spring St</AddressLine>
      <Options>1</Options>
      <GenerateAddress>IfNeeded</GenerateAddress>
      <DrillDown></DrillDown>
      <StartFrom>0</StartFrom>
    <Properties SOAP-ENC:arrayType="ns1:NameValuePair[3]">
      <item>
        <Name>UserId</Name>
        <Value>SoapUser</Value>
      </item>
      <item>
        <Name>Password</Name>
        <Value>4u0gVee6</Value>
      </item>
      <item>
```

```

        <Name>Delimiter</Name>
        <Value>|</Value>
    </item>
</Properties>
</ns1:ValidateAddressWithOptions>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Output / Response

Address	The validated address, in Australian address record format . The <code>sResult</code> parameter in the ValidateFullAddress(Id) function.
AddressQualityFlag	An indication of the quality of the address in <code>AddressLine</code> and <code>LocalityLine</code> : 0 (good), 1 (OK) or 2 (bad).
Addresses	An array of matching address, each address in Australian address record format .
ErrorCode	zero for success, or an error code number and optional error string
FindStatus	0 (all results returned), 1 (too many results, so not all returned), 2 (search took too long, so not all returned)
ErrorString	An error message if applicable.

Example Response

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://intechsolutions.com.au/soap/">
  <SOAP-ENV:Body SOAP-
    ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns1:ValAddrResult>
      <Address>NSW|2022|BONDI
        JUNCTION|SPRING|ST|||44664340|35|||||L|7|||||||F|||||||0||||1301011111
        202011101100302003013102113|correct|Level 7 35 Spring
        Street|||||||</Address>
      <AddressQualityFlag>0</AddressQualityFlag>
      <ErrorCode>0</ErrorCode>
      <FindStatus>0</FindStatus>
      <ErrorString></ErrorString>
    </ns1:ValAddrResult>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Notes

The first thing this function does is to validate the address passed in the `AddressLine` and `LocalityLine` parameters. This address will be parsed and matched against the Address Reference File specified by the `Database` parameter. The result of the validation will be returned in the `Result` and `AddressQualityFlag` parameters.

Then, if `GenerateAddress` is “Always”, or if it is “IfNeeded” and a unique address couldn’t be found above, then a list of possible matching addresses will be returned in the `Addresses` parameter.

If `DrillDown` is set with an address, then sub-addresses of this address will be returned in the `Address` parameter. For example, if `DrillDown` contains only a locality, then all streets in this locality will be returned. If `DrillDown` contains a locality and street, then all street numbers in this street will be returned.

If `DrillDown` is set, then `GenerateAddress` is ignored and considered as “Never”.

On the first call to `ValidateAddressWithOptions` with `DrillDown` set, pass zero as the `StartFrom` parameter. If the `FindStatus` returned is 1, then there were too many addresses to be returned at once. To retrieve more addresses, call `ValidateAddressWithOptions` again, with `StartFrom` set to the number of addresses returned so far. This will then retrieve more addresses. If need be, this can be continued until the `FindStatus` returned is 0.

If an error occurred, the error number is returned in `ErrorCode`, and a description in `ErrorString`.

PolygonSearch

This SOAP function encapsulates the [BeginPolygonSearch function](#) and [GetNextAddressInPolygon function](#).

Input / Request Parameters

Database	The name of the Address Reference File, or blank for the default. The <code>sDatabase</code> parameter in the OpenArfld function .
Polygon	The latitude and longitude coordinates of the points of the polygon. Lat1,Long1,Lat2,Long2,... The <code>sPolygon</code> parameter in the BeginPolygonSearch function
Start	The latitude and longitude (separated by a comma) of the starting point. May be blank. The <code>sStart</code> parameter in the BeginPolygonSearch function .
Attachment	Whether to return the results as a MTOM attachment (1 or “true”) or to return them in the <code>Addresses</code> array (0 or “false”).
ZipAttachment	Whether to compress the MTOM attachment (1 or “true”) using the gzip (.gz) format. Only applies if <code>Attachment</code> is true.
Properties	An array of properties to set.

Output / Response

Addresses	An array of found addresses. Each address is in Australian address record format . Only populated if <code>Attachment</code> is not true.
AddressesAttachment	A text file as an MTOM attachment with the found addresses. Only populated if <code>Attachment</code> is true. If <code>ZipAttachment</code> is true, the text file will be zipped.
ErrorCode	A numeric error code. Zero means no error.
ErrorString	An error message if applicable.
AttachmentChecksum	The CRC-32 check sum of the MTOM attachment. Only populated if <code>Attachment</code> is true.

Notes

The `Properties` array must include setting “GeographicalCodes” to include a Geographical file with Latitudes and Longitudes (see [GeographicalCodes property](#)). The “UserId” and “Password” properties may also need to be set.

If `Attachment` is true, then the client should call the `FileAcknowledge` SOAP function after receiving the MTOM attachment.

FileAcknowledge

The client calls this SOAP function to acknowledge receiving a MTOM attachment.

The server can then perform logging if required.

Input / Request Parameters

FileName	The name of the attachment received.
Properties	An array of properties to set. This may include the UserId and Password properties.

Output / Response

ErrorCode	A numeric error code. Zero means no error.
ErrorMessage	An error message if applicable.

GetSearchStatistics

This SOAP function is the equivalent of the [GetSearchStatistics function](#).

Input / Request Parameters

Database	The name of the Address Reference File, or blank for the default. The sDatabase parameter in the OpenArfld function .
Search	The name of the search function. Currently only “Polygon” is supported.
Statistic	The name of the statistic(s) to retrieve.
Param1	A parameter that would be parsed to the search function.
Param2	A parameter that would be parsed to the search function.
Properties	An array of properties to set.

Output / Response

Result	The statistics returned from the search..
ErrorCode	A numeric error code. Zero means no error.
ErrorMessage	An error message if applicable.

Notes

See [GetSearchStatistics function](#) for more information.

For the “Polygon” search, the *Properties* array must include setting “GeographicalCodes” to include a Geographical file with Latitudes and Longitudes (see [GeographicalCodes property](#)). The other properties that affect a polygon search may also need to be set ([ExpandGifLists property](#) and [PolygonSearchFilter property](#)). The “UserId” and “Password” properties may also need to be set.

RapidJNI Java class

The RapidJNI Java class provides the address searching functions of the [PAF Libraries](#) and [RapidSvr](#).

This class accesses the PAF libraries via a Java Native Interface (JNI), and so requires a dynamic library (RapidJNI.dll on Windows, and iqpfjni on Unix).

The TestRapidJni class provides an example of using this RapidJNI class.

Function Reference

Functions provided by the RapidJNI class are listed below.

```
public String GetAddress(String sRec);
public String GetHintAddress(String sLocality, String sGroup);
public String [] GetAddresses(String sRec);
public String [] GetHintAddresses(String sLocality, String sGroup);
public String Properties(String sName);
```

```

public void SetProperties(String sName, String sValue);
public String FormatAddress(String sAddress, int iOptions);
public String DpidToAddress(int Dpid);
public String DpidToBarcode(String Dpid, String FormatCode, String CustInfo,
    boolean NCoding);
public int PostcodeToBsp(int Postcode);
public int PostcodeToNpsp(int Postcode);
public int FindStatus();
public String ParsedInput();
public void ChangeDatabase (int iOptions, String Database);
public String ValidateFullAddress(String sIn, int iOptions);
public String GetPopupHint(String sLocality, String sGroup, int iOptions);
public void Cleanup ();

```

RapidJNI.GetAddress

This function lists addresses or partial addresses based on given criteria.

The first time GetAddress is called, the criteria of what to find are passed as the parameter, and the first address in the list is returned. Subsequent calls pass a null or zero-length String and return subsequent addresses in the list. When there are no more addresses in the list, an [Empty record](#) (just delimiters) is returned.

Syntax

```
public String GetAddress(String sRec);
```

Parameters

sRec string containing the criteria of what to find, which must be in [Australian address record format](#) (see [Input record](#)).

Return Value

Address or partial address found, in [Australian address record format](#) (see [Output record](#)).

See Also

[GetAddresses](#), [GetHintAddress](#), [FormatAddress](#).

RapidJNI.GetHintAddress

This function searches for addresses based on some address information (a hint).

The first time GetHintAddress is called, the criteria of what to find are passed as parameters, and the first matching address found is returned. Subsequent calls pass null parameters, and return subsequent matching addresses. When there are no more matching addresses, an [Empty record](#) is returned.

The results are returned in order with the closest match first.

Syntax

```
public String GetHintAddress(String sLocality, String sGroup);
```

Parameters

sLocality Some (or no) information about the locality (see [Locality Line](#)).

sGroup Some (or no) street or postal information (see [Address Line](#)).

Return Value

Address or partial address found, in [Australian address record format](#).

See Also

[GetHintAddresses](#), [GetAddress](#), [FindStatus](#), [FormatAddress](#), [TimeLimit property](#).

RapidJNI.GetAddresses

This function will return an array of addresses or partial addresses based on given criteria.

It is equivalent to multiple calls to the [GetAddress](#) function.

Syntax

```
public String [] GetAddresses(String sRec);
```

Parameters

sRec string containing the criteria of what to find, which must be in [Australian address record format](#) (see [Input record](#)).

Return Value

Address or partial address found, in [Australian address record format](#) (see [Output record](#)).

See Also

[GetAddress](#), [GetHintAddresses](#), [FormatAddress](#).

RapidJNI.GetHintAddresses

This function returns an array of addresses found based on some address information (a hint). It is equivalent to multiple calls to the [GetHintAddress](#) function.

Syntax

```
public String [] GetHintAddresses(String sLocality, String sGroup);
```

Parameters

sLocality Some (or no) information about the locality (see [Locality Line](#)).

sGroup Some (or no) street or postal information (see [Address Line](#)).

Return Value

Address or partial address found, in [Australian address record format](#).

See Also

[GetHintAddress](#), [GetAddresses](#), [FindStatus](#), [FormatAddress](#), [TimeLimit property](#).

RapidJNI.Properties

This function returns specific information about RapidJNI and how it will function.

Syntax

```
public String Properties(String sName);
```

Parameters

sName the name of the property.

Return Value

Property value or null if a property does not exist. See [Read-Only properties](#) and [Read/Write properties](#) for a list of available properties and their meanings.

See Also

[SetProperties](#).

RapidJNI.SetProperties

This function sets specific information about how RapidJNI is to function.

Syntax

```
public void SetProperties(String sName, String sValue);
```

Parameters

sName	the name of the property. See Read/Write properties for a list of properties that can be set.
sValue	new value of the property

Return Value

If a property does not exist or is read-only, an exception will be thrown.

See Also

[Properties.](#)

RapidJNI.FormatAddress

This functions returns a formatted address from the address fields returned from many of the other RapidJNI functions.

Syntax

```
public String FormatAddress(String sAddress, int iOptions);
```

Parameters

sAddress	the address fields in Australian address record format as the first parameter.
iOptions	formatting options, possible values defined in FormatAddress method

Return Value

Formatted address.

RapidJNI.DpidToAddress

This function will return the address with the given DPID.

Syntax

```
public String DpidToAddress(int Dpid);
```

Parameters

DPID	DPID
------	------

Return Value

Address with this DPID in [Australian address record format](#).

See Also

[FormatAddress](#)

RapidJNI.DpidToBarcode

This function will return a barcode string based on the DPID (see [Barcodes](#)).

Syntax

```
public String DpidToBarcode(String Dpid, String FormatCode, String CustInfo,  
    boolean NCoding);
```

Parameters

Dpid	8-digit DPID
------	--------------

FormatCode	FCC value specifying type of Barcodes : 11 – Standard customer barcode (37 bars, no customer information) 59 – Customer barcode 2 (52 bars including 16 bars of customer information) 62 – Customer barcode 3 (67 bars including 31 bars of customer information)
CustInfo	customer information to be encoded into the barcode.
Ncoding	specifies the Customer Information coding : False (zero) for C-coding or True (non-zero) for N-coding.

Return Value

Barcode string.

RapidJNI.PostcodeToBsp

This function will return the Barcode Sort Plan (BSP) number for the given postcode.

Syntax

```
public int PostcodeToBsp(int Postcode);
```

Parameters

Postcode Postcode on which to enquire

Return Value

BSP corresponding to postcode.

See Also

[PostcodeToNpsp](#)

RapidJNI.PostcodeToNpsp

This function will return the National PreSort Plan (NPSP) number for the given postcode.

Syntax

```
public int PostcodeToNpsp(int Postcode);
```

Parameters

Postcode Postcode on which to enquire

Return Value

NPSP corresponding to postcode.

See Also

[PostcodeToBsp](#)

RapidJNI.FindStatus

This function will return the status of the previous call to [GetHintAddress](#) or [GetHintAddresses](#).

Syntax

```
public int FindStatus();
```

Parameters

None.

Return Value

Number indicating result:

0 – OK, all results returned

- 1 – results exceeded [MaxAddressesReturned property](#), not all results returned
- 2 – time exceeded [TimeLimit property](#), not all results returned

See Also

[GetHintAddress](#), [GetHintAddresses](#)

RapidJNI.ParsedInput

This function returns the input of the previous call to the [GetHintAddress](#) or [GetHintAddresses](#).

This input was in the AddressLine and LocalityLine parameters, and is returned here parsed into [Australian address record format](#).

Syntax

```
public String ParsedInput();
```

Parameters

None.

Return Value

Results of previous call to the [GetHintAddress](#) or [GetHintAddresses](#) parsed into [Australian address record format](#).

See Also

[GetHintAddress](#), [GetHintAddresses](#)

RapidJNI.ChangeDatabase

When the RapidJNI object is created, the default database (the Address Reference File) is opened. Use this function to change the default database and open an alternate database.

If this function fails and raises an error, there will be no open database to work with. A successful call to this function must be made before using most of the other functions.

Syntax

```
public void ChangeDatabase(int iOptions, String Database);
```

Parameters

iOptions	determines how the Address Reference File is loaded into memory when it is opened: 0 (Registry), 1 (Disk), 2 (Memory) or 3 (Mapping), see ARF loading options .
Database	the Address Reference File to open, see ARF file naming .

RapidJNI.ValidateFullAddress

This function parses a free-form address, standardising it and looking it up in the Address Reference File.

Syntax

```
public String ValidateFullAddress(String sIn, int iOptions);
```

Parameters

sIn	free-form address
iOptions	1 (stricter matching) 2 (more lenient matching)

Return Value

String with the address found, in [Australian address record format](#).

RapidJNI.GetPopupHint

This function is used by IQ Rapid Address to provide the user with a popup list to help the user complete the address.

The first time GetPopupHint is called, the criteria of what to find are passed as parameters, and the first popup item is returned. Subsequent calls pass no parameters (blank and zero), and return subsequent popup items. When there are no more items on the popup list, an empty string is returned.

Syntax

```
public String GetPopupHint(String sLocality, String sGroup, int iOptions)
```

Parameters

sLocality	Some locality information, such as a locality name of the beginning thereof, the state or the postcode.
sGroup	Some address line information, such as a street address, building name, postal address or the beginning of any of these.
iOptions	One or more option values added (bit-wise OR-ed), see list below.

Option values

- 1 popup information for the locality line
- 2 popup information for the address line (mutually exclusive with 1 above)
- 4 don't list building names (used with option 2 above)
- 8 don't list postal types (used with option 2 above)
- 16 don't list level types (used with option 2 above)
- 32 don't list flat/unit types (used with option 2 above)
- 64 don't list street names (used with option 2 above)

RapidJNI.Cleanup

Releases resources. Call this function when finished using RapidJNI and do not subsequently call any other functions, as results will be unpredictable.

Syntax

```
public void Cleanup();
```

IQ Rapid Address

IQ Rapid Address is an application that enables users to rapidly search for addresses and localities, enter addresses using minimal keystrokes, and validate addresses in the PAF/ARF. For help on using the IQ Rapid Address application, see the [Rapid Address User Manual](#).

IQ Rapid Address is also available via Win32 API calls using [IQRapidForm32](#), and via a COM object using [IQRapidForm](#).

IQ Rapid Address Installation

This section describes steps required for a custom installation, or repair of a standard IQ Rapid Address installation performed by the Setup program.

RapidSvr

The IQ Rapid Address window iRapid.exe needs RapidSvr installed as described in [RapidSvr Installation](#) (one of RapidSvr.exe, RapidSvr.dll or iPaf32.dll).

iRapid attempts to find the RapidSvr COM library in `RapidSvr.exe` or `RapidSvr.dll`, but if neither of these is installed will look in `iPaf32.dll`.

To force iRapid to use `iPaf32.dll` even if RapidSvr is installed, add the following Windows Registry entries:

```
REGEDIT4
[HKEY_LOCAL_MACHINE\Software\Intech\iRapid]
"UseDll"=""
```

StanRt

The IQ Rapid Address window `iRapid.exe` needs StanRt to parse changes made to the final address. It also allows a look-up based on the final address via StanRt. For this, StanRt (`StanRt.exe`, `StanRt.dll` or `Stan32.dll`) needs to be installed, accessible to iRapid, see [StanRt Installation](#).

iRapid will first try to create the StanRt COM object to use. If this fails, it will try to load the `Stan32.dll` library. If this also fails, editing the final address and the StanRt look-up will not be available.

To force iRapid to use `Stan32.dll` instead of the StanRt COM object, make the following registry setting:

```
REGEDIT4
[HKEY_LOCAL_MACHINE\Software\Intech\iRapid]
"Stan32"=""
```

or

```
[HKEY_LOCAL_MACHINE\Software\Intech\iRapid]
"StanRt"="32"
```

To disable the StanRt look-up, set the above registry value to “No” instead of “32”.

Machine wide options

The IQ Rapid Address window options set in the General, Search and Advanced Options screens are stored separately for each user. To set options for all users on a machine, click the **Defaults...** button on the options screen, choose **Set as machine defaults**, then make one of the following registry settings:

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\iRapid]
"MachineWideOptions"=dword:00000007

[HKEY_CURRENT_USER\SOFTWARE\Intech\iRapid]
"MachineWideOptions"=dword:00000007
```

The `MachineWideOptions` is a bit-code which defines the options as 1 (General Options), 2 (Search Options) or 4 (Advanced Options) and can be OR’ed together, for example, 7.

The values of this setting in `HKEY_LOCAL_MACHINE` and `HKEY_CURRENT_USER` are OR’ed together. For example, if HKLM has a value of 1 and HKCU a value of 2, then General and Search options will be determined by the machine defaults, and the user won’t be able to modify them.

IQRapidForm32

The IQRapidForm32 Windows 32-bit API library `IQRapidForm32.dll` provides functions to display the [IQ Rapid Address](#) screen to the user. The address and locality lines are passed to the DLL, and the address selected by the user is returned.

It is functionally equivalent to [IQRapidForm](#), except with a Win32 API interface.

Using IQRapidForm32

Call the [IQRapidShowModal function](#) to display the IQ Rapid Address screen, allowing the user to search for an address. If this function returns `iqrAccept` as the user's choice, then the user has chosen an address. The address can be retrieved via the [IQRapidGetAddressFieldsDelimited function](#), [IQRapidGetAddressField function](#) and [IQRapidFormatAddress function](#).

Use the [IQRapidSetFormOptions function](#) to have more control on how the form is displayed.

To display the form as a modeless form instead, call the [IQRapidShow function](#) instead. A callback function will be called to signal when the form has been closed. If the [IQRapidGetUsersChoice function](#) returns `iqrAccept`, then an address has been chosen, and it can be retrieved as above.

There may be a delay the first time the form is displayed, as it connects to the PAF files.

Before closing your application, call the [IQRapidUnload function](#) to free resources.

IQRapidForm32 Reference

Data Types

The following data types are used in the IQRapidForm32 function reference.

Data type	Description
INT	32-bit signed integer
PINT	Pointer to a 32-bit signed integer
LPSTR	Pointer to a null-terminated string of 8-bit (ANSI) characters.
LPCSTR	Pointer to a constant null-terminated string of 8-bit (ANSI) characters.
BOOL	32-bit boolean value (zero for false, non-zero for true)

IQRapidShowModal function

Shows the IQ Rapid Address screen, allowing the user to search for an address. The window is shown as a modal window, meaning that the function only returns when the window is closed.

Syntax

```
INT IQRapidShowModal (  
    LPCSTR sAddressLine,  
    LPCSTR sLocalityLine,  
    PINT piUsersChoice  
)
```

Parameters

`sAddressLine` Pointer to a null-terminated string that specifies the address line to be searched for.

`sLocalityLine` Pointer to a null-terminated string that specifies the locality line to be searched for.

`piUsersChoice` Pointer to a variable to receive information about how the window was closed.

Return Value

If the function succeeds, the return value is zero.

If the function fails, a message box will be displayed indicating the error, and the return value is non-zero.

Notes

This function puts the address and locality lines on the Search tab of the IQ Rapid Address screen, then click the **Search** button to search for the address.

The results of the user's search are available via the [IQRapidGetAddressFieldsDelimited function](#) and [IQRapidGetAddressField function](#).

The button used to close the window is placed in `piUsersChoice`, and is one of the [IQUsersChoiceEnum](#) constants.

Use the [IQRapidSetFormOptions function](#) to set the options for the window before calling this function.

To show a modeless window, use the [IQRapidShow function](#) instead.

IQRapidShow function

Shows the IQ Rapid Address screen, allowing the user to search for an address. The window is shown as a modeless window, meaning that the function returns immediately after the window is shown. To indicate when the user has closed the window, a callback function will be called.

Syntax

```
INT IQRapidShow (
    LPCSTR sAddressLine,
    LPCSTR sLocalityLine,
    CALLBACKPROC pOnClose
)
```

Parameters

`sAddressLine` Pointer to a null-terminated string that specifies the address line to be searched for.

`sLocalityLine` Pointer to a null-terminated string that specifies the locality line to be searched for.

`pOnClose` Pointer to an application-defined callback function, that takes no parameters and has no return value. It is called by **IQRapidForm32** when the user closes the window.

Return Value

If the function succeeds, the return value is zero.

If the function fails, a message box will be displayed indicating the error, and the return value is non-zero.

Notes

This function puts the address and locality lines on the Search tab of the IQ Rapid Address screen, then clicks the **Search** button to search for the address.

The results of the user's search are available via the [IQRapidGetAddressFieldsDelimited function](#), [IQRapidGetAddressField function](#) and [IQRapidGetUsersChoice function](#).

Use the [IQRapidSetFormOptions function](#) to set the options for the window if required before calling this function.

To show a modal window instead, use the [IQRapidShowModal function](#).

IQRapidGetAddressFieldsDelimited function

Retrieves the components of the address the user selected.

These components are tab delimited and in the [IQAddressFieldsEnum](#) order.

To retrieve an individual address component, split up this property into fields, or use the [IQRapidGetAddressField function](#) which will do this for you.

Syntax

```
INT IQRapidGetAddressFieldsDelimited (
    LPSTR sFields,
    INT nBufSize
)
```

Parameters

`sFields` Pointer to the buffer to receive the null-terminated string containing the address fields.

`nBufSize` Specifies the length in characters of the above buffer.

Return Value

If the function succeeds, the return value is zero.

If the buffer is too small, the return value is the size of the required buffer.

IQRapidGetAddressField function

Retrieves a component of the address selected by the user.

Syntax

```
INT IQRapidGetAddressField (  
    INT iField,  
    LPSTR sFields,  
    INT nBufSize  
)
```

Parameters

`iField` The index of the address component desired – one of the [IQAddressFieldsEnum](#) constants.

`sFields` Pointer to the buffer to receive the null-terminated string containing the address field.

`nBufSize` Specifies the length in characters of the above buffer.

Return Value

If the function succeeds, the return value is zero.

If the buffer is too small, the return value is the size of the required buffer.

Notes

If no address has been chosen, or if the index `iField` is out of bounds, an empty string is retrieved.

The address components can also be retrieved by the [IQRapidGetAddressFieldsDelimited function](#).

IQRapidGetAddressLine function

Retrieves the text currently in the address line on the IQ Rapid Address screen.

Syntax

```
INT IQRapidGetAddressLine (  
    LPSTR sLine,  
    INT nBufSize  
)
```

Parameters

`sLine` Pointer to the buffer to receive the null-terminated string containing the address line.

`nBufSize` Specifies the length in characters of the above buffer.

Return Value

If the function succeeds, the return value is zero.

If the buffer is too small, the return value is the size of the required buffer.

See Also

[IQRapidGetLocalityLine function](#)

IQRapidGetLocalityLine function

Retrieves the text currently in the locality line on the IQ Rapid Address screen.

Syntax

```
INT IQRapidGetLocalityLine (  
    LPSTR sLine,  
    INT nBufSize  
)
```

Parameters

sLine Pointer to the buffer to receive the null-terminated string containing the locality line.

nBufSize Specifies the length in characters of the above buffer.

Return Value

If the function succeeds, the return value is zero.

If the buffer is too small, the return value is the size of the required buffer.

See Also

[IQRapidGetAddressLine function](#)

IQRapidGetFormOptions function

Returns the options used when displaying the IQ Rapid Address screen.

Syntax

```
INT IQRapidGetFormOptions (VOID)
```

Return Value

Any combination of the [IQFormOptionsEnum](#) constants OR'ed together.

IQRapidSetFormOptions function

Sets the options used when displaying the IQ Rapid Address screen.

Syntax

```
VOID IQRapidSetFormOptions (  
    INT iOptions  
)
```

Parameters

iOptions Any number of the [IQFormOptionsEnum](#) constants OR'ed together.

Notes

Set these options before calling the [IQRapidShow function](#) or the [IQRapidShowModal function](#).

The default value of this property is zero, unless set in the [IQRapidForm Registry Entries](#).

See Also

See [IQRapidSetFormOptions function](#).

IQRapidGetUsersChoice function

Returns one of the [IQUsersChoiceEnum](#) constants, indicating how the IQ Rapid Address screen was closed.

Syntax

```
INT IQRapidGetUsersChoice (VOID)
```

Return Value

One of the [IQUsersChoiceEnum](#) constants.

IQRapidFormatAddress function

Formats the address selected by the user.

Syntax

```
INT IQRapidFormatAddress (  
    INT iOptions,  
    LPSTR sAddress,  
    INT nBufSize  
)
```

Parameters

iOptions	Any number of the IQFormatOptionsEnum constants OR'ed together, specifying the format desired.
sAddress	Pointer to the buffer to receive the null-terminated string containing the formatted address.
nBufSize	Specifies the length in characters of the above buffer.

Return Value

If the function succeeds, the return value is zero.

If the buffer is too small, the return value is the size of the required buffer.

Note

The formatted address is also returned as part of the address components, available via the [IQRapidGetAddressFieldsDelimited](#) function and the [IQRapidGetAddressField](#) function.

IQRapidGenerateBarcode function

Generates a barcode string based on the DPID of the address chosen by the user and optionally some other customer information, (see [Barcodes](#)).

Syntax

```
INT IQRapidGenerateBarcode (  
    INT iFormat,  
    LPCSTR sCustInfo,  
    BOOL NCoded,  
    LPSTR sBarcode,  
    INT nBufSize  
)
```

Parameters

iFormat	The barcode format code – usually one of the IQBarcodeFormatsEnum constants.
sCustInfo	Pointer to a null-terminated string that specifies the customer information to be included in the barcode string. If a character in this string cannot be coded, an empty barcode-string will be retrieved. If the number of characters in this string is greater than the maximum, it will be truncated.
NCoded	Specifies the Customer Information coding : zero for C-coding or non-zero for N-coding.
sBarcode	Pointer to the buffer to receive the null-terminated string containing the barcode string.
nBufSize	Specifies the length in characters of the above buffer.

Return Value

If the buffer is too small, the return value is the size of the required buffer. If the buffer is big enough, the return value is zero.

Note

The StandardBarcode string is returned as the `pafBarcode37` address component, retrieved via the [IQRapidGetAddressFieldsDelimited function](#) and the [IQRapidGetAddressField function](#).

IQRapidUnload function

Unloads the IQ Rapid form, and frees any resources.

This function should be called before terminating your application.

Syntax

```
VOID IQRapidUnload (VOID)
```

IQRapidForm

IQRapidForm is a COM object providing functions to display the [IQ Rapid Address](#) screen to the user. The address and locality lines are passed to the Object, and the address selected by the user is returned.

It is functionally equivalent to [IQRapidForm32](#), except with a COM interface.

IQRapidForm Installation

This section describes steps required for a custom installation, or repair of a standard IQRapidForm installation performed by the Setup program.

IQRapidForm COM object

The file `IQRapidForm.dll` supplies this COM object.

To register the COM object, type the following from the command prompt or from the Run menu:

```
Regsvr32 IQRapidForm.dll
```

To unregister the COM object, type:

```
Regsvr32 /u IQRapidForm.dll
```

Where appropriate, substitute the full path of `IQRapidForm.dll` above.

PAF file access

To connect to the PAF files, this COM object needs either the [PAF Libraries](#) to be installed locally with the PAF files stored locally, or the [RapidSvr](#) COM object (which can also be accessed remotely).

IQRapidForm Registry Entries

The default value of the [FormOptions property](#) and [IQRapidSetFormOptions function](#) can be set in the Windows registry.

This registry entry example below is shown in Registry Files (.reg) format.

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\iRapid]
"IQRapidForm_FormOptions"=dword:00000000

[HKEY_CURRENT_USER\SOFTWARE\Intech\iRapid]
"IQRapidForm_FormOptions"=dword:00000000
```

The setting can be in either `HKEY_LOCAL_MACHINE` or `HKEY_CURRENT_USER`, the latter overriding the former.

Using IQRapidForm

Call the [ShowModal method](#) to display the IQ Rapid Address screen, allowing the user to search for an address. If this method returns [IQUsersChoiceEnum](#) `iqrAccept`, the user has chosen an address, which can then be retrieved using the [AddressFieldsDelimited property](#), [AddressFields property](#) and [FormatAddress method](#).

Use the [FormOptions property](#) to have more control on how the form is displayed.

To display the form as a modeless form instead, call the [Show method](#). The [OnClose event](#) will signal when the form has been closed. If the [UsersChoice property](#) equals `iqrAccept`, then an address has been chosen, and it can be retrieved as above.

There may be a delay the first time the form is displayed, as it connects to the PAF files.

Use ARF [Properties property](#) to specify the Address Reference File used.

IQRapidForm Summary

Methods

[ShowModal method](#) – show IQ Rapid Address screen in modal form

[Show method](#) – show IQ Rapid Address screen in modeless form

[FormatAddress method](#) – return formatted address

[GenerateBarcode method](#) – return barcode string of DPID

Properties

[FormOptions property](#) – set or return IQ Rapid Address screen [IQFormOptionsEnum](#) options

[Properties property](#) – set or return ARF used

[AddressFields property](#) – return formatted address component

[AddressFieldsDelimited property](#) – return formatted complete address

[AddressLine property](#) – return address line from screen

[LocalityLine property](#) – return locality line from screen

[UsersChoice property](#) – return [IQUsersChoiceEnum](#) value

Enums

[IQFormOptionsEnum](#) – defines IQ Rapid Address screen buttons

[IQAddressFieldsEnum](#) – defines order of fields returned

[IQBarcodeFormatsEnum](#) – defines barcode formats

[IQFormatOptionsEnum](#) – defines address formatting

[IQUsersChoiceEnum](#) – defines how IQ Rapid Address screen was closed

Events

[OnClose event](#) – notify IQ Rapid Address screen closed

IQRapidForm Reference

ShowModal method

Shows the IQ Rapid Address screen as a modal form, allowing the user to search for an address.

Syntax

`Object.Show (AddressLine, LocalityLine)`

Syntax description

Object an IQRapidForm object.

AddressLine a string containing the address line to be searched for.

LocalityLine a string containing the address line to be searched for.

Return Value

This method returns an integer indicating how the form was closed, the same value as the [UsersChoice property](#).

Notes

This method shows IQ Rapid Address screen as a modal form, puts the address and locality lines on the Search tab, then clicks the **Search** button to search for the address.

The results of the user's search are available via the [AddressFieldsDelimited property](#), [AddressFields property](#) and [UsersChoice property](#).

Use the [FormOptions property](#) to set the options for the form if required before calling this method.

To show a modeless form, use the [Show method](#) instead.

Show method

Shows the IQ Rapid Address screen as a modeless form, allowing the user to search for an address.

Syntax

Object.Show (AddressLine, LocalityLine)

Syntax description

Object an IQRapidForm object.

AddressLine a string containing the address line to be searched for. This may be empty.

LocalityLine a string containing the address line to be searched for. This may be empty.

Notes

This method shows IQ Rapid Address screen as a modeless form, puts the address and locality lines on the Search tab, then clicks the **Search** button to search for the address. When the closes the screen an [OnClose event](#) is fired.

The results of the search are available via the [AddressFieldsDelimited property](#), [AddressFields property](#) and [UsersChoice property](#).

Use the [FormOptions property](#) to set the options for the form if required before calling this method.

To show a modal form, use the [ShowModal method](#) instead.

AddressFields property

Returns a String with a component of the address the user chose.

Syntax

Object.AddressFields (Index)

Syntax description

Object an IQRapidForm object.

Index one of the [IQAddressFieldsEnum](#) constants, indicating which component is desired.

Notes

If no address has been chosen, or if the *Index* is out of bounds, an empty string is returned.

The address components can also be retrieved by the [AddressFieldsDelimited property](#).

AddressFieldsDelimited property

Returns a String with all the components of the address selected by the user.

These components are tab delimited and in the [IQAddressFieldsEnum](#) order.

To retrieve an individual address component, split up this property into fields, or use the [AddressFields property](#) which will perform this operation.

AddressLine property

Returns a String with the text currently in the address line on the IQ Rapid Address screen.

LocalityLine property

Returns a String with the text currently in the locality line on the IQ Rapid Address screen.

FormatAddress method

Returns a String with the formatted address selected by the user .

Syntax

`Object.FormatAddress (Options)`

Syntax description

Object an IQRapidForm object.

Options any number of the [IQFormatOptionsEnum](#) OR'ed together, specifying the format desired.

Notes

The address is formatted from the individual address components available via the [AddressFieldsDelimited property](#).

FormOptions property

Returns or sets the options used when displaying the IQ Rapid Address screen.

Set these options before calling the [Show method](#) or [ShowModal method](#).

These options are any number of the [IQFormOptionsEnum](#) constants OR'ed together.

The default value of this property is zero, unless set in the [IQRapidForm Registry Entries](#) .

GenerateBarcode method

Returns a barcode String using the DPID of the address selected by the user (see [Barcodes](#)).

Syntax

`Object.GenerateBarcode (Format, CustomerInfo, NCoded)`

Syntax description

Object an IQRapidForm object

Format an integer specifying the barcode format code, usually one of [IQBarcodeFormatsEnum](#)

CustomerInfo a string containing other information to be included in the barcode. If a character in this string cannot be coded, an error will be generated, and if the number of characters in the string is greater than the maximum, it will be truncated.

NCoded specifies the [Customer Information coding](#): False (zero) for C-coding or True (non-zero) for N-coding.

IQAddressFieldsEnum

This enumerated type defines the order of the fields returned by the [AddressFieldsDelimited property](#) and the [IQRapidGetAddressFieldsDelimited function](#). It is also used to specify the required fields when using the [AddressFields property](#) and the [IQRapidGetAddressField function](#).

Values

```
pafState = 0
pafPostcode = 1
pafLocality = 2
pafStreetName = 3
pafStreetType = 4
pafStreetSuffix = 5
pafPostalType = 6
pafDpid = 7
pafHouseNumber1 = 8
pafHouseSuffix1 = 9
pafHouseNumber2 = 10
pafHouseSuffix2 = 11
pafFlatType = 12
pafFlatNumber = 13
pafLevelType = 14
pafLevelNumber = 15
pafBuildingName1 = 16
pafBuildingName2 = 17
pafLotNumber = 18
pafPostalNumber = 19
pafPostalPrefix = 20
pafPostalSuffix = 21
```

The address components as they appear in the PAF.

```
pafPrimaryPointFlag = 22
```

This field appears here as it does in the PAF and indicates whether the address is a primary point: R (Real Primary Point), P (Phantom Primary Point), or empty (not a primary point).

```
pafSynonymLocalityFlag = 23
```

Indicates whether the synonym locality name chosen is valid: V (synonym is valid and can appear on the letter piece) or U (synonym is unacceptable and the real locality name must appear on the letter piece). This flag will be empty if `pafSynonymLocality` is empty.

```
pafSynonymLocality = 24
```

The synonym of a locality known by more than one name. For example, ASHGROVE QLD 4060 is also known as ST JOHNS WOOD (the locality synonym), so a search for ST JOHNS WOOD would return ST JOHNS WOOD in the `pafSynonymLocality` field and ASHGROVE in the `pafLocality` field. However, a search for ASHGROVE would return ASHGROVE in the `pafLocality` field and nothing in the `pafSynonymLocality` field.

```
pafAddress1 = 25
```

```
pafAddress2 = 26
```

```
pafAddress3 = 27
```

Formatted address lines without the locality line. For more options on formatting the address lines use the [FormatAddress method](#) or [IQRapidFormatAddress function](#).

pafBarcode37 = 28

The standard customer barcode of 37 digits with value 0, 1, 2 or 3, representing the DPID of the chosen address. See [Printing Barcodes](#). To include other information in the barcode, use the [GenerateBarcode method](#) or [IQRapidGenerateBarcode function](#).

pafBsp = 29

The Barcode Sort Plan (BSP) number used for sorting in bulk mail lodgements.

pafNpsp = 30

The National PreSort Plan (NPSP) number used for sorting in bulk mail lodgements.

pafOtherUserInfo = 31

If the user manually edited the address, this field will contain what the user typed after the address. What the user typed before the address will go in pafAddress1.

pafBorderPostcode = 33

pafBorderName = 32

If the user entered a locality, and the address found was in a bordering locality, then the address and postcode the user entered will be returned in these two fields, and the locality of the found address will be returned in the pafLocality and pafPostcode fields.

pafAlternateStreetName = 34

pafAlternateStreetType = 35

pafAlternateStreetSuffix = 36

Some streets are known by alternate names. If the user entered such a street, then the street the user entered will be returned in these three fields, and the 'official' street name will be returned in the pafStreetName, pafStreetType and pafStreetSuffix fields.

pafLocalityDeliveryID = 37

The Delivery ID of the locality of the address found (LocalityDID), may be used in place of the DPID only if there is no GroupDID and no DPID for the address.

pafGroupDeliveryID = 38

The Delivery ID of the group (street) of the address found (GroupDID), may be used in place of the DPID only if there is no DPID for the address.

IQBarcodeFormatsEnum

This enumerated type defines the barcode formats available via the [GenerateBarcode method](#) and [IQRapidGenerateBarcode function](#) (see [Barcodes](#)).

Values

iqrStandardBarcode = 11

Standard customer barcode (37 bars, no customer information)

iqrBarcode2 = 59

Customer barcode 2 (52 bars including 16 bars of customer information)

iqrBarcode3 = 62

Customer barcode 3 (67 bars including 31 bars of customer information).

IQFormOptionsEnum

This enumerated type defines the values used by the [FormOptions property](#), [IQRapidGetFormOptions function](#) and [IQRapidSetFormOptions function](#). Any number of these options can be combined via a bitwise OR. The [IQUsersChoiceEnum](#) can be retrieved with the [UsersChoice property](#) or [IQRapidGetUsersChoice function](#).

Values

`iqrStopButton = 1`

A **Stop** will be displayed on the form. Clicking this button will close the form and set the [IQUsersChoiceEnum](#) to `iqrStop`

`iqrSkipButton = 2`

A **Skip** will be displayed on the form. Clicking this button will close the form and set the [IQUsersChoiceEnum](#) to `iqrSkip`

`iqrCancelButton = 4`

A **Cancel** will be displayed on the form. Clicking this button will close the form and set the [IQUsersChoiceEnum](#) to `iqrCancel`

`iqrAutoAccept = 8`

If there is only one address matching the input address and locality lines, the form won't be displayed, and the found address will be returned. Similarly, if at any stage while the form is displayed, only one address matches the search criteria the user enters, the found address will be automatically accepted. Without this option, the user needs to click on the **Accept** button to close the form and return the chosen address.

IQFormatOptionsEnum

This enumerated type defines the way the address is formatted via the [FormatAddress method](#) and the [IQRapidFormatAddress function](#). Any number of these options can be combined via a bitwise OR.

Values

`iqrOneLine = 1`

The formatted address will only take up one line. Otherwise, it might span several lines, with a CarriageReturn LineFeed combination separating the lines.

`iqrNoLocality = 2`

The locality, state and postcode are not included in the formatted address.

`iqrExpandFlatLevel = 4`

Flat (unit) and level types will be fully spelled out, e.g. "Suite 3 Level 10". Without this option, their abbreviations will be used, e.g. "SE 3 L 10".

`iqrEmbedSynonym = 8`

If a synonym locality name has been chosen, then the real locality name will appear in parenthesis after the synonym name. Without this option, only one name will appear: the synonym name if it is a valid synonym, or the real locality name if not. This option is not applicable if the **iqrNoLocality** option is set.

`iqrExpandStreet = 16`

Street types and suffixes will be fully spelled out, eg "Main Street East". Without this option, abbreviations will be used, e.g. "Main St E".

IQUsersChoiceEnum

This enumerated type defines how the user closed the IQ Rapid Entry form. It is returned by the [UsersChoice property](#), the [ShowModal method](#), the [IQRapidGetUsersChoice function](#) and the [IQRapidShowModal function](#).

Values

`iqrAccept = 0`

The form was closed because an address was chosen.

`iqrCancel = 1`

The form was closed either by clicking the **Cancel** button, or one of the usual ways to close windows, such as clicking on the X at the top right of the window, using the menu at the top left of the window, or pressing Alt+F4) or .

`iqrStop = 2`

The form was closed by clicking the **Stop** button.

`iqrSkip = 3`

The form was closed by clicking the **Skip** button.

Note that the **Cancel**, **Stop** and **Skip** buttons are only displayed if set in the [FormOptions property](#) / [IQRapidSetFormOptions function](#).

Properties property

Set or query a given named property.

Syntax

`Object.Properties (Name)`

Syntax description

Object an IQRapidForm object.

Name the name of the property to set or query. Currently the only property name supported is "ARF". Use this property to set or query the Address Reference File (ARF) to be used for the next call to the [Show method](#) or the [ShowModal method](#).

UsersChoice property

Returns one of the [IQUsersChoiceEnum](#) constants, indicating how the IQ Rapid Address screen was closed.

OnClose event

This event is fired when the IQ Rapid Address screen is closed after being shown by the [Show method](#). It has no parameters.

Reference

Barcodes

Australia Post uses a 4-state barcode to barcode and automatically sort mail, which is composed of a sequence of the digits 0, 1, 2 and 3. There are 37, 52 or 67 such digits, which are converted into actual bars to make the barcode and grouped in fields as shown below.

Field	Bars	Description
Start Bars	2	Identifies start of barcode, values 1 and 3.
FCC	4	Two digit number that identifies the type of barcode:

		11 – Standard customer barcode (37 bars)
		87 – Routing barcode (37 bars)
		45 – Reply Paid barcode (37 bars)
		92 – Redirection barcode (37 bars)
		59 – Customer barcode 2 (52 bars including 16 bars of customer information)
		62 – Customer barcode 3 (67 bars including 31 bars of customer information)
		44 – Currently Reserved
DPID	16	Unique mail delivery identifier
Customer Information	16 or 31	For customers to store their own information in customer barcode 2 (52 bar) and 3 (67 bar).
Reed Solomon Error Correction	12	Quality control bars which minimise errors due to faulty printing, smudging or reflectance.
Stop Bars	2	Identifies end of barcode, values 1 and 3.

Customer Information coding

Customer Information can be in any required format, or encoded by calling the functions to use one of the following methods:

- N-coding – numbers, each represented as two bar state values
- C-coding – a total of 64 characters (uppercase letters, lowercase letters, numbers, space and hash), each represented as three bar state values

These encoding methods enable information to be stored as shown below

FCC	Description	Total Bars	Customer Info Bars	Max Digits (N coding)	Max Chars (C coding)
11	Standard customer barcode	37	0	n/a	n/a
59	Customer barcode 2	52	16	8	5
62	Customer barcode 3	67	31	15	10

Printing Barcodes

To print barcodes, print the digits generated by a barcode function using the 4-state barcode font supplied with this SDK in 18pt size to conform to the Australia Post standards.

Record Formats

Australian address record format

The format of the address fields passed to and from many of the [RapidSvr](#) and [PAF Libraries](#) functions is a string delimited by the [Delimiter property](#) (default is tab character), with the fields groups as listed below. The output format can be defined as XML if required in the [OutputType property](#).

Locality – State, Postcode and Locality Name

- 1 State
- 2 Postcode
- 3 Locality Name

Group – Street Name, Street Type, Street Suffix & Postal Type

- 4 Street Name
- 5 Street Type
- 6 Street Suffix

7 Postal Type

Point – DPID through to Primary Point Flag

8 DPID (or Address ID)

9 Street Number 1

10 Street Number 1 Suffix

11 Street Number 2

12 Street Number 2 Suffix

13 Flat/Unit Type

14 Flat/Unit Number

15 Level

16 Level Number

17 Building Name 1

18 Building Name 2

19 Lot Number

20 Postal Number

21 Postal Number Prefix

22 Postal Number Suffix

23 [Primary Point Flag](#)

[Synonym Localities](#) – Synonym Locality Flag and Synonym Locality.

24 Synonym Locality Flag

25 Synonym Locality

26 [Final Address Flag](#)

27 Border Name

28 Border Postcode

29 Alternate Street Name

30 Alternate Street Type

31 Alternate Street Suffix

32 Locality Delivery ID

33 Group Delivery ID

34 [Status Flag](#)

35 [Amended Flag](#)

Pre Address, Post Address, Barcode37 and Flag String – only populated by the [ValidateFullAddress\(Id\) function](#) and the [ValidateFullAddress method](#).

36 Pre Address

37 Post Address

38 Barcode37

39 Flag String

Address lines – do not contain the locality, state and postcode, only returned if the [ReturnFormattedAddress property](#) is set to True.

40 Address Line 1

41 Address Line 2

GIF Flags – only populated by the [GetDpid\(Id\) function](#), and [GetDpidNonAmas\(Id\) function](#).

42 GIF Flags

GIF Fields – only populated if the Geographical Codes are set by the [GeographicalCodes property](#).

43-52 GIF fields

Empty record

An empty record is made up of only delimiters defined by the [Delimiter property](#).

Primary Point Flag

A primary point is a street number address (with a value in the Street Number 1 field), and with no secondary information (Street Number Suffix, Flat/Unit Type, Flat/Unit Number, Level or Level Number).

This flag can take have the following values:

R (real primary point) a primary point that is a deliverable address

P (phantom primary point) not a deliverable address, but if flat/unit or level information or a house number suffix is provided on the address, then it can be delivered to.

Synonym Localities

Some localities are known by different names, but are the same locality, for example PYRMONT NSW 2009 (the real locality name) is also known as DARLING ISLAND (the synonym locality name).

This flag can take have the following values:

V valid synonym that can be used on the mail piece

U unacceptable synonym which must be corrected

If the record does not contain a synonym locality, this flag will be empty.

Backward compatibility

In previous releases, the synonym name was included within the *Locality* field, with the real name in parenthesis, e.g. “DARLING ISLAND (PYRMONT)”. There was no *Synonym Locality* field.

For backward compatibility, this option is still available by setting the [EmbedSynonym property](#) to **True**. In this case, the *Synonym Locality* field will be always be empty.

Final Address Flag

A final address that is passed to the [GetAddress\(Id\) function](#) or the [GetAddress method](#) will return the same address. A non-final address will usually return a list of sub-addresses.

This flag can take have the following values:

F address returned from the [RapidSvr](#) and [PAF Libraries](#) functions is a final address

(empty) partial address returned

The examples below show the value of this flag when using the [GetAddressHint\(Id\) function](#) or [GetHintAddress method](#) with PAF 2004.3.

Example 1

A search for a locality “Bondi” returns (amongst others) “Bondi Beach NSW 2026” and “Bondi Forest NSW 2632”. Bondi Beach is not a final address, as there are many sub-addresses (streets) in Bondi Beach. Bondi Forest is a final address, as there are no sub-addresses (streets or postal deliveries) in the PAF for this locality.

Example 2

A search for “Spring Street”, “Bondi” returns (amongst others) “35 Spring St, Bondi” and “2 Spring St, Bondi”. Number 35 is not a final address, as there are sub-addresses (level or suites) at number 35. Number 2 is a final address, as there are no sub-addresses (units or levels) at this number.

Example 3

A search for “35 Spring St” “Bondi”, returns (amongst others) “35 Spring St” and “Level 7 35 Spring St”. They are both marked as final, even though “35 Spring St” has sub-addresses. This is because it is also a final address in its own right (it is a Primary Point), and its sub-addresses such as “Level 7 35 Spring St”, have already been returned.

Status Flag

This flag is returned by the [ValidateFullAddress\(Id\) function](#) and the [ValidateFullAddress method](#). It is the [Flag](#) field returned by the DpidGeocode grammar file, which indicates how the input address matched an address in the ARF, or why it didn’t match.

The [GetAddressHint\(Id\) function](#), [GetNextAddressHint\(Id\) function](#), [GetHintAddress method](#) and [GetHintAddresses method](#) return a confidence rating in this flag, indicating how well the returned address matched the search criteria.

Amended Flag

This flag is returned by the [ValidateFullAddress\(Id\) function](#) and the [ValidateFullAddress method](#). It is the [Amended Flag Code](#) field in the DpidGeocode grammar file that indicates how the address was amended to match an address in the PAF.

It is also used by the [GetHintAddress method](#), the [GetHintAddresses method](#) and the [GetAddressHint\(Id\) function](#), with the [SearchPostcodes property](#) set to True. If a locality is returned only because it is in the same postcode as another found locality, then this flag will be given a value of “W”.

Input record

The input record is a list of fields in [Australian address record format](#), all of which must be in uppercase, which is input into the [GetAddress\(Id\) function](#), [GetAddress method](#) and [GetAddresses method](#) to determine what is listed.

The content of the input record determines the content of the [Output record](#). Use the options below to select the content of the output record.

All localities

Pass an empty record, either an empty string, or one with any number of delimiters.

All localities with a specific postcode

Pass a record with just the Postcode in the Postcode field (a delimiter, the postcode and any number of delimiters).

All localities in a specific state (or NZ City)

Pass a record with just the State in the State field (the first field).

All locality names with a specific beginning

Pass a record with just the beginning of the locality name in the Locality Name field, e.g. SYD in the locality field will return SYDNEY, SYDENHAM, SYDNEY UNIVERSITY etc.

All streets and postal types (e.g. PO Box) in a specific locality

Pass a record with the exact locality information only (State, Postcode and Locality Name) in the respective fields.

All house numbers in a specific street

Pass a record with the exact locality information (State Postcode and Locality Name), and the street information (Street Name, Street Type & Street Suffix) in the respective fields.

All units and levels for a specific house number

Pass a record with the exact locality and street information (as above) and the exact house number (House Number 1, House Number 1 Suffix, House Number 2 and House Number 2 Suffix) in the respective fields.

All sub-addresses matching specific unit/level

Pass a record with the exact locality, street and house number (as above) in the respective fields. Include one or more of the desired Unit Type, Unit Number, Level and Level Number.

All postal numbers for a specific postal type

Pass a record with the exact locality information (State, Postcode & Locality Name) and the exact Postal Type in the respective fields.

A list of street names with a specific beginning

Pass a record with the beginning of the street name followed by a star in the Street Name field. (e.g. "SPRIN*" will return Spring, Sprint etc.).

All streets with a specific street name

Pass a record with the street name in the Street Name field.

To limit the list to streets with a given street type, pass the street type (or the beginning of the street type followed by a star) in the Street Type field.

To limit the list to a certain locality, pass the locality name (or the beginning of the locality name followed by a star) in the Locality Name field.

All streets in a given postcode

Pass a record with the Postcode in the Postcode field, and an asterisk in the Street Name field.

An address with a specific DPID or AddressId

Pass a record with the DPID or AddressId in the DPID/AddressId field.

This functionality requires an index on the DPID/AddressId.

All addresses with a specific Geographical code

Pass a record with the Geographical Code(s) to search for in its appropriate field.

The names of the Geographical codes need to be first set via the [GeographicalCodes property](#). This functionality requires an index on the Geographical Code or nothing will be returned.

Output record

The content of the output record depends on the content of the [Input record](#).

Localities

When requesting localities, a record with only the locality information (State, Postcode and Locality Name) will be returned.

Groups

When requesting streets and postal types, a record with the locality information and street information (Street Name, Street Type, Street Suffix) and Postal Type will be returned.

Points

When requesting house numbers, unit and level numbers or postal numbers, full PAF records will be returned.

When requesting house numbers, only one record will be returned for every house number, even though there might be several units and/or levels at that house number. This record might be a full PAF record with a DPID.

Street Names

When requesting street name, a record with only the Street Name field populated will be returned.

Streets

When requesting streets, a record with the street information (Street Name, Street Type and Street Suffix) and locality information (State Postcode and Locality Name) will be returned.

Lookups

When requesting addresses with a certain ID or Geographical code, the full address record will be returned.

New Zealand Record Format

The format of a New Zealand address record differs slightly from an [Australian address record format](#) as shown in the table below.

Field	Australia	New Zealand
1	State	City or Mail Town
2	Postcode	
3	Locality Name	Suburb, RD number or Lobby
4	Street Name	
5	Street Type	
6	Street Suffix	
7	Postal Type	
8	DPID	
9	Street Number 1	
10	Street Number 1 Suffix	
11	Street Number 2	Not used
12	Street Number 2 Suffix	Not used
13	Flat/Unit Type	
14	Flat/Unit Number	
15	Level Type	
16	Level Number	
17	Building Name 1	Building or Lobby name
18	Building Name 2	Not used
19	Lot Number	Not used
20	Postal Number	
21	Postal Number Prefix	
22	Postal Number Suffix	

23	Primary Point Flag	
24	Synonym Locality Flag	
25	Synonym Locality	
26	Final Address Flag	
27	Border Name	
28	Border Postcode	
29	Alternate Street Name	
30	Alternate Street Type	
31	Alternate Street Suffix	
32	Locality Delivery ID	Not used
33	Group Delivery ID	Not used
34	Status Flag	
35	Amended Flag	
36	Pre Address	
37	Post Address	
38	Barcode37	Not used
39	Flag String	
40	Address Line 1	
41	Address Line 2	
42	GIF Flags	
43-52	GIF fields	

There are two different formats for the New Zealand PAF. They differ in the position of the Lobby name and Rural Delivery number.

First Format

For urban addresses, the suburb is in the third field (Locality Name field).

For postal addresses, the third field is empty, and the Postal Lobby name is in the Building Name field (field 17).

For rural addresses, the third field is empty, and the Rural Delivery number is available via a GIF field.

Second Format

The third field (Locality Name field) is used according to the type of the address:

For an urban address, it contains the suburb.

For a postal addresses, it contains the Postal Lobby name.

For a rural address it contains the Rural Delivery number (e.g. “RD 1”).

For Oamaru rural addresses, the Rural Delivery number may only be a letter (e.g. “RD C”) at locality and street level, and a letter with a number (e.g. “RD 1C”) at (street and) street number level.

Address Reference Files

ARF loading options

There are four options which define if and how an Address Reference File is loaded into memory when it is opened.

0 – Registry

The option used depends on the [Open PAF configuration setting](#).

1 – Disk

The Address Reference Files are read from the disk and are not loaded into memory.

This is the simplest option.

2 – Memory

The Address Reference Files are pre-loaded into memory when opened.

This option should only be used if there is enough free RAM to contain all the files, but will provide greater performance when batch processing addresses. Pre-loading into memory will cause a small delay while opening the files.

3 – Mapping

The Address Reference Files are accessed via the File Mapping method, in which the operating system loads them into memory when required, and unloads them when memory runs low.

This option provides greater performance when batch processing addresses.

ARF file naming

The *Database* parameter passed to the [OpenArfId function](#) or [ChangeARF method](#) can be one of the following:

6. An absolute directory path, e.g.

```
\Intech\Arf
C:\Program Files\Intech\Paf\
/usr/local/intech/arf3
```

7. The name of the sub-directory of the [ARF base directory](#). In this case, ensure that *Database* ends with a slash but does not begin with one, e.g.

```
Paf2007.3\
jun05/
```

8. The name of the ARF stored in the [ARF Paths](#).
9. An empty string, indicating that the default [PAF Directory](#) is used.

PAF files

The Postal Address File (PAF) is a database of addresses in Australia provided by Australia Post which is over 850Mb in size. Intech Solutions supply these files in a sorted, compressed and indexed format to save storage space and improve performance.

A standard installation of the PAF Files is performed by the Setup program. If a custom installation is required, or the installation becomes unstable install the files below in the defined [PAF Directory](#).

The following 16 files make up the compressed PAF files:

```
Altern.db
Altern.p
Border.cdb
Group.db
Group.in1 or Street.tin
Group.in2
Group.p
Locality.db
Locality.in2 or Locality.tin
Locality.in4
Locality.p
PAF.ver
```

Point.cbd
Postcode.idx
Synonym.db
Synonym.p

Locality	Locality name, Postcode and State. Each Locality record points to a range of Group records in this locality.
Group	Street name or Postal type (e.g. PO Box). Each Group record points to a range of Points that in this locality.
Point	House number, Lot number or Postal number, and sometimes unit, level and/or building information.
Synonym	Synonym (vanity) names for localities. Each Synonym record points to the Locality record of which it is a synonym.
Border	Bordering localities per locality. Each Locality has any number of bordering localities.

The following 6 files are optional:

Dpid.in1 or AddrId.hin
Point.in3
Point.in4
Street.in1
Bsp.db
Npsp.db

PAF.ver	Stores the version and expiry date of the PAF files. It is an AMAS requirement that after the expiry date, DPIDs are no longer allocated.
Dpid.in1	Index on DPID
Point.in3	Index on building name
Point.in4	Index on building name
Street.in1	Index on streets
Bsp.db	Contains the Barcode Sort Plan numbers
Npsp.db	Contains the National PreSort Plan numbers.

PAF abbreviations

The following abbreviations are used in the records returned from the PAF. Other Address Reference Files (ARFs) may use other abbreviations.

State Abbreviations

PAF state abbreviations are listed below.

AAT AUSTRALIAN ANTARCTIC TERRITORY

ACT AUSTRALIAN CAPITAL TERRITORY

NSW NEW SOUTH WALES

NT NORTHERN TERRITORY

QLD QUEENSLAND

SA SOUTH AUSTRALIA

TAS TASMANIA

VIC VICTORIA

WA WESTERN AUSTRALIA

Street type abbreviations

PAF street abbreviations are listed below.

ACCS	ACCESS
ALLY	ALLEY
ALWY	ALLEYWAY
AMBL	AMBLE
ANCG	ANCHORAGE
APP	APPROACH
ARC	ARCADE
ART	ARTERY
AVE	AVENUE
BASN	BASIN
BCH	BEACH
BDGE	BRIDGE
BDWY	BROADWAY
BEND	
BLK	BLOCK
BRAE	
BRCE	BRACE
BRK	BREAK
BROW	
BVD	BOULEVARD
BYPA	BYPASS
BYWY	BYWAY
CAUS	CAUSEWAY
CCT	CIRCUIT
CDS	CUL-DE-SAC
CH	CHASE
CIR	CIRCLE
CL	CLOSE
CLDE	COLONNADE
CLT	CIRCLET
CMMN	COMMON
CNR	CORNER
CNWY	CENTREWAY
CON	CONCOURSE

COVE	
COWY	CROSSWAY
CPS	COPSE
CRCS	CIRCUS
CRD	CROSSROAD
CRES	CRESCENT
CRSG	CROSSING
CRSS	CROSS
CRST	CREST
CSO	CORSO
CT	COURT
CTR	CENTRE
CTTG	CUTTING
CTYD	COURTYARD
CUWY	CRUISEWAY
DALE	
DELL	
DEVN	DEVIATION
DIP	
DR	DRIVE
DRWY	DRIVEWAY
DSTR	DISTRIBUTOR
EDGE	
ELB	ELBOW
END	
ENT	ENTRANCE
ESP	ESPLANADE
EST	ESTATE
EXP	EXPRESSWAY
EXTN	EXTENSION
FAWY	FAIRWAY
FITR	FIRETRAIL
FLAT	
FOLW	FOLLOW
FORM	FORMATION
FRNT	FRONT
FRTG	FRONTAGE

FSHR	FORESHORE
FTRK	FIRE TRACK
FTWY	FOOTWAY
FWY	FREEWAY
GAP	
GDN	GARDEN
GDNS	GARDENS
GLD	GLADE
GLEN	
GLY	GULLY
GR	GROVE
GRA	GRANGE
GRN	GREEN
GRND	GROUND
GTE	GATE
GTES	GATES
HILL	
HRD	HIGHROAD
HTS	HEIGHTS
HWY	HIGHWAY
INTG	INTERCHANGE
INTN	INTERSECTION
JNC	JUNCTION
KEY	
LDG	LANDING
LEES	
LINE	
LINK	
LKT	LOOKOUT
LNWY	LANEWAY
LOOP	
LT	LITTLE
LWR	LOWER
MALL	
MNDR	MEANDER
MT	MOUNT
MWY	MOTORWAY

NOOK	
OTLK	OUTLOOK
PARK	
PART	
PASS	
PATH	
PDE	PARADE
PHWY	PATHWAY
PIAZ	PIAZZA
PKLD	PARKLANDS
PKT	POCKET
PKWY	PARKWAY
PL	PLACE
PLAT	PLATEAU
PLZA	PLAZA
PNT	POINT
PORT	
PROM	PROMENADE
QDGL	QUADRANGLE
QDRT	QUADRANT
QUAD	
QY	QUAY
QYS	QUAYS
RAMP	
RCH	REACH
RD	ROAD
RDGE	RIDGE
RDS	ROADS
RDSD	ROADSIDE
RDWY	ROADWAY
RES	RESERVE
REST	
RGWY	RIDGEWAY
RIDE	
RING	
RISE	
RMBL	RAMBLE

RND	ROUND
RNDE	RONDE
RNGE	RANGE
ROW	
ROWY	RIGHT OF WAY
RSBL	ROSEBOWL
RTE	ROUTE
RTT	RETREAT
RTY	ROTARY
RUE	
RUN	
RVR	RIVER
RVRA	RIVIERA
RVWY	RIVERWAY
SBWY	SUBWAY
SDNG	SIDING
SHWY	STATE HIGHWAY
SLPE	SLOPE
SND	SOUND
SPUR	
SQ	SQUARE
ST	STREET
STPS	STEPS
STRA	STRAND
STRP	STRIP
STRS	STAIRS
SWY	SERVICE WAY
TARN	
TCE	TERRACE
THOR	THOROUGHFARE
TKWY	TRUNKWAY
TLWY	TOLLWAY
TOP	
TOR	
TRI	TRIANGLE
TRK	TRACK
TRL	TRAIL

TRLR	TRAILER
TURN	
TWRS	TOWERS
UPAS	UNDERPASS
UPR	UPPER
VDCT	VIADUCT
VIEW	
VLLS	VILLAS
VSTA	VISTA
WADE	
WAY	
WHRF	WHARF
WKWY	WALKWAY
WYND	
YARD	

Flat/Unit type abbreviations

These are the flat or unit type abbreviations in the PAF:

APT	APARTMENT
CTGE	COTTAGE
DUP	DUPLEX
F	FLAT
FY	FACTORY
HSE	HOUSE
KSK	KIOSK
MB	MARINE BERTH
MSNT	MAISONETTE
OFF	OFFICE
PTHS	PENTHOUSE
RM	ROOM
SE	SUITE
SHED	
SHOP	
SITE	
SL	STALL
STU	STUDIO
TNHS	TOWNHOUSE
U	UNIT

VLLA	VILLA
WARD	
WE	WAREHOUSE

Street suffix abbreviations

These are the street suffix abbreviations in the PAF:

CN	CENTRAL
E	EAST
EX	EXTENSION
LR	LOWER
N	NORTH
NE	NORTH EAST
NW	NORTH WEST
S	SOUTH
SE	SOUTH EAST
SW	SOUTH WEST
UP	UPPER
W	WEST

Floor/Level type abbreviations

These are the floor or level type abbreviations in the PAF:

B	BASEMENT
FL	FLOOR
G	GROUND FLOOR
L	LEVEL
LG	LOWER GROUND FLOOR
M	MEZZANINE
UG	UPPER GROUND FLOOR

Postal delivery type abbreviations

These are the postal delivery type abbreviations in the PAF:

CARE PO	CARE OF POST OFFICE
CMA	COMMUNITY MAIL AGENT
CMB	COMMUNITY MAIL BAG
CPA	COMMUNITY POSTAL AGENT
GPO BOX	GENERAL POST OFFICE BOX
LOCKED BAG	
MS	MAIL SERVICE
PO BOX	POST OFFICE BOX
PRIVATE BAG	

RMB	ROADSIDE/RURAL MAIL BAG/BOX
RMS	ROADSIDE MAIL SERVICE
RSD	ROADSIDE DELIVERY

Configuration Settings

[RapidSvr](#), the [PAF Libraries](#) and other components rely on configuration settings defining certain behaviours, locations of files, etc. Refer to the [IQ Office Configuration Manual](#) for details of system configuration settings

Windows

In Windows, these settings are stored in the Windows Registry, which is shown in Registry File (*.reg) format here, e.g.

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech]
"HomeDir"="C:\\Program Files\\Intech"
```

This example shows that under the key

```
HKEY_LOCAL_MACHINE\SOFTWARE\Intech
```

there is a string value named HomeDir with value

```
C:\Program Files\Intech
```

For ease of reading, the line “REGEDIT4” is often omitted.

UNIX / LINUX

In UNIX / LINUX, these settings are stored in a configuration text file which is searched for in the following order:

1. A file called ‘.intech’ in the current directory
2. A file specified by the INTECH_CONFIG_FILE environment setting
3. A file called ‘.intech’ in the HOME directory
4. The file ‘/usr/etc/.intech’.

This file contains one key/value pair per line in the form “key/subkey=value”, e.g.:

```
HomeDir=/home/bob/intech
RapidSvr/Properties/TimeLimit=4
```

Default property values

The default values for all the [Read/Write properties](#) can be defined in the [Configuration Settings](#).

Windows platform

For each property whose default value is to be set, create a string value under the key:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\RapidSvr\Properties]
```

For example:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\RapidSvr\Properties]
"TimeLimit"="4"
"Compliance Type"="AMAS"
"Output Type"="Delimited"
```

UNIX / LINUX platform

For each property whose default value is to be set, create a key beginning with “RapidSvr/Properties/” and the property name.

For example:

```
RapidSvr/Properties/TimeLimit=4
RapidSvr/Properties/Compliance Type=AMAS
RapidSvr/Properties/Output Type=Delimited
```

Backward compatibility - Windows

For backward compatibility, the following registry settings are also valid:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\RapidSvr]
"TimeLimit"=dword:00000004
"Compliance Type"="AMAS"
"Output Type"="Delimited"
"MaxAddressesReturned"=dword:00000064
```

Backward compatibility - UNIX/LINUX

For backward compatibility, the following configuration file settings are also valid:

```
RapidSvr/TimeLimit=4
RapidSvr/Compliance Type=AMAS
RapidSvr/Output Type=Delimited
RapidSvr/MaxAddressesReturned=100
```

StanCOM registry setting

This [Configuration Settings](#) is only for Windows platforms. This setting specifies whether the StanRt COM object is used to parse the address line. If set to `Yes`, then the StanRt COM object will be used. If not set or set to `No`, then Stan32.dll will be used directly.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\RapidSvr]
"StanCOM"="Yes"
```

Home Directory

This [Configuration Settings](#) specifies the Intech home directory, in which licence file(s) must be located. The actual directory may be different to those in the examples below.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech]
"HomeDir"="C:\\Program Files\\Intech"
```

Configuration Text File example

```
HomeDir=/home/intech
```

PAF Directory

This [Configuration Settings](#) specifies the default location of the compressed Address Reference Files (ARF/PAF). The actual directory may be different to those in the examples below.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech]
"PafDir"="C:\\Program Files\\Intech\\Paf"
```

Configuration Text File example

```
PafDir=/home/intech/paf
```

Grammar Directory

This [Configuration Settings](#) specifies the location of the grammar files. The actual directory may be different to those in the examples below.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\StanRt]
"GrammarPath"="C:\\Program Files\\Intech\\Grammars"
```

Configuration Text File example

```
StanRt/GrammarPath=/home/intech/grammars
```

Open PAF configuration setting

This [Configuration Settings](#) specifies how the [PAF Libraries](#) will load the PAF files into memory. It can have one of three values: “Disk”, “Mem” and “Map” as described in [ARF loading options](#). If none are specified, the default of “Disk” is used and the files are not loaded into memory but read from disk.

Note that this setting does not affect [RapidSvr](#), which always opens the PAF files with the “Disk” option, unless otherwise specified via the [ChangeARF method](#).

Windows Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech]
"OpenPaf"="Map"
```

Configuration Text File example

```
OpenPaf=Map
```

ARF base directory

This [Configuration Settings](#) specifies the base (parent) directory of other Address Reference Files.

When opening an Address Reference File via the [ChangeARF method](#) or [OpenArfId function](#), the name of the Address Reference File can be specified as a sub-directory of the base ARF directory. In this setting, one specifies this base directory.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech]
"ArfBaseDir"="C:\\Program Files\\Intech\\ARFs"
```

Configuration Text File example

```
ArfBaseDir=/home/intech/arf/
```

Windows Registry example (for before version 4.0.1.43)

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\Arf]
"BaseDir"="C:\\Program Files\\Intech\\ARFs"
```

Configuration Text File example (for before version 4.0.1.43)

```
Arf/BaseDir=/home/intech/arf/
```

ARF Paths

This [Configuration Settings](#) specifies the location of Address Reference Files.

When opening an Address Reference File via the [OpenArfId function](#) or [ChangeARF method](#), the name of the Address Reference File can be specified as a pre-defined path here.

Windows platform

For each pre-defined path, create a string value under the key:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\ARF]
```

For example:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\ARF]
"PAF2007.1"="C:\\Program Files\\Intech\\PAF\\2007.1"
"PAF2007.4"="C:\\Program Files\\Intech\\PAF\\2007.4"
"GNAF_V6"="C:\\Program Files\\Intech\\GNAF\\V6"
```

UNIX / LINUX platform

For each pre-defined path, create a key beginning with “ARF/” and the ARF name.

For example:

```
ARF/paf2007.1=/home/intech/paf/2007/1
ARF/paf2007.4=/home/intech/paf/2007/4
ARF/gnaf6=/home/intech/gnaf/6
```

TcpHost and TcpPort setting

This [Configuration Settings](#) specifies the name and the port of the socket server to use if configuring a [Socket Client](#).

Define both the TcpHost and TcpPort to specify the socket server, but do not define only the host or port. TcpHost can be either a domain name or IP address.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\RapidSvr]
"TcpHost"="myserver.mycompany.com"
"TcpPort"="4000"
```

Configuration Text File example

```
RapidSvr/TcpHost=myserver.mycompany.com
RapidSvr/TcpPort=4000
```

Open GIF configuration setting

This [Configuration Settings](#) specifies how the IQ Geographical Index Sub System (iqgiss.dll) will load the Geographical Information Files (GIF) files into memory. It can have one of three values: “Disk”, “Mem” and “Map” as described in [ARF loading options](#). If none are specified, the default of “Disk” is used and the files are not loaded into memory but read from disk.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech]
"OpenGif"="Map"
```

Configuration Text File example

```
OpenGif=Map
```

COM Audit File configuration setting

This [Configuration Settings](#) is used for debugging [RapidSvr](#). If this setting is set, then each time a RapidSvr COM object is created, a log entry will be written to this file.

Example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\RapidSvr]
"ComAuditFile"="C:\\Temp\\RapidSvr_COM_Audit.log"
```

Socket Server Autoload

This [Configuration Settings](#) specifies which Address Reference Files (ARF), if any, are to be loaded when the [Socket Server](#) is started. These ARFs will remain loaded, and prevent the need for the socket server to open and close frequently used ARFs.

The example below indicates to autoload the default ARF (indicated by the empty string) and the ARF called GNAF. The names of the entries (“1” & “2” in this example) are ignored, and anything can be used.

Windows platform

```
[HKEY_LOCAL_MACHINE\Software\Intech\IQSocSvr\AutoloadARFs]
"1"=""
"2"="GNAF"
```

UNIX / LINUX platform

```
IQSocSvr/AutoloadARFs/1=
IQSocSvr/AutoloadARFs/2=GNAF
```

SOAP Server Autoload

This [Configuration Settings](#) specifies which Address Reference Files (ARF), if any, are to be loaded when the [SOAP Server](#) is started. These ARFs will remain loaded, and prevent the need for the SOAP Server to open and close frequently used ARFs.

The example below indicates to autoload the default ARF (indicated by the empty string) and the ARF called GNAF. The names of the entries (“1” & “2” in this example) are ignored, and anything can be used.

Windows platform

```
[HKEY_LOCAL_MACHINE\Software\Intech\IQSoapSv\AutoloadARFs]
"1"=""
"2"="GNAF"
```

UNIX / LINUX platform

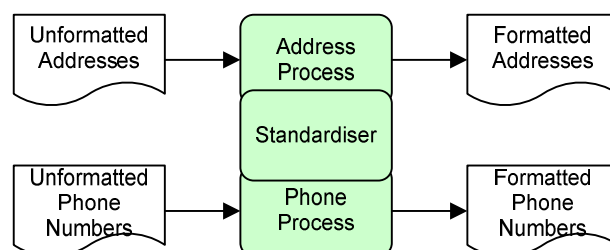
```
IQSoapSv/AutoloadARFs/1=
IQSoapSv/AutoloadARFs/2=GNAF
```


IQ Office Grammar File Reference

Overview

Standardising is the process of converting unformatted text data such as addresses, names and telephone numbers into data in a *standard format* performed by IQ Office Standardiser, a suite of applications and library functions supplied with IQ Office DTS and Enterprise editions. This typically cleanses, transforms, enhances and validates the input data, providing greatly improved and high quality output data.

Standardising is performed by one or more standardising *processes*. Each process takes one type of alphanumeric input data such as addresses, names or telephone numbers, *parses* the input data to break it into components, and returns each required component in a specified format.

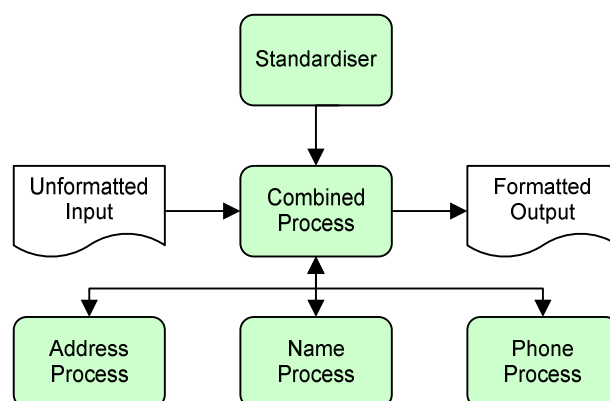


Each standardising process has a *grammar file*, which defines how the data will be standardised and formatted. The Standardiser reads and interprets the grammar file before performing the defined standardising processes.

Grammar files are text files written in a simple programming language. They contain definitions of the number, type and format of input data fields; instructions on how Standardiser is to process input data; references to lookup-lists such as standard abbreviations; and definitions of the number, type and format of output data fields.

Existing grammar files can be easily edited to *customise* them to the specific requirements of an organisation, for example, to process address data received in a variety of formats, and output data in the specific format used by a particular database or automated mailing system. New files can also be written to fulfil the needs of new business processes, for example, if an organisation installs a new account management system.

Grammar files can contain *calls* to other grammar files, for example, the “Name, Address and Phone” grammar file would call three separate grammar files and combine the returned output.



Grammar files can also contain calls to other processes, data sources, libraries and *address reference files*. For example, the *DpidGeocode* grammar file supplied with IQ Office Standardiser uses the IQ Office PAF Libraries to look up an address in the Postal Address File (PAF) containing all Australian postal addresses supplied by Australia Post, and returns the formatted output and DPID.

About this manual

This manual provides a reference to the grammar files supplied with IQ Office. It should be read by anyone using the grammar files to standardise data. The actual files supplied depend on the IQ Office edition and data licence which has been purchased.

Each grammar file description includes:

- A summary of the function of the file
- The expected input fields into the file
- The output fields generated when the file is run
- Examples of typical input and output from the file showing how data is processed
- A description of how to perform minor customisation of the file by editing internal parameters or external files.

Refer to the following manuals for details of other IQ Office components:

- The [Grammar File SDK Reference Manual](#) contains detailed instructions on how to create, edit and run grammar files.
- The [Standardising Overview](#) summarises the standardising process and the IQ Office Standardiser components which standardise data.

Installation of Grammar Files

Grammar files are installed with a standard IQ Office installation to the following Windows directory:

```
C:\Program Files\Intech\Grammars\*.grm
C:\Program Files(x86)\Intech\Grammars\*.grm
```

This directory also contains a number of grammar files beginning with an underscore _. These are internal system files which must not be edited or deleted. It also contains a number of text files (*.txt) containing list and tables which must not be edited unless otherwise indicated in the description of each grammar file.

Summary of Grammar Files

IQ Office can supply the following categories of grammar file, depending on the software edition and data licence. All except System files can be customised to specific customer requirements.

System	Internal system files providing common functions to other grammar files. These files must not be edited or deleted.
Standard	Supplied with all IQ Office Editions. Standardise and validate Australian addresses, including BSP, Barcodes and DPID look up.
Enhanced	Only supplied with IQ Office DTS and Enterprise Editions. Standardise names, phone numbers, email addresses and organisations, and generate phonetic algorithm codes for words.
Data	Specific to the licensed and installed data package. Standardise Australian addresses, and provide brief or detailed Australian Bureau of Statistics (ABS) statistical boundary information, electoral administrative boundaries and geocodes.
Country	Specific to a country, such as New Zealand. Standardise and validate New Zealand addresses.

Grammar	Category	Data Type	Data Processed
AddressEnhancer	Standard	Addresses	Australian addresses.

AddressField	Standard	Addresses	Checks presence of address components.
ArfGeocode	Standard	Addresses	Australian addresses and geographical coordinates.
CustInfoBarcode	Standard	Addresses	Return barcode from DPID.
Dpid3Pass	Standard	Addresses	Australian addresses using lenient matching rules.
DpidDefault	Standard	Addresses	Australian addresses.
DpidDefault3Pass	Standard	Addresses	Australian addresses.
DpidGeocode	Standard	Addresses	Australian addresses using AMAS-compliant matching.
DpidToBarcode	Standard	Addresses	Return barcode from DPID.
PostcodeToSortPlans	Standard	Addresses	Postcode to BSP and NPSP.
AccountNames	Enhanced	Persons and Organisations	Relationships between and names of organisations or individuals.
AliasNames	Enhanced	Persons	Nicknames and common names.
AusPhone	Enhanced	Telephone numbers	Convert pre-1997 to current Australian telephone numbers.
Company	Enhanced	Organisations	Parse names of companies.
DecodeBarcodeScanner	Enhanced	Addresses	Decodes output of barcode scanner.
Email	Enhanced	Addresses	Parse email addresses.
FixLastLine	Enhanced	Addresses	Fix last line of Australian addresses.
FixLastLine2	Enhanced	Addresses	Fix last line of Australian addresses.
IndNames	Enhanced	Persons	Full or abbreviated names of up to two individuals, including titles and initials.
IndNames3a	Enhanced	Persons	Full or abbreviated names of up to five individuals, including honorifics, titles, initials and suffixes.
Organisation3	Enhanced	Organisations	Names of up to three organisations.
OrgOrIndividual	Enhanced	Persons and Organisations	Names of up to three organisations and eight individuals.
Phone	Enhanced	Telephone numbers	Australian telephone numbers.
Phone2	Enhanced	Telephone numbers	Australian and international telephone numbers.
RemoveAddress	Enhanced	Addresses	Remove address from phrase.
RuleInList	Enhanced	Values	Compare values, see Rule Grammars.
RuleInSizeRange	Enhanced	Values	Compare values.
RuleInValueRange	Enhanced	Values	Compare values.
RuleIsValidAddress	Enhanced	Addresses	Validate addresses.
Soundex	Enhanced	Words	Generate phonetic algorithm codes for words.

NzGeocode	Country	Addresses	Replaced by NzAddress
NzAddress	Country	Addresses	New Zealand address details.
NZDefault	Country	Addresses	New Zealand address summary.
NZValidate	Country	Addresses	Validate New Zealand addresses.
ABSGeocode	Data	Addresses	Australian addresses and ASGC 2006 ABS information.
AdminGeocode	Data	Addresses	Australian addresses and detailed ABS information including administrative electoral boundaries.
RapidFormat	System	Formatting	Customise output format of IQ Rapid Address

Geographical coordinate accuracy

The `reliability` field in various grammar files is an indication of the accuracy of the geographical coordinate shown in the tables below.

New Zealand

Level	Description
1	XY Coordinates are an exact match to LINZ reference address point.
2	XY Coordinates are an exact match ignoring subdwelling address elements.
3	Match to street - manual geocoding match to a street.

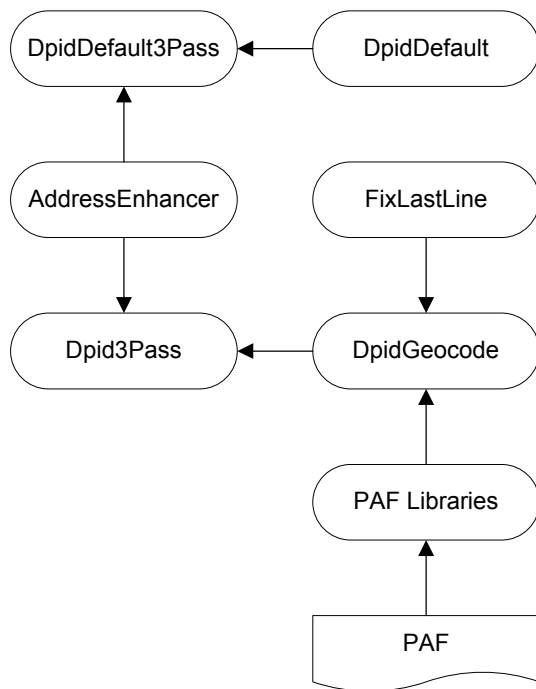
Australia (GNAF)

Level	Description	Example
1	Geocode resolution recorded to appropriate surveying standard.	Address level geocode was manually geocoded with a GPS.
2	Geocode resolution sufficient to place centroid within address site boundary.	Address level geocode was automatically calculated by centroiding the cadastre parcel to which it correlated.
3	Geocode resolution sufficient to place centroid near or possibly within address site boundary.	Address level geocode was automatically calculated by calculating where on the road the address was likely to appear based on other bounding geocoded addresses.
4	Geocode resolution sufficient to associate address site with a unique road feature.	Street level geocode automatically calculated by using the road centreline reference data.

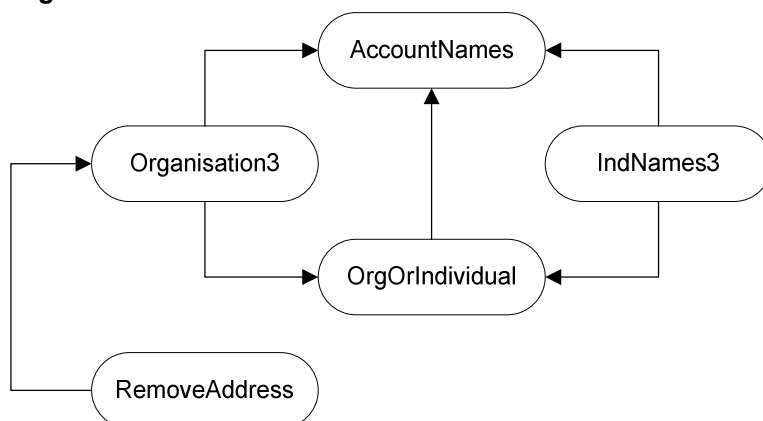
Dependencies

The diagrams below show dependencies which exist between some Standard and Enhanced grammar files.

DPID and Geocode



Organisations and Individuals



Customising with DEFINES

DEFINES are internal settings in some grammar files which can be easily turned on and off to customise the grammar file. These make use of the hash (#) character, which indicates that anything after the hash is a comment and ignored. Remove the hash (#) at the beginning of the DEFINE line to turn the setting on, or add a hash to turn it off, as shown in the examples below.

This setting is on, with a comment after the setting:

```
\DEFINE CHECKFOR_SURNAME_INITIAL # turned on 1 Oct 1010
```

This setting is off:

```
#\DEFINE DOUBLE_SPACE_INVERT
```

This setting defines the value of the ARF_name variable as “GNAF” with a comment above the setting.

```
# Changed GNAF file name to use new data
\DEFINE sARF_NAME GNAF2012
```

This setting has turned off the previously defined value of sGIF_NAME

```
# \DEFINE sGIF_NAME GIF2012
```

Codes

Flag codes

These codes are displayed in the `flag` output field of the ABSGeocode, AdminGeocode, ArfGeocode, Dpid3Pass, DpidGeocode and NzAddress grammar files.

There are three categories of flag: [Error Flag](#), [Correct Flag](#) and [Bit Flag](#).

Error Flags

These flags indicate why a DPID could not be appended to the address.

- A Insufficient locality info
e.g. 29 George St, SYDNEY
- B Cannot match locality info
e.g. 29 George St, SYDNEY VIC
- E PAF validation not performed, an error occurred preventing accessing the PAF files.
- F No street of that name in this locality
e.g. 2 Mad St, SYDNEY NSW 2000
- G No such postal type in that locality
e.g. Private Bag 1, SYDNEY NSW 2001 (There are only GPO Boxes and Locked Bags in 2001)
- I No street name in address
e.g. Melbourne, VIC 3000
- J Postal prefix or suffix doesn't match
e.g. GPO BOX 1B, MELBOURNE VIC 3001 (should be GPO Box 1A)
- K Number range invalid (first number greater than second)
e.g. 6-2 Mill St, Perth WA 6000
- L No house number in that street
e.g. 10000 CAMPBELL ST, HOBART TAS 7000
- M No such postal number
e.g. GPO BOX 999999, BRISBANE QLD 4001
- N No such lot number
e.g. Lot 4 CARRINGTON ST, ADELAIDE SA 5000
- O No such postal address with no number
e.g. CMB, DARWIN NT 0801 (CMB in Darwin have numbers)
- P Missing house number in address
e.g. CAVENAGH ST, DARWIN NT 0800
- Q Ambiguous street (or NZ Lobby)
e.g. 2 MACQUARIE AVE, SYDNEY NSW 2000 (there is Macquarie Street and Place, but no Avenue)
- R Ambiguous house number or within range
e.g. 13-17 CONSTITUTION AVE, CANBERRA ACT 2600 (there is a 13-15 and a 17)
e.g. 341 BRISBANE ST, BEAUDESERT QLD 4285 (the number is 339-347)
- S Matches to more than one address
e.g. 222 Pacific Highway, 2065 NSW (this address appears in several localities Crows Nest, St Leonards and Greenwich)
- U Too many differences. The amendedFlag field will list the differences.
e.g. 29 George, SYDNEY NSW (postcode missing, and street type missing)

- V G/PO Box address with incorrect locality, or changed from GPO to PO Box and postcode incorrect.
e.g. GPO BOX 1, NSW 2001 (missing the locality SYDNEY)
e.g. GPO BOX 1, NORTH SYDNEY NSW 2060 (there is a PO Box 1, North Sydney NSW 2059)
- X Missing unit or level information. This address is not deliverable without unit/level information (Phantom Primary Point)
e.g. 35 Spring Street BONDI JUNCTION NSW 2022 (adding “Level 99” for example to the address would make it acceptable).
- Z Unit or level not found, and no corresponding Primary Point in the PAF
e.g. 1/67 Nerang St, NERANG QLD 4211

Correct Flags

These flags indicate that a DPID could be appended to the address.

- 0 Address correct
- 1 Address matched to primary point only.

There is secondary address information (such as level, unit number and/or house suffix) that could not be found in the PAF, so the DPID for primary point (the address without any secondary information) was returned.
- 2 Address didn’t exactly match, but was left as is.

AMAS rules allow an incorrect locality to be kept in the address, if it exists within the postcode. Similarly, a border locality, some synonym localities or an alternate street name can be kept. This code can only be returned if the flag in “DpidGrammar.txt” is set to keep these address elements. The [Amended Flag code](#) will indicate which address elements differ from the record in the PAF but weren’t corrected. For example, (“The Rocks” is incorrect, but is a locality in the 2000 postcode in 309 Kent St, The Rocks NSW 2000
- 4 Address amended.

The [Amended Flag code](#) will contain a list of address elements amended.
- 8 Address amended on second pass.

Applies only to the Dpid3Pass grammar file, indicating that the address couldn’t be given a DPID according to AMAS matching rules, but after address enhancement, a DPID was found.

The [Amended Flag code](#) will contain a list of address elements amended.
- 16 Address matched to street only.

The street number provided doesn’t exist in the PAF, but the PAF does contain a Group Delivery ID for the street. This Group Delivery ID is returned in the DPID field.
- 32 Address matched to locality only.

The street provided doesn’t exist in the PAF, but the PAF does contain a Locality Delivery ID, which is returned in the DPID field.
- 64 Additional sub address information exists.

Additional secondary address information (such as level, unit number and/or house suffix) is available for that address.
- 128 Geocode match only.

All matching records had same Geocode, so are returning the found Geocode, even though a unique address cannot be found.

Bit Flags

The bit flags can be combined. Examples include:

- 5 Address amended and matched to a primary point only
- 6 Some elements of address were amended and some were left as are.
- 9 Address amended on second pass, and matched to a primary point only.
- 12 Address amended on second pass.
- 20 Address amended and matched to street only.
- 24 Address amended on second pass and matched to street only.
- 36 Address amended and matched to locality only.
- 40 Address amended on second pass and matched to locality only.
- 65 Primary property match and sub addresses exist.
- 68 Sub addresses exist and address was amended.

Amended Flag codes

These codes are displayed in the `amendedFlag` output field of the `ABSGeocode`, `AdminGeocode`, `ArfGeocode`, `Dpid3Pass`, `DpidGeocode` and `NzAddress` grammar files.

Although there are two separate files for Australia (`AmendedFlagCodes.txt`) and New Zealand (`NzAmendedFlagCodes.txt`), all codes are listed together below with country differences noted.

These codes indicate one of the following options:

- what was corrected in the address ([Correct Flag 4](#))
- which elements differed but were not corrected ([Correct Flag 2](#))
- which element differed making a match impossible ([Flag code U](#))

More than one of these flags can be returned if more than one item of the address is corrected.

If some elements were corrected and others not (Correct Flag 6), then the flags indicating corrected elements will come first, followed by a space, followed by flags indicating elements not corrected.

- a Added missing Locality (Aus) or Suburb (NZ)
- b Corrected Locality (Aus) or Suburb (NZ) spelling
- c Changed incorrect Locality (Aus) or Suburb (NZ)
- d Substituted Synonym Locality (Aus) or Suburb (NZ)
- e Removed extra Locality (Aus) or City (NZ)
- f Added missing Postcode
- g Changed incorrect Postcode
- h Added missing State (Aus) or City (NZ)
- i Changed incorrect State (Aus) or City (NZ)
- j Corrected Street Name spelling
- k Added missing Street Type
- l Changed incorrect Street Type
- m Removed extra Street Type
- n Added missing Street Suffix

- o Changed incorrect Street Suffix
- p Removed extra Street Suffix
- q Corrected house number from Single to Range
- r Corrected House number Range
- s Corrected house number from Range to Single
- t Added missing Postal Suffix
- u Added missing Postal Prefix
- v Changed Dash to Slash
- w Substituted Border Locality (Aus) or Suburb (NZ)
- x Substituted Alternate Street
- y Unit or Level incorrect
- z Matched by building
- 2 Lot number corrected, amended or removed
- 3 Corner address
- 4 Address incorrectly formatted (NZ only)
- 5 Rural Delivery (RD) number corrected, amended or removed (NZ only)
- 6 Corrected Lobby (NZ only)
- 7 Corrected City spelling (NZ only)
- 8 Substituted Synonym City (NZ only)

GIF Flags

The `gifFlags` field returned by some grammars indicates how the geographical information associated with the address was obtained by “dump” Coding Aggregation. See also [Geographical coordinate accuracy](#).

There is one character per geographical field, or group of geographical fields. The order of the fields is specific to each grammar file and is linked to the order of the GIF request. Values of the possible flags per group are listed below.

- 0 No geographical information for this address.
- 1 Geographical information associated with the exact street address.
- 2 Geographical information associated with street number (primary point).
- 3 Geographical information associated with possible matching addresses.
- 4 Geographical information associated with closest street address with information.
- 5 Geographical information associated with street.
- 6 Geographical information associated with possible matching streets.
- 7 Geographical information associated with locality.
- 8 Geographical information associated with possible matching localities.
- 9 Geographical information associated with street intersection.
- Z Not licensed for the geographical information requested.
- Y Geographical information file doesn't exist.

System Grammars

System grammar file names begin with an underscore _, and must not be edited or deleted, except for the following.

_RapidFormat

This system grammar is only used to change the displayed output of address fields in IQ Rapid Address if the [User Customizations \(_RapidFormat.grm\)](#) option is checked.

The file can be edited by the user, for example, to change the street abbreviation of Avenue from AVE to AV by changing the streetTypeTable.

Expected Input

Address with standard fields.

Output Fields

Address formatted with changed fields.

Standard Grammars

Standard grammar files supplied with all IQ Office Editions are summarised below.

Grammar	Category	Data Type	Data Processed
AddressEnhancer	Standard	Addresses	Australian addresses.
AddressField	Standard	Addresses	Checks presence of address components.
ArfGeocode	Standard	Addresses	Australian addresses and geographical coordinates.
CustInfoBarcode	Standard	Addresses	Return barcode from DPID.
Dpid3Pass	Standard	Addresses	Australian addresses using lenient matching rules.
DpidDefault	Standard	Addresses	Australian addresses.
DpidDefault3Pass	Standard	Addresses	Australian addresses.
DpidGeocode	Standard	Addresses	Australian addresses using AMAS-compliant matching.
DpidToBarcode	Standard	Addresses	Return barcode from DPID.
PostcodeToSortPlans	Standard	Addresses	Postcode to BSP and NPSP.

AddressEnhancer

This standard grammar file parses an address, looks it up in the PAF, and returns the corrected address. It uses more lenient matching criteria than [DpidGeocode](#) and therefore finds more addresses.

AddressEnhancer is an example of a data enhancement program described in Section 16-4 of the Australia Post AMAS handbook: “It is understood that many organisations offering AMAS approved software will have data enhancement programs in place to improve address information in order to gain higher match rates. This will normally involve those addresses that fail to match through the first cycle of address matching.”

Expected Input

A free form address, possibly with address lines divided by a new-line or tab character.

Output Fields

The field length is shown in brackets (n).

- output (256) The improved cleaned address lines, separated by tab characters.
- flag (10) The [Flag code](#) where applicable, indicating why the address could not be found. This field may be blank even if no address is found.
- amendedFlag (10) The [Amended Flag code](#) where applicable, indicating what was changed in the address to make a match.

Examples

	1 Bank St Bulli NSW 2516	1 Bank Bulli NSW 2516	1 Bank St Bulli NSW	1 Bank Bulli	1 Ban Bulli
output	<address>	<address>	<address>	<address>	<address>
flag					U
amendedFlag		k	f	fbhk	fbjk
	1 Bank St Bulli NSW 2516	1 Bank Bulli NSW 2516	1 Bank St Bulli NSW	1 Bank Bulli	1 Ban Bulli
output	<address>	<address>	<address>	<address>	<address>

AddressField

This standard grammar file parses an address, and returns five flag corresponding to the presence of the address, locality, postcode, other information and state components in the address.

Expected Input

A free form address.

Output Fields

The field length is shown in brackets (n).

- AddressFlag (1) Address component flag: Y (present) or blank
- LocalityFlag (1) Locality component flag: Y (present) or blank
- PostcodeFlag (1) Postcode component flag: Y (present) or blank
- OtherFlag (1) Other component flag: Y (present) or blank
- StateFlag (1) 3 letter state abbreviation

Examples

	1 Bank St Bulli NSW 2516	1 Bank Bulli NSW 2516	1 Bank St Bulli NSW	c/- Bulli Post Office NSW 2516
AddressFlag	Y		Y	
LocalityFlag	Y	Y	Y	Y
PostcodeFlag	Y	Y		Y
OtherFlag				Y
StateFlag	NSW	NSW		NSW

ArfGeocode

This standard grammar parses an Australian address against an Address Reference File (ARF) and also returns the geographical coordinates of the address.

This grammar is similar to [Dpid3Pass](#), except that the name of the ARF can be specified in the input, and, depending on licensing, the latitude and longitude may be returned.

Expected Input

A free form address, with address lines divided by optional new-line or tab character.

Use any of the following prefixes followed by a semicolon before the address:

<ARFname>; Specify the name of the Address Reference File (ARF) to use, e.g. GNAF; This must be the first of the prefixed items.

AmasRules; Use stricter matching rules.

Output Fields

The field length is shown in brackets (n). The output fields are similar to [DpidGeocode](#), except for the following.

Extra fields

latitude (12)	The latitude component of the geographic coordinates of the address.
longitude (12)	The longitude component of the geographic coordinates of the address.
reliability (1)	An indication of the Geographical coordinate accuracy .
gifFlags (10)	A representation of how the geographical information was associated with the address, see GIF Flags .

The DPID field is replaced by:

addressId	An address Id associated with the address returned
-----------	--

Customising

To customise this grammar, edit the following DEFINE parameters (see [Customising with DEFINES](#)).

- `sARF_NAME` – specify the name of the ARF to be used if none is specified in the input, e.g. `\DEFINE sARF_NAME GNAF.`

CustInfoBarcode

Creates an Australia Post barcode code string, which can be printed on a mail item, see [Barcodes Explained](#).

Expected Input

Only the 8-digit DPID if no customer information is required on the barcode, or the following:

- 8-digit DPID
- The type of barcode required (52N, 52C, 67N or 67C)
- The customer information, separated by white space or a punctuation character.

Output Fields

The barcode and the input information. If the input is invalid nothing is returned.

Dpid3Pass

This standard grammar file combines the [DpidGeocode](#) and [AddressEnhancer](#) grammar files.

It first passes the input address to DpidGeocode, and if the address is found in the PAF it is returned in the output. If it is not found, it is passed to AddressEnhancer.

If the address is now found in the PAF, it is passed back to DpidGeocode to get a DPID, as only AMAS-approved software can allocate DPIDs.

Expected Input

The same as [DpidGeocode](#).

Output Fields

The same as [DpidGeocode](#), including [Flag codes](#).

- If the address only went through one pass, there will only be one character in the flag.
- If the address went through two passes, there will be two characters in the flag, one per pass, with the first character the flag from the second pass.
- If a DPID could only be found on the second pass, the flag will be 8 or 9.

DpidDefault

This standard grammar file parses an Australian address, separates it into components, looks up the corresponding DPID, and returns address components, DPID, barcode and barcode sort plan number. It is AMAS approved and normally only used by IQ Easy Post.

Expected Input

A free form address, possibly with address lines divided by a new-line or tab character.

Output Fields

The field length is shown in brackets (n).

address1 (100)	First line of the address, formatted into a 3-line address.
address2 (100)	Second line of the above (if there is a second line).
address3 (100)	Third line of the above (if there is a third line).
locality (30)	Locality name
state (3)	3 letter state abbreviation
postCode (4)	4-digit post code
flag (3)	A Flag code indicating if the address was correct, needed to be corrected, or why no DPID was found.
bsp (3)	Barcode Sort Plan number, created from the postcode.
barcode37 (37)	The 37-digit 4-state barcode sequence, created from the DPID
dpid (8)	DPID (if found)

DpidDefault3Pass

This standard grammar file used by IQ Easy Post combines the [DpidDefault](#) and [AddressEnhancer](#) grammar files.

It first passes the input address to DpidDefault, and if the address is found in the PAF it is returned in the output. If it is not found, it is passed to AddressEnhancer.

If the address is now found in the PAF, it is passed back to DpidDefault to get a DPID, as only AMAS-approved software can allocate DPIDs. It is normally only used by IQ Easy Post.

Expected Input

A free form address, possibly with address lines divided by a new-line or tab character.

Output Fields

The field length is shown in brackets (n).

address1 (100)	First line of the address, formatted into a 3-line address.
----------------	---

address2 (100)	Second line of the above (if there is a second line).
address3 (100)	Third line of the above (if there is a third line).
locality (30)	Locality name
state (3)	3 letter state abbreviation
postCode (4)	4-digit post code
flag (3)	A Flag code indicating if the address was correct, needed to be corrected, or why no DPID was found.
bsp (3)	Barcode Sort Plan number, created from the postcode.
barcode37 (37)	The 37-digit 4-state barcode sequence, created from the DPID
dpid (8)	DPID (if found)

DpidGeocode

This standard grammar file parses an Australian address, separates it into components, looks up the corresponding DPID, and returns address components, DPID, barcode, sort plan numbers, address lines and Soundexes. This grammar file is approved by Australia Post AMAS.

Expected Input

A free form address, possibly with address lines divided by a new-line or tab character.

Output Fields

The field length is shown in brackets (n).

state (3)	3 letter state abbreviation
postCode (12)	4-digit post code
locality (46)	Locality name
streetName (30)	Name of the street for a street address
streetType (4)	Street type abbreviation, e.g. RD, ST, AVE
streetSuffix (2)	Street suffix abbreviation, e.g. W in “Circular Quay West”
postalType (11)	For a postal address, the postal type, e.g. PO BOX
dpid (8)	DPID (if found)
houseNo1 (5)	First house number, e.g. 5 in “5 Main St”, or in “5-7 Main St
houseSuffix1 (1)	House number suffix, e.g. A in “5A Main St”
houseNo2 (5)	Second house number in a range, e.g. 7 in “5-7 Main St
HouseSuffix2 (1)	Not used
flatType (7)	Flat or unit type abbreviation, e.g. U in “Unit 5 / 10 Railway St”
flatNo (7)	Flat or unit number, e.g. 5 in “Unit 5 / 10 Railway St”
levelType (2)	Building level type abbreviation, e.g. L in “Level 4”, or G in “Ground Floor”
levelNo (5)	Building level number, e.g. 4 in “Level 4”
buildingName1 (50)	Name of the building
buildingName2 (30)	Second line of the building name
lotNo (6)	Allotment number, e.g. 6 in “Lot 6 Forrest Lane”

postalNo (5)	Postal number, e.g. 123 in “PO Box X123YY”
postalPrefix (3)	Postal number prefix, e.g. X in “PO Box X123YY”
postalSuffix (3)	Postal number suffix, e.g. YY in “PO Box X123YY”
flag (10)	A Flag code indicating if the address was correct, needed to be corrected, or why no DPID was found.
flagString (80)	The <code>flag</code> and <code>amendedFlag</code> fields in words..
barcode37 (37)	The 37-digit 4-state barcode sequence, created from the DPID
bsp (3)	Barcode Sort Plan number, created from the postcode.
npSP (3)	National PreSort Plan number, created from the postcode.
streetSoundex (4)	4 character Soundex code of the street name (see Phonetic Algorithm).
streetRSoundex (4)	4 character Soundex code of the reverse of the street name
localitySoundex (4)	4 character Soundex code of the street name
flatTypeMatch (7)	FlatType (above) with type “F”, “U” and “APT” cleared, as they are equivalent to no flat type.
levelTypeMatch (2)	Level (above) with “FL” (floor) changed to “L” (level), as they are equivalent.
preAddress1 (50)	Information that appears in the input before the address components
preAddress2 (50)	Second line of the above
address1 (50)	First line of the address, formatted into a 3-line address
address2 (50)	Second line of the above (if there is a second line)
address3 (50)	Third line of the above (if there is a third line)
postAddress (30)	Information that appears in the input after the address components
amendedFlag (10)	Flag indicating which if any part of the address was corrected, see Amended Flag code . The <code>FlagString</code> field holds this flag in words.

Examples

Input: 1 Bank St Bulli NSW 2516

state	NSW
postCode	2516
locality	BULLI
streetName	BANK
streetType	ST
dpid	86554932
houseNo1	1
flag	0
flagString	correct
barcode37	13010122201212113010023232322120313
bsp	013
npSP	209
streetSoundex	B520

streetRSoundex K510
localitySoundex B400
address1 1 Bank St

DpidToBarcode

This standard grammar returns the corresponding barcode from an input DPID.

Expected Input

An eight digit DPID. Returns nothing if invalid DPID input.

Output Fields

The field length is shown in brackets (n).

dpid (8) DPID (if found)
barcode (37) The 37-digit 4-state barcode sequence, created from the DPID

Examples

Input: 86554932

dpid 86554932
barcode 13010122201212113010023232322120313

PostcodeToSortPlans

This standard grammar file looks up an input postcode and returns the corresponding Barcode Sort Plan number (BSP) and National PreSort Plan number (NPSP or NSP).

Expected Input

A postcode. Since the first number in the range 1-9999 is taken as the postcode, do not include any other numbers before the postcode input, e.g. RMB 1040 Pyramid Hill VIC 3575 will return an incorrect postcode of 1040.

Output Fields

The field length is shown in brackets (n).

pcode (4) Four digit Australian postcode
bsp (3) Barcode Sort Plan number (BSP)
npsp (3) National PreSort Plan number (NPSP or NSP)

Examples

	3039	Tas 7150	Derby WA 6728	Locked Bag Penrith NSW 1793
pcode	3039	7150	6728	1793
bsp	023	054	053	008
npsp	314	054	656	204

IQ Office Fix Address

Overview

IQ Office Fix Address is a simple utility for processing address data files such as those output from IQ Standardiser, manually searching for addresses in the PAF in the same way as IQ Rapid Address, and assigning them a DPID. Refer to the IQ Standardiser User Manual for details of using Standardiser, and the Rapid Address User Manual for details of searching.

Fix Address steps through records in an input file, prompts for records that do not have a DPID, and enables addresses to be manually found in the PAF and allocated a corresponding DPID. This creates a copy of the input file incorporating all the corrections.

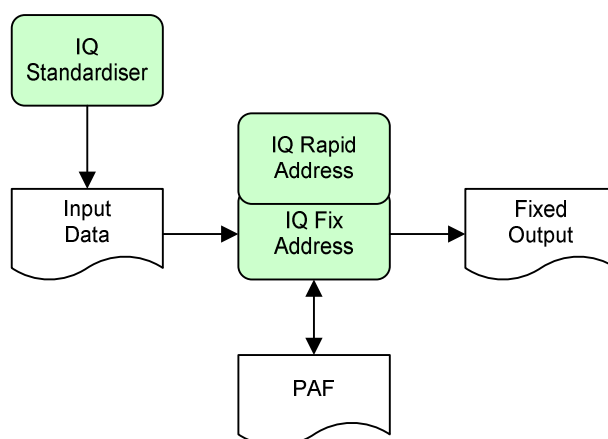
This input file is often produced from IQ Standardiser running the DpidGeocode or Dpid3Pass grammar file. The input file must have:

- Format as flat text with delimited or fixed-width records.
- Either or both a DPID and Barcode37 field.
- Some address fields which enable an address search, which can be updated with any search results.

Fix Address works by reading each record in the input file. If a record already has a DPID or a Barcode, it is copied to the output file as is. If not, FixAddress prompts to search for the address and corresponding DPID in the PAF.

- If the final address is found it can be accepted to copy the record to the output file with the address fields, DPID and/or barcode updated.
- If the final address cannot be found it can be skipped to copy the record to the output file as is without a DPID.
- The manual fix process can be paused at any time and later resumed.

A copy of the processed output file incorporating all corrections is created after manually fixing addresses. The diagram below summarises the manual fix process.



Using Fix Address

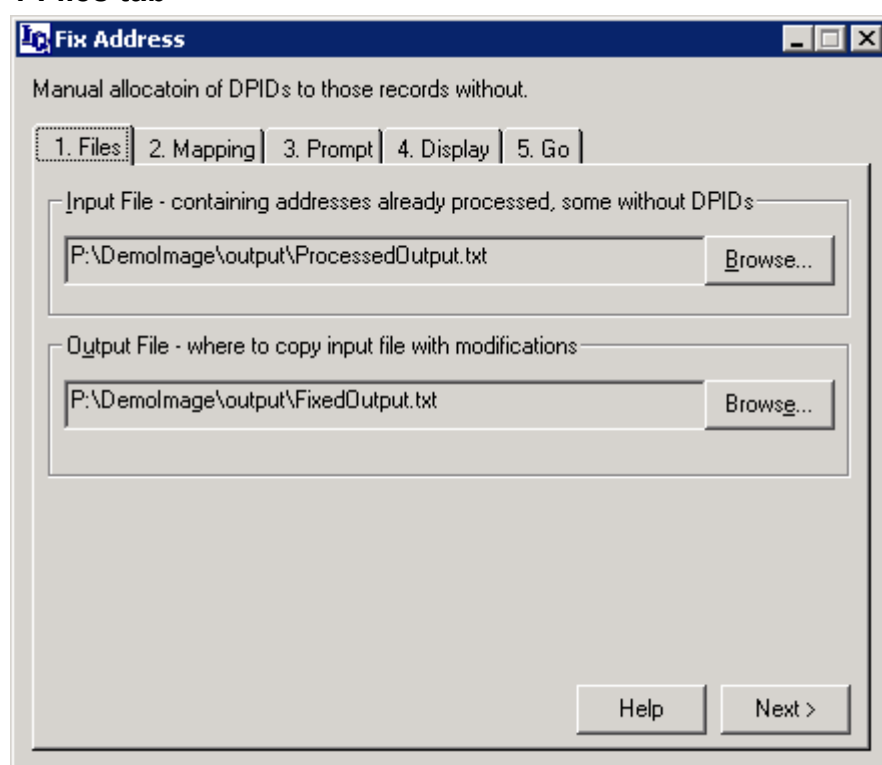
To start Fix Address, select it from the Intech | Other Utilities program menu. This opens the Fix Address panel.

Use Fix Address by filling in each tab in the order below. To select the next tab either click the tab name at the top of the screen, or the Next button at the bottom of the screen.

10. Use the [Files tab](#) to select the input and output data files, then the [Input Text File Format screen](#) to define the format of the input fields. File formats can be loaded or saved using the [Saved Text File Formats](#) dialog box.
11. Use the [Mapping tab](#) to map fields in the input file to fields in the PAF.
12. Use the [Prompt tab](#) to determine the fields in the input file which are used to compose the address and locality line which are used to search for the address.
13. Use the [Display tab](#) to select the fields in the original address file which are displayed while searching for an address, enabling it to be matched to the search results.
14. Use the [Go tab](#) to start reading and searching for records from the input file, and resume address searching.

To exit Fix Address click the X button in the top right of the window, or press Alt+F4.

1 Files tab



Use the Files tab to select the input file of addresses to be processed, and the output file to contain processed addresses.

Input File

Click the Browse button and browse to the name of the input file. The [Saved Text File Formats](#) is then displayed to specify the file format. There is an option to accept the format which has been correctly guessed.

Output File

Click the Browse button and browse to the name of the file. The file will be created if it does not exist. The output file will have the same format as the input file, and for each record in the input file, a record will be created in the output file with the address corrected and DPID and/or Barcode incorporated.

Next

Click to go to the next tab required to fix addresses.

Input Text File Format screen

Text File Format - P:\DemoImage\input\AddressInput.txt

☐ Fixed width
☒ Delimited
☒ First Row Contains Field Names

Field Delimiter: {tab} Text Qualifier: {none}

Ok Cancel Help

Field Name: Organisation Max Field Size: 24

Next Field Previous Field Load... Save...

ID	Organisation	Title	Firstname	Surname	Address
1	Abercrombie Holdings	Mr	Aaron	Aarons	33 Le...
2	Abercrombie Enterprises	Mrs	Aaron	Abba	Le...
3	Aberdare Pty Ltd	Miss	Aron	Abbey	Le...
4	Aberdeen Holdings	Ms	Ari	Abbeywood	99...
5	Aberfeldie Enterprises	Dr	Abbey	Abbotsbury	29...
6	Aberfeldy Pty Ltd	Prof	Gail	Abbotsford	Ur...
7	Aberfoyle Holdings	Sir	Gayle	Abbotsham	10...
9	Aberfoyle Enterprises	Lady	Abbey	Abels	Le...

Click on column header to select it. Then change its max size

Use this screen to define the format of the input file specified in the [Files tab](#). A preview of the input file is displayed at the bottom of the screen. It may also be useful to first view the contents of the file using a text file viewer or editor.

There are two variations of this screen depending on whether each field in the file is delimited by a character such as a tab or comma, or a fixed width.

Delimited

When a delimited file is opened, lines are automatically added at each delimiter (field). Manually edit the fields as required to define the size and name of each input field. All fields must be named, and names must be unique.

Check the option button then select other options:

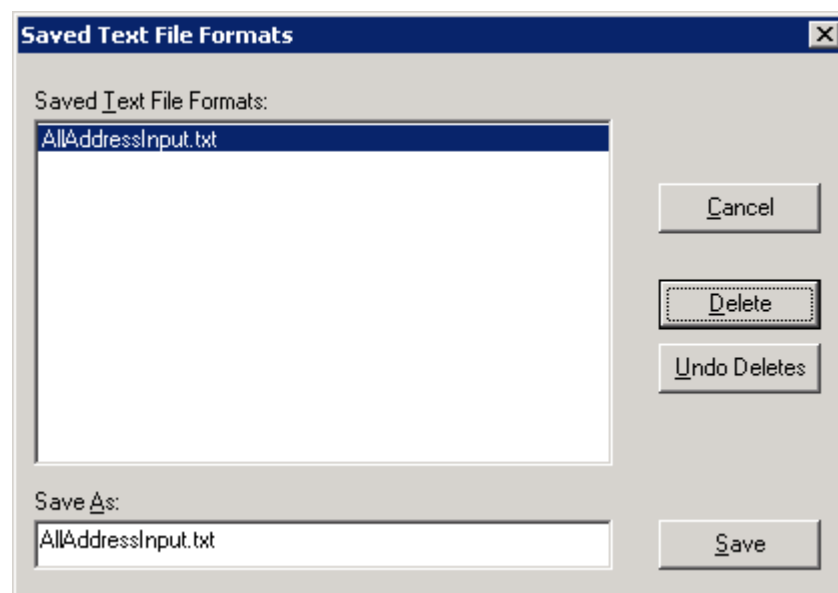
- **First Row Contains Field Names** – check this option if the first row of the file contains the names of the fields. If not, the name of each field must be defined.
- **Field Delimiter** – select the single character that delimits the fields in each row:
, (comma) ; (semicolon) {tab} {space}
- **Text Qualifier** – (optional) select the character that surrounds text in each field:
{none} ' (single quote) " (double quote)
- **Max Field Size** – (optional) enter the maximum width of a field in numbers of characters.
- Use the **Next Field** and **Previous Field** buttons to scroll between fields.
- Click the **Load** or **Save** button to load or save field format definitions in [Saved Text File Formats](#).
- Click the **OK** button when all the fields have been defined.

Fixed Width

When a fixed-width file is opened, lines are automatically added at the assumed edge of each field. Manually edit the fields as required to define the size and name of each input field. All fields must be named, and names must be unique.

- Type the name of the field in the **Field Name** box, e.g. `GivenNames`
- Either click on the ruler to add a line between fields, or enter the width in number of characters in the **Field Size** box, e.g. `20`
- Click an existing line between fields to remove it.
- Use the **Next Field** and **Previous Field** buttons to scroll between fields.
- Click the **Load** or **Save** button to load or save field format definitions in [Saved Text File Formats](#).
- Click the **OK** button when all the fields have been defined.

Saved Text File Formats



Use this dialog box to save or delete text format definitions created in the [Input Text File Format screen](#). These files should be given clear descriptive names.

Open	Open a saved text file format.
Delete	Delete a saved text file format.
Undo Delete	Recover a deleted text file format.
Save	Enter a name and save the format.

2 Mapping tab

Manual allocation of DPIDs to those records without.

1. Files | 2. Mapping | 3. Prompt | 4. Display | 5. Go

Field Names in Input & Output Files	Corresponding PAF fields
address2	Address2
address3	Address3
locality	Locality
state	State
postCode	Postcode
flag	
barcode37	Barcode37
dpid	Dpid

Help Next >

Use the Mapping tab to map fields in the input file to fields in the PAF.

For each field in the input file on the left, select the corresponding PAF field from the drop-down list on the right as follows:

- <field name> Update input file field with the PAF field
- <no field> Do not update input file field, leave as is
- (set to blank) Remove field, use to clear irrelevant data
- (set to *) Only update input file field if an address is found in the PAF
- (set to #) Only update input file field if an address is found in the PAF

3 Prompt tab

Fix Address

Manual allocation of DPIDs to those records without.

1. Files | 2. Mapping | 3. Prompt | 4. Display | 5. Go

Fields Used in Prompting for Address Lookup

☒ Automatic (default)

☐ Using these fields:

Locality Line

Address Line

Help Next >

Use the Prompt tab to determine the fields in the input file which are used to compose the address and locality line which are used to search for the address. These are the same as the Address and Locality line in the IQ Rapid Address window.

Automatic

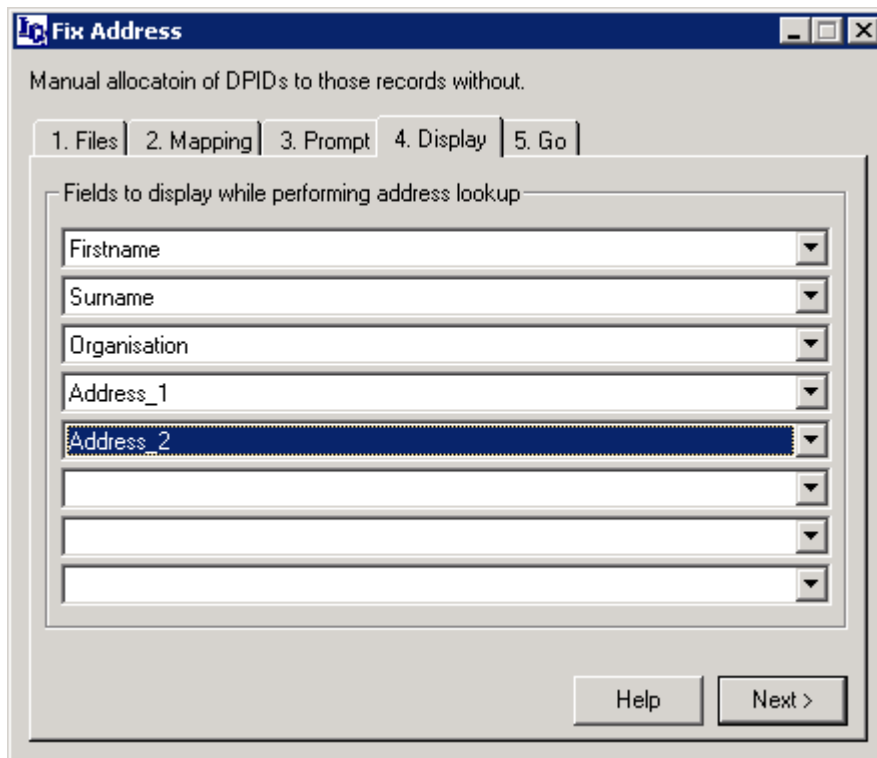
Check this option to compose the Address and Locality line from the fields mapped in the [Prompt tab](#) as follows:

- | | |
|---------------|---|
| Locality Line | Locality and Postcode fields. If these are both empty, then the last non-empty field of Address1, Address2 and Address3 will be used. |
| Address Line | Second last of Address1, Address2 and Address3. If these fields do not exist in the file, the address will be composed from the individual address components, e.g. StreetName + StreetType + StreetSuffix. |

Using these fields

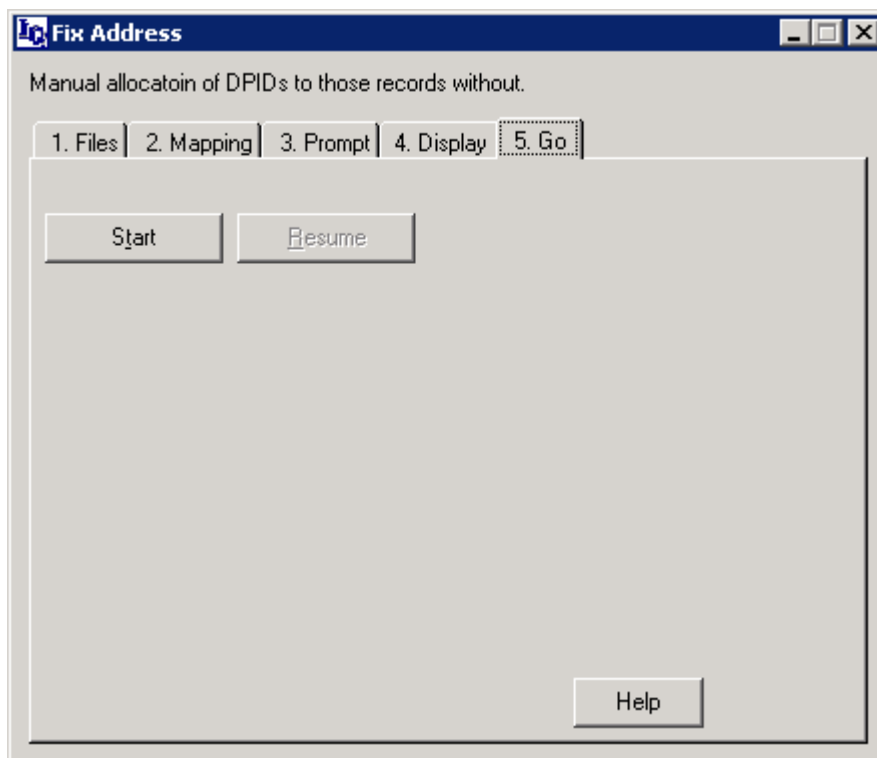
Check this option to select the fields used to compose the Locality and Address lines. The selected fields are concatenated, separated by a space.

4 Display tab



Use the Display tab to select the fields in the original address file which are displayed in a small window while searching for an address, enabling it to be matched to the search results. These are usually the input address lines to IQ Standardiser, or “Address1”, “Address2” and “Address3”.

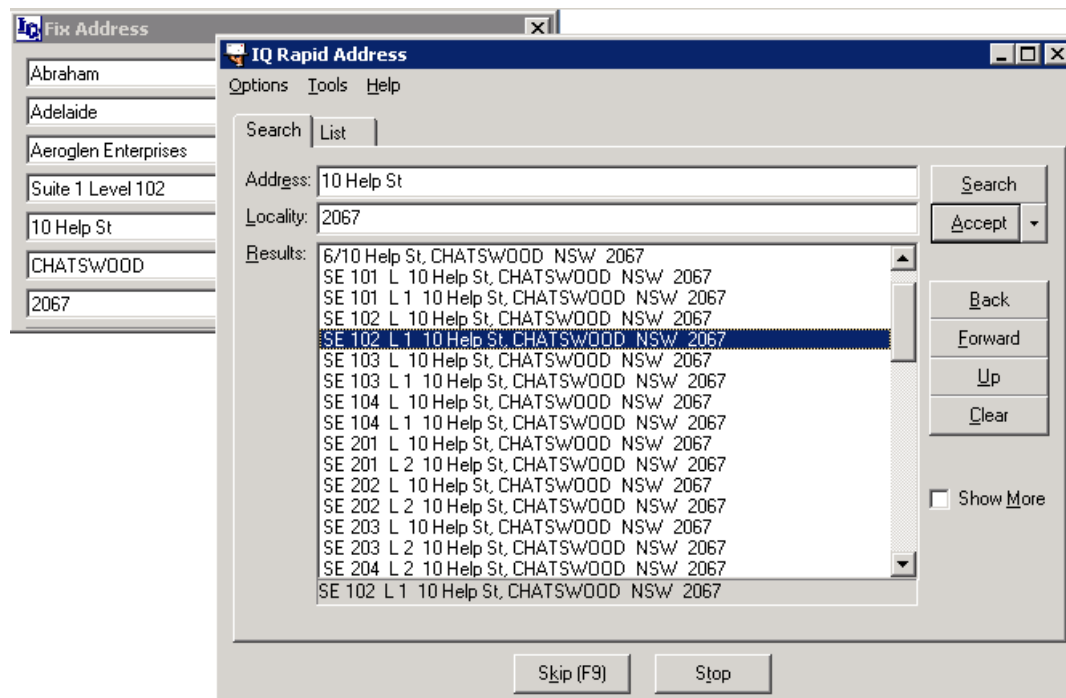
5 Go tab



Use the Go tab to start reading and searching for records from the input file, and resume address searching. Addresses are searched for using IQ Rapid Address.

Start

Click to start searching the PAF, reading records from the input file one at a time. The IQ Rapid Address window displays fields selected in the [Prompt tab](#), and a small window displays fields selected in the [Display tab](#). Use the IQ Rapid Address window to search for the address, and the small window to match the original address with the found address.



Search for the correct address using the [Rapid Address search functions](#), and accept or skip it using the following buttons.

- Accept** Find the correct address then click to accept it as correct. This copies the record to the output file with the address fields, DPID and/or barcode updated, and goes to the next record.
- Skip** If unable to find the correct address, click to skip the current address and copy the record to the output file as is without DPID.
- Stop** Pause manual corrections, which can later be resumed at the same record by clicking the Resume button.

Resume

Click to open the IQ Rapid Address window at the address being manual fixed when the **Stop** button was clicked. You can also exit Fix Address and continue manually fixing records if the processed data file has remained unchanged as follows:

15. Click the Stop button.
16. Exit Fix Address .
17. Start Fix Address when you want to resume manual fixing. A prompt is displayed:

Output file: <name> already exists.
Click <YES> to replace it
Click <NO> to continue from where you left off.

18. Click No to open the Go tab at the last record being checked.

Rapid Address search functions

A summary of the Rapid Address search functions is provided below. Refer to the Rapid Address User Manual for complete details of searching and configuration settings.

Search tab

Find addresses by typing in some details of the street address in the Address line, and some details of the locality, postcode or state in the Locality line, and press Enter to display all possible matching addresses. Select an address or partial address from the list, then press Enter to narrow down the search until the full address is found

Search Tree tab

Find addresses by expanding and collapsing a hierarchical tree view of locality, street, property, level and unit number.

List tab

Find addresses by choosing the locality, then the street, then the property number, and finally a unit or level number if applicable. As you type, choices are displayed in the Results panel, which can then be selected, saving the need to type the full word.

Address line

Type partial or full details of the following address information: Street address, Lot number, Unit or level, Postal box, bag or address, Building name. Type any unit or level numbers in the address before the house number. Street spellings will be corrected.

Locality line

Type partial or full details of the following locality information: Locality name, Postcode, state abbreviation.

The locality name tolerates one letter spelling errors. You can type the first few letters of words in the locality, such as `Syd A` for `Sydney Airport`, according to the rules below:

- Each word beginning must be in the locality for it to be found, but you don't need to type all the word beginnings. For example, `syd` will display `Sydney`, `Sydenham`, `Sydney Markets`, `North Sydney` etc, but `n syd` will find `North Sydney` but not `Sydney`.
- The word beginnings must be in order, e.g. `sy m` will display `Sydney Markets`, but `m sy` will not.
- Abbreviations (N, S, E and W) are recognised and need not be in order, e.g. both `me n` and `n me` will display both `North Melbourne` and `Merah North`.
- State – Enter a state abbreviation to only find only addresses in the state, depending on the selected options.
- No locality or postcode – Enter either nothing or only a state to search all localities or all localities in the state.
- Locality or postcode – Enter a postcode or part of a locality name to search only addresses matching the locality.
- Locality and postcode – Enter both a postcode and some locality name information to search all localities matching either the name or the postcode, with a preference to those localities that match both.
- Exact matching – Enter all locality elements enclosed in square brackets to find an exact match, e.g. `[SYDNEY 2000 NSW]` will only find `SYDNEY NSW 2000`. Without brackets this will find all localities containing the word “SYDNEY” such as “NORTH SYDNEY”, as well as all localities with postcode 2000.

Popup List

A popup hint box listing addresses or localities matching any partial information entered into the address or locality line can be automatically displayed after a defined number of characters have been typed. Press Ctrl-Space to manually show the list at any time.

ID line

Displays ID, DPID and additional fields.

Results / Address panel

Displays found addresses.

Buttons

Use the buttons to search and navigate through addresses.

Search	Search for information entered in the Address line and Locality line.
Accept	Zoom to address details or accept address.
Back	Returns to previous search results. Same as pressing Esc key.
Forward	Goes forward through search results.
Up	Goes up through locality and address tree
Clear	Clears the address line, locality line and all found results, enabling another search.
Stop Search	Displayed during a long search. Click to cancel the search.

Show More checkbox

Check this box to display bordering or synonym localities, alternate street names and border synonyms matched to addresses as permitted under the 2004 AMAS rules. This can vastly increase the number of results returned for a partial address match.

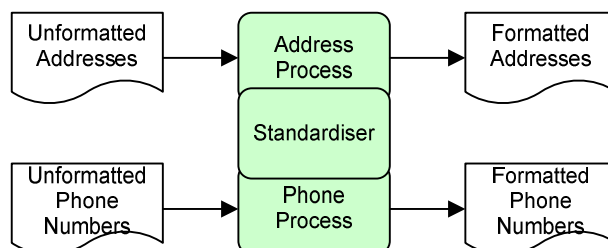
Clear this box to display only addresses matching the search criteria

IQ Standardiser

Overview

Standardising is the process of converting unformatted text data such as addresses, names and telephone numbers into data in a *standard format* performed by IQ Office Standardiser, a suite of applications and library functions supplied with IQ Office DTS and Enterprise editions. This typically cleanses, transforms, enhances and validates the input data, providing greatly improved and high quality output data.

Standardising is performed by one or more standardising *processes*. Each process takes one type of alphanumeric input data such as addresses, names or telephone numbers, *parses* the input data to break it into components, and returns each required component in a specified format.

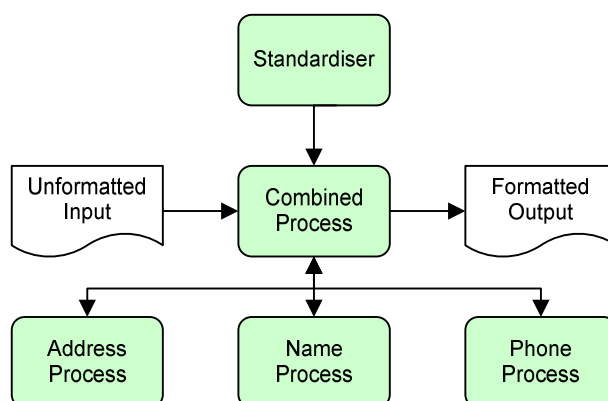


Each standardising process has a *grammar file*, which defines how the data will be standardised and formatted. The Standardiser reads and interprets the grammar file before performing the defined standardising processes.

Grammar files are text files written in a simple programming language. They contain definitions of the number, type and format of input data fields; instructions on how Standardiser is to process input data; references to lookup-lists such as standard abbreviations; and definitions of the number, type and format of output data fields.

Existing grammar files can be easily edited to *customise* them to the specific requirements of an organisation, for example, to process address data received in a variety of formats, and output data in the specific format used by a particular database or automated mailing system. New files can also be written to fulfil the needs of new business processes, for example, if an organisation installs a new account management system.

Grammar files can contain *calls* to other grammar files, for example, the “Name, Address and Phone” grammar file would call three separate grammar files and combine the returned output.



Grammar files can also contain calls to other processes, data sources, libraries and *address reference files*. For example, the *DpidGeocode* grammar file supplied with IQ Office Standardiser uses the IQ Office PAF Libraries to look up an address in the Postal Address File (PAF) containing all Australian postal addresses supplied by Australia Post, and returns the formatted output and DPID.

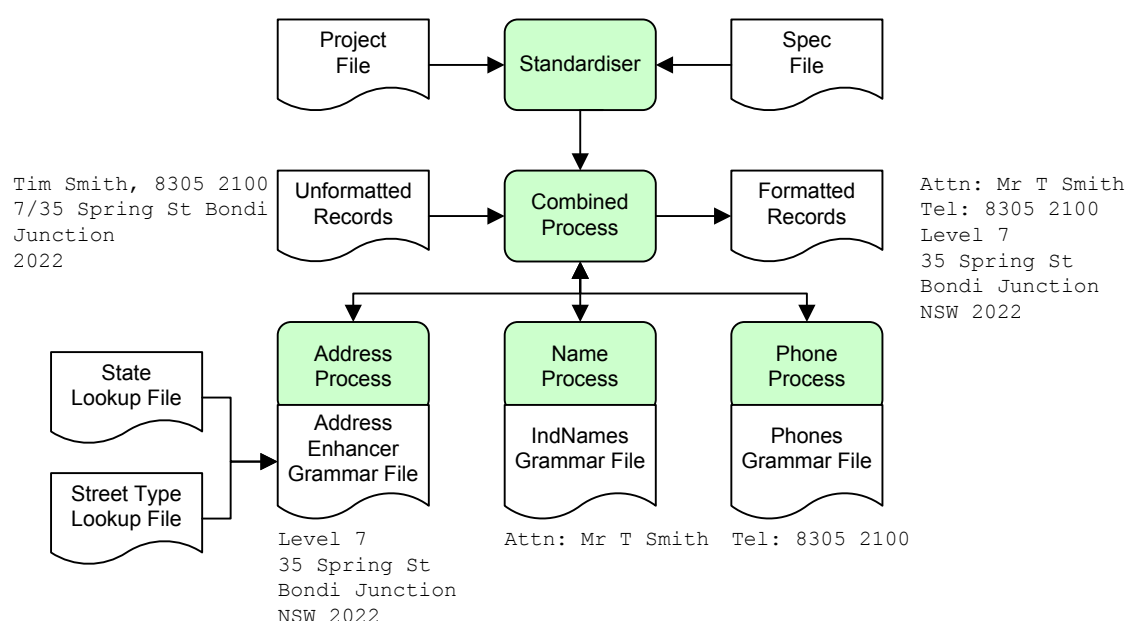
The grammar files supplied with IQ Standardiser depend on the IQ Office Edition and licence, and are described in the [Grammar File Reference Manual](#).

Standardising processes are specified in a *project* which defines the processes performed, and input and output data and fields. A project can be created, edited, saved, and opened again using IQ Standardiser, or run in Batch mode by IQ Stan Console with no user or screen interaction.

A project is defined in two text files:

- The *specification* or *spec file* (extension `.sta`) lists the input fields passed to each process, and the output fields of each process. Spec files can also specify fields which *pass through* the process intact, for example, a record number.
- The *project file* (extension `.ist`) lists the input and output files, input and output fields of each process, and other parameters.

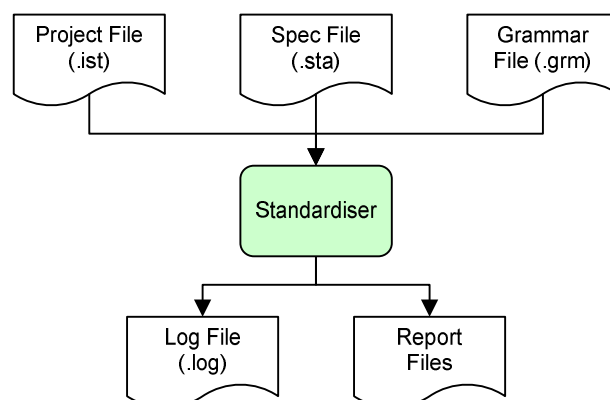
The diagram below shows all components of an example Standardiser project.



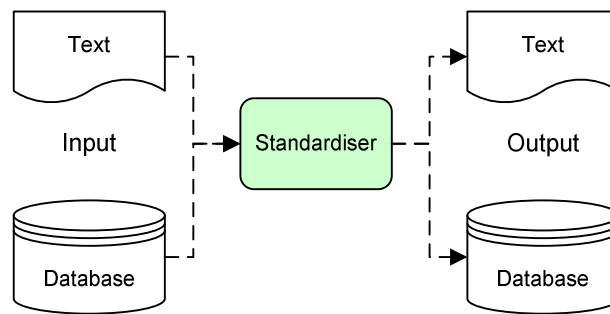
Errors occurring during standardising can be recorded in a *log file* (extension `.log`).

Standardiser can also create *reports* based on the standardised output. Currently the only available reports are for Australian and New Zealand bulk mail lodgements. Output for these reports must be created using a standardising process which assigns an AddressID.

The diagram below summarises Standardiser files.



Standardiser input and output data can be a combination of either *text files* in fixed-width or delimited format, or different types of *database* supporting OLE-DB connections.

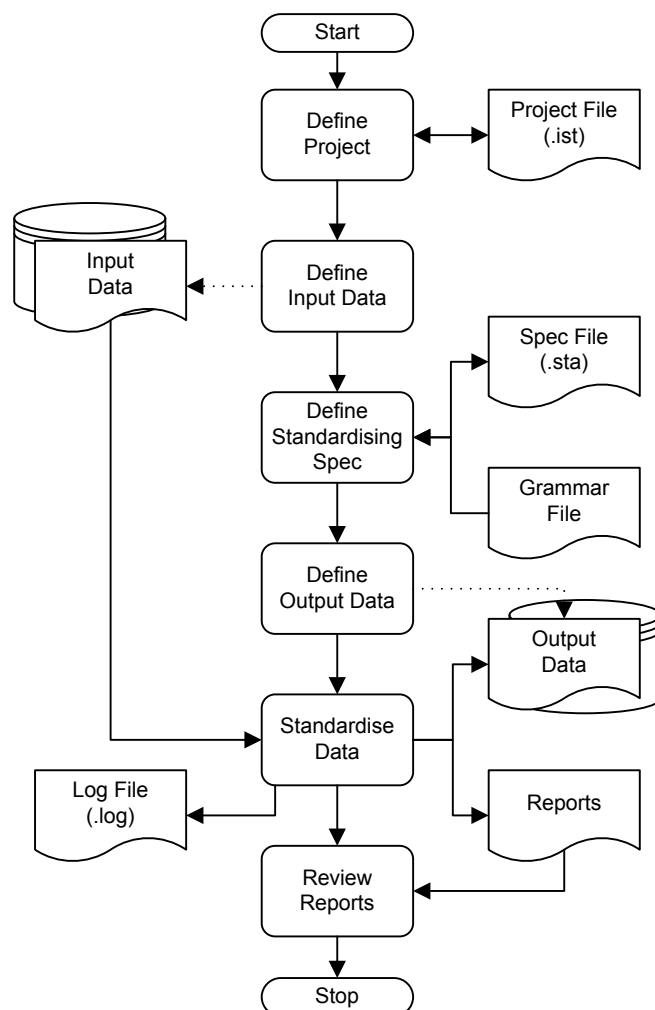


Data can be input from one type of data source into another, for example, from a text file to a database or vice-versa. In the case of databases, data inputs and outputs can be a table, view, query, stored procedure or SQL statement. Data can be input from one database and output to another, or output to a different table in the source database either by creating a new table, or by updating the same table from which the data was read, via a direct table write or stored procedure.

These flexible options, combined with batch mode operation, provide a number of powerful methods to standardise and update data.

Workflow

IQ Standardiser workflow is summarised in the flowchart and steps below.



Workflow

19. Start IQ Standardiser with an existing or new project, see [Starting and Exiting](#). This opens the [Standardiser screen](#).

20. Define a new project, or open an existing project, see [Managing Projects](#).
21. Define the input data sources using as text using the [Input Text File Format Screen](#), or as a data source using the [OLE DB Source Selection Screen](#). Optionally load or save defined [Saved Text File Formats](#).
22. Define the way in which data will be standardised according to the Standardising Spec using the [Standardising Specifications screen](#).
23. Define format of the standardised data output as text using the [Output Text File Format screen](#) or to a data source using the [OLE DB Source Selection Screen](#).
24. Standardise the input data according to the defined spec and save the output data.
25. Review the results of [Reporting](#) generated according to defined [Report Options](#).
26. Exit IQ Standardiser, see [Starting and Exiting](#).

Using IQ Standardiser

Starting and Exiting

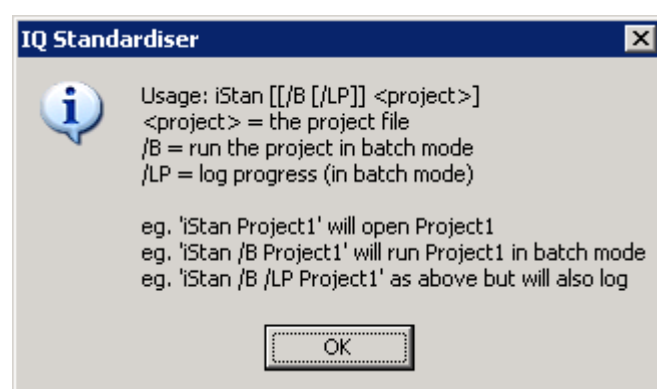
Starting

To start IQ Standardiser, select it from the Intech program menu, double-click the desktop icon, or run it from the command line with the following optional parameters:

```
iStan [[/b [/lp]] Project]
```

```
iStan /?
```

Project	Name of project file. Starts IQ Standardiser with project loaded.
/b	Batch mode. Runs project to input data, standardise it, and save output data with no screens displayed or user interaction. The project must already have been defined, including input data, output data and spec, and all specified files must exist in the defined directories.
/lp	Log Progress. Write a log file containing the time and details of processing, named <project>.log
/?	Display the command line parameter dialog box.



Examples

Start IQ Standardiser with open Phones project:

```
iStan Phones
```

Run Phones project in batch mode:

```
iStan /b Phones
```

Run Phones in batch mode and write a log file:

```
iStan /b /lp Phones
```

Display command line parameter help.

```
iStan /?
```

Example Log File

```
19/10/2010 10:14:01 AM. Begin. Project:Phones ;Input:Phones-Input.txt;  
Output:Phones-Output.txt  
19/10/2010 10:14:12 AM. Done. Project:Phones ;Input:Phones-Input.txt;  
Output:Phones-Output.txt
```

Exiting

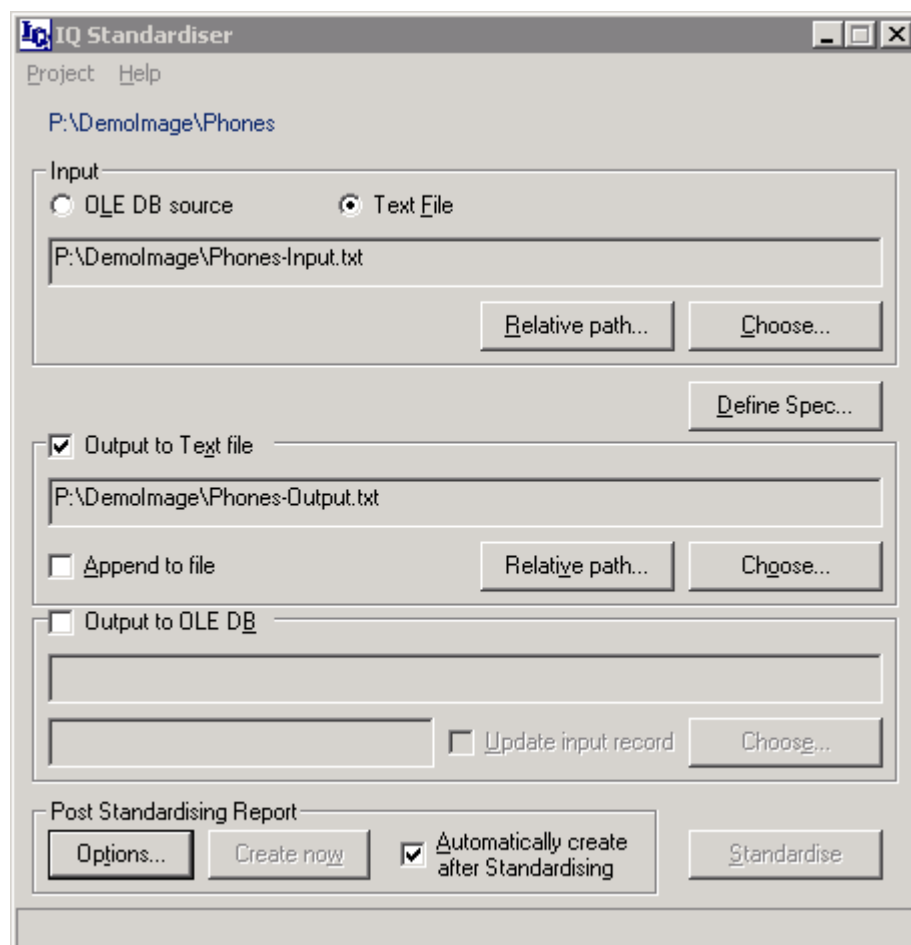
To exit IQ Standardiser, first save any open project, then select **Exit** from the **Project** menu, click the X button in the top right of the window, or press Alt+F4.

Managing Projects

To manage projects, select an option from the IQ Standardiser **Project** menu:

New	Create a new project.
Save	Save the current project.
Save As...	Save the current project as a new project with a different name.
Close	Close the current project.
<project>	Select a recently used project name from the list to open it.

Standardiser screen



Use the IQ Standardiser screen to define input and output data; specify the way in which data will be standardised; standardise the defined data; and manage reports. The name and path of the currently open project is displayed at the top of the screen.

Input

Use this panel to define the input data:

- **OLE DB source** – click the Choose button to open the [OLE DB Source Selection Screen](#) to define the input data as an OLE-DB data source.
- **Text File** – click the **Choose** button to browse to and select an input text file, then open the [Input Text File Format Screen](#) to define the format of the input text file data. You can also click the **Relative path** button to use [Relative Path Names](#).

Define Spec

Click the **Define Spec** button to open the [Standardising Specifications screen](#) and define the way in which data will be standardised.

Output

Use this panel to define the output data:

- **Output to Text file** – check this option then click the **Choose** button to browse to and select an output text file, then open the [Output Text File Format screen](#) to define the format of the output text file data. You can also click the **Relative path** button to use [Relative Path Names](#).

Append to file – check this option to append standardised data to an output text file which has already been defined.

- **Output to OLE DB** – click the Choose button to open the [OLE DB Source Selection Screen](#) to define the output data as an OLE-DB data source. This can be the same as, or different from, the input data source.

Update input record – check this option to update existing records in the OLE-DB input data source. In this case the input, processed, and output fields must match. For example, an input database table containing non-standardised telephone numbers could be standardised by a Phones process and the standardised telephone numbers written back to the table.

Post Standardising Report

Use this panel to manage [Reporting](#).

- **Options** – click to define the [Report Options](#).
- **Automatically create after Standardising** – check to create reports immediately after standardising
- **Create now** – click to create reports now. This can be used to recreate reports with different options.

Standardise

Click the **Standardise** button to load the input data, standardise it, and save the output data. Any errors occurring during standardising are written to a log file with the same name as the project file and .log extension.

OLE DB Source Selection Screen

Use this screen to define the format of the database data input to the Standardiser.

Saved configuration

Type a name for the connection to store all configuration parameters for later use. The configuration is saved when you click the **Open** button. To use a saved configuration, select it from the list. To delete a saved configuration, click the Delete button.

Connection

Enter the database connection details:

- **Provider** – select the OLE DB data provider from the dropdown list of providers installed on your machine.
- **Prov String** – enter the provider or connection string passed to the OLE DB provider.

For the "Microsoft.Jet" provider (Access database), this is the name of the database file including extension, e.g. `Organisations.mdb`

For the "SQLOLEDB" provider (SqlServer), this is the server and database name, e.g.
`server=Bighost;database=AddressDB;`
- **User Id** – enter the username to connect to the data source.
- **Password** – enter the password to connect to the data source.

Click the **Connect** button to connect to the data source.

RecordSet

Enter the name of the table, view, query, stored procedure or SQL statement to fetch or store data. Some data sources enable tables to be listed by clicking the **List tables** button.

Advanced

Click the Advanced button to open a panel to define advanced connection options. Select options from each list, or leave as (default). The available options may depend on the type of data source.

- **Cursor type** – e.g. default, Forward only, Keyset, Dynamic, Static
- **Lock type** – e.g. Read only, Pessimistic, Optimistic, Batch only
- **Cursor location** – e.g. Client, Server
- **Evaluate source as** – e.g. Text, Table, Table direct, Stored proc, File type, unknown
- **Cache size** – size of cache
- **Command timeout** – time after which commands time out

The following options are only available for an output data source:

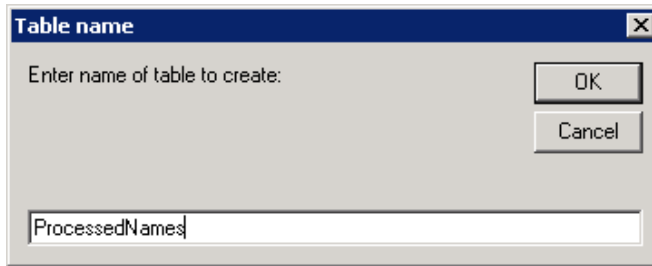
- **Create New Table...** – click to [Create New Output Table](#) in the output database.
- **Stored Proc** – check this option to use the [Stored Procedure Parameters Screen](#) to create a stored procedure to handle the output fields.

Click the **Open** button to open the data source and close this screen.

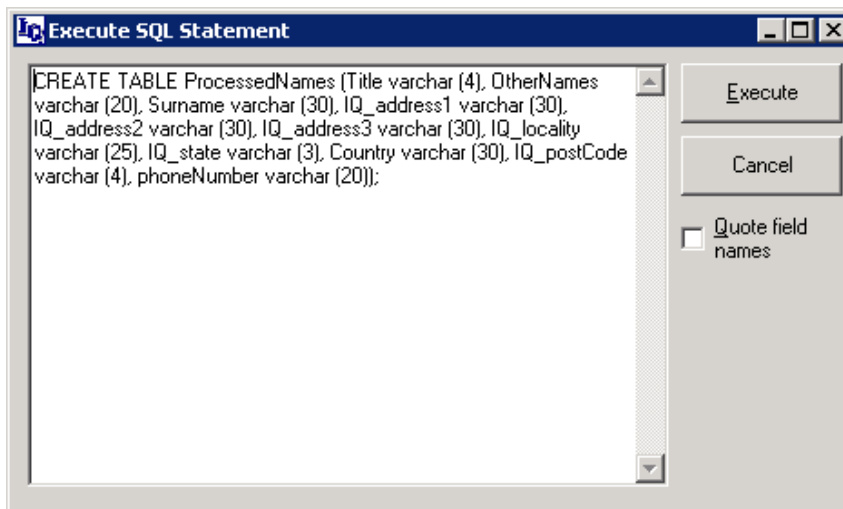
Create New Output Table

To create a new table to hold output data:

1. Connect to a data source from the **Output to OLE DB** panel to open the [OLE DB Source Selection Screen](#).
2. Click the **Create New Table** button.
3. Enter the new table name in the **Table name** dialog box and click **OK**.

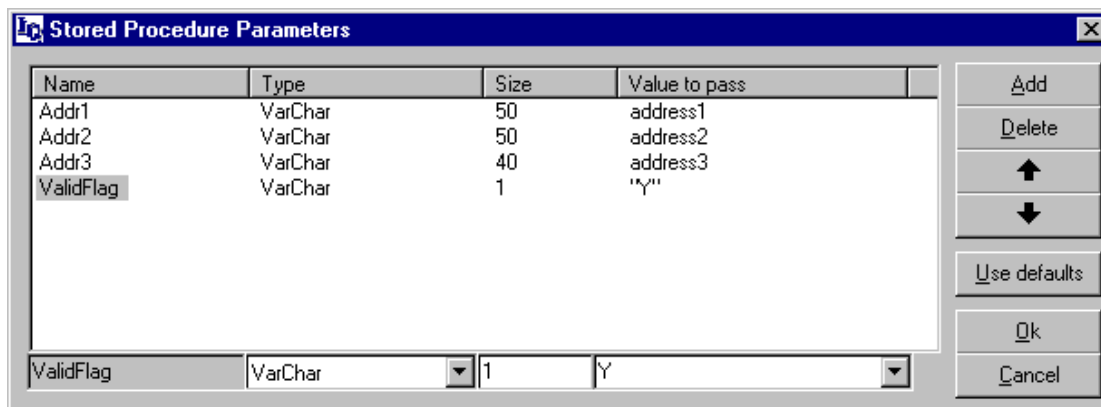


4. Check to view or edit the SQL in the **Execute SQL Statement** dialog box. Check **Quote field names** to view the names of the fields in quotes.



5. Click the **Execute** button to create the table in the database.

Stored Procedure Parameters Screen



Use this screen to define the order, type and size of the parameters expected by the stored procedure handling output data.

To create a new table to hold output data:

1. Connect to a data source from the **Output to OLE DB** panel to open the [OLE DB Source Selection Screen](#).
2. Check the **Stored Proc** option.
3. Type the name of the stored procedure in the Stored Proc panel, e.g. *ProcessedNamesHandler*
4. Click the **Stored Proc Parameters** button to open the Stored Procedure Parameters screen.

Use the buttons on the right side of the screen to edit the stored procedure parameters:

- Up and down arrows – click to change the order of the parameters
- **Add** – add a new parameter
- **Delete** – delete an existing parameter
- **Use defaults** – reset to the default parameters

Edit the fields at the bottom of the screen to define the values placed into the parameters. These can be either an output field, or a constant string shown by double quotes.

Input Text File Format Screen – Delimited

id	title	lastName	firstName	add1
458547	Miss	MITCHELL	NICOLE	P.O.BOX 67
1362499	Ms	BADGERY	ELMA	GLENGARRY
160174	Mrs	JOWETT	LORRAINE	WILLUNGA
1301869	Ms	COLEMAN	NORMA	DOON DOON
397769	Mrs	MARTYN	MARGARET	1111 EYRE ST
633459	Mrs	LEMMICH	GLENDA	26 HULLS RD
806517	Mrs	DAVIDSON	JANETTE	SILVERTON
1851698	Mrs	FOWLER	KYLIE	CAMPANIA

Use this screen to define the format of the text file data input to the Standardiser. A preview of the input file is displayed at the bottom of the screen. It may also be useful to first view the contents of the file using a text file viewer or editor.

For **Fixed Width** format, check the option button and use the [Input Text File Format Screen – Fixed-width](#) to define the file format.

For **Delimited** format, check the option button then select other options:

- **First Row Contains Field Names** – check this option if the first row of the file contains the names of the fields. If not, the name of each field must be defined.
- **Field Delimiter** – select the single character that delimits the fields in each row:
, (comma) ; (semicolon) {tab} {space}
- **Text Qualifier** – (optional) select the character that surrounds text in each field:
{none} ' (single quote) " (double quote)
- **Max Field Size** – (optional) enter the maximum width of a field in numbers of characters.
- Use the **Next Field** and **Previous Field** buttons to scroll between fields.
- Click the **Load** or **Save** button to load or save field format definitions in [Saved Text File Formats](#).

- Click the **OK** button when all the fields have been defined.

Input Text File Format Screen – Fixed-width

Text File Format - P:\DemoImage\Phones-Input-Fixed.txt

☒ Fixed width
☐ Delimited

Field Delimiter:

Text Qualifier:

Field Name:

Field Size:

Next Field Previous Field Load... Save...

Title	Given Names	Field3	Field4
MR	Tim John	Doak	Federal
MR	TIBOR	Doak	Freecall
MR	ROBERT	Horgan	Call 180
MR	VICTOR	Wright	PO BOX 7
MR	IMRE	Buich	Marine B
MR	Matthew James	Holm	17-21 Be
MR	LESLIE	Robinson	GPO BOX
			PO BOX 3

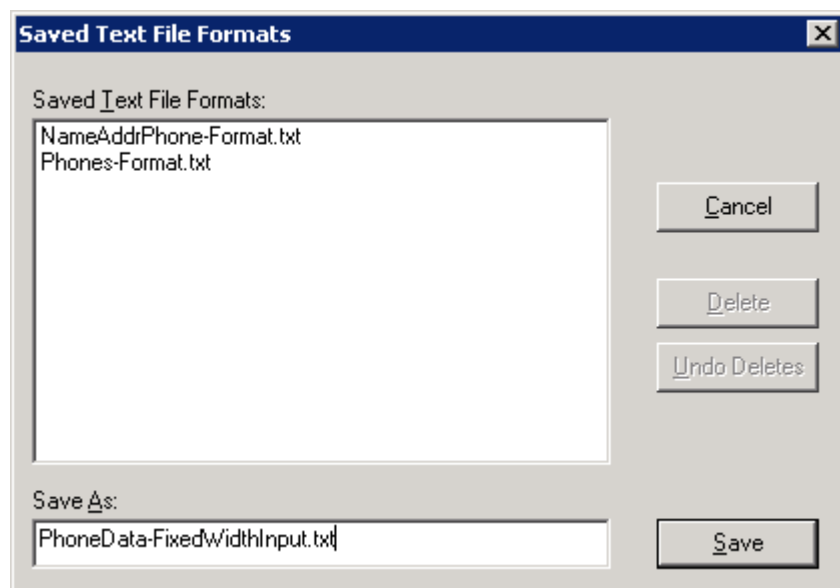
Click on ruler to Add/Remove lines. Click on column header to select it. Then change its name

Use this screen to define the fields of fixed-width input text files.

When a fixed-width file is opened, lines are automatically added at the assumed edge of each field. Manually edit these lines to define the size and name of each input field. All fields must be named, and names must be unique.

- Type the name of the field in the **Field Name** box, e.g. `GivenNames`
- Either click on the ruler to add a line between fields, or enter the width in number of characters in the **Field Size** box, e.g. `20`
- Click an existing line between fields to remove it.
- Use the **Next Field** and **Previous Field** buttons to scroll between fields.
- Click the **Load** or **Save** button to load or save field format definitions in [Saved Text File Formats](#).
- Click the **OK** button when all the fields have been defined.

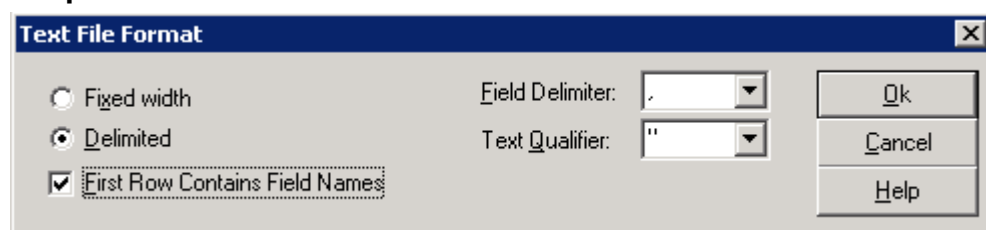
Saved Text File Formats



Use this dialog box to save or delete text format definitions created in the [Input Text File Format Screen](#) or [Input Text File Format Screen – Fixed-width](#). These files should be given clear descriptive names, e.g. `PhoneData-FixedWidthInput.txt`

- | | |
|--------------------|-------------------------------------|
| Open | Open a saved text file format. |
| Delete | Delete a saved text file format. |
| Undo Delete | Recover a deleted text file format. |
| Save | Enter a name and save the format. |

Output Text File Format screen



Use this screen to define the format of standardised data output as text.

For **Fixed Width** format, check the option button. No other options can be selected.

For **Delimited** format, check the option button then select other options:

- **Field Delimiter** – select the single character that will delimit fields in each row:
, (comma) ; (semicolon) {tab} {space}
- **Text Qualifier** – (optional) select the character that will surround text in each field:
{none} ' (single quote) " (double quote)
- **First Row Contains Field Names** – (optional) check this option to include the names of the [Output fields](#) at the top of the output file.

Examples

Fixed Width format:

Mr	Matthew James	Holm
Ms	Sheila	Cowper-Smith

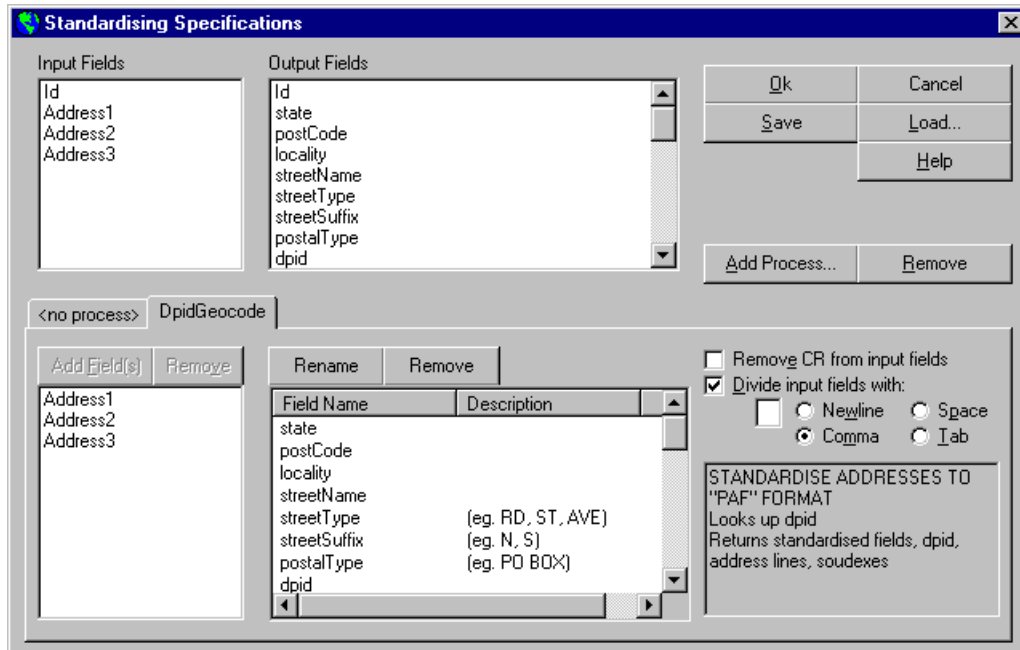
Delimited format, Field Delimiter comma, Text Qualifier {none}

Title,Names,LastName
Mr,Matthew James,Holm
Ms,Sheila,Cowper-Smith

Delimited format, Field Delimiter semi-colon, Text Qualifier double quotes:

"Title";"Names";"LastName"
"Mr";"Matthew James";"Holm"
"Ms";"Sheila";"Cowper-Smith"

Standardising Specifications screen



The 'Standardising Specifications' dialog box is used to configure data standardization. It features two main sections: 'Input Fields' and 'Output Fields'. The 'Input Fields' section contains a list of fields: Id, Address1, Address2, and Address3. The 'Output Fields' section contains a list of fields: Id, state, postCode, locality, streetName, streetType, streetSuffix, postalType, and dpid. Below these lists are buttons for 'Add Process...', 'Remove', 'Add Field(s)', and 'Remove'. A 'Process' dropdown menu is set to 'DpidGeocode'. A table with 'Field Name' and 'Description' columns lists the output fields with their descriptions: state, postCode, locality, streetName, streetType (eg. RD, ST, AVE), streetSuffix (eg. N, S), postalType (eg. PO BOX), and dpid. To the right of the table are checkboxes for 'Remove CR from input fields' and 'Divide input fields with:' (Newline, Space, Comma, Tab). A text box at the bottom right explains the 'STANDARDISE ADDRESSES TO "PAF" FORMAT' process, which looks up dpid and returns standardised fields, dpid, address lines, and soudexes. Buttons for 'Ok', 'Cancel', 'Save', 'Load...', and 'Help' are located at the top right.

Use this screen to define the way in which data will be standardised.

Input Fields

This box displays all fields in the input data. The box directly below displays fields from the input data which will feed into the specified process.

Check the **Show all** box to display all fields in the input data if some fields have been removed from the lower list.

To add fields from the upper list to the lower list, select the fields using click, Shift+Click or Ctrl+Click, then either click the **Add** button or drag and drop.

You can also add the following special fields from the arrow on the right of the **Add** button:

AutoNum A value which automatically increases by 1 from a specified starting number, e.g. 1, 2, 3 ..

Date A date in a specified format, e.g. mm/dd/yyyy, yy/mmm/dd

Constant A numeric or text constant, e.g. Attn:

To remove fields from the lower list, select the fields using click, Shift+Click or Ctrl+Click, then either click the **Remove** button or press the Del key.

Output fields

:This box initially displays all output fields in the process (grammar file). The box directly below displays the fields from this list which you can re-order, rename or delete. You cannot edit the upper list, but it changes as the lower list is changed.

To edit the lower list:

- To change the position of a field in the list, click on it then either drag it up or down the list, or click **Up** or **Down** button.
- To rename a field, click on it then click **Rename** and type the new field name.
- To delete a field, click on it then click **Remove** or press the Del key.

Process tabs

There is one process tab for each currently specified process.

<No Process> displays a list of fields copied directly from the input to the output without being standardised. You can also edit these fields, or add special [Input Fields](#).

Click on a process tab to display the input and output of the process, and a description of the process in the box on the bottom right of the Standardising Specifications screen.

Use the following buttons to edit the processes in the spec:

Add Process... click to open the [Add Process screen](#) and add a new process

Remove click the process tab then click this button to delete a process

Delimiter options

s:Use these options to manage delimiters in the input fields.

- **Remove CR from input fields** – check to remove line feed and carriage return characters from text input fields, before passing them to processes which cannot handle these characters as input. The default (unchecked) is usually OK.
- **Divide input fields with...** – If more than one field is input into a process, fields will be concatenated. Check this option then type or select a character to place between fields before they are passed to the process. The default is usually OK.

Buttons

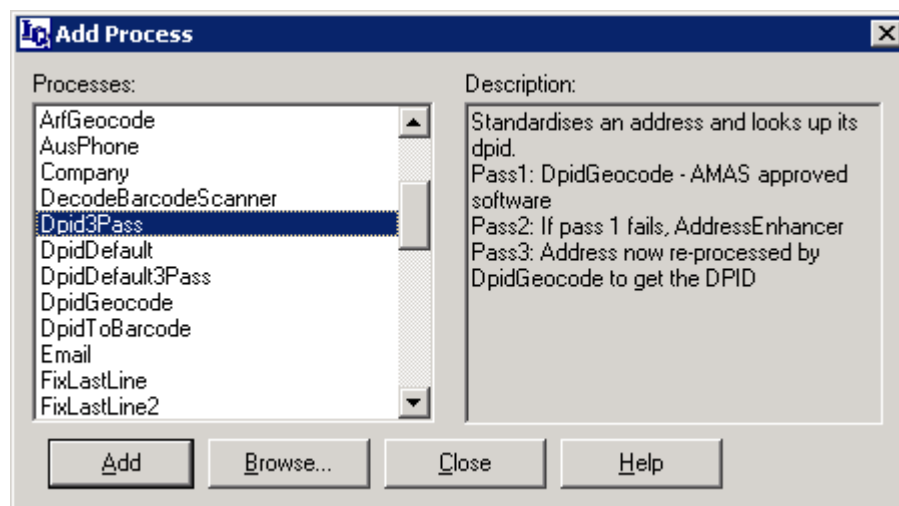
OK Save the spec and close the screen.

Close Close the screen without saving the spec.

Save Save the current spec leaving the screen open.

Load... Browse to and open an existing spec (.sta) file

Add Process screen



Use this screen to add processes to a spec in the [Standardising Specifications screen](#). Click on a process in the list to display a description of the process. The available processes depend on the IQ Office Edition and licence.

Click the buttons to add processes to the spec:

- | | |
|-----------------|--|
| Add | Click on a process in the list then click this button to add it to the spec. |
| Browse.. | Click to browse to a process which is not in the list. When selected it is automatically added to the list without needing to click the Add button. |
| Close | Click to return to the spec when finished adding processes. |

Reporting

IQ Standardiser can create reports based on the standardised output. Currently the only available reports are for Australian bulk mail lodgement, which are required for Australia Post PreSort mailing lodgements. Output for these reports must be created using a standardising process which assigns a DPID and/or barcode, and provides a field with the BSP number, such as the Dpid3Pass process. Various [Report Options](#) can also be defined.

The following three reports are created:

- Manifest report – the volume, weight and number of letters mailed to each state
- Address Matching Summary report – details of AMAS-compliant software used to append the DPIDs (IQ Standardiser), list owner and manager, DPID processing results and compliance statement to be signed
- Lodgement Information report – the number of letters lodged to each state and sort level

Report generation creates a subdirectory of the output directory with the same name as the project, and each report file in both text and HTML formats. The Manifest report is also created in comma separated value (CSV) format to enable automatic processing. For example, an Address project would create the following files:

```
\Mailout  
  
Address.txt  
NSW_Address.txt  
WA_Address.txt  
...  
  
\Mailout\Address\
```

AddressMatchingSummaryReport.htm
AddressMatchingSummaryReport.txt
LodgementInfo.htm
LodgementInfo.txt
ManifestReport.csv
ManifestReport.htm
ManifestReport.txt

Report Options

IQ Stan Report Options

☒ **Manifest Report**

State of Lodgement: NSW *
Number of letters per Tray: 400 *
Weight per letter in grams: 10
Customer Name: BigMail Company
Job Number: 102

☒ **Address Matching Summary Report**

List Processor's Name: Mr List Processor
Name of Address List: Monthly Mailout
List Owner/Manager: Ms List Owner

☒ **Resort Output Data**

☒ **Also produce separate files per:**

☒ State ☐ Sort Level (direct tray, residue, unbarcoded) ☐ Both

Save as Default Restore Default Advanced <<< Cancel Ok

State Field: state
Bsp Field: bsp
Dpid Field: dpid
Amended/Flag Field: amendedFlag

Default Fields

Manifest Report options

Check the **Manifest Report** box to create this report, and enter the following options. The first two options are mandatory.

- **State of Lodgement** – the Manifest Report requires the state of lodgement, as same-state mail is given greater discounts.
- **Number of letters per Tray** – enter the maximum number of letters that can fit into one tray, which will determine the number of trays required, and is used to calculate the number of trays per group, e.g. per BSP.
- **Weight per letter in grams** – (Optional) weight of each letter. This is used to provide the weight per group in the Manifest Report and to determine the minimum number of letters to make a direct tray: 300 or 4kg. If omitted, no weight information will appear on the report.

- **Customer Name** – (Optional) name of the mailout customer to appear on the top of the report.
- **Job Number** – (Optional) number of the job to appear on the top of the report.

Address Matching Summary Report options

Check the **Address Matching Summary Report** box to create this report, which indicates the number of addresses to which a DPID:

- was appended without amending the address
- was appended by amending the address
- could not be appended

It also includes information about the software used to append the DPIDs (IQ Standardiser), database processed, and compliance statement to be signed.

Enter the following options which are included in the Address Matching Summary Report:

- **List Processor's Name** – name of the person responsible for processing the list
- **Name of Address List** – name of the mailout list itself
- **List Owner/Manager** – name of person responsible for managing the list

File Processing options

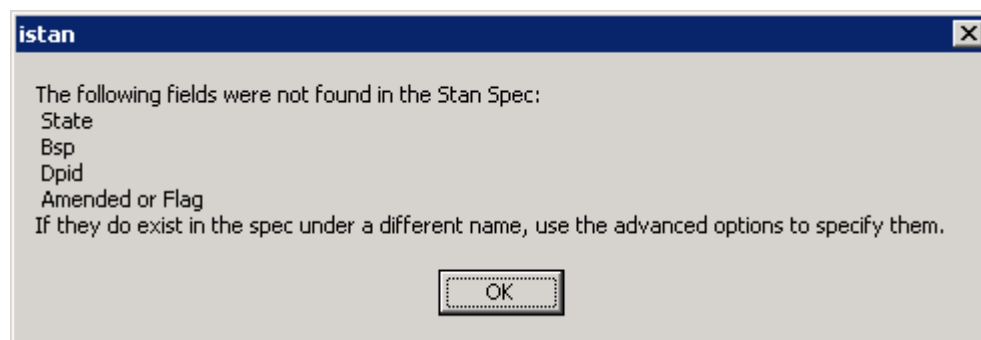
Check **Resort Output Data** to sort the standardised file by SortLevel then by BSP number when the report is created ready for printing.

Check **Also produce separate files per:** to create separate files per state or sort level in the same directory as the output file:

- **State** – create separate files per state with the state prefix, e.g. NSW_, WA_ etc
- **Sort Level** – create separate files per sort level with the sort level prefix, i.e. BDT_ (Barcode Direct Tray), BR_ (Barcode Residue) or UR_ (Unbarcoded residue)
- **Both** – create separate files per state and sort level, creating up to 24 separate files.

Advanced options

Click the **Advanced** button to change the mapping of the State, BSP, DPID and Amended Flag fields, which are required for the report. If these fields do not already exist in a defined Spec, a warning message is displayed when the options button is clicked.



Click the **Default Fields** button to return to the default mapping.

Saving options

Click one of the following buttons to save or restore all defined report options:

Save as Default Save the current options as the default for all subsequent reports.

Restore Default Restore the saved default options.

Advanced	Define advanced options.
Cancel	Discard the current options.
OK	Save the current options for this report.

Relative Path Names

Relative path name aliases enclosed by dollar signs may be used at the beginning of text file names and OLE-DB Provider Strings, for example, \$WorkPath\$

The alias is stored with the project, and substituted with the value found in the Windows registry under the following key when the project is run:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Intech\Paths
```

For example, if the above key has the value

```
"WorkPath"="C:\Program Files\Intech\Working"
```

then the file name

```
$WorkPath$\MyDir\MyFile.txt
```

is translated to

```
C:\Program Files\Intech\Working\MyDir\MyFile.txt
```

To select relative path name aliases, click the **Relative path...** button instead of the **Choose...** button. A list of relative path names defined in the registry is displayed. Click the required path, then browse to the file.

To see the translated file name, place the mouse pointer over the path name.

IQ Standardiser SDK

Overview

Standardising is the process of converting unformatted text data such as addresses, names and telephone numbers into data in a *standard format* performed by IQ Office Standardiser, a suite of applications and library functions supplied with IQ Office DTS and Enterprise editions. This typically cleanses, transforms, enhances and validates the input data, providing greatly improved and high quality output data.

The IQ Standardiser SDK is an extensive set of Information Quality Application Programming Interfaces and test code which enables these functions to be integrated into new and existing business systems.

About this manual

This manual provides a complete reference to:

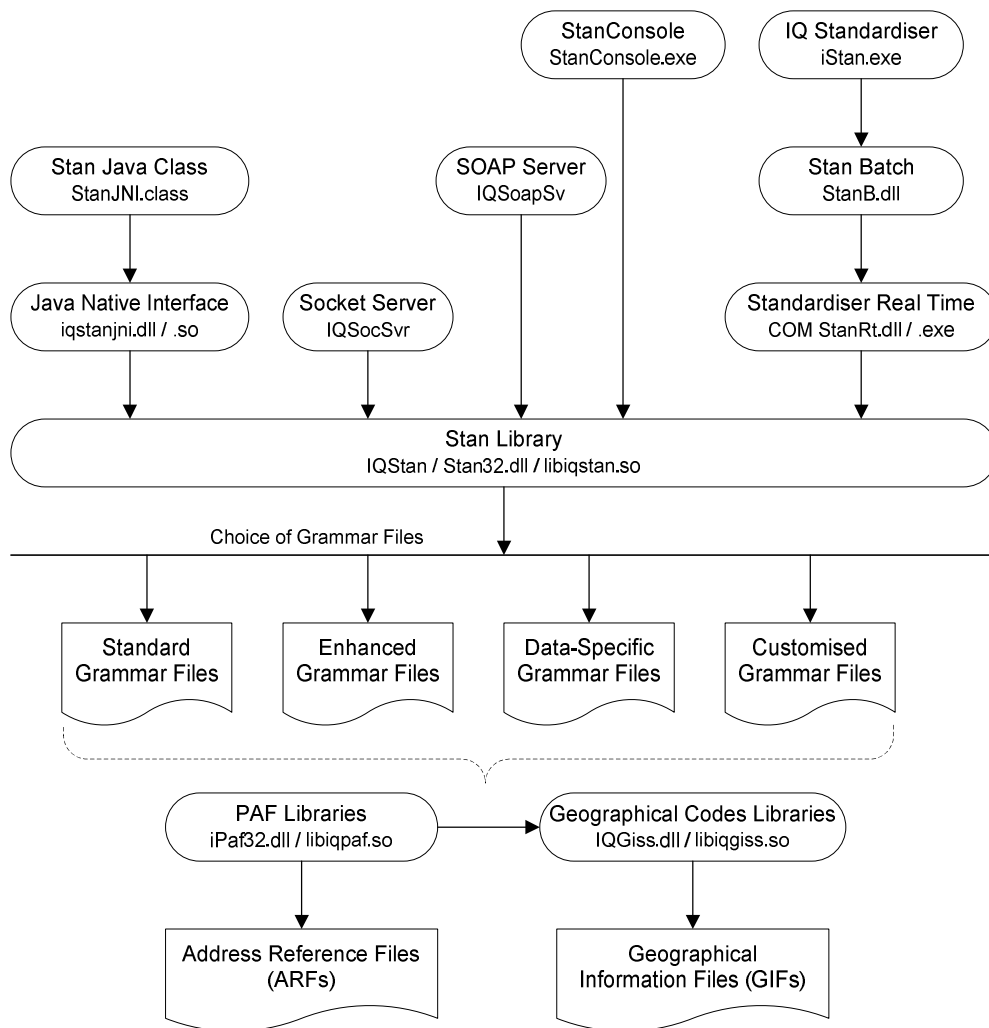
- The Stan Libraries which provide the standardising engine, and are available as a Windows 32 or 64 bit API DLL or C function library which can be integrated into other systems.
- The Standardiser Real Time (StanRt) engine exposed as a COM / DCOM object which can run as a Windows service.
- Java, Socket and SOAP interfaces to the Stan Library functions.
- The Standardiser Batch COM object library which uses StanRt to standardise a batch of records.
- The StanConsole application which can standardise data stored in text files and be used to debug grammar files.

Refer to the following manuals for details of other IQ Office components:

- The [Standardising Overview](#) summarises the standardising process and the IQ Office Standardiser components which standardise data.
- The [IQ Standardiser User Manual](#) describes how to use the standalone IQ Office Standardiser application.
- The [Grammar File Reference Manual](#) describes the various grammar files supplied with different IQ Office editions and data licences.
- The [Grammar SDK Reference Manual](#) contains detailed instructions on how to create, edit and run grammar files.
- The [IQ Office Configuration Manual](#) describes system configuration settings.
- The [IQ Office Component Setup Manual](#) describes how to edit configuration settings, and how to configure machines as a Rapid Address or Standardiser socket, SOAP or DCOM client or server, and install, remove or manage services.

Components

The diagram summarises IQ Standardiser components described in this manual and their dependencies.



[Stan Libraries](#)

The standardising engine, which standardises a single piece of data such as a field or a string, at a time, in real-time. Available as Windows 32 and 64 bit API DLLs, and as C function library for several UNIX and other operation systems.

[StanRt](#)

(Standardiser Real Time) The standardising engine exposed as a COM / DCOM object that can run as a Windows service.

[StanJNI Java Class](#)

A Java interface to the standardising engine.

[Socket Server](#)

A socket server providing access to the Stan Library functions.

[SOAP Server](#)

A SOAP / XML server providing access to the Stan Library.

[StanB](#)

(Standardiser Batch) A COM object library that uses StanRt to standardise a whole table of records.

[StanConsole](#)

A console application that can standardise data and multi-thread text file based projects. It can also be used in interactive mode to test the standardising engine.

Grammar Files

These files define the rules for standardising data that are used by the standardising engine. Each grammar file (or process) standardises a different type of data, such as addresses, names and telephone numbers. Different grammar files are supplied with the various editions and data releases of IQ Office as described in the [Grammar File Reference Manual](#). Grammar files can

also be customised according to specific customer requirements as described in the [Grammar SDK Reference Manual](#) .

Windows Installation

This section describes steps required for a custom installation, or repair of a standard Standardiser SDK installation performed by the Setup program.

Stan Libraries Installation

The Intech Home and PAF Files directories are described in the [IQ Office Configuration Manual](#) and can be configured using IQ Office Component Setup [Client Server Setup](#).

Install the Stan Libraries as follows:

1. Copy the `Stan32.dll` file to the Windows System directory:
`C:\Windows\System32`
`C:\WINDOWS\SysWOW64`
2. Copy the `iStan.lic` licence file to the Intech Home directory. This directory is set in the Windows Registry.
3. Copy grammar files to the Intech [Grammar Directory](#), and other required data, grammar and library files to the relevant directories. For example, the DpidGeocode Grammar File requires the PAF Library file `iPaf32.dll` in the Windows System directory, and the compressed PAF files placed in the Intech PAF Files directory.

StanRt Installation

StanRt requires [Stan Libraries Installation](#). StanRt can be registered as described below, or by using IQ Office Component Setup [Client Server Setup](#).

StanRt as a Windows Service

1. Copy `StanRt.exe` to the Intech Home Directory or Windows System directory.
2. Register StanRt as a DCOM service by typing the following from the command prompt or from the Run menu:

```
StanRt /Service
```

3. To unregister StanRt, type the following:

```
StanRt /UnregServer
```

StanRt as a COM server (Executable version)

1. Copy `StanRt.exe` to the Intech Home Directory or Windows System directory.
2. Register StanRt as a COM server by typing the following from the command prompt or from the Run menu:

```
StanRt /RegServer
```

3. To unregister StanRt, type the following:

```
StanRt /UnregServer
```

StanRt as a COM server (Dynamic Library version)

1. Copy `StanRt.dll` to the Intech Home Directory or Windows System directory.
2. Register StanRt as a COM server by typing the following from the command prompt or from the Run menu (Where appropriate, substitute the full path of `StanRt.dll`):

```
Regsvr32 StanRt.dll
```

3. To unregister StanRt, type the following:

```
Regsvr32 /u StanRt.dll
```

StanConsole Installation

StanConsole requires [Stan Libraries Installation](#). To install StanConsole, copy the `StanConsole.exe` executable file to any directory.

StanB Installation

StanB requires [StanRt Installation](#). StanRt can be registered as described below, or by using IQ Office Component Setup [Client Server Setup](#).

Install StanB as follows:

1. Copy `StanB.dll` to the Intech Home Directory or Windows System directory.
2. Register StanRt as a COM server by typing the following from the command prompt or from the Run menu (Where appropriate, substitute the full path of `StanB.dll`):

```
Regsvr32 StanB.dll
```

3. To unregister StanRt, type the following:

```
Regsvr32 /u StanB.dll
```

StanRt

StanRt is a (D)COM object library, which allows the Standardiser to be accessed from other applications.

There are two COM objects that are used to access the Standardiser:

- [StanRt object](#) – used to standardise records.
- [Manager object](#) – used to manage the grammar files that are accessed by all StanRt objects.

StanRt object

This object is used to standardise records, and get information about the standardising process (the grammar file).

Each instance of this object is associated with one grammar file.

Use the [Open method](#) to open the grammar file, then the [Stan method](#) to standardise a record, then the [Close method](#) to release resources used when finished.

The properties give information about the grammar file and the format of the output.

Open method

Open the grammar file.

Syntax

```
Object.Open GrammarName, DescriptionOnly
```

Syntax description

Object a StanRt object.

GrammarName a String with the name of the grammar file. This can be an absolute path, or relative to the default directory (see [DefaultDirectory property](#)).

DescriptionOnly an optional integral expression. If non-zero, the grammar file will not be fully loaded, only information about it will be read. If zero or missing, the grammar file will be fully loaded.

Notes

After the **Open** method had been called, the properties of the StanRt object become available.

After the **Open** method had been called without *DescriptionOnly*, the method **Stan** becomes available.

The grammar file remains in memory, until the [Close method](#) is called or the object is unloaded, See [Loading grammar files with StanRt](#).

If the **Open** method has already been called, calling it again will first unload the previous grammar file, then load it with the new one.

An error will be triggered if there is an error in loading the grammar file.

Stan method

Returns a standardised record.

Syntax

Object.Stan (*Input*)

Syntax description

Object a StanRt object.

Input a String with the input to the standardiser.

Notes

The grammar file must be open (by calling the [Open method](#) without *DescriptionOnly*), else an error is generated.

The output will be in a format as specified in the [OutputFormatType property](#).

Close method

Closes the open grammar file and releases resources.

See [Loading grammar files with StanRt](#).

Syntax

Object.Close

Syntax description

Object a StanRt object.

Description property

Returns a String with the description of the standardising process (the grammar file).

Notes

The grammar file must be open (by calling the [Open method](#)), else an error is generated when trying to read this property.

GrammarName property

Returns a String with the name of the grammar file, as used when calling the [Open method](#).

Notes

If an error is generated when trying to access this property, it means that the grammar file has not yet been opened with the [Open method](#).

OutputFormatType property

Sets or returns the format of the output fields of the open grammar file. This can be one of the following

- 0 **output_Fixed** for fixed-length output fields
- 1 **output_Delimited** for delimited output fields
- 2 **output_XML** for an output in XML format

Notes

The grammar file defines a default output format. This output format can be read using this property. If desired, setting this property can change this output format.

If the output is fixed-width, the lengths of the output fields are specified by the [OutputFieldSizes property](#).

If the output is delimited, the character that delimits the fields is specified by the [OutputDelimiter property](#). The character that acts as a text-qualifier (if any) is specified by the [OutputQualifier property](#). If the output format is changed to “delimited”, an output delimiter needs to be set.

The grammar file must be open (by calling the [Open method](#)), else an error is generated when trying to access this property.

OutputDelimiter property

Sets or returns a String with the character that delimits the output fields.

Notes

This property is only applicable if the output type is delimited (See [OutputFormatType property](#)). Otherwise reading this property will return nothing.

The grammar file defines a default output delimiter. Setting this property can change this default output delimiter.

The grammar file must be open (by calling the [Open method](#)), else an error is generated when trying to access this property.

OutputQualifier property

Sets or returns a String with the character that is used as the output text-qualifier (such as the single or double quotes character).

Notes

This property is only applicable if the output type is delimited (See [OutputFormatType property](#)). Otherwise reading this property will return nothing.

The grammar file defines a default output qualifier. Setting this property can change this default output qualifier.

The grammar file must be open (by calling the [Open method](#)), else an error is generated when trying to access this property.

OutputFieldCount property

Returns an Integer with the number of fields in the output.

Notes

The grammar file must be open (by calling the [Open method](#)), else an error is generated when trying to read this property.

OutputFieldNames property

Returns a String with the name of the output field.

Syntax

`Object.OutputFieldNames (Index)`

Syntax description

Object a StanRt object.

Index an Integer, with the index of the output field. This index is base 1, so an index of 1 will return the first field, 2 will return the second field, etc. If the index is out of bounds, an empty String ("") will be returned.

Notes

The grammar file must be open (by calling the [Open method](#)), else an error is generated when trying to read this property.

OutputFieldSizes property

Returns an Integer with the size of the output field.

Syntax

`Object.OutputFieldSizes (Index)`

Syntax description

Object a StanRt object.

Index an Integer, with the index of the output field. This index is base 1, so an index of 1 will return the first field, 2 will return the second field, etc. If the index is out of bounds, zero will be returned.

Notes

If the output format is fixed length (see the [OutputFormatType property](#)), then this will be the length of the output field. Otherwise this will be the maximum length of the output field.

The grammar file must be open (by calling the [Open method](#)), else an error is generated when trying to read this property.

OutputFieldDescriptions property

Returns a String with the description of the output field.

Syntax

`Object.OutputFieldDescriptions (Index)`

Syntax description

Object a StanRt object.

Index an Integer, with the index of the output field. This index is base 1, so an index of 1 will return the first field, 2 will return the second field, etc. If the index is out of bounds, an empty String ("") will be returned.

Notes

The grammar file must be open (by calling the [Open method](#)), else an error is generated when trying to read this property.

SuggestedDivider property

Returns a String with the character suggested to use to divide input fields.

Notes

The input to the Standardiser (to the [Stan method](#)) is one String, which usually corresponds to one field. Sometimes it is desired that several fields be the input to the standardiser. In this case, the input fields are

concatenated with a possible divider/delimiter between then. The grammar file can suggest a divider it is expecting.

If the grammar file does not suggest a divider, then this property will return an empty String ("").

The grammar file must be open (by calling the [Open method](#)), else an error is generated when trying to read this property.

Manager object

This object is used to manage the grammar files, applying to all StanRt objects.

KeepLoaded method

Marks a loaded grammar file to stay loaded, even if it has no more clients. See [Loading grammar files with StanRt](#).

Syntax

Object.KeepLoaded (*GrammarName*)

Syntax description

Object a Manager object.

GrammarName a String with the name of the loaded grammar. The same string used to load the grammar file with the [Open method](#).

Return Value

True (non-zero) Grammar file was loaded and successfully marked.

False (zero) Grammar file isn't loaded, so cannot mark it.

DontKeepLoaded method

.Marks a grammar file to unload when there are no more clients using this grammar file

If there are no current clients, the grammar file will unload now. See [Loading grammar files with StanRt](#).

Syntax

Object.DontKeepLoaded (*GrammarName*)

Syntax description

Object a Manager object.

GrammarName a String with the name of the loaded grammar. The same string used to load the grammar file with the [Open method](#). If no string is passed, all loaded grammar files will be marked.

Return Value

True (non-zero) Grammar file was loaded and successfully marked.

False (zero) Grammar file isn't loaded, so cannot mark it.

DefaultDirectory property

Returns or sets a String containing the default directory for the grammar files.

Notes

The [Open method](#) is called with the name of the grammar file. This name will be relative to the directory set here. The grammar file name can be an absolute path (such as "C:\Intech\Grammars\DpidGeocode.grm"), in which case this default directory will be ignored.

When running as a service, this default directory itself is relative to the location of the service executable (usually on the Systems directory).

DefaultProcesses property

Returns a String containing a default process.

Syntax

Object.DefaultProcesses (*Index*)

Syntax description

Object a Manager object.

Index an Integer, with the index of the process. This index is base 1, so an index of 1 will return the first process, 2 will return the second process, etc. If the index is out of bounds, nothing will be returned (an empty string "").

Notes

The default processes are all files in the default directory (see [DefaultDirectory property](#)) with the grammar file extension “.grm”.

To get all default processes, access this property with index 1, then 2, etc, until an empty string is returned.

Licence property

Returns a string with information on the iStan licence, installed on the machine.

Properties property

Returns a variant data type with specific information about StanRt.

Syntax

Object.Properties (*PropertyName*)

Syntax description

Object a Manager object.

PropertyName a String with the name of the property desired. If no such property exists, nothing will be returned.

Supported Properties

The following are the property names currently supported:

<i>Name</i>	Returns a String with the name of the COM module
<i>LoadedGrammars</i>	Returns a String with the names of the currently loaded grammar files, and how many clients each has.
<i>LicenceName</i>	Returns a String with the name that appears in the licence file.
<i>LicenceCompany</i>	Returns a String with the company that appears in the licence file.
<i>LicenceEdition</i>	Returns a String with the edition that appears in the licence file.
<i>LicenceDate</i>	Returns a String with the start date that appears in the licence file.
<i>LicenceExpires</i>	Returns a String with the expiry date that appears in the licence file.
<i>LicenceComputerName</i>	Returns a String with the name of the computer that appears in the licence file.
<i>LicenceValid</i>	Returns a Boolean value, indicating whether the licence allows StanRt to function.
<i>LicenceCPUs</i>	Returns an Integer with the CPU limit that appears on the licence file.
<i>LicenceClients</i>	Returns an Integer with the client limit that appears on the licence file.

<i>ClientCount</i>	Returns an Integer with the number of current clients.
<i>PafVersionInfo</i>	Returns a String with information on the version and expiry date of the installed PAF files.
<i>PafVersionCycle</i>	Returns an Integer with the major version number. This version number is read directly from the version file, and the validity of the version file is not checked.
<i>PafVersionRelease</i>	Returns an Integer with the minor version number. This version number is read directly from the version file, and the validity of the version file is not checked.
<i>AmasCertifiedDate</i>	Returns a String with the date the software was approved by the AMAS program. This string is read from the AMAS Certified Date configuration setting .
<i>StanVersion</i>	The version of the Stan Libraries currently used. (eg. 5.3.37.1).
<i>StanMachine</i>	The name of the machine/computer running the Stan Libraries. This may be a remote machine, for example if DCOM or the socket server is being used.
<i>Properties</i>	Returns a String with the names of all the (above) supported properties. The names are delimited by semi-colons.

PAF libraries properties

Properties from the PAF Libraries [Properties property](#) are also available. Preface the property with "PAF:". See example 3 below.

Example 1

```
Manager1.Properties("LoadedGrammars")
```

might return

```
"DpidGeocode.grm<02>IndNames.grm<01>"
```

indicating that there are 2 clients currently using the grammar file "DpidGeocode", and one currently using "IndNames".

Example 2

```
Manager1.Properties("PafVersionInfo")
```

might return

```
"PAF V2008.4, Expires 12/2008."
```

Example 3

```
Manager1.Properties("PAF:ARFs")
```

will return a semi-colon delimited list of available Address Reference Files.

Read/Write properties

There are a few properties that may be set. See [Read/Write Properties](#).

Update method

Causes all future connections to a particular grammar file, to use an updated version of this grammar file.

Useful in updating grammar files real time, without the need to shut down the server, or close the clients.

Syntax

```
Object.Update (GrammarName)
```

Syntax description

Object a Manager object.

GrammarName a String with the name of the loaded grammar. The same string used to load the grammar file with the [Open method](#). If no string is passed, all loaded grammar files will be updated.

Return value

True (non-zero) Grammar file was loaded and successfully marked.

False (zero) Grammar file isn't loaded, so cannot mark it.

Notes

As explained in [Loading grammar files with StanRt](#), a call to the [Open method](#) will not reload a grammar file, if it is already in memory. After a call to this method (**Update method**), the next call to the **Open method**, will reload the grammar file. This allows an update of a grammar file, without affecting current objects. For an object to make use of the new grammar file, it needs to call the [Close method](#), and then the **Open method** again.

If the grammar file has been marked to remain in memory (by the [KeepLoaded method](#), or by auto loading) then the new version of the grammar file will be loaded and kept loaded.

Loading grammar files with StanRt

When a client calls the [Open method](#), the appropriate grammar file is loaded into memory. If another client then calls the Open method with the same grammar file, it will not reload the grammar file, but rather use the grammar file already loaded by the first client. When all clients using a particular grammar file have called the [Close method](#), or no longer exist, the grammar file will be unloaded.

To retain a particular grammar file in memory, even when all clients have closed, use the [KeepLoaded method](#). This will ensure that even when all clients are finished with this grammar file, it will remain in memory. This is useful if one is expecting this grammar file to be often Opened and Closed, as it saves loading time. Note that the Open method must be called before the KeepLoaded method.

To undo this action, use the [DontKeepLoaded method](#). This will mark the grammar file to unload when all clients are finished with it, and if it has no clients at present, it will unload now.

Alternatively, or in addition to, you can set which grammars are to be kept loaded via the [KeepLoaded](#) configuration setting.

Auto-loading

The server can be configured to [Autoload](#) particular grammar files when it starts.

Loaded grammar files limit

The maximum number of grammar files that can be simultaneously loaded is defined by the [Max Grammars](#) configuration setting. If an attempt is made to open a grammar file after this limit has been reached, an error will be returned.

Updating grammar files real-time

To update a grammar file while the server is running and clients are using the grammar file:

1. Overwrite the old grammar file(s) with the new updated one(s).
2. Call the [Update method](#).

Current StanRt objects using this grammar file will continue to use the old version. All future StanRt objects that call the [Open method](#), will be given the new version of the grammar file.

Take care not to go over the [Max Grammars](#) limit. Both the new and the old version of the grammar files might remain in memory, until all clients using the old version have closed.

Stan Libraries

The Stan Libraries provide access to the standardising engine in real-time.

On Windows platforms, these libraries are multi-thread safe, allowing multi-threaded clients to access its functions and take advantage of multi-processor machines.

These libraries are very similar to those available in a COM library form (see [StanRt](#)).

Win32 DLL

The Windows 32-bit DLL version of this library is in the file `stan32.dll`

The functions in this DLL use the **stdcall** calling convention, whereby the parameters are passed from right to left, and the called function is responsible for removing the parameters off the stack.

C Libraries

A C-Library is available for several other platforms.

A header file “StanLib.h” is provided with this SDK containing the function prototypes.

Function summary

Basic functions

[Open function](#) - :opens a grammar-file (a standardising process).

[Stan function](#) - standardises a record.

[stanStanW function](#) - standardises a Unicode record.

[Close function](#) - closes the grammar-file. To be called when finished.

Grammar-file information functions

[Get_GrammarName function](#)

[Get_Description function](#)

[Get_SuggestedDivider function](#) expected format of the input

Grammar-file output format functions

[Get_OutputFormatType function](#)

[Put_OutputFormatType function](#)

[Get_OutputDelimiter function](#)

[Put_OutputDelimiter function](#)

[Get_OutputQualifier function](#)

[Put_OutputQualifier function](#)

[Get_OutputFieldCount function](#)

[Get_OutputFieldNames function](#)

[GetOutputFieldDescription function](#)

[Get_OutputFieldSizes function](#)

Default grammar files functions

[Get_DefaultDirectory function](#)

[Put_DefaultDirectory function](#)

[Get_DefaultProcesses function](#)

Control which grammar files remain loaded in memory

[KeepLoaded function](#)

[DontKeepLoaded function](#)

[Update function](#)

Other functions

[Get_Licence function](#)

[Get_Properties function.](#)

[Put_Properties function](#)

[stanPutProperty function](#)

[stanGetProperty function](#)

Error codes

Most of these functions return an integer, which is zero indicating success, else, an error code. These error codes can be converted into a meaningful message using the [GetErrorMessage function](#), or looked up in the Grammar SDK [Error Messages](#).

Loading grammar files with the Stan Libraries

When the [Open function](#) is called, the specified grammar file is loaded into memory. If another called to **Open** is made specifying the same grammar file, the Stan Library will not reload the grammar file, but rather use the grammar file already loaded by the first call. When all calls to **Open** using a particular grammar file have been matched with a call to the [Close function](#), the grammar file will be unloaded.

To retain a particular grammar file in memory, even after **Close** has been called, use the [KeepLoaded function](#). This is useful if one is expecting this grammar file to be often Opened and Closed, as it saves loading time. Note **Open** must be called to load the grammar file before calling **KeepLoaded**.

To undo this action, use the [DontKeepLoaded function](#). This will mark the grammar file to unload when the appropriate calls to **Close** have been made. If these calls to **Close** have already been made, the grammar file will unload now.

See the [Get_Properties function](#) on how to know which grammars are currently loaded.

Auto-loading

The Stan Libraries can be configured to [Autoload](#) particular grammar files on the first call to the libraries.

Loaded grammar files limit

The maximum number of grammar files that can be simultaneously loaded is defined by the [Max Grammars](#) configuration setting. If an attempt is made to open a grammar file after this limit has been reached, an error will be returned.

Updating grammar files real-time

To update a grammar file while the server is running and clients are using the grammar file:

1. Overwrite the old grammar file(s) with the new updated one(s).
2. Call the [Update function](#).

Current Ids to this grammar file will continue to use the old version. All future calls to **Open** however will be given the new version of the grammar file.

Take care not to go over the [Max Grammars](#) limit. Both the new and the old version of the grammar files might remain in memory, until all old versions have been closed.

Function reference

Data Types Equivalents

The data types in this documentation are shown as C data types. The following table shows the equivalent data types for Win32 API.

C data type	Win32 API data type	Description
int	INT	32-bit signed integer
int *	LPINT	32-bit pointer to an INT
short	SHORT	16-bit signed integer
short *	PSHORT	32-bit pointer to a SHORT
char	CHAR	An 8-bit (ANSI) character.
char *	LPSTR	Pointer to a null-terminated string of 8-bit (ANSI) characters
char *	PCHAR	Pointer to an 8-bit (ANSI) character
const char *	LPCSTR	Pointer to a constant null-terminated string of 8-bit (ANSI) characters

Open function

Opens a grammar file and allocates memory needed for standardising

Syntax

```
int Open (  
    const char *    sGrammarFile,  
    int bDescriptionOnly,  
    int *          id  
);
```

Parameters

- sGrammarFile* Pointer to a null-terminated string that specifies the grammar file to open. This can be an absolute path, or relative to the default directory (see [Get_DefaultDirectory function](#)).
- bDescriptionOnly* Specifies whether to fully open the grammar file, or just read its description and information. Non-zero to just read its description; Zero to fully load.
- Id* Pointer to a variable to receive the identifier (ID). This ID is subsequently used in other functions.

Return Value

- Zero Function succeeded.
- Error Code Function failed. An error is usually the result of the grammar file being incorrect (error 74), or an error in the grammar file itself.

Notes

When **Open** is called, it provides an ID. This ID is then used to get information about this grammar file (e.g. with the [Get_OutputFieldNames function](#)) and to standardise records with the [Stan function](#).

For each grammar file used, another call to **Open** needs to be made, receiving another ID.

Also, if more than one thread is to standardise records simultaneously (even if using the same grammar file), each should use a different ID. Do this by calling **Open** more than once, and obtaining more than

one ID. The Stan Library will only load the grammar file once, but it will allocate separate memory for standardising, each call to **Open**.

Each call to **Open** should be matched with a call to [Close function](#) (when finished), to free resources.

This maximum number of IDs that can be allocated is specified in the [Max Stans](#) configuration setting.

Stan function

Standardises a record.

Syntax

```
int Stan (  
    int id,  
    const char *    sIn,  
    char *    sOut,  
    int maxlen  
)
```

Parameters

<i>id</i>	Specifies the identifier of the grammar file to use, as obtained from the Open function .
<i>sIn</i>	Pointer to a null-terminated string that is the input to be standardised.
<i>sOut</i>	Pointer to the buffer to receive the standardised output.
<i>maxlen</i>	Specifies the length, in characters, of the buffer for the standardised output.

Return Value

Zero	Function succeeded.
Error Code	Function failed. Common errors are 1102 (invalid ID); 61 (output length greater than maxlen); 50 (too much information in input to be standardised); 110 (the grammar file not open, or open for description only).

Notes

The output will be in a format as specified by the output format (see in the [Get_OutputFormatType function](#)).

For double-bytes strings, use the [stanStanW function](#).

stanStanW function

Standardises a record. Identical to the [Stan function](#), except with wide-character (double byte) strings.

Syntax

```
int stanStanW (  
    int id,  
    const wchar_t * sIn,  
    wchar_t * sOut,  
    int maxlen  
)
```

Notes

maxlen is in number of characters, not number of bytes.

Close function

Unloads a grammar from memory, and frees memory allocated for standardising.

Syntax

```
int Close (  
    int id  
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).

Return Value

Zero Function succeeded.

Error Code Function failed. A common error is 1102 (invalid ID).

Notes

The grammar file will not be unloaded until **Close** is called for each call to **Open** with this grammar file name.

Get_Description function

Retrieves the description of the grammar file.

Syntax

```
int Get_Description (  
    int id,  
    char *    sDescription  
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).

sDescription Pointer to the buffer to receive the description. This buffer should be at least [MAX_DESCRIPTION](#) (Default 255) in length.

Return Value

Zero Function succeeded.

Error Code Function failed.

Notes

This description is provided by the grammar file for information only.

Get_GrammarName function

Retrieves the name of the grammar file, as used when calling the [Open function](#).

Syntax

```
int Get_GrammarName (  
    int id,  
    char *    sGrammarName  
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).

sGrammarName Pointer to the buffer to receive the grammar name. This buffer should be at least 260 bytes in length.

Return Value

Zero Function succeeded.

Error Code Function failed.

Get_OutputFormatType function

Retrieves the format type of the output fields: 0 (fixed-width), 1 (delimited), 2 (XML).

Syntax

```
int Get_OutputFormatType (  
    int id,  
    short *   iType  
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).

iType Pointer to the variable to receive the format type.

Return Value

Zero Function succeeded.

Error Code Function failed.

Notes

If the format type is fixed-width (0), then the length of each field is specified by the [Get_OutputFieldSizes function](#).

If the format type is delimited (1), then the delimiting character is specified by the [Get_OutputDelimiter function](#), and the text-qualifier (if any) is specified by the [Get_OutputQualifier function](#).

Put_OutputFormatType function

Sets the format type of the output fields: 0 (fixed-width), 1 (delimited), 2 (XML).

Syntax

```
int Put_OutputFormatType (  
    int id,  
    short   iType  
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).

iType The format type (0, 1 or 2).

Return Value

Zero Function succeeded.

Error Code Function failed.

Notes

If the format type is set to fixed-width (0), then the length of each field is specified by the [Get_OutputFieldSizes function](#).

If the format type is set to delimited (1), then set the delimiting character via the [Put_OutputDelimiter function](#).

Get_OutputDelimiter function

Retrieves the character that delimits the output fields.

Syntax

```
int Get_OutputDelimiter (
    int id,
    char *    cDelimiter
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).
cDelimiter Pointer to the variable to receive the delimiter character.

Return Value

Zero Function succeeded.
 Error Code Function failed.

Notes

If the output is not delimited (see the [Get_OutputFormatType function](#)) then the *cDelimiter* variable will receive zero.

Put_OutputDelimiter function

Sets the character that delimits the output fields.

Syntax

```
int Put_OutputDelimiter (
    int id,
    char    cDelimiter
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).
cDelimiter The character to be used as delimiter.

Return Value

Zero Function succeeded.
 Error Code Function failed.

Notes

This value is only of meaning, if the output format is delimited (see the [Get_OutputFormatType function](#)).

Get_OutputQualifier function

Retrieves the character that is used as the output text-qualifier (such as the single or double quotes character).

Syntax

```
int Get_OutputQualifier (
    int id,
    char *    cQualifier
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).
cQualifier Pointer to the variable to receive the text-qualifier character.

Return Value

Zero Function succeeded.

Error Code Function failed.

Notes

If the output is not delimited (see the [Get_OutputFormatType function](#)) or if there is no text-qualifier, then the *cQualifier* variable will receive zero.

Put_OutputQualifier function

Sets the character that is used as the output text-qualifier (such as the single or double quotes character).

Syntax

```
int Put_OutputQualifier (
    int id,
    char            cQualifier
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).

cQualifier The character to be used as text-qualifier.

Return Value

Zero Function succeeded.

Error Code Function failed.

Notes

This value is only of meaning, if the output format is delimited (see the [Get_OutputFormatType function](#)). If no text-qualifier is desired, set *cQualifier* to zero.

Get_OutputFieldCount function

Retrieves the number of fields in the output.

Syntax

```
int Get_OutputFieldCount (
    int id,
    int *            iCount
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).

iCount Pointer to the variable to receive the field count.

Return Value

Zero Function succeeded.

Error Code Function failed.

Notes

For each field in the output, information about that field is available via the [Get_OutputFieldNames function](#), [Get_OutputFieldSizes function](#) and [GetOutputFieldDescription function](#). The index passed to these functions is between 1 and *iCount* inclusive.

Get_OutputFieldNames function

Retrieves the name of an output field.

Syntax

```
int Get_OutputFieldNames (
    int id,
    int index,
    char *    sName
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).

index Specifies the index of the output field. This index is base 1, so an index of 1 will return the first field, 2 will return the second field, etc. If the index is out of bounds, a zero length string will be placed in the *sName* buffer.

sName Pointer to the buffer to receive the field name. This buffer should be at least [MAX_NAME](#) (Default 30) + 1 bytes in length.

Return Value

Zero Function succeeded.

Error Code Function failed.

Get_OutputFieldSizes function

Retrieves the size of an output field.

Syntax

```
int Get_OutputFieldSizes (
    int id,
    int index,
    int *    size
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).

index Specifies the index of the output field. This index is base 1, so an index of 1 will return the first field, 2 will return the second field, etc. If the index is out of bounds, size will be given the value of zero.

size Pointer to the variable to receive the field size.

Return Value

Zero Function succeeded.

Error Code Function failed.

Notes

If the output format is fixed length (see the [Get_OutputFormatType function](#)), then this size will be the length of the output field. Otherwise this size will be the maximum length of the output field.

Get_OutputFieldDescriptions function

Retrieves the description of an output field.

This function has been superseded by the [GetOutputFieldDescription function](#).

Syntax

```
int Get_OutputFieldDescriptions (
    int id,
    int index,
```

```
char *    sDescription
)
```

Parameters

<i>id</i>	Specifies the identifier of the grammar file, as obtained from the Open function .
<i>index</i>	Specifies the index of the output field. This index is base 1, so an index of 1 will return the first field, 2 will return the second field, etc. If the index is out of bounds, a zero length string will be placed in the <i>sDescription</i> buffer.
<i>sDescription</i>	Pointer to the buffer to receive the field description. This buffer should be at least MAX_FIELD_DESCRIPTION (Default 50) + 1 bytes in length.

Return Value

Zero	Function succeeded.
Error Code	Function failed.

Notes

The grammar file might not provide a description for a field, in which case, a zero length string will be placed in *sDescription*.

GetOutputFieldDescription function

Retrieves the description of an output field.

Syntax

```
int GetOutputFieldDescription (
    int id,
    int index,
    char *    sDescription
    int maxlen
)
```

Parameters

<i>id</i>	Specifies the identifier of the grammar file, as obtained from the Open function .
<i>index</i>	Specifies the index of the output field. This index is base 1, so an index of 1 will return the first field, 2 will return the second field, etc. If the index is out of bounds, a zero length string will be placed in the <i>sDescription</i> buffer.
<i>sDescription</i>	Pointer to the buffer to receive the field description.
<i>maxlen</i>	Maximum length of a description to place in above buffer. Ie. size of the above buffer minus one.

Return Value

Zero	Function succeeded.
Error Code	Function failed.

Notes

The grammar file might not provide a description for a field, in which case, a zero length string will be placed in *sDescription*.

Get_SuggestedDivider function

Retrives the character suggested by the grammar file to use to divide input fields.

Syntax

```
int Get_SuggestedDivider (
    int id,
    char *    cDivider
)
```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).

cDivider Pointer to the variable to receive the suggested character.

Return Value

Zero Function succeeded.

Error Code Function failed.

Notes

The input to the Standardiser (to the [Stan function](#)) is one string, which usually corresponds to one field. Sometimes it is desired that several fields be the input to the standardiser. In this case, the input fields are concatenated with a possible divider/delimiter between them. The grammar file can suggest a divider it is expecting.

If the grammar file does not suggest a divider, then the *cDivider* variable will receive the null character.

KeepLoaded function

Marks a loaded grammar file to stay loaded, even after calls to the [Close function](#). See [Loading grammar files with the Stan Libraries](#).

Syntax

```
int KeepLoaded (
    const char *    sGrammarFile,
)
```

Parameters

sGrammarFile Pointer to a null-terminated string that specifies the grammar file to keep loaded, as used with the [Open function](#).

Return Value

Zero Grammar was not previously loaded.

Non-Zero Function succeeded.

DontKeepLoaded function

Marks a grammar file to unload when after a call to the [Close function](#) is made for each call to the [Open function](#) with this grammar file name. If these calls to **Close** have already been made, the grammar file will unload now.

Syntax

```
int DontKeepLoaded (
    const char *    sGrammarFile,
)
```

Parameters

sGrammarFile Pointer to a null-terminated string that specifies the grammar file as used with the [Open function](#). If this is NULL, or a pointer to a zero-length string, all grammar files will be marked.

Return Value

Zero	Grammar was not previously loaded.
Non-Zero	Function succeeded.

Notes

See [Loading grammar files with the Stan Libraries](#).

Get_DefaultDirectory function

Retrieves the default [Grammar Directory](#) for the grammar files.

Syntax

```
int Get_DefaultDirectory (  
    char *    sDirectory,  
    int maxlen  
)
```

Parameters

<i>sDirectory</i>	Pointer to the buffer to receive the directory.
<i>maxlen</i>	Specifies the length, in characters, of the above buffer.

Return Value

Zero	Function succeeded.
Non-Zero	Length of the required buffer if the buffer is too small.

Notes

The [Open function](#) is called with the name of the grammar file. This name will be relative to the directory set here. The grammar file name can be an absolute path (such as “C:\Intech\Grammars\DpidGeocode.grm”), in which case this default directory will be ignored.

To set this directory use the [Put_DefaultDirectory function](#).

Put_DefaultDirectory function

Sets the default directory for the grammar files.

[This function is only available on Windows platforms.]

Syntax

```
int Put_DefaultDirectory (  
    const char *    sDirectory,  
)
```

Parameters

<i>sDirectory</i>	Pointer to a null-terminated string that specifies the directory.
-------------------	---

Return Value

Zero	Function succeeded.
Non-Zero	Function failed - Windows system error code.

Notes

The [Open function](#) is called with the name of the grammar file. This name will be relative to the directory set here. The grammar file name can be an absolute path (such as “C:\Intech\Grammars\DpidGeocode.grm”), in which case this default directory will be ignored.

To retrieve this directory, use the [Get_DefaultDirectory function](#).

This default directory is stored in the [Grammar Directory](#) configuration setting, so alternatively it may be set or retrieved directly from the registry/configuration file.

Get_DefaultProcesses function

Retrieves the default processes (the grammar files on the default directory).

Syntax

```
int Get_DefaultProcesses (
    int index
    char *    sGrammar,
    int maxlen
)
```

Parameters

<i>index</i>	Specifies the index of the default process. This index is base 1, so an index of 1 will retrieve the first process, 2 the second, etc. If there are no more processes (the index is out of bounds), a zero length string will be placed in the <i>sGrammar</i> buffer.
<i>sGrammar</i>	Pointer to the buffer to receive the grammar file (process) name.
<i>maxlen</i>	Specifies the length, in characters, of the above buffer.

Return Value

Zero	Function succeeded.
Non-Zero	Length of the required buffer if the buffer is too small.

Notes

The default processes are all files on the default directory (see [Get_DefaultDirectory function](#)) with the grammar file extension “.grm”.

To get all default processes, call this function with index 1, then 2, etc, until a zero length string is retrieved.

Get_Licence function

Retrieves information on the iStan licence, installed on the machine.

Syntax

```
int Get_Licence (
    char *    sLicence,
    int maxlen
)
```

Parameters

<i>sLicence</i>	Pointer to the buffer to receive the licence information.
<i>maxlen</i>	Specifies the length, in characters, of the above buffer.

Return Value

Zero	Function succeeded.
Non-Zero	Length of the required buffer if the buffer is too small.

Get_Properties function

Retrieves specific information about the Standardising Engine.

Syntax

```
int Get_Properties (
    const char *    sName,
    char *          sValue,
    int maxlen
)
```

Parameters

sName Pointer to a null-terminated string that specifies the property name.

sValue Pointer to the buffer to receive the property value.

maxlen Specifies the length, in characters, of the above buffer.

Return Value

Zero Function succeeded.

Non-Zero Length of the required buffer if the buffer is too small.

Supported Properties

See [Properties property](#) for a list of supported properties. Note that for the Stan Library, the properties returned are all converted into Strings.

Put_Properties function

Sets specific functionality desired from the Standardising Engine.

Syntax

```
int Put_Properties (
    const char *    sName,
    const char *    sValue
)
```

Parameters

sName Pointer to a null-terminated string that specifies the property name. See [Read/Write Properties](#) for a list of supported property names.

sValue Pointer to a null-terminated string that specifies the property value.

Return Value

Zero Function succeeded.

Error Code Function failed. A common error is 1106 – “Invalid property name”.

Read/Write Properties

The following valid property names and values can be used with the [Put_Properties function](#):

UseSockets True (use a socket server) or False (do not use a socket server)

TcpHost <hostname> of socket server

TcpPort <portnumber> of socket server

AutoLoad True (enable [Autoload](#)) or False (disable Autoload)

Socket Server

To specify that the Stan Libraries redirect requests to a socket server use the following settings before calling any other functions:

UseSockets True

TcpHost <hostname> of socket server

TcpPost <portnumber> of socket server

To specify not to use a socket server, set the `UseSockets` property to `False`

Autoload

To disable auto loading of grammar files, set `AutoLoad` to `False` before opening a grammar or requesting the `LoadedGrammars` [Properties property](#), otherwise auto loading may have already taken place.

See also [Loading grammar files with the Stan Libraries](#) and the [Autoload](#) configuration setting.

stanGetProperty function

Retrieves specific information about the standardising or the grammar. (Not to be confused with the [Get_Properties function](#), which is not specific to the open grammar).

Syntax

```
int stanGetProperty (
    int id,
    const char * sName,
    char * sValue,
    int maxlen
)
```

Parameters

<i>id</i>	Specifies the identifier of the grammar file, as obtained from the Open function .
<i>sName</i>	Pointer to a null-terminated string that specifies the property name.
<i>sValue</i>	Pointer to the buffer to receive the property value.
<i>maxlen</i>	Specifies the length, in characters, of the above buffer.

Return Value

Zero	Function succeeded.
Non-Zero	Function failed. Common errors are 1115 (buffer too small) and 1102 (invalid id).

Supported Properties

Currently the only properties supported are “LastOutputLength” and “LastOutput”. If the most recent call to the [Stan function](#) or the [stanStanW function](#) returns an error 61 (buffer too small), then the result (output of the standardising) can be retrieved as follows:

1. Call `stanGetProperty(“LastOutputLength”)`
2. Call `stanGetProperty(“LastOutput”)`, using a buffer whose size is one more than the result obtained in (1) above.

If the last call to `Stan` or `stanStanW` succeeded (or returned a different error), then these properties will return 0 and empty respectively. If the last call was to `stanStanW`, then “LastOutput” will return a result in UTF-8 encoding. Instead of calling this function, you could call the `Stan` or `stanStanW` function again with a bigger buffer.

stanPutProperty function

Sets specific information about the standardising or the grammar. (Not to be confused with the [Put_Properties function](#), which is not specific to the open grammar).

Syntax

```
int stanPutProperty (
    int id,
```

```

const char * sName,
const char * sValue,
)

```

Parameters

id Specifies the identifier of the grammar file, as obtained from the [Open function](#).

sName Pointer to a null-terminated string that specifies the property name.

sValue Pointer to a null-terminated string that specifies the property value.

Return Value

Zero Function succeeded.

Non-Zero Function failed. Common errors are 1106 (unknown property name) and 1102 (invalid id).

Supported Properties

Currently the only properties supported are “PARAMETERS” and “PARAMETER:<param-name>”. A grammar file may have parameters defined (see [Parameter definitions](#)). These are variables whose initial values can be set programmatically. To do this, call:

```
stanPutProperty(id, "PARAMETER:<param-name>", "<value>")
```

where <param-name> is the name of the parameter to set, and <value> is the initial value to set it to. To set several parameters in one call:

```
stanPutProperty(id, "PARAMETERS", "<p1>=<v1>;<p2>=<v2>...")
```

where <p1>, <p2> ... are the parameter names, and <v1>, <v2>, ... are their initial values.

Each time data is standardised (via the [Stan function](#) or the [stanStanW function](#)), the parameters will be given their initial values as set here. If the parameter’s value is changed during standardising, it will be reset to its initial value before standardising again. If no initial value is given programmatically, then its initial value will be empty.

Update function

Causes all future calls to **Open** to a particular grammar file, to use an updated version of this grammar file.

Useful in updating grammar files real time, without the need to close grammar files in use.

Syntax

```

int Update (
    const char *    sGrammarFile,
)

```

Parameters

sGrammarFile Pointer to a null-terminated string that specifies the grammar file as used with the [Open function](#). If this is NULL, or a pointer to a zero-length string, all grammar files will be updated.

Return Value

Zero Grammar was not previously loaded.

Non-Zero Function succeeded.

Notes

As explained in [Loading grammar files with the Stan Libraries](#), a call to the [Open function](#) will not reload a grammar file, if it is already in memory. After a call to the **Update** function, the next call to **Open** will

reload the grammar file. This allows an update of a grammar file, without affecting threads currently using this grammar file. For a thread to make use of the new grammar file, it needs to call the [Close function](#), and then **Open** again.

If the grammar file has been marked to remain in memory (by the [KeepLoaded function](#) or by auto loading) then the new version of the grammar file will be loaded and kept loaded.

GetErrorMessage function

Retrieves a textual interpretation of an error number.

Syntax

```
int GetErrorMessage (
    int errorId,
    char *    sError
)
```

Parameters

iError Specifies the error number, as returned from one of the other functions.

sError Pointer to the buffer to receive the error message. A buffer of size 256 is amply enough.

Return Value

This function always returns 0, indicating success.

Configuration Settings

The Stan Libraries rely on configuration settings defining certain behaviours, locations of files etc.

Windows platform

On the Windows platform, these settings are stored in the Windows Registry.

In this documentation, the registry settings are shown in Registry File (*.reg) format. For example:

```
REGEDIT4
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\StanRt]
"GrammarPath"="C:\\Program Files\\Intech\\Grammars"
```

This example would indicate that under the key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Intech\StanRt
```

there is a string value of name `GrammarPath` and of value

```
C:\Program Files\Intech\Grammars
```

For ease of reading, the line “REGEDIT4” is often omitted.

UNIX / LINUX platform

In UNIX / LINUX, these settings are stored in a configuration text file which is searched for in the following order:

1. A file called ‘.intech’ in the current directory
2. A file specified by the INTECH_CONFIG_FILE environment setting
3. A file called ‘.intech’ in the HOME directory
4. The file ‘/usr/etc/.intech’.

This configurations text file contains one key/value pair per line in the form “key/subkey=value”, e.g.

```
StanRt/GrammarPath=/home/intech/grammars
```

Autoload

These [Configuration Settings](#) specify which grammar files (if any) are to be loaded when the [Stan Libraries](#) are first called.

These grammar files will remain loaded, as if the [KeepLoaded function](#) was called.

When running StanRt Com as a Windows Service, opening of these grammar files will be logged in the Windows Event Log.

See also the [KeepLoaded](#) configuration setting.

The example below indicates that the grammar files “DpidGeocode.grm” and “AusPhone.grm” are to be loaded when **Stan** is started. The names of the entries (“1” & “2” in this example) are ignored, and anything can be used.

Window Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\StanRt\Autoload]
"1"="DpidGeocode.grm"
"2"="AusPhone.grm"
```

Configuration Text File example

```
StanRt/Autoload/1=DpidGeocode.grm
StanRt/Autoload/2=AusPhone.grm
```

KeepLoaded

These [Configuration Settings](#) specify which grammar files (if any) are to be kept loaded once opened.

When one of these grammar files is opened, it will remain loaded, as if the [KeepLoaded function](#) was called.

If a grammar is already in the [Autoload](#) configuration setting, it doesn’t need to be put in this KeepLoaded setting.

The example below indicates that the grammar files “DpidGeocode.grm” and “Phone.grm” are to be kept loaded. The names of the entries (“1” & “2” in this example) are ignored, and anything can be used.

Window Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\StanRt\KeepLoaded]
"1"="DpidGeocode.grm"
"2"="Phone.grm"
```

Configuration Text File example

```
StanRt/KeepLoaded/1=DpidGeocode.grm
StanRt/KeepLoaded/2=Phone.grm
```

COM Audit File

See [Configuration Settings](#).

This setting is used for debugging StanRt. If this setting is set, then each time a StanRt COM object is created, a log entry will be written to this file.

Example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\StanRt]
"ComAuditFile"="C:\Temp\StanRt_COM_Audit.log"
```

Grammar Directory

These [Configuration Settings](#) specify the location of the grammar files. The actual directory may be different to those in the examples below.

Window Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\StanRt]
```

```
"GrammarPath"="C:\\Program Files\\Intech\\Grammars"
```

Configuration Text File example

```
StanRt/GrammarPath=/home/intech/grammars
```

Locale

These [Configuration Settings](#) specify which locale to use. This will affect uppercase and lowercase conversions, and word recognition.

The example below will set the locale to the system-default. Without this setting, a neutral locale is used, which doesn't recognise characters above ASCII value 127. See Grammar SDK [Locale setting](#) for more details.

Window Registry example

```
[HKEY_LOCAL_MACHINE\\SOFTWARE\\Intech\\StanRt]  
"Locale"=""
```

Configuration Text File example

```
StanRt/Locale=
```

Max Grammars

These [Configuration Settings](#) specify the maximum number of grammars that can be simultaneously loaded by the Stan Library. The example below sets this to 20, which is the default if this setting is missing.

Window Registry example

```
[HKEY_LOCAL_MACHINE\\SOFTWARE\\Intech\\StanRt]  
"MaxGrammars"="20"
```

Configuration Text File example

```
StanRt/MaxGrammars=20
```

Max Stans

These [Configuration Settings](#) specify the maximum number of IDs that can be allocated by the Standardising Library functions. An Id is allocated on a call to the [Open function](#), and released on a call to the [Close function](#)

The example below shows this set to 100, which is the default if this setting is missing.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\\SOFTWARE\\Intech\\StanRt]  
"MaxStans"="100"
```

Configuration Text File example

```
StanRt/MaxStans=100
```

Stan Sockets

These [Configuration Settings](#) instruct the Stan Libraries to redirect all functions call to the socket server, instead of processing them locally.

The Host can be an IP address, a machine name, or a remote server address.

This example tells the Stan Library to use the socket server on machine 192.168.1.49 on port 4000.

Windows Registry example

```
[HKEY_LOCAL_MACHINE\\SOFTWARE\\Intech\\StanRt]  
"TcpHost"="192.168.1.49"  
"TcpPost"="4000"
```

Configuration Text File example

```
StanRt/TcpHost=192.168.1.49
StanRt/TcpPort=4000
```

StanConsole application

StanConsole is a console application that can run:

1. In interactive mode, to test a grammar file
2. In batch mode, to standardise records in a text file.
3. An IQ Standardiser project.

Run StanConsole without any parameters to see command-line usage.

Interactive mode

In StanConsole interactive mode, the grammar file is read, and any error in it displayed with the line where the error occurred. If the grammar file runs without errors, records can be entered one-by-one, and the output of the Standardiser displayed.

To quit, type *Ctrl-Z* and *Enter* (*Ctrl-D* on UNIX/LINUX), when prompted for the input record.

Usage

```
StanConsole [options] grammar-file
```

Options

- d Show description of grammar file only
- n Show a list of output fields with their names
- s Show output fields delimited by |
- i Show the modified input string
- t Show all tokens found
- g Show the grammar file as understood by StanConsole
- o Don't show output string
- k Each record spans multiple lines, divided by blank line

Notes

The *grammar-file* may be an absolute path name. Otherwise, the grammar file is searched for in the current directory. If not found, it is looked for in the [Grammar Directory](#).

Batch mode

In StanConsole batch mode, the grammar file is read, each line in the input file is standardised, and the output written to the output file.

Usage

```
StanConsole [options] grammar-file input-file output-file
```

Parameters

- | | |
|---------------------|---|
| <i>grammar-file</i> | the name of the grammar file. An absolute path; relative to the current directory; or relative to the Grammar Directory . |
| <i>input-file</i> | the name of the file containing the records to be standardised. |
| <i>output-file</i> | the name of the file to write the output to. If it already exists, it will be overwritten. |

Options

-cCOUNT	Show record count to stderr every COUNT (default 100)
-lLINES	Each record spans LINES number of lines
-k	Each record spans multiple lines, divided by blank line
-fSTART	Start data column (must also use -e)
-eLEN	Number of data columns (only with -f)
-aDELIM	ASCII value of char that delimits fields (only with -f), 0 for fixed-width

Running an IQ Standardiser project

IQ Standardiser creates a project file with an “.ist” extension, and an associated “.sta” file. If the IQ Standardiser project processes text-files only, then you can use StanConsole to run such a project file, instead of using [IQ Standardiser](#).

Usage

```
StanConsole -b project-file
```

Description

-b The flag to indicate running an IQ Standardiser project. On Windows, “/b” may be used instead.

project-file the name of the IQ Standardiser project file, with or without the “.ist” extension. A relative or absolute path may be used.

StanJNI Java Class

The StanJNI Java class provides the same standardising engine as the [Stan Libraries](#).

Installation requirements

StanJNI uses a Java Native Interface (JNI), and so requires the dynamic library `iqstanjni`, which in turn requires the library `iqstan` (`Stan32.dll` on Windows).

Sample code

The TestStanJni class provides an example of using this StanJNI class.

StanJNI reference

Functions provided by the StanJNI class are listed below:

Basic functions

```
void Open(String sGrammarFile, boolean bDescriptionOnly)
```

```
String Stan (String sInput)
```

```
void Close ()
```

Grammar-file information functions

```
String Description ()
```

```
String GrammarName ()
```

Grammar-file output format functions

```
int OutputFormatType ()
```

```
char OutputDelimiter ()
```

```
char OutputQualifier ()
```

```
int OutputFieldCount ()
```

```
String OutputFieldNames(short iField)
int OutputFieldSizes(short iField)
String OutputFieldDescriptions(short iField)
char SuggestedDivider ()
void OutputFormatType(int iType)
void OutputDelimiter(char cDelimiter)
void OutputQualifier(char cQualifier)
```

Default grammar files functions

```
String DefaultDirectory()
void DefaultDirectory(String sDirectory)
String DefaultProcesses(short iIndex)
```

Control which grammar files remain loaded in memory

```
boolean DontKeepLoaded(String sGrammar)
boolean KeepLoaded(String sGrammar)
boolean Update(String sGrammar)
```

Other functions

```
String Licence()
String Properties(String sPropertyName)
```

StanJNI.Open

Opens the grammar file specified by the first parameter.

Syntax

```
void Open(String sGrammarFile, boolean bDescriptionOnly)
```

Notes

After **Open** has been called, all the methods that return information about the grammar file become available.

After **Open** has been called with **bDescriptionOnly False**, the method [Stan](#) becomes available.

The grammar file remains in memory, until [Close](#) is called.

If **Open** had already been called, calling it again will first unload the previous grammar file, then load it with the new one.

An exception is thrown if there is an error in grammar file or in finding the grammar file.

StanJNI.Stan

Standardises a record.

Syntax

```
String Stan(String sInput)
```

Notes

A grammar file must have been opened with the [Open](#) method.

The format of the returned standardised record is specified by [OutputFormatType](#).

StanJNI.Close

Closes the grammar file opened via the [Open](#) method.

Syntax

```
void Close()
```

StanJNI.Description

Returns a description of the grammar file opened via the [Open](#) method.

Syntax

```
String Description()
```

StanJNI.GrammarName

Returns the name of the grammar file opened via the [Open](#) method.

Syntax

```
String GrammarName()
```

StanJNI.OutputFormatType

Returns or sets the format of the standardised record to be returned by the [Stan](#) method.

Syntax

```
int OutputFormatType()
```

```
void OutputFormatType(int iType)
```

Parameters

iType	Type of output format: 0 (fixed-length output fields), 1 (delimited output fields) or 2 (XML format)
-------	--

Notes

The grammar file defines a default output format. This output format can be read using the first method. If desired, use the second method to change this output format.

If the output is fixed-width, the lengths of the output fields are specified by [OutputFieldSizes](#).

If the output is delimited, the character that delimits the fields is specified by [OutputDelimiter](#). The character that acts as a text-qualifier or quotes (if any) is specified by [OutputQualifier](#). If the output format is changed to “delimited”, an output delimiter needs to be set.

The grammar file must be open (by calling [Open](#)), else an exception will be thrown.

StanJNI.OutputDelimiter

Returns or sets the character that will delimit the fields returned by the [Stan](#) method.

Syntax

```
char OutputDelimiter()
```

```
void OutputDelimiter(char cDelimiter)
```

Notes

Only applicable if [OutputFormatType](#) is 1 (Delimited).

The grammar file defines a default output delimiter. Use the first method to retrieve this output delimiter. If desired, use the second method to change this output delimiter.

The grammar file must be open (by calling [Open](#)), else an exception will be thrown.

StanJNI.OutputQualifier

Returns or sets the character (if any) that will quote the fields returned by the [Stan](#) method.

Syntax

```
char OutputQualifier()  
void OutputQualifier(char cQualifier)
```

Notes

Only applicable if [OutputFormatType](#) is 1 (Delimited).

The grammar file defines a default output qualifier. Use the first method to retrieve this output qualifier. If desired, use the second method to change this output qualifier.

The grammar file must be open (by calling [Open](#)), else an exception will be thrown.

StanJNI.OutputFieldCount

Returns the number of fields that are returned by the [Stan](#) method.

Syntax

```
int OutputFieldCount()
```

StanJNI.OutputFieldNames

Returns the names of the fields returned by the [Stan](#) method.

Syntax

```
String OutputFieldNames(short iField)
```

Parameters

iField The index to the field, between 1 and [OutputFieldCount](#) inclusive. If the parameter is out of bounds, null will be returned.

StanJNI.OutputFieldSizes

Returns the sizes of the fields returned by the [Stan](#) method.

Syntax

```
int OutputFieldSizes(short iField)
```

Parameters

iField The index to the field, between 1 and [OutputFieldCount](#) inclusive. If the parameter is out of bounds, null will be returned.

If [OutputFormatType](#) is 0 (fixed-width), then this method will return the size of the field. Otherwise it will return the maximum size of the field.

StanJNI.OutputFieldDescriptions

Returns a description (if any) of the fields returned by the [Stan](#) method.

Syntax

```
String OutputFieldDescriptions(short iField)
```

Parameters

iField The index to the field, between 1 and [OutputFieldCount](#) inclusive. If the parameter is out of bounds, null will be returned.

StanJNI.SuggestedDivider

Returns the character suggested to use to divide input fields.

Syntax

```
char SuggestedDivider()
```

Notes

The input to the Standardiser (the [Stan](#) method) is one String, which usually corresponds to one field. Sometimes it is desired that several fields be the input to the standardiser. In this case, the input fields are concatenated with a possible divider/delimiter between them. The grammar file can suggest a divider it is expecting.

If the grammar file does not suggest a divider, then this property will return null.

StanJNI.DefaultDirectory

Returns or sets the directory in which to look for grammar files when calling the [Open](#) method.

Syntax

```
String DefaultDirectory()  
void DefaultDirectory(String sDirectory)
```

StanJNI.DefaultProcesses

Returns the grammar files (files ending with “.grm”) in the [DefaultDirectory](#).

Syntax

```
String DefaultProcesses(short iIndex)
```

Parameters

iIndex	Passing 1 will return the first grammar file, 2 the second etc. When there are no more grammar files, null will be returned.
--------	--

StanJNI.Licence

Returns information about the IQ Standardiser licence installed on this machine.

Syntax

```
String Licence()
```

StanJNI.Properties

Returns specific information about the standardising engine.

Syntax

```
String Properties(String sPropertyName)
```

Notes

See [Properties property](#) for a list of supported property names.

StanJNI.DontKeepLoaded

Marks a grammar file to be unloaded from memory when no longer in use. This is the default, unless the grammar file was marked [KeepLoaded](#).

Syntax

```
boolean DontKeepLoaded(String sGrammar)
```

Notes

Pass a null or empty string to mark all loaded grammars.

Returns whether the grammar file was loaded before calling **DontKeepLoaded**.

StanJNI.KeepLoaded

Marks a grammar file to remain loaded in memory even after calling [Close](#). This will allow the grammar to [Open](#) faster.

Syntax

```
boolean KeepLoaded(String sGrammar)
```

Marks a grammar file to remain loaded in memory even after calling [Close](#). This will allow the grammar to [Open](#) faster.

Pass a null or empty string to mark all loaded grammars.

Returns whether the grammar file was loaded. If the grammar file was not loaded, **KeepLoaded** has no affect.

StanJNI.Update

Mark the grammar file to be reloaded by all new calls to [Open](#). This can be useful in updating the grammar files without closing current objects using the grammar files.

Syntax

```
boolean Update(String sGrammar)
```

Pass a null or empty string to mark all loaded grammars.

Returns whether the grammar file was loaded. If the grammar file was not loaded, **Update** has no affect.

Socket Server

The Socket Server provides the standardising functions provided by the [Stan Libraries](#).

This same socket server also provides access to the PAF Libraries described in the Rapid SDK Reference Manual, and Match Libraries described in the Matcher SDK Reference Manual.

Socket Server Installation

The Socket Server usually runs as Service on Windows (`IQSocSvr.exe`) and a daemon on UNIX (`iqsocsvr`).

The Socket Server requires [Stan Libraries Installation](#).

Windows

To install the socket server as a Service (listening on port 4000, for example), type:

```
IQSocSvr.exe -i 4000
```

Start the service using Windows Services Manager or Service Controller (`sc.exe`).

Use `-h` to see a full list of options:

```
C:\>IQSocSvr.exe -h
```

```
iqsocsvr [-h | -d | -D | -s ] [-lFile] [hostname] port
iqsocsvr -i [ [-cFile] [hostname] port]
iqsocsvr -u
```

```
-h    Show this help message
-d    Debug. Show data sent and received via sockets
-D    Debug connections only
-s    Silent. No writing to terminal
-l    Log errors to file specified
-i    Install as a Window's service
-u    Uninstall Window's service
```

To change the port, stop the service, uninstall it, and reinstall it.

UNIX

To listen on port 4000 for example, type:

```
iqsocsvr 4000
```

To run as a daemon, type:

```
iqsocsvr -a 4000
```

To see a full list of options, type without any parameters:

```
iqsocsvr
```

Calling the socket server

This section describes how to write a client to call the socket server to access the functions in the [Stan Libraries](#).

Conventions

- Each message sent to the socket server must be terminated with a New-Line character (ASCII 10) or a Carriage-Return character followed by a New-Line character (ASCII 13 then ASCII 10).
- Each message returned from the socket server will be terminated by a New-Line character.
- In each message (both sent and received), the tab character (ASCII 9) is used to divide parameters.
- Messages are case sensitive.
- Send a `quit` message to the server to close the connection.
- Most messages returned from the socket server will begin with an error code. This is the same as the return value of the corresponding Library function.
- If the call was successful, there might be return parameters after the zero error code.
- If the call was unsuccessful, there might be an error message after the error code.

Function Summary

An equivalent message can be sent to the socket server to call most of the functions in the [Stan Libraries](#). The table below lists each message sent, and the result of the function returned by the socket server. Parameters in the table are shown separated by a comma and a space, but this is a tab in the messages. The RV (Return Value) is the value returned by the equivalent library function, usually an error code or zero for success.

See each topic in the list for a description of the function and its parameters.

Function Called	Message Sent	Message Returned
Close function	stanClose, id	RV
DontKeepLoaded function	stanDontKeepLoaded, sGrammarFile	RV
Get_DefaultDirectory function	stanGetDefaultDirectory	RV, sDirectory
Get_DefaultProcesses function	stanGetDefaultProcesses, index	RV, sGrammar
Get_Description function	stanGetDescription, id	RV, sDescription
Get_GrammarName function	stanGetGrammarName, id	RV, sGrammarName

Get Licence function	stanGetLicence	RV, sLicence
Get OutputDelimiter function	stanGetOutputDelimiter, id	RV, Ascii Value of cDelimiter
Get OutputFieldCount function	stanGetOutputFieldCount, id	RV, iCount
GetOutputFieldDescription function	stanGetOutputFieldDescriptions, id, index	RV, sDescription
Get OutputFieldNames function	stanGetOutputFieldNames, id, index	RV, sName
Get OutputFieldSizes function	stanGetOutputFieldSizes, id, index	RV, size
Get OutputFormatType function	stanGetOutputFormatType, id	RV, iType
Get OutputQualifier function	stanGetOutputQualifier, id	RV, Ascii Value of cQualifier
Get Properties function	stanGetProperties, sName	RV, sValue
stanGetProperty function	stanGetProperty, id, sName	RV, sValue
Get SuggestedDivider function	stanGetSuggestedDivider, id	RV, Ascii Value of cDivider
GetErrorMessage function	stanGetErrorMessage, errorId	sError
KeepLoaded function	stanKeepLoaded, sGrammarFile	RV
Open function	stanOpen, bDescriptionOnly, sGrammarFile	RV, id or Error Description
Put DefaultDirectory function	stanPutDefaultDirectory, sDirectory	RV
Put OutputDelimiter function	stanPutOutputDelimiter, id, Ascii Value of cDelimiter	RV
Put OutputFormatType function	stanPutOutputFormatType, id, iType	RV
Put OutputQualifier function	stanPutOutputQualifier, id, Ascii Value of cQualifier	RV
stanPutProperty function	stanPutProperty, id, sName, sValue	RV
Stan function	stanStan, id, sIn	RV, sOut
Update function	stanUpdate, sGrammarFile	RV

Example

The following example will open the DpidGeocode Grammar file and standardise an address. The id returned here is 1, but it could be another value.

Send: stanOpen<tab>0<tab>DpidGeocode.grm

Receive: 0<tab>1

Send: stanStan<tab>1<tab>L7 35 spring st, bondi junction NSW

Receive: 0<tab>NSW<tab>2022<tab>BONDI JUNCTION<tab>SPRING ST
<tab><tab><tab>44664340<tab>35 <tab><tab><tab><tab><tab> L <tab>
7<tab><tab><tab><tab><tab><tab> 4<tab> amended missing
Postcode<tab> 1301011111202011101100302003013102113 <tab>010
<tab>208 <tab>S165 <tab>G561 <tab>B532 <tab><tab> L
<tab><tab><tab> Level 7 <tab>35 Spring St <tab><tab><tab> f

Send: stanClose<tab>1

Receive: 0

SOAP Server

The SOAP/XML Server provides the standardising functions provided by the [Stan Libraries](#). It also provides access to some of the address searching functions provided by the PAF Libraries.

SOAP Server Installation

Installing the SOAP Server

The SOAP Server handles its own socket connections and HTTP requests, so does not require a Web Server. However, it must be configured to use a different port to any existing Web Servers already running on the machine. Check which ports are already in use by any existing web server (e.g. 80) before starting the SOAP Server.

The SOAP Server usually runs as Service on Windows (IQSoapSv.exe) and a daemon on UNIX (iqsoapsv). However, it can also run as a stand-alone executable, usually for debugging.

The SOAP Server requires [Stan Libraries Installation](#).

Windows

To install the SOAP server as a Windows service (listening on port 82, for example), type:

```
IQSoapSv.exe -i 82
```

Start the service using Windows Services Manager or Service Controller (sc.exe).

Use the -h option to see a full list of options:

```
IQSoapSv.exe -h

IQSoapSv [-d | -s ] [-llogfile] port
IQSoapSv -i [port]
IQSoapSv [-h | -u | -v]

-d    Debug. Show data sent and received
-s    Silent. No writing to terminal
-l    Log errors to file specified
-i    Install as a Window's service
-h    Show this help message
-u    Uninstall Window's service
-v    Show version number
```

To change the port, stop the service, uninstall it, and reinstall it.

UNIX

To listen on port 82 for example, type:

```
iqsoapsv 82
```

To run as a daemon, type:

```
iqsoapsv -a 82
```

To see a full list of options, type:

```
iqsoapsv -h
```

Calling the SOAP Server

This section describes how to make requests to the SOAP Server to access the [Stan Libraries](#) functions. SOAP functions are defined in `iqoffice.wsdl` or `iqoffice2.wsdl`

The interface to the SOAP server is different from the other Java, sockets, and COM interfaces, because SOAP is a connectionless interface. The full request must be sent to the SOAP server in one call, and the full result retrieved by the call.

Standardise

This SOAP Service function standardises one item of data.

Input / Request Parameters

Grammar	The grammar file name.
Input	The data to standardise.

Example request 1

```
POST / HTTP/1.1
Host: www.yourcompany.com
Content-Type: text/xml; charset=utf-8
Content-Length: 609
Connection: close
SOAPAction: "http://intechsolutions.com.au/soap/"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://intechsolutions.com.au/soap/">
  <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns1:Standardise>
      <Grammar>DpidGeocode.grm</Grammar>
      <Input>level 7 35 spring st, bondi junction nsw</Input>
    </ns1:Standardise>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Output / Response

Name	Value	A Name-Value pair array, each item having the name of the output field and its value.
ErrorCode		Error code number, zero for success. If the error code is non-zero, there may be more information in the returned error string.
Optional		Error string.

Sample response 1

```
HTTP/1.1 200 OK
Server: IQSoapSv/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: 2721
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:ns1="http://intechsolutions.com.au/soap/"
  <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns1:StandardiseResult>
      <NameValuePairs xsi:type="SOAP-ENC:Array" SOAP-
ENC:arrayType="ns1:NameValuePair[39]">
        <item><Name>state</Name><Value>NSW</Value></item>
        <item><Name>postCode</Name><Value>2022</Value></item>
        <item><Name>locality</Name><Value>BONDI JUNCTION</Value></item>
        <item><Name>streetName</Name><Value>SPRING</Value></item>
        <item><Name>streetType</Name><Value>ST</Value></item>
        <item><Name>streetSuffix</Name><Value></Value></item>
        <item><Name>postalType</Name><Value></Value></item>
        <item><Name>dpid</Name><Value>44664340</Value></item>
        <item><Name>houseNo1</Name><Value>35</Value></item>
        <item><Name>houseSuffix1</Name><Value></Value></item>
        <item><Name>houseNo2</Name><Value></Value></item>
        <item><Name>houseSuffix2</Name><Value></Value></item>
        <item><Name>flatType</Name><Value></Value></item>
        <item><Name>flatNo</Name><Value></Value></item>
        <item><Name>levelType</Name><Value>L</Value></item>
        <item><Name>levelNo</Name><Value>7</Value></item>
        <item><Name>buildingName1</Name><Value></Value></item>
        <item><Name>buildingName2</Name><Value></Value></item>
        <item><Name>lotNo</Name><Value></Value></item>
        <item><Name>postalNo</Name><Value></Value></item>
        <item><Name>postalPrefix</Name><Value></Value></item>
        <item><Name>postalSuffix</Name><Value></Value></item>
        <item><Name>flag</Name><Value>4</Value></item>
        <item>
          <Name>flagString</Name>
          <Value>amended missing Postcode</Value>
        </item>
        <item>
          <Name>barcode37</Name>
          <Value>1301011111202011101100302003013102113</Value>
        </item>
        <item><Name>bsp</Name><Value>010</Value></item>
        <item><Name>npsp</Name><Value>208</Value></item>
        <item><Name>streetSoundex</Name><Value>S165</Value></item>
        <item><Name>streetRSoundex</Name><Value>G561</Value></item>
        <item><Name>localitySoundex</Name><Value>B532</Value></item>
        <item><Name>flatTypeMatch</Name><Value></Value></item>
        <item><Name>levelTypeMatch</Name><Value>L</Value></item>
        <item><Name>preAddress1</Name><Value></Value></item>
        <item><Name>preAddress2</Name><Value></Value></item>
        <item><Name>address1</Name><Value>Level 7</Value></item>
        <item><Name>address2</Name><Value>35 Spring St</Value></item>
        <item><Name>address3</Name><Value></Value></item>
        <item><Name>postAddress</Name><Value></Value></item>
        <item><Name>amendedFlag</Name><Value>f</Value></item>
      </NameValuePairs>
      <ErrorCode>0</ErrorCode>
    </ns1:StandardiseResult>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Sample request 2

```

POST / HTTP/1.1
Host: www.yourcompany.com

```

```

Content-Type: text/xml; charset=utf-8
Content-Length: 579
Connection: close
SOAPAction: "http://intechsolutions.com.au/soap/"

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://intechsolutions.com.au/soap/">
  <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns1:Standardise>
      <Grammar>NonExistant.grm</Grammar>
      <Input>Dummy Data</Input>
    </ns1:Standardise>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Sample response 2

```

HTTP/1.1 200 OK
Server: IQSoapSv/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: 602
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ns1="http://intechsolutions.com.au/soap/">
  <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <ns1:StandardiseResult>
      <ErrorCode>74</ErrorCode>
      <ErrorString>
        ERR_OPEN_GRAMMAR_FILE. File: C:\Intech\Grammars\NonExistant.grm
      </ErrorString>
    </ns1:StandardiseResult>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

BatchStan

This SOAP Service function standardises several records of data.

Input / Request Parameters

Grammar The grammar file name.

InputRecords An array of records of data to standardise.

OutputDelimiter The character to delimit output fields in a given record.

Example request 1

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"

```

```

xmlns:ns2="http://intechsolutions.com.au/soap2/">
<SOAP-ENV:Body>
  <ns2:BatchStan>
    <ns2:Grammar>DpidGeocode.grm</ns2:Grammar>
    <ns2:InputRecords>
      <ns2:item>Level 7 35 spring st, Bondi Junction NSW</ns2:item>
      <ns2:item>6 Macquarie Street, Sydney NSW</ns2:item>
    </ns2:InputRecords>
    <ns2:OutputDelimiter>|</ns2:OutputDelimiter>
  </ns2:BatchStan>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Output / Response

FieldNames A list of field names. Each output record will be composed of these fields.

OutputRecords An array of standardised records. One per input record.

ErrorCode Error code number, zero for success.

ErrorString Description of error, if ErrorCode non-zero.

Sample response 1

```

HTTP/1.1 200 OK
Server: IQSoapSv/5.4
Content-Type: text/xml; charset=utf-8
Content-Length: 1466
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xop="http://www.w3.org/2004/08/xop/include"
  xmlns:ns1="http://intechsolutions.com.au/soap/"
  xmlns:ns2="http://intechsolutions.com.au/soap2/">
  <SOAP-ENV:Body>
    <ns2:BatchStanResult>
      <ns2:FieldNames>
state|postCode|locality|streetName|streetType|streetSuffix|postalType|dpid|
houseNo1|houseSuffix1|houseNo2|houseSuffix2|flatType|flatNo|levelType|level
No|buildingName1|buildingName2|lotNo|postalNo|postalPrefix|postalSuffix|fla
g|flagString|barcode37|bsp|npsp|streetSoundex|streetRSoundex|localitySoundex|
flatTypeMatch|levelTypeMatch|preAddress1|preAddress2|address1|address2|ad
dress3|postAddress|amendedFlag
      </ns2:FieldNames>
      <ns2:OutputRecords>
        <ns2:item>
NSW||BONDI JUNCTION|SPRING|ST|||35|||||L|7|||||U|too many differences:
missing Postcode, Single number to Range|||S165|G561|B532||L|||Level 7 35
Spring Street|||fq
        </ns2:item>
        <ns2:item>
NSW|2000|SYDNEY|MACQUARIE|ST|||68541829|6|||||PALIAMENT
HOUSE|||||4|amended missing
Postcode|1301012022121101220230330200002221313||M260|E625|S350|||||Paliame
nt House|6 Macquarie Street|||f
        </ns2:item>
      </ns2:OutputRecords>
    </ns2:BatchStanResult>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

```

    <ns2:ErrorCode>0</ns2:ErrorCode>
    <ns2:ErrorString></ns2:ErrorString>
  </ns2:BatchStanResult>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

GetGrammarInfo

This SOAP Service function provides information about a grammar – its output fields, its description etc.

Input / Request Parameters

Grammar The grammar file name.

Example request 1

```

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns2="http://intechsolutions.com.au/soap2/">
  <SOAP-ENV:Body>
    <ns2:GetGrammarInfo>
      <ns2:Grammar>Soundex.grm</ns2:Grammar>
    </ns2:GetGrammarInfo>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Output / Response

Description The description of the grammar, as specified in the grammar itself.

OutputFormat The default output format: Fixed or Delimited.

OutputDelimiter The default delimiting character, if the output format is delimited.

OutputQualifier The default output qualifier, if any, if the output is delimited.

SuggestedDivider The character the grammar suggests to delimit parts of the input.

OutputFields An array of information about the output fields – their names, maximum sizes and descriptions.

ErrorCode Error code number, zero for success.

ErrorString Description of error, if ErrorCode non-zero.

Sample response 1

```

HTTP/1.1 200 OK
Server: IQSoapSv/5.4
Content-Type: text/xml; charset=utf-8
Content-Length: 1199
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xop="http://www.w3.org/2004/08/xop/include"
  xmlns:ns1="http://intechsolutions.com.au/soap/"
  xmlns:ns2="http://intechsolutions.com.au/soap2/">
  <SOAP-ENV:Body>
    <ns2:GrammarInfoResult>
      <ns2:Description>

```

```

    Returns SOUNDEX, NYSIIS & METAPHONE of input
</ns2:Description>
<ns2:OutputFormat>Fixed</ns2:OutputFormat>
<ns2:OutputDelimiter></ns2:OutputDelimiter>
<ns2:OutputQualifier></ns2:OutputQualifier>
<ns2:SuggestedDivider></ns2:SuggestedDivider>
<ns2:OutputFields>
  <ns2:item>
    <ns2:Name>Soundex</ns2:Name>
    <ns2:Description></ns2:Description>
    <ns2:Size>4</ns2:Size>
  </ns2:item>
  <ns2:item>
    <ns2:Name>Nysiis</ns2:Name>
    <ns2:Description></ns2:Description>
    <ns2:Size>8</ns2:Size>
  </ns2:item>
  <ns2:item>
    <ns2:Name>Metaphone</ns2:Name>
    <ns2:Description></ns2:Description>
    <ns2:Size>8</ns2:Size>
  </ns2:item>
</ns2:OutputFields>
<ns2:ErrorCode>0</ns2:ErrorCode>
<ns2:ErrorString></ns2:ErrorString>
</ns2:GrammarInfoResult>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

StanB

StanB is a COM object library that provides methods to standardise a table of records using any number of standardising processes (grammar files). The ways in which records are standardised are stored in a Specifications file ([Stan Spec File](#)).

To use StanB:

1. Create a [StanB object](#).
2. Name the Spec File using the [SpecFile property](#).
3. Open an OleDb RecordSet with the data to be standardised.
4. Call the [DefineSpec method](#), which will prompt you to define how you want to standardise the records.
5. Open an updateable OleDb RecordSet to store the standardised records.
6. Call the [Run method](#) to standardise the records.

Notes

You will need to create a table in which to store the standardised records. Use the [OutputFields property](#) and [OutputFieldSizes property](#) to get a list of field names and lengths for this table.

If you want to store the standardised records in a text file, then skip step 5.

If the input is a text-file, then skip step 3.

The standardised record may also be sent to a stored procedure using the [Run method](#).

StanB object

The StanB object is the only object in the StanB object library.

SpecFile property

Returns or sets a String containing the name of the Spec File, which stores the ways in which the records in a file are to be standardised.

This is a full path name, without any extension, e.g. C:\Intech\StanB\Spec1

Notes

This property must be set before the [DefineSpec method](#) or [Run method](#) is called.

DefineSpec method

Displays a screen prompting the user to define how the records will be standardised.

Syntax

```
Object.DefineSpec (InputFields [, ObligatoryFields])
```

Syntax description

Object a [StanB object](#).

InputFields the fields that will be the input to the standardiser. This is either an OleDb RecordSet with these fields, or a string array with the field names.

ObligatoryFields Optional. A Variant that is a String Array with the names of the fields that must be used in the [NoProcess](#) (fields that will be placed in the output without being standardised). This will place these fields into the NoProcess and not allow them to be removed.

Return value

A Boolean, whether a valid spec was saved.

Note

The [SpecFile property](#) must be set before calling this method.

The spec file may be created manually, see [Stan Spec File](#).

IsSpecValid method

Determines whether the standardising spec is valid for the given input fields.

Syntax

```
Object.IsSpecValid (InputFields [, ObligatoryFields])
```

Syntax description

Object a [StanB object](#).

InputFields the fields that will be the input to the standardiser. This is either an OleDb RecordSet with these fields, or a string array with the field names.

ObligatoryFields Optional. A Variant that is a String Array with the names of the fields that must be used in the [NoProcess](#) (fields that will be placed in the output without being standardised).

Return value

A String with a message describing what is invalid about the spec. If the spec is valid, the string will be empty.

Note

The [SpecFile property](#) must be set before calling this method.

Run method

Standardises the records according to a predefined spec.

Syntax

Object.Run (*Input*, *Output1*, *Output2*, *InteractWithDesktop*)

Syntax description

Object a [StanB object](#).

Input The records to be input, one of the following:

- an OleDb RecordSet with the records to be standardised. The RecordSet must be positioned at its beginning (or at the first record to be standardised).
- a String with the description of a text file, see [Text-file description](#).

Output1 Where to store the standardised records, one of the following:

- an updateable OleDb RecordSet having the required fields
- a String with the description of a text-file, see [Text-file description](#).
- an Array with the stored procedure settings, see [Stored Procedure description](#).
- Null, indicating that the input RecordSet should be updated with the output.

Output2 Another location to store the standardised records. As above, or Empty if only one output is required.

Ineract A boolean expression whether there should be any interaction with the Desktop.

True - a screen will be displayed showing the progress of the standardising

False - there will be no interaction with the desktop.

Return value

A long integer with the number of records standardised.

Notes

The [SpecFile property](#) must be set, and the [Stan Spec File](#) defined before calling this method.

If two outputs are used, then one and only one must be a text-file output. In other words, one must be a text-file output, and the other a database output.

Text-file description

The *Input* and *Output* parameter of the [Run method](#) can be a description of a text-file.

The format of this description string is the name/path of the text-file, followed by any of the following clauses, separated by semi-colons. Extra spaces should not appear in this string.

Syntax

```
FieldDelimiter=FieldDelimiter
TextQualifier=TextQualifier
Fields=Field1,Field2,...
FieldSizes=Size1,Size2,...
NamesFirst=True
Append=True
```

Parameters

FieldDelimiter The character which delimits the fields. If not defined, the file is in fixed format, in which case **Field Sizes** must be defined.

TextQualifier	(Optional) The character used to enclose the fields, can be only one character in length.
NamesFirst	If defined the names of the fields appear in the first line of the text-file, in which case Fields does not need to be defined. This is only valid for delimited and not fixed-length, format.
Append	If defined the output will be appended to the file, otherwise the file will be overwritten. This is not applicable to the <i>Input</i> parameter.

Example

```
C:\Intech\Data\Customers.csv;FieldDelimiter=,;TextQualifier="";Fields=Name,Addr1,Addr2,Phone;
```

This defines the file C:\Intech\Data\Customers.csv as comma delimited format, with the quotation mark (") as a text-qualifier, having four fields (Name, Addr1, Addr2 and Phone).

Stored Procedure description

The *Output* parameter of the [Run method](#) can be a description of a stored procedure. In this case, it needs to be an array with the following elements:

1. An ADODB.Connection object with an active connection to the database.
2. A String with the name of the Stored Procedure
3. A String Array with the Stored Procedure Parameter Names
4. An Integer Array with the Stored Procedure Parameter data types
5. An Integer Array with the Stored Procedure Parameter Sizes
6. A Variant Array with the values to be passed to each parameter of the Stored Procedure.
An Integer element signals the index to an output field to pass.
Anything else will be passed directly.

Data types

```
Boolean=11, BSTR=8, Char=129, Currency=6, Date=7, DBDate=133, DBFileTime=137, DBTime=134, DBTimeStamp=135, Decimal=14, Double=5, GUID=72, Integer=3, Numeric=131, Single=4, SmallInt=2, TinyInt=16, UnsignedInt=19, UnsignedSmallInt=18, UnsignedTinyInt=17, VarChar=200, VarWChar=202, Wchar=130.
```

Visual Basic Example

The following Visual Basic code will create the vOutput parameter to be passed to the Run method, to describe the stored procedure:

```
Dim vOutput ' the output parameter
Dim MyConnection As ADODB.Connection
Dim StoredProcName As String
Dim ParameterNames (1 To 3) As String
Dim ParameterTypes (1 To 3) As Integer
Dim ParameterSizes (1 to 3) As Integer
Dim ParameterValues (1 To 3) As Variant
Set MyConnection = New ADODB.Connection
MyConnection.Provider = "SQLOLEDB"
MyConnection.Open "server=COMPUTER1;database=pubs"
StoredProcName = "putAddress"
ParameterNames(1) = "Param1"
ParameterNames(2) = "Param2"
ParameterNames(3) = "Param3"
```

```

ParameterTypes(1) = adVarChar
ParameterTypes(2) = adVarChar
ParameterTypes(3) = adVarChar
ParameterSizes(1) = 50
ParameterSizes(2) = 50
ParameterSizes(3) = 2
ParameterValues(1) = 1      ' The first output field
ParameterValues(1) = 2      ' The second output field
ParameterValues(1) = "Y"    ' The letter Y
vOutput = Array(MyConnection, StoredProcName, ParameterNames,
ParameterTypes, ParameterSizes, ParameterValues)

```

OutputFields property

A read-only Variant, with a String array containing the names of the output fields.

Notes

This list of fields can be used (together with the [OutputFieldSizes property](#)) to create a table to store the standardised records – the output of the [Run method](#).

The [SpecFile property](#) must be set, and the [Stan Spec File](#) defined before accessing this property.

OutputFieldSizes property

A read-only Variant, with an Integer array containing the sizes of the output fields.

Notes

This property is to be used with the [OutputFields property](#)

The [SpecFile property](#) must be set, and the [Stan Spec File](#) defined before accessing this property.

Licence property

Returns a string with information on the iStan licence, installed on the machine.

This property returns the same value as the [Licence property](#) of the [Manager object](#).

Properties property

Returns a variant data type with specific information about StanB.

Syntax

Object.Properties (*PropertyName*)

Syntax description

Object a StanB object.

PropertyName a String with the name of the property desired. If no such property exists, nothing will be returned.

Notes

Currently, the only property supported is *PafVersionInfo*, which will return a String containing the version and expiry date of the installed PAF files, e.g.

```
PAF V2008.4, Expires 12/2008
```

Stan Spec File

The Stan Specifications (Spec) File stores information on how StanB standardises records.

This file is read by the [Run method](#), the [OutputFields property](#) and the [OutputFieldSizes property](#).

This file can be created by either of the following two methods:

- Using the StanB [DefineSpec method](#), which creates the file via a Graphical User Interface
- Manually creating the file using the following guidelines.

File format

Each spec file can contain one or more processes (grammar files). Below is the format for one process:

```
PROCESS [process-name]
OUT index length name
IN field-type length name
[DELIMITER delimiter]
[REMOVE_CR]
```

Format description

PROCESS

process-name	The name of the grammar file (without the .grm extension), relative to the default directory (see DefaultDirectory property), or an absolute path.
OUT line	One such line for each output field.
index	The index of the output field of the grammar file.
length	The defined length of the output field, available from the StanRt OutputFieldSizes property .
name	The name of the output field. This field name must be in the output table passed to the Run method , unless a text-file output is used.
IN	One line for each input field.
field-type	The type of data in the field. Use one of the values in the data types table.
length	The defined maximum length of the input field.
name	The name of the input field. A field of this name must exist in the input table passed to the Run method .
DELIMITER	
delimiter	The ASCII value of the character delimiting the input fields, if there are more than one, before being passed to the process.
REMOVE_CR	(Optional) Removes Carriage-Return Line-Feed combinations from text fields before they are passed to the process.

NoProcess

If no process name is provided the input fields are copied to the output fields without modification, this is called the “No Process”. In this case, the field on the first “IN” line, will be copied to the field on the first “OUT” line. So the number of IN lines must equal the number of OUT lines for this “No Process”.

Notes

The output length value is used for fixed-length text-file outputs, in the [OutputFieldSizes property](#), and in the [DefineSpec method](#). If none of these are going to be used, then a zero value may safely be used instead of the correct value.

The input field type and length are used mainly with the [DefineSpec method](#), to check if the input table definition has changed. Hence, zero values may safely be used instead of the correct values.

Each grammar file defines a certain number of output fields and gives them names (see [OutputFieldNames property](#)). However, the output of StanB as defined in the Spec file, can change these names. Hence, the grammar file output fields are referred to by their index, not their name. This is a base-zero index, so 0 is the first field, 1 the second, etc.

Data types

Boolean=11, BSTR=8, Char=129, Currency=6, Date=7, DBDate=133, DBFileTime=137, DBTime=134, DBTimeStamp=135, Decimal=14, Double=5, GUID=72, Integer=3, Numeric=131, Single=4, SmallInt=2, TinyInt=16, UnsignedInt=19, UnsignedSmallInt=18, UnsignedTinyInt=17, VarChar=200, VarWChar=202, Wchar=130.

Example

```

PROCESS
OUT          0      6      ID
OUT          0     30     InAddress1
OUT          0     30     InAddress2
OUT          0     30     InAddress3
IN           0      6      ID
IN           0     30     Address1
IN           0     30     Address2
IN           0     30     Address3

PROCESS DpidGeocode
OUT          0      3      state
OUT          1     12     postCode
OUT          2     30     locality
OUT          3     30     streetName
OUT          4      4     streetType
OUT          5      2     streetSuffix
OUT          6     11     postalType
OUT          7      8     dpid
OUT          8      5     houseNo1
OUT          9      1     houseSuffix1
OUT         10      5     houseNo2
OUT         11      1     houseSuffix2
OUT         12      7     flatType
OUT         13      7     flatNo
OUT         14      2     levelType
OUT         15      5     levelNo
OUT         16     30     buildingName1
OUT         17     30     buildingName2
OUT         18      6     lotNo
OUT         19      5     postalNo
OUT         20      3     postalPrefix
OUT         21      3     postalSuffix
OUT         22      1     flag
OUT         28     50     preAddress1
OUT         29     50     preAddress2
OUT         30     50     address1
OUT         31     50     address2
OUT         32     40     address3
OUT         33     50     postAddress
IN           0     29     Address1
IN           0     31     Address2
IN           0     28     Address3
DELIMITER    44

```


Enhanced Grammars

Summary

Enhanced grammar files supplied with IQ Office DTS and Enterprise Editions are summarised below.

Grammar	Category	Data Type	Data Processed
AccountNames	Enhanced	Persons and Organisations	Relationships between and names of organisations or individuals.
AliasNames	Enhanced	Persons	Nicknames and common names.
AusPhone	Enhanced	Telephone numbers	Convert pre-1997 to current Australian telephone numbers.
Company	Enhanced	Organisations	Parse names of companies.
DecodeBarcodeScanner	Enhanced	Addresses	Decodes output of barcode scanner.
Email	Enhanced	Addresses	Parse email addresses.
FixLastLine	Enhanced	Addresses	Fix last line of Australian addresses.
FixLastLine2	Enhanced	Addresses	Fix last line of Australian addresses.
IndNames	Enhanced	Persons	Full or abbreviated names of up to two individuals, including titles and initials.
IndNames3a	Enhanced	Persons	Full or abbreviated names of up to five individuals, including honorifics, titles, initials and suffixes.
Organisation3	Enhanced	Organisations	Names of up to three organisations.
OrgOrIndividual	Enhanced	Persons and Organisations	Names of up to three organisations and eight individuals.
Phone	Enhanced	Telephone numbers	Australian telephone numbers.
Phone2	Enhanced	Telephone numbers	Australian and international telephone numbers.
RemoveAddress	Enhanced	Addresses	Remove address from phrase.
RuleInList	Enhanced	Values	Compare values, see Rule Grammars .
RuleInSizeRange	Enhanced	Values	Compare values.
RuleInValueRange	Enhanced	Values	Compare values.
RuleIsValidAddress	Enhanced	Addresses	Validate addresses.
Soundex	Enhanced	Words	Generate phonetic algorithm codes for words.

AccountNames

This enhanced grammar determines whether the input is an organisation or individual, then parses the names of up to eight individuals or three organisations which have some relationship between them.

Expected Input

A free format organisation name or individual name, or several organisations or several people.

Use any of the following Entity Hint or Type prefixes followed by a semicolon to indicate the type of entity:

<code>ETHint:P;</code>	Entity is more likely to be a person
<code>ETHint:C;</code>	Entity is more likely to be a company/organisation
<code>ETType:P;</code>	Entity is a person
<code>ETType:C;</code>	Entity is a company/organisation
<code>SaluteHint:</code>	Salutation hint, usually the name to which a letter is addressed, must be after <code>ETHint</code> and <code>ETType</code> . This enhances the title and gender choice, e.g. <code>SaluteHint:Mr & Mrs Smith;John & Mary Smith</code>
<code>GenderHint:</code>	Gender hint which enhances gender choice, either M (male) or F (female), must be after <code>ETHint</code> and <code>ETType</code> . This will only be used if the gender cannot be otherwise determined. For example, <code>GenderHint:M;Kim Smith</code>

Output Fields

The field length is shown in brackets (n).

`inputType (20)` Whether the first entity in the input is an `Org` (Organisation), a `Person`, or `Ambiguous`.

Individual fields – the following seven fields are repeated for up to eight individuals, suffixed by the number of the individual, i.e. from `title1` to `title8`

<code>title1 (5)</code>	Title of the individual, e.g. Mr, Mrs, Dr
<code>firstName1 (20)</code>	First given name of the individual.
<code>middleName1 (20)</code>	Second and other given names of the individual.
<code>lastName1 (25)</code>	Surname of the individual.
<code>suffix1 (10)</code>	Suffix of the individual, e.g., Senior.
<code>typicalGender1 (1)</code>	The assumed gender of the individual: M or F (based on given names), m or f (based on title) or underscore _ (name is unisex or in gender-conflict with other given names or with the title.
<code>flag1 (12)</code>	Morbidity indicator, e.g. <code>Estate</code> or <code>The Late</code> .

Organisation fields – the following nine fields are repeated for up to three organisations, prefixed by `prim` (first) `sec` (second) or `third` (third) organisation.

<code>primName (60)</code>	Organisation key name.
<code>primQual (30)</code>	Organisation qualifiers.
<code>primType (15)</code>	Organisation type
<code>primNysiisName (8)</code>	NYSIIS of organisation key name
<code>primRsndxName (4)</code>	Reverse Soundex of organisation name
<code>primNysiisFull (8)</code>	NYSIIS of organisation key name and qualifiers
<code>primCleanOrg (120)</code>	Locality and punctuation removed from end
<code>primPreOrg (50)</code>	Text before the company name
<code>primPostOrg (50)</code>	Text after the <code>primQual</code> or <code>primType</code>

The following fields contain the relationships between entities.

checkFlag	Flag indicating anomalies: Y (data truncated), 1 (one letter surname), P (PreOrg or PostOrg fields are populated).
RelationshipType1 (20)	The first relationship in the input, e.g. "As Trustee For " in "Mr & Mrs Cook As Trustee For Cook Family Trust".
RelationshipFrom1 (20)	The entity (or entities) in the first relationship. In the above example, "i1,i2", meaning the first two individual entities (Mr Cook & Mrs Cook), returned in title1 to flag1 and title2 to flag2 fields.
RelationshipTo1 (20)	The entity (or entities) in the first relationship. In the above example, "o1", meaning the first organisation entity (Cook Family Trust), returned in the primName to primPostOrg fields.
RelationshipType2 (20)	The second relationship in the input, e.g. "Trading As" in "Mr & Mrs Cook As Trustee For Cook Family Trust Trading As The Cook Company".
RelationshipFrom2 (20)	In the above example, this will be "o1", meaning the first organisation (Cook Family Trust) in the primName to primPostOrg fields.
RelationshipTo2 (20)	In the above example, this will be "o2", meaning the second organisation (The Cook Company) in the secName to secPostOrg fields.

Customising

This grammar can be customised by customising the following grammars which it uses: [OrgOrIndividual, Organisation3](#) and [IndNames3a](#). To customise this grammar, you can also edit the following files:

- `Site_Custom_Person_Details.txt` – edit entries in the lists used to parse individual names according to the description of each list
- `Site_Custom_Company_Details.txt` – edit entries in the lists used to parse organisation names according to the description of each list

AliasNames

This enhanced grammar converts between first names and nicknames.

Expected Input

A single nickname, abbreviation or diminutive such as Rob, Pat, Dan or Sam, or a first name such as Robert or Robyn, Patricia or Patrick, Danielle or Daniel, or Samuel and Samantha.

Output Fields

The field length is shown in brackets (n).

commonAlias (50)	The common form of the name
commonAliasNysiis (8)	The NYSIIS code of the common name (see Phonetic Algorithm).
aliasNamesList (50)	A list of names commonly associated with this nickname.
nameFound (1)	A flag indicating that the name was found: Y (yes) or blank
TypicalGender (1)	The assumed gender of the individual: M or F (based on given names), m or f (based on title) or underscore _ (name is unisex or in gender-conflict with other given names or with the title).

Examples

Input: pat

commonAlias	PATRICIA
commonAliasNysiis	PATRAC

aliasNamesList	PATRICIA,PATRICK,PAT,RICK,TRISH,TRICIA,TRISHA
nameFound	Y
TypicalGender	F

AusPhone

This enhanced grammar converts Australian landline telephone numbers in the format before the 1997 Telecommunications Numbering Plan to current ten digit Full National Numbers comprising a two digit area code and eight local digits.

Expected Input

A valid Australian pre-1997 landline telephone number with area code, e.g. 075 938 123

Output Fields

The field length is shown in brackets (n).

areaCode (3)	Old Australian area code, e.g. 075
localNumber (7)	Old local number, e.g. 938123
austelAreaCode (2)	New two digit area code, e.g. 07
austelLocalNumber (8)	New eight digit local number, e.g. 55938123
midDigits (3)	Middle digits of old number, e.g. 938

Company

This enhanced grammar processes the names of companies.

Expected Input

A free-format company name.

Output Fields

The field length is shown in brackets (n).

name (30)	Corporation key name
qual (30)	Corporation qualifiers
type (15)	Corporation type
miscInfo (15)	Division or misc info
outletType (4)	Branch or outlet type
outletId (6)	Branch or outlet identification
nysiisName (8)	NYSIIS of corporation key name (see Phonetic Algorithm)
rsndxName (4)	Reverse Soundex of corporation name
nysiisFull (8)	NYSIIS of corporation key name and qualifiers
cleanCompany (40)	Locality and punctuation removed from end

Examples

	Cook & Son assoc	Cook-Baker Consulting p / l
name	COOK	COOK BAKER
qual	SONS	CONSULTING
type	ASSOCIATES	PTY LTD

nysiisName	CAC	CACBACAR
rsndxName	K200	R212
nysiisFull	CACSAN	CACBACAR
cleanCompany	Cook & Son Assoc	Cook-Baker Consulting P / L

DecodeBarcodeScanner

This enhanced grammar decodes the output of a barcode scanner used for processing returned mail. The barcode contains address details in the form of a DPID and optional customer details. Customer information is decoded using C-coding and N-coding but no validation is performed. See [Barcodes Explained](#).

Expected Input

Output of the barcode scanner, a comma delimited string of barcodetype, dpid, customerinfo, and parityNumbers fields.

Output Fields

The field length is shown in brackets (n).

barcodetype (2)	The type of barcode
dpid (8)	DPID
custinfoC (10)	Customer information in C-coding.
custinfoN (16)	Customer information in N-coding.
rsParity (12)	Reed Solomon Error Correction Parity values used for error correction

Email

This enhanced grammar parses email addresses containing the @ symbol.

Expected Input

A phrase containing an email address.

Output Fields

The field length is shown in brackets (n).

UserName (80)	Text before the @ symbol
DomainName (80)	Text after the @ symbol
TopLevelDomain (50)	Top level domain name
Domain (50)	Domain name
EmailAddress (170)	Full email address stripped of other text
Valid (1)	Validity flag: Y (valid) or blank (invalid)

Examples

	Mailto: info@nsw.gov.au	pat.rat@race.com.au	go@you.z
UserName	info	pat.rat	go
DomainName	nsw.gov.au	race.com.au	you.z
TopLevelDomain	.gov.au	.com.au	
Domain	nsw	race	

EmailAddress	info@nsw.gov.au	pat.rat@race.com.au	go@you.z
Valid	Y	Y	

FixLastLine

This enhanced grammar attempts to correct the state, postcode and locality information in the last line of an address. It is used by [Dpid3Pass](#). The FixLastLine2 grammar performs the same function, but uses the PAF Library, see [iPaf32_FixLastLine](#).

Expected Input

Tab-delimited State <tab> Pcode <tab> Locality.

Output Fields

The field length is shown in brackets (n).

state (3) 3 letter state abbreviation
pcode (12) 4-digit post code
locality (30) Locality name

Examples

	NT 0810 Casuarina	2042 Newtown
state	NT	NSW
pcode	0810	2042
locality	Casuarina	Newtown

IndNames

This enhanced grammar parses the names and titles of up to two individuals. [IndNames3a](#) also handles honorifics, titles, gender-ambiguous names, nicknames, abbreviations, awards and suffixes.

Expected Input

Full or abbreviated names of up to two individuals, including titles and initials.

Output Fields

The field length is shown in brackets (n).

The following fields are repeated for up to two individuals:

title (5) Title of the individual, e.g. Mr, Mrs, Dr, Major, Rev
firstName (20) First given name of the individual.
middleName (20) Second and other given names of the individual.
initials (2) Initials of individual.
lastName (25) Surname of the individual.
nysiisLast (8) NYSIIS code of last name
sndxFIRST (4) Soundex code of first name of the individual (see Phonetic Algorithm).
nysiisFirst (8) NYSIIS code of first name (see Phonetic Algorithm).

Examples

	Mr Joseph Albert Torrens	Basil St John	Mr Hood
--	-----------------------------	---------------	---------

title	Mr		Mr
firstName	Joseph	Basil	
middleName	Albert		
initials	JA	B	
lastName	Torrens	St John	Hood
sndxFirst	J210	B240	
nysiisFirst	JASAF	BASAL	

	Andrew & Pat Smith	Dr Beth and Mr Paul Jens van der Heiden
title		Dr
firstName	Andrew	Beth
middleName		
initials	A	B
lastName	Smith	Van Der Heiden
sndxFirst	A536	B300
nysiisFirst	ANDR	BAT
nysiisLast	SNAT	VANDARAD
firstName2	Pat	Paul
middleName2		Jens
Initials2	P	
lastName2	Smith	Van Der Heiden
sndxFirst2	P300	P400
nysiisFirst2	PAT	PAL

IndNames3a

This standard grammar parses the names and titles of up to five individuals. It handles honorifics, titles, gender-ambiguous names, nicknames, abbreviations, awards and suffixes.

Expected Input

Full or abbreviated names of up to five individuals, including honorifics, titles, initials and postnominals.

Output Fields

The field length is shown in brackets (n). The following fields are repeated for up to five individuals:

style (20)	Honorific, e.g. The Honourable, Her Excellency
title (5)	Title of the individual, e.g. Mr, Mrs, Dr, Major, Rev
firstName (20)	First given name of the individual.
middleName (20)	Second and other given names of the individual.
lastName (25)	Surname of the individual.
suffix (10)	Suffix of the individual including postnominals, e.g., Senior, QC, OBE
typicalGender (1)	The assumed gender of the individual: M or F (based on given names), m or f (based on title) or underscore _ (name is unisex or in gender-conflict with other given names or with the title).

flag (12) Morbidity indicator, e.g. Estate or The Late.

Examples

	The Late Mr Joseph Albert Torrens Senior Junior	Basil St John QC	Robin Hood	Mr Hood
title	Mr			Mr
firstName	Joseph	Basil	Robin	
middleName	Albert Torrens			
lastName	Senior	St John	Hood	Hood
suffix	JNR	QC		
typicalGender	M	M	–	m
flag	Deceased			

	AE and PR Smith	Dr Beth and Mr Paul J van der Heiden	The Hon Sir Albert and Lady Amelia Fforsythe
style			The Honourable
title		Dr	Sir
firstName	A	Beth	Albert
middleName	E		
lastName	Smith	Van Der Heiden	Fforsythe
typicalGender		F	M
title2		Mr	Lady
firstName2	P	Paul	Amelia
middleName2	R	J	
lastName2	Smith	Van Der Heiden	Fforsythe
typicalGender		M	F

Customising

To customise this grammar, edit the following files:

- `UnisexNames.txt` – add a list of unisex names which will indicate gender-uncertainty with an underscore in the `typicalGender` field, e.g. Robin, Kim, Pat
- `Surname.txt` – add a list of surnames, which sometimes help to distinguish between the given name and surname.
- `Site_Custom_Person_Details.txt` – edit entries in the lists used to parse individual names according to the description of each list

The grammar can also be customised by editing the following DEFINE parameters, see [Customising with DEFINES](#).

- `CHECKFOR_SURNAME_INITIAL` – This will parse a name such as "Smith B", where Smith is the surname and B is the initial of the first name. Without this defined, Smith will be the first name and B the surname.
- `GENDER_CLASH` as `Nothing`, `Title` or `Name` – If the gender of the first name is not the same as the gender of the title, `GENDER_CLASH` indicates what gender to return in the `typicalGender` field. For

example "Mr Mary Jones". Nothing will return an underscore, Title will return the gender of the title (M in this example), and Name will return the gender of the first name (F in this example).

- **DOUBLE_SPACE_INVERT** – If there are two words in the input, the grammar will assume that the first word is the given name and the second is the surname. If this parameter is defined, and there are two spaces between the words, the grammar will swap the given name with the surname, i.e. it will assume that the surname is first, followed by the given name(s).
- **DOUBLE_SPACE_INVERT_CORRECTION** – When used with **DOUBLE_SPACE_INVERT** this will prevent swapping the given name with the surname if the first word is a known given name, and the second word is not.
- **UPPERCASE_INVERT** – if this is defined and the first word is in upper-case, and subsequent words are in lower or title-case, the grammar will assume that the first word is the surname and the other words are the given names.

Organisation3

Parses the names of up to three organisations or companies, separated by "and" or the ampersand character "&". Corrects minor spelling errors and variations such as P/L, P-L, etc., and removes an address from the end of the input.

Expected Input

Names of one or up to three organisations or companies.

Output Fields

The field length is shown in brackets (n).

The following fields are repeated for each company or organisation:

name (60)	Corporation key name
qual (30)	Corporation qualifiers
type (15)	Corporation type
nysiisName (8)	NYSIIS code of corporation key name (see Phonetic Algorithm)
rsndxName (4)	Reverse Soundex of corporation name
nysiisFull (8)	NYSIIS of corporation key name and qualifiers
clean (12)	Locality and punctuation removed from end
pre (50)	Anything before the name
post (50)	Anything after the <code>qual</code> or <code>type</code>

Examples

	Roberston FT	Bulli Community Bank p/l, 1 Bank St Bulli NSW 2516	Cook, Baker and Assoc AND Cook Partners p/l, Accountants
name1	ROBERSTON	BULLI	COOK BAKER AND ASSOCIATES
qual1		COMMUNITY BANK	
type1	FAMILY TRUST	PTY LTD	
nysiisName1	RABARSTA	BAL	CACBACAR
rsndxName1	N326	I410	S322
nysiisFull1	RABARSTA	BALACANA	CACBACAR

clean1	Roberston FT	Bulli Community Bank P/L	Cook, Baker And Assoc
post1		1 BANK ST BULLI NSW 2516	
name2			COOK
qual2			PARTNERSHIP
type2			PTY LTD
nysiisName2			CAC
rsndxName2			K200
nysiisFull2			CACPARTN
clean2			Cook Partners P/L
post2			, ACCOUNTANTS

Customising

To customise this grammar, edit the following files:

- `Site_Custom_Company_Details.txt` – adding or changing entries in lists used in parsing. This file contains a description of each list and how to edit it.

The grammar can also be customised by editing the following DEFINE parameters (see [Customising with DEFINES](#)):

- `DO_REMOVE_IN_BRACKETS` – removes anything in parentheses and puts it in the post1 field.
- `DO_REMOVE_LOCALITY_IN_BRACKETS` – removes locality in parentheses
- `DO_CLEAN_COMPANY` – removes extra punctuation from company/organisation name.
- `DONT_REMOVE_ADDRESS_FROM_END` – removes address from end of company/organisation name and puts it in the post1 field
- `DO_REMOVE_LOCALITY_AT_END` – removes locality from end of company/organisation name and puts it in the post1 field.

OrgOrIndividual

This enhanced grammar determines whether the input is an organisation or individual, then parses the names of up to eight individuals or three organisations.

Expected Input

A free format organisation name or individual name, or several organisations or several people.

Use any of the following Entity Hint or Type prefixes followed by a semicolon before the input to indicate the type of entity:

```
ETHint:P;      Entity is more likely to be a person
ETHint:C;      Entity is more likely to be a company/organisation
ETType:P;      Entity is a person
ETType:C;      Entity is a company/organisation
```

Output Fields

These are the same as [Organisation3](#) and [IndNames3a](#)

Customising

This grammar can be customised by customising the following grammars which it uses: [Organisation3](#) and [IndNames3a](#). The grammar can also be customised by editing the following DEFINE parameters (see [Customising with DEFINES](#)).

- `DO_LOCALITY_CHECK` – In certain cases, if the input begins or ends with a locality name, the grammar will mark it as an organisation.
- `DO_ABBREVIATION_CHECK` – In certain cases, if the first word in the input is short, with many consecutive consonants or vowels, the grammar will mark it as an organisation.

Phone

This enhanced grammar parses Australian landline, mobile, international and Freecall telephone numbers in the same way as [Phone2](#), except that it does not handle:

- Non-Australian international numbers, e.g. `+(64)9-303 5717`
- Non-numeric prefixes, e.g. `Freecall 1300 101 202`,
`Int +(612)83052100`
- Australian numbers before 1996, e.g. `075 938 123`
- The `countryName` and `comments` fields

Expected Input

A valid Australian telephone number with or without punctuation or prefix.

Output Fields

The field length is shown in brackets (n).

`countryCode` (3) Country prefix after the “+”.

`areaCode` (2) 2-digit Australian area code.

`number` (20) Number without the area code.

`fullNumber` (20) Full number formatted with spaces.

`type` (1) Flag indicating type of number: L (land line), M (mobile), S (special), U (unknown), I (international), O (pre-1996 Australian numbers)

`valid` (1) Number validity: Y (yes) or N (no)

Phone2

This enhanced grammar parses Australian and international telephone numbers, including the following valid formats:

- Land lines with or without including area code, e.g. `8305 2100`, `02 8305 2100`, `(08) 8952 1024`
- Landline without area code but with state, e.g. `NT 8944 9900`, `Tas 6230 5605`
- Mobiles, e.g. `0404 570 670`, `0404570670`, `+61404 570 670`
- Freecall 1800, and local rate 1300 and 13 numbers, e.g. `1800 787 898`, `131313`, `1300 975 707`, `Freecall 1300 101 202`
- Australian numbers before 1996, e.g. `075 938 123`
- International numbers including prefix, e.g. `+61-2-8305 2100`, `+(612)83052100`, `+61404 570 670`, `+61 2 83052100`, `+(64)9-303 5717`, `0011+33 01.64.22.01.57`

Invalid formats include:

- Too few digits, e.g. 550234
- Letters, e.g. 1300FRUIT, 13DRAIN
- Invalid area code, e.g. (09) 1234 5678

Expected Input

A valid Australian or international telephone number with or without punctuation or prefix.

Output Fields

The field length is shown in brackets (n). The output fields are the same as [Phone](#), except for the following.

countryName (40) Name of the country with the given country code.

comments (50) Other information in the input, before and/or after the number.

Examples

	Int +(612)83052100	+(64)9-303 5717	0011+33 01.64.22.01.57
countryCode	61	64	33
areaCode	02		
number	83052100	93035717	164220157
fullNumber	02 8305 2100	+64 93035717	+33 164220157
type	L	I	I
valid	Y	Y	Y
countryName	Australia	New Zealand	France
Comments	Int		

	NSW 6882 2844	Freecall 1300 101 202	0409111777	(09) 1234 5678
countryCode				
areaCode				
number	68822844	1300101202	0409111777	0912345678
fullNumber	02 6882 2844	1300 101 202	0409 111 777	09 1234 5678
type	L	S	M	U
valid	Y	Y	Y	N
countryName				
Comments		Freecall		

RemoveAddress

This enhanced grammar removes the address from any input words. It is used by [Organisation3](#).

Expected Input

A phrase containing an address

Output Fields

The field length is shown in brackets (n).

output (255) Non-address component of input phrase

address (255) Address component of input phrase

Examples

	Great Cakes, 10 Hordern St Newtown 2042	BigCorp, Unit 15 Redfern Technology Park	Ms Sue Cook, PO Box 12 Newtown 2042
output	Great Cakes	BigCorp	Ms Sue Cook, PO Box 12 Newtown 2042
address	10 Hordern St Newtown 2042	Unit 15 Redfern Technology Park	

Rule Grammars

These enhanced grammars are provided for various value-comparison and address validation functions used by IQ Office Profiler.

RuleInList

This enhanced grammar determines if an item is in a defined alphanumeric list.

Expected Input

Comma delimited list of items <tab> comparison

Output Fields

Result Result of test : Y (in list) N (not in list) X (null)

Examples

	1,2,3,4 10	1,2,3,4 2	1,2,3,4	St,Rd,Cl,Av Ave
Result	N	Y	X	N

RuleInValueRange

This enhanced grammar determines if a numeric value is within a defined range.

The range is calculated as: Minimum <= Value <= Maximum

Expected Input

Tab-delimited Minimum <tab> Maximum <tab> Value

Output Fields

Result Result of numeric test : Y (in range) N (out of range)

Examples

	1 10 1	1 10 3	1 10 4.5	1 10 11
Result	Y	Y	Y	N

RulesValidAddress

This enhanced grammar validates an input address and outputs a confidence rating and the fields amended to correct the address.

Expected Input

Free-form address.

Output Fields

The field length is shown in brackets (n).

Confidence (1) Rating that address is valid: 1 (Correct), 2 (High), 3 (Moderate), 4 (Low), 5 (Insufficient), 6 (Unknown)

Amended (1) Fields amended to correct address, combined from [Amended Flag codes](#): A (Locality, codes a – e), B (State, codes h, i), C (Postcode, codes f, g, w), D (Street, codes j – p, x)

Examples

	1 Bank St Bulli NSW 2516	1 Bank Bulli NSW 2516	1 Bank St Bulli NSW	1 Bank Bulli NSW	1 Bank St NSW
Confidence	1	2	2	4	5
Amended		D	C		

Soundex

This enhanced grammar generates the corresponding Soundex, NYSIIS and Metaphone phonetic algorithm codes for any input words.

Expected Input

Any words.

Output Fields

The field length is shown in brackets (n).

Soundex (4) Soundex code of input words (see Phonetic Algorithm).

Nysiis (8) NYSIIS code of input words

Metaphone (8) Metaphone code of input words

Examples

	Bill	Beth van der Heiden	Parramatta	101 Main Street
Soundex	B400	B315	P653	
Nysiis	BAL	BATHVAND	PARANAT	MANSTRAT
Metaphone	BL	B0FNTRHT	PRMT	MNSTRT

Data-specific Grammars

Summary

Data-specific grammar files specific to the IQ Office licence and installed data package are summarised below.

Grammar	Category	Data Type	Data Processed
ABSGeocode	Data	Addresses	Australian addresses and ASGC 2006 ABS information.
AdminGeocode	Data	Addresses	Australian addresses and detailed ABS information including administrative electoral boundaries.

ABSGeocode

This data-specific grammar parses an Australian address and provides Australian Bureau of Statistics (ABS) ASGC 2006 statistical boundary information.

Expected Input

A free form address, with address lines divided by optional new-line or tab character.

Use any of the following prefixes followed by a semicolon before the address:

`AmasRules;` Use stricter matching rules. This must be the first of the prefixed items.

`ARF=<ARFname>;` Specify the name of the Address Reference File (ARF) to use, e.g. `ARF=GNAF;`

`GIF=<GIFname>;` Specify the name of the Geographical Information File (GIF) to use for the MB, CD and SLA boundary information, e.g. `GIF=ABSGif;`

Output Fields

The field length is shown in brackets (n). The output fields are similar to [DpidGeocode](#) with the following extra fields:

latitude (12) The latitude component of the geographic coordinates of the address.

longitude (12) The longitude component of the geographic coordinates of the address.

reliability (1) An indication of the [Geographical coordinate accuracy](#)

MB (11) Mesh Block – the smallest geographic region used by the Australian Bureau of Statistics (ABS).

CD (7) Collection District – the second smallest ABS geographical area, represented by a unique seven digit code. These are the smallest Census collection and processing units, which serve as building blocks for aggregating statistics info to larger Census geographic areas.

SLA (9) Statistical Local Area – an ABS area comprising all or part of a Local Government Area, consisting of one or more Collection Districts (CDs).

gifFlags (10) A representation of how the geographical information was associated with the address, see [GIF Flags](#).

Customising

To customise this grammar, edit the following DEFINE parameters (see [Customising with DEFINES](#)).

- `sARF_NAME` – specify the name of the ARF to be used if none is specified in the input, e.g. `\DEFINE sARF_NAME GNAF.`

- `sGIF_NAME` – specify the name of the GIF to be used (for MB, CD & SLA) if none is specified in the input, set to the name of the GIF, e.g. `\DEFINE sGIF_NAME ABSGif`

AdminGeocode

This data-specific grammar parses an Australian address and provides detailed Australian Bureau of Statistics (ABS) statistical information including administrative electoral boundaries.

Expected Input

A free form address, with address lines divided by optional new-line or tab character.

Use any of the following prefixes followed by a semicolon:

- `sARF=<ARFname>;` Specify the name of the Address Reference File (ARF) to use, e.g.
`sARF=GNAF;`
- `sGIF=<GIFname>;` Specify the name of the Geographical Information File (GIF) to use for the MB, CD and SLA boundary information, e.g. `sGIF=ABSGif;`

Output Fields

The field length is shown in brackets (n). The output fields are similar to [DpidGeocode](#) with the following extra fields:

latitude (12)	The latitude component of the geographic coordinates of the address.
longitude (12)	The longitude component of the geographic coordinates of the address.
reliability (1)	An indication of the Geographical coordinate accuracy
gifFlags (10)	A representation of how the geographical information was associated with the address, see GIF Flags .
MB (11)	Mesh Block – the smallest geographic region used by the Australian Bureau of Statistics (ABS).
CD (7)	Collection District – the second smallest ABS geographical area, represented by a unique seven digit code. These are the smallest Census collection and processing units, which serve as building blocks for aggregating statistics info to larger Census geographic areas.
SLA (9)	Statistical Local Area – an ABS area comprising all or part of a Local Government Area, consisting of one or more Collection Districts (CDs).
SLA_Name (20)	Statistical Local Area name
LGA_PID (6)	Local Government Area code
LGA_Name	Local Government Area name
SE_PID (6)	State Electoral Division (SED) code. An ABS geographical region.
SE_Name (20)	State Electoral Division (SED) name.
CE_PID (6)	Commonwealth Electoral Divisions (CED) code. An ABS geographical region.
CE_Name (20)	Commonwealth Electoral Divisions (CED) name.

Customising

To customise this grammar, edit the following DEFINE parameters (see [Customising with DEFINES](#)).

- `sARF_NAME` – specify the name of the ARF to be used if none is specified in the input, e.g.
`\DEFINE sARF_NAME GNAF`

- `sGIF_NAME` – specify the name of the GIF to be used (for MB, CD & SLA) if none is specified in the input, set to the name of the GIF, e.g. `\DEFINE sGIF_NAME ABSGif`

Country-specific Grammars

Summary

Country-specific grammar files specific to the IQ Office licence are summarised below.

Grammar	Category	Data Type	Data Processed
NzGeocode	Country	Addresses	Replaced by NzAddress
NzAddress	Country	Addresses	New Zealand address details.
NZDefault	Country	Addresses	New Zealand address summary.
NZValidate	Country	Addresses	Validate New Zealand addresses.
NzAddress TwoArfsCNAR	Country	Addresses	New Zealand address details incorporating both NZ PAF and C-NAR data

NzAddress

This country-specific grammar parses a New Zealand address against an Address Reference File (ARF) and also returns the geographical coordinates of the address.

Expected Input

A free form address, with address lines divided by optional new-line or tab character.

To use a different ARF from the default `NZPaF` use the following prefixes followed by a semicolon:

ARF=<ARFname>; Specify the name of the Address Reference File (ARF) to use, e.g.
ARF=NZNewPaF;

Output Fields

The field length is shown in brackets (n).

city (30)	Mail town or city.
postcode (4)	4-digit post code.
suburb (30)	Name of suburb (for an urban address).
streetName (30)	Name of the street (for a street address).
streetType (12)	Street type abbreviation, e.g. RD, ST, AVE
streetSuffix (10)	Street suffix abbreviation, e.g. W in “Circular Quay West”
postalType (16)	For a postal address, the postal type, e.g. PO BOX
addressId (8)	An address Id associated with the address returned
houseNo1 (6)	First house number, e.g. 5 in “5 Main St”, or in “5-7 Main St
houseSuffix1 (1)	House number suffix, e.g. A in “5A Main St”
houseNo2 (6)	Not used.
houseSuffix2 (1)	Not used.
flatType (10)	Flat or unit type abbreviation, e.g. U in “Unit 5 / 10 Railway St”
flatNo (10)	Flat or unit number, e.g. 5 in “Unit 5 / 10 Railway St”
levelType (15)	Building level type abbreviation, e.g. L in “Level 4”, or G in “Ground Floor”

levelNo (5)	Building level number, e.g. 4 in “Level 4”
buildingName1 (50)	Name of the building. Lobby name for postal addresses.
buildingName2 (30)	Not used
lotNo (6)	Not used
postalNo (6)	Postal number, e.g. 123 in “PO Box X123YY”
postalPrefix (3)	Postal number prefix, e.g. X in “PO Box X123YY”
postalSuffix (3)	Postal number suffix, e.g. YY in “PO Box X123YY”
flag (10)	A Flag code indicating if the address was correct, needed to be corrected, or why no AddressId was found.
flagString (160)	The flag and amendedFlag fields in words.
preAddress1 (50)	Information that appears in the input before the address components.
preAddress2 (50)	Second line of the above.
address1 (50)	First line of the address, formatted into a 3-line address.
address2 (50)	Second line of the above (if there is a second line).
address3 (40)	Third line of the above (if there is a third line).
postAddress (50)	Information that appears in the input after the address components.
amendedFlag (10)	Flag indicating which if any part of the address was corrected, see Amended Flag code The flagString field holds this flag in words.
latitude (12)	The latitude component of the geographic coordinates of the address.
longitude (12)	The longitude component of the geographic coordinates of the address.
reliability (1)	An indication of Geographical coordinate accuracy
MB (11)	The associated Mesh Block or aggregated mesh block value.
rdNumber (6)	The Rural Delivery (RD) number associated with the address
gifFlags (10)	A representation of how the Geographical information (geo-coordinates, the Mesh Block and RD number) was associated with the address, see GIF Flags .
validFlag (10)	The Statement of Accuracy (SOA) flag that can be used for SendRight lodgement based on the input address: VALID_U (uniquely valid), VALID_B (valid to base address) or INVALID
validFlagCorrected (10)	The SOA flag that can be used for SendRight lodgement based on the output (corrected) address. One of the above validFlag values, or blank if address not corrected.

Customising

To customise this grammar, edit the following DEFINE parameters (see [Customising with DEFINES](#)).

- SARF_NAME – specify the name of the ARF to be used if none is specified in the input, e.g. \DEFINE SARF_NAME NZNewPaf.

NZValidate

This country-specific grammar validates a New Zealand address against an Address Reference File (ARF) and returns a flag indicating the address validity. It is only used by [NzAddress](#).

Expected Input

A free form address, with address lines divided by optional new-line or tab character.

Use any of the following prefixes followed by a semicolon before the address:

ARF=<ARFname>; Specify the name of the Address Reference File (ARF) to use, e.g.
ARF=NZNewPaf;

Lenient; Use less strict matching rules.

GIF=<GIFname>; Specify the name of the Geographical Information File (GIF) to use, e.g.
GIF=NZGif;

Output Fields

The field length is shown in brackets (n).

validFlag (10) Same as [NzAddress](#) validFlag field. The SOA flag that can be used for SendRight lodgement based on the input address: VALID_U (uniquely valid), VALID_B (valid to base address), INVALID or ERROR.

NZDefault

This country-specific grammar parses a New Zealand address against an Address Reference File (ARF) and returns summary address information. It is normally only used by IQ Easy Post.

Expected Input

A free form address, with address lines divided by optional new-line or tab character.

To use a different ARF from the default NZPaf use the following prefixes followed by a semicolon:

ARF=<ARFname>; Specify the name of the Address Reference File (ARF) to use, e.g.
ARF=NZNewPaf;

Output Fields

The field length is shown in brackets (n). These fields are a subset of [NzAddress](#).

preAddress1 (50) Information that appears in the input before the address components.

address1 (50) First line of the address, formatted into a 3-line address.

address2 (50) Second line of the above (if there is a second line).

address3 (40) Third line of the above (if there is a third line).

suburb (30) Name of suburb (for an urban address).

city (30) Mail town or city.

postcode (4) 4-digit post code.

flag (10) A [Flag code](#) indicating if the address was correct, needed to be corrected, or why no AddressId was found.

validFlag (10) The Statement of Accuracy (SOA) flag that can be used for SendRight lodgement based on the input address: VALID_U (uniquely valid), VALID_B (valid to base address) or INVALID

Examples

	1 Panui Koutu Rotorua 3010	1 Panui St Koutu 3010	PO Box 1 Timaru Timaru 7940	Box 1 Timar
address1	1 Panui Road	1 Panui Road	PO BOX 1	PO BOX 1

address2				
suburb	KOUTU	KOUTU	TIMARU	TIMARU
city	ROTORUA	ROTORUA	TIMARU	TIMARU
postcode	3010	3010	7940	7940
flag	0	4	0	4
validFlag	VALID_U	INVALID	VALID_U	INVALID

NzAddressTwoArfsCNAR

This country-specific grammar parses a New Zealand address against both the New Zealand Postal Address File and the Critchlow C-NAR Address Reference File (ARF) returning the best match address from either dataset including the geographical coordinates of the address and other boundary attributes..

Expected Input

A free form address, with address lines divided by optional new-line or tab character.

Output Fields

The field length is shown in brackets (n).

city (30)	Mail town or city.
postcode (4)	4-digit post code.
suburb (30)	Name of suburb (for an urban address).
streetName (30)	Name of the street (for a street address).
streetType (12)	Street type abbreviation, e.g. RD, ST, AVE
streetSuffix (10)	Street suffix abbreviation, e.g. W in “Circular Quay West”
postalType (16)	For a postal address, the postal type, e.g. PO BOX
addressId (8)	An address Id associated with the address returned
houseNo1 (6)	First house number, e.g. 5 in “5 Main St”, or in “5-7 Main St
houseSuffix1 (1)	House number suffix, e.g. A in “5A Main St”
houseNo2 (6)	Not used.
houseSuffix2 (1)	Not used.
flatType (10)	Flat or unit type abbreviation, e.g. U in “Unit 5 / 10 Railway St”
flatNo (10)	Flat or unit number, e.g. 5 in “Unit 5 / 10 Railway St”
levelType (15)	Building level type abbreviation, e.g. L in “Level 4”, or G in “Ground Floor”
levelNo (5)	Building level number, e.g. 4 in “Level 4”
buildingName1 (50)	Name of the building. Lobby name for postal addresses.
buildingName2 (30)	Not used
lotNo (6)	Not used
postalNo (6)	Postal number, e.g. 123 in “PO Box X123YY”
postalPrefix (3)	Postal number prefix, e.g. X in “PO Box X123YY”
postalSuffix (3)	Postal number suffix, e.g. YY in “PO Box X123YY”

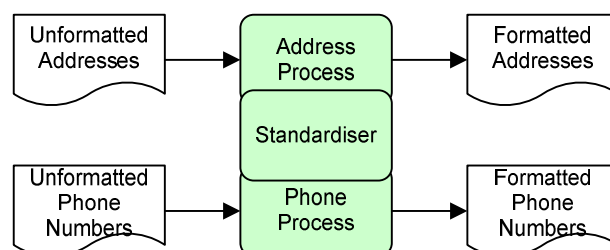
flag (10)	A Flag code indicating if the address was correct, needed to be corrected, or why no AddressId was found.
flagString (160)	The <code>flag</code> and <code>amendedFlag</code> fields in words.
preAddress1 (50)	Information that appears in the input before the address components.
preAddress2 (50)	Second line of the above.
address1 (50)	First line of the address, formatted into a 3-line address.
address2 (50)	Second line of the above (if there is a second line).
address3 (40)	Third line of the above (if there is a third line).
postAddress (50)	Information that appears in the input after the address components.
amendedFlag (10)	Flag indicating which if any part of the address was corrected, see Amended Flag code The <code>flagString</code> field holds this flag in words.
rdNumber (6)	The Rural Delivery (RD) number associated with the address
validFlag (10)	The Statement of Accuracy (SOA) flag that can be used for SendRight lodgement based on the input address: <code>VALID_U</code> (uniquely valid), <code>VALID_B</code> (valid to base address) or <code>INVALID</code>
validFlagCorrected (10)	The SOA flag that can be used for SendRight lodgement based on the output (corrected) address. One of the above <code>validFlag</code> values, or blank if address not corrected.
gifFlags (10)	A representation of how the Geographical information (geo-coordinates, the Mesh Block and RD number) was associated with the address, see GIF Flags .
latitude (12)	The latitude component of the geographic coordinates of the address.
longitude (12)	The longitude component of the geographic coordinates of the address.
reliability (1)	An indication of Geographical coordinate accuracy .
MB (20)	The associated Mesh Block or aggregated mesh block value.
Unused (20)	Will be returned as blank.
ParcelId (20)	Land Parcel Identifier
Dpid (20)	The New Zealand Post Delivery Point Identifier.
CNAR_ID (20)	The Critchlow C-NAR Id.
DEP_INDEX (20)	Deprivation rating against the official Mesh Bloch pattern.
QUINTILE (20)	Quintile version of deprivation rating.
DOMICILE (20)	Domcile code, commonly used by clients in a wider health sector
AREA_UNIT (20)	Statistical area (grouping of Mesh Blocks)
Source (20)	An Indication as to which ARF dataset had the better match “NZPaf” or “NZCNAR”
SourceLatLong (20)	An Indication as to which dataset produced the LatLong output “NZPaf” or “NZCNAR”

IQ Office Grammar File SDK

Overview

Standardising is the process of converting unformatted text data such as addresses, names and telephone numbers into data in a *standard format* performed by IQ Office Standardiser, a suite of applications and library functions supplied with IQ Office DTS and Enterprise editions. This typically cleanses, transforms, enhances and validates the input data, providing greatly improved and high quality output data.

Standardising is performed by one or more standardising *processes*. Each process takes one type of alphanumeric input data such as addresses, names or telephone numbers, *parses* the input data to break it into components, and returns each required component in a specified format.

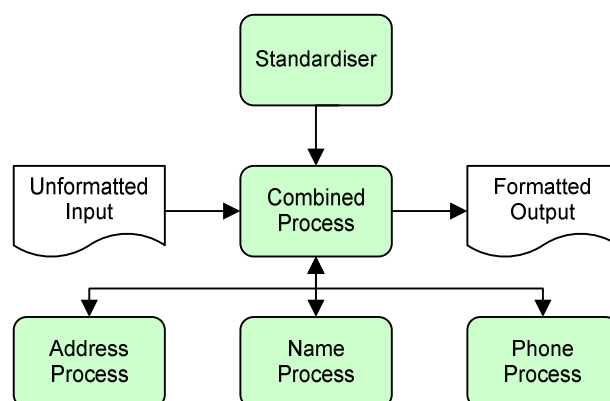


Each standardising process has a *grammar file*, which defines how the data will be standardised and formatted. The Standardiser reads and interprets the grammar file before performing the defined standardising processes.

Grammar files are text files written in a simple programming language. They contain definitions of the number, type and format of input data fields; instructions on how Standardiser is to process input data; references to lookup-lists such as standard abbreviations; and definitions of the number, type and format of output data fields.

Existing grammar files can be easily edited to *customise* them to the specific requirements of an organisation, for example, to process address data received in a variety of formats, and output data in the specific format used by a particular database or automated mailing system. New files can also be written to fulfil the needs of new business processes, for example, if an organisation installs a new account management system.

Grammar files can contain *calls* to other grammar files, for example, the “Name, Address and Phone” grammar file would call three separate grammar files and combine the returned output.



Grammar files can also contain calls to other processes, data sources, libraries and *address reference files*. For example, the *DpidGeocode* grammar file supplied with IQ Office Standardiser uses the IQ Office PAF Libraries to look up an address in the Postal Address File (PAF) containing all Australian postal addresses supplied by Australia Post, and returns the formatted output and DPID.

About this manual

This manual contains detailed instructions on how to create, edit and run grammar files. It should be read by system developers and administrators who are responsible for developing and managing the systems and processes used to standardise data using the IQ Office suite of applications.

This manual contains the following major sections:

- [Grammar file structure](#) – details each section in a grammar file
- [Parsing](#) – describes how input into a grammar file is parsed, including the token definitions which define how the input is parsed, and output actions performed to extract the parsed data
- [Commands](#) – lists commands which use variables defined in the grammar file to manipulate the input text and output fields
- A list of the defined [Limits](#) imposed on grammar files, and [Error messages](#) which may be returned when an error has occurred.

Refer to the following documentation for details of other IQ Office products.

The [Grammar File Reference Manual](#) describes the various grammar files supplied with different IQ Office editions and data licences.

The [IQ Standardiser SDK Reference Manual](#) provides a complete reference to:

- The Stan Libraries which provide the standardising engine, and are available as a Windows 32 or 64 bit API DLL or C function library which can be integrated into other systems.
- The Standardiser Real Time (StanRt) engine exposed as a COM / DCOM object which can run as a Windows service.
- Java, Socket and SOAP interfaces to the Stan Library functions.
- The Standardiser Batch COM object library which uses StanRt to standardise a batch of records.
- The StanConsole application which can standardise data stored in text files and be used to debug grammar files.

A simple grammar file

To get a quick understanding of how the grammar file works, examine the following simple example and explanation.

Example

```
# Parse first and last name
\OUTPUT_FIELDS DELIMITED ","
20         first_name
30         last_name

\GRAMMAR
Name       : WORD
FullName   : Name(first_name)  Name(last_name)
```

Explanation

This grammar will take the first two words of the input, and place them in the output, separated by a comma.

So “John Smith” will produce:

“John,Smith”.

The first line is just a comment, as is any line beginning with a hash (“#”).

The next lines define the output fields, in this case, two of them, of maximum length 20 and 30 characters respectively.

The `\GRAMMAR` section defines what to look for in the input.

First we look for every “Name”, which is a word (a continuous sequence of letters).

Then we look for a “FullName”, which is two names. The first name we put in the “first_name” of the output, and the second name we put in the “last_name” of the output.

Tips

The tips below should be used when writing grammar files. [Debugging with StanConsole](#) can be used to test and debug grammar files.

- Too much nesting can cause overflow and/or slow down
- Take care with priorities, use search from back when needed, and order tokens when using [Any order](#).
- How to handle foreign words with “çěà” etc.
- Make full use of separators.
- Beware of directly outputting from a [Composed](#) type.
- Allowing junk in between is slower.
- Using *Must be first* or *Must be last* is quicker, unless *In any order*
- [Priorities - optional token](#)
- [Priorities - first come first serve](#)

Priorities - optional token

If a token is optional, then a fit including the optional token will be given priority over a fit without the token.

Example

```
\TABLE ranks
JR
\GRAMMAR
Word          : WORD
Rank          : TABLE ranks
Name          : Word Rank?
```

Assume an input of “JAMES JR”.

Both “JAMES JR” and “JAMES” will fit the *Name* token, but the priority will be given to “JAMES JR”, as it includes the optional token.

Priorities - first come first serve

If there are several occurrences of a token, the first one found will always be given priority.

Example 1

```
\TABLE Localities
SYDNEY
MELBOURNE

\GRAMMAR
PostCode      : NUMBER
Locality      : TABLE Localities
Address       : PostCode Locality ~
```

Note the *Any order qualifier* (~).

Here, an input of “SYDNEY 2000” will fit the *Address* token.

However, with an input of “GPO 1234 SYDNEY 2000”, both “1234 SYDNEY” and “SYDNEY 2000” will fit the *Address* token, and “1234 SYDNEY” will be given priority, as it was found first.

To remedy this, we could look for the *PostCode* from the end, so that “2000” will be given priority over “1234”, changing the line to:

```
PostCode          : NUMBER !
```

Example 2

```
\TABLE streetTypes
ST
AVE

\TABLE directions
EAST=E
WEST=W
NORTH=N
SOUTH=S

\GRAMMAR
StreetType      : TABLE streetTypes
Direction       : TABLE directions
StreetName      : WORD
Suburb          : WORD
FullStreet      : StreetName StreetType Direction?
FullSuburb      : Direction? Suburb
Address         : FullStreet FullSuburb
```

Note the *Optional qualifier* (?) for direction.

Now take an input of “GEORGE ST EAST SYDNEY”.

For the token *FullStreet*, two fits will be found, i.e. “GEORGE ST EAST” and “GEORGE ST”.

Similarly, for the token *FullSuburb*, two fits will be found, i.e. “EAST SYDNEY” and “SYDNEY”.

Now, when looking for the *Address*, “EAST” will be given to *FullStreet* (and not to *FullSuburb*) as it appears first in the definition of *Address*. So it will be “GEORGE ST EAST”, “SYDNEY”.

If we wanted the direction to be given to *FullSuburb*, then we could use the [Any order](#) qualifier as follows:

```
Address          : FullSuburb FullStreet ~
```

Since *FullSuburb* comes before *FullStreet*, “EAST” will be given to it.

The drawback with this is that if the suburb comes before the street, it will also fit. e.g. “SYDNEY GEORGE ST”. To prevent this, we can use [Order digits](#) as follows:

```
Address          : FullSuburb FullStreet ~21
```

Debugging with StanConsole

StanConsole is a tool that can be used to debug grammar files, in particular for displaying the tokens found. [StanConsole](#) is described in detail in the IQ Standardiser SDK Reference Manual.

Calling StanConsole

StanConsole is a command line application. Call it with the “-t” option to display the found tokens. The “-n” option is also useful - it will display the output fields, e.g.

```
stanconsole -tn MyGrammar.grm
```

Type in the input and StanConsole will display all found tokens.

The example below illustrates the output of StanConsole using an example grammar file and example input record.

Example Grammar File

```
\TABLE StreetTypeTable
STREET
ROAD
AVENUE

\GRAMMAR
StreetType      : TABLE StreetTypeTable
StreetSuffix    : "EAST"
                : "WEST"
Word            : WORD
Words           : Word Word Word?
                : Word
StreetName      : Words StreetType StreetSuffix?
```

Screen dump

```
>Stanconsole -tn MyGrammar.grm

Enter record: MAIN STREET EAST

0: StreetType (5-11) STREET
1: StreetSuffix.1 (12-16) EAST
2: Word (0-4) MAIN
3: Word (5-11) STREET
4: Word (12-16) EAST
5: Words.1 (0-16) MAIN STREET EAST[2,3,4]
6: Words.1 (0-11) MAIN STREET[2,3,]
7: Words.1 (5-16) STREET EAST[3,4,]
8: Words.2 (0-4) MAIN[2]
9: Words.2 (5-11) STREET[3]
10: Words.2 (12-16) EAST[4]
11: StreetName (0-16) MAIN STREET EAST[8,0,1]
```

Explanation

Stanconsole -tn MyGrammar.grm

The application is called with the -t and -n options and the grammar file name.

Enter record: MAIN STREET EAST

The input line entered in our example is “MAIN STREET EAST”.

0: StreetType (5-11) STREET

Each token found is displayed on a separate line. The tokens found are numbered starting from zero.

Following the found-token-number is the name of the token found – StreetType.

Then comes the position in the input string that the token is found – in this case, from characters 5 to 11.

Then comes the text of the token found – STREET.

1: StreetSuffix.1 (12-16) EAST

If a token has more than one definition, then the definition number follows. For example, *StreetSuffix* has two definitions, “EAST” & “WEST”. The found token “EAST” was the first definition, so the name of the token has a 1 after it.

```
5: Words.1 (0-16) MAIN STREET EAST[2,3,4]
```

For a composite token, the sub-tokens are displayed in square brackets. This found *Word* token is made up of 3 found sub-tokens, numbers 2, 3 and 4.

```
6: Words.1 (0-11) MAIN STREET[2,3,]
```

Here, there are only two found sub-tokens (2 & 3), the third sub-token being optional.

```
11: StreetName (0-16) MAIN STREET EAST[8,0,1]
```

To understand how the grammar finally parsed the input, you need to start with the last line and work up. *StreetName* was parsed as three tokens – 8,0 and 1. 8 is a *Words* token made up of one *Word* token 2. So the input was parsed as follows:

- 11. *StreetName*
- 8. *Words*
 - 2. *Word* – MAIN
- 0. *StreetType* – STREET
- 1. *StreetSuffix* – EAST

Grammar file structure

The grammar file is divided into sections, each with a header to mark the new section. The section ends where the next section begins.

Most sections are optional.

The first three sections must come first (if they are present). Besides this, sections may appear in any order, except that references can only be made to something defined in a previous section.

The sections are:

- [Description](#) – just for informative purposes.
- [Suggested divider](#) – a message to the user of the process.
- [Output fields](#) – the defined structure of the standardised data – the output of this process.
- [Tables](#) – lists of token strings (words) to recognise or convert.
- [Grammar definitions](#) – the structure of the expected input text – how to parse the text.
- [Pre and Post-parse commands](#) – to be done before or after the text is parsed.
- [Initialise and Finalise commands](#) – to be done once only.
- [Variables](#) – variables used in the commands.
- [Dynamic Linked Library \(DLL\) definitions](#) – functions of external DLLs that are to be used.
- [Sub Grammars](#) – other grammars called by this one.
- [Global Options](#) – some rules that will apply to the whole file or to the rest of the file.

[General syntax rules](#) apply to all sections of the file.

Description

This section provides a description of the process defined in the grammar file.

Syntax

```
\DESCRIPTION
  Description
```

Notes

Each line after this header line, until the next section will be considered as a description.

This description will allow the standardiser to provide the user with some information about this process.

It will not be used in the standardising itself.

Example

```
\DESCRIPTION
This process will parse names.
It will recognise up to 2 first names.
```

Suggested divider

This section suggests to the user of this grammar file the character to use to divide several input fields.

Syntax

```
\SUGGESTED_DIVIDER Divider
```

Syntax description

Divider The character to suggest to be used as the input field divider. Place the character in double quotes. [Escape sequences](#) may be used. Alternatively, the ACSII number may just be used.

Notes

Some processes might expect several fields as an input, for example several address lines. The grammar file might be written to recognise the separate address lines broken up by a certain character, such as a comma or new line character. Use this section to suggest the character to place between each input field to best make use of the grammar file.

Example 1

```
\SUGGESTED_DIVIDER  ", "
```

Example 2

The new line character:

```
\SUGGESTED_DIVIDER  "\n"
```

Example 3

The tab character, or use "\t"

```
\SUGGESTED_DIVIDER  9
```

Output fields

This mandatory section defines the output of this standardising process, i.e. the structure of the standardised data.

Syntax

```
\OUTPUT_FIELDS [ DELIMITED delimiter [qualifier] | XML ]
  Length Name Description
```

Syntax description

Delimiter The character to delimit the output fields. This is either a character in double quotes including Escape sequences, or the ASCII character number.

<i>Qualifier</i>	Optional. The character to qualify the output fields, such as single or double quotes. This is either a character in double quotes including Escape sequences, or the ASCII character number.
<i>Length</i>	The size in characters of the output field (maximum size for delimited or XML output)
<i>Name</i>	The name of the output field
<i>Description</i>	Optional. A description of this output field.

Several output fields may (and usually are) defined, each one on a separate line.

Notes

The output of the standardiser is a string with fields in either fixed positions in the string, or delimited by the given delimiter, or in XML format.

If neither DELIMITED nor XML is specified the output will be fixed-length.

The output field names used here are made reference to later in the grammar file.

For a fixed width output, the positions of the output fields are defined in this section. Otherwise, the maximum length of each field is defined.

The test client (StanRtClient.exe) does not support the XML format.

Example 1

```
\OUTPUT_FIELDS
46      locality
4       postcode
3       state
```

This defines the output of the standardiser as a string of length 53. The *locality* will be position 1-46, the *postcode* in position 47-50 and the *state* in position 51-53.

Example 2

```
\OUTPUT_FIELDS DELIMITED "," "' '
46      locality
4       postcode
3       state
```

This defines the output of the standardiser as three fields (locality, postcode and state), separated by a comma. Each field will be enclosed by quotes. If a quote appears in a field, it will be doubled. The maximum sizes of each field are 46, 4 and 3 respectively.

Example 3

```
\OUTPUT_FIELDS DELIMITED 9
46      locality      Suburb name
4       postcode      4-digit post code
3       state3-letter state abbreviation
```

This defines the output of the standardiser as three fields (locality, postcode and state), separated by a tab character (ASCII 9). The maximum sizes of each field are 46, 4 and 3 respectively. Each field also has a simple description.

Example 3

```
\OUTPUT_FIELDS XML
46      locality
4       postcode
3       state
```

This defines the output of the standardiser as three fields (locality, postcode and state), in XML format. The maximum sizes of each field are 46, 4 and 3 respectively.

Tables

This section defines a table of values, which can be used later to find key words in the input, or as a table of conversions.

There can be any number of tables in a grammar file, or none at all. Each table definition has one header line followed by any number of table entries comprising either values, or value/abbreviation pairs.

Syntax

```
\TABLE  name  options
        value=abbreviation
        value2=abbreviation2
        ...
```

Syntax description

<i>name</i>	Table name
<i>options</i>	Optional. See below.
<i>value</i>	Text to look for.
<i>abbreviation</i>	Optional. Text to replace the found value.

Notes

The table is a list of values, which each value mapped to an optional second value or abbreviation. If no second value is defined, there must be no equals sign, and the value itself is the second value.

Tables can be used in parsing to find key words, see [TABLE](#) and [TABLE SINGLE WORDS](#) tokens. Tables can also be used in the commands to look up values, see the [LOOKUP](#) command.

Example

```
\TABLE  States  CASE_INSENSITIVE
NSW
VIC
NEW SOUTH WALES=NSW
VICTORIA=VIC
```

This table can be used to find state names in addresses, and to replace the state with its abbreviation.

The second line NSW is equivalent to NSW=NSW.

Options

Any of the following options may be specified in the header line of a table definition:

- CASE_INSENSITIVE or CASE_SENSITIVE (see [Table Case Sensitivity](#))
- ACCENT_INSENSITIVE or ACCENT_SENSITIVE (see [Table Case Sensitivity](#))
- NO_DUPLICATES, ALLOW_DUPLICATES, IGNORE_DUPLICATES, OVERWRITE_DUPLICATES or RANDOM_DUPLICATES (see [Duplicates in a table](#))
- FIXED_LENGTH (see [Fixed length table entries](#))
- RESET and/or IFUSED (see [Duplicate table definitions](#)).

Table Case Sensitivity

On the header line of a table definitions (see [Tables](#)), one may specify one or two (or none) of the following options.

If an option is not set, then the default value is used, as explained below. If an option is set, then it overrides the default.

CASE_SENSITIVE

When finding values in a table with the [TABLE](#) token, [TABLE_SINGLE_WORDS](#) token or [LOOKUP](#) command, searching will be case-sensitive. By default, tables are case-sensitive, unless set otherwise in [Global Option - Table Case Sensitivity](#).

CASE_INSENSITIVE

When finding values in a table with the **TABLE** token, **TABLE_SINGLE_WORDS** token or **LOOKUP** command, searching will be case-insensitive. This setting will also convert all values in the table to upper case, although not the abbreviations.

ACCENT_SENSITIVE

When finding values in a table with the **TABLE** token, **TABLE_SINGLE_WORDS** token or **LOOKUP** command, no specific consideration will be given to accented characters. So “café” will not match “caf  ”. By default, tables are accent-sensitive.

ACCENT_INSENSITIVE

When finding values in a table with the **TABLE** token, **TABLE_SINGLE_WORDS** token or **LOOKUP** command, accented characters will be matched to their equivalent non-accented characters. This option is designed to work with the Latin-1 (ISO 8859-1) character encoding. It will convert any of          to A,    to C,        to E,      to I,    to N,          to O,        to U and    to Y. The same applies for the lower case of these characters. This setting will also remove all accents from the values in the table, although not from the abbreviations. So an entry in the table “caf  ” will become “cafe”, but “caf  =caf  ” will become “cafe=caf  ”.

Example

```
\TABLE Table1 CASE_INSENSITIVE ACCENT_INSENSITIVE
Caf  =Caf  
R  sum  
```

This will find any of CAF  , CAFE, caf  , cafe, etc. and return caf  . It will find any of r  sum  , resume, resum  , R  SUM   etc, and will return RESUME.

Duplicates in a table

Entries in a table are value-abbreviation pairs (see [Tables](#)). When searching a given table, only one abbreviation is returned for a given value. This means that the values in a given table must be unique. If duplicate values occur in a table definition, one of the values must be discarded to keep the values unique. Via one of the options described here, one can specify how to handle duplicate entries in a table definition.

If no option is specified, then the default value is used, as set in [Global Option - Duplicate Table Entries](#). If an option is specified, then it overrides the default.

ALLOW_DUPLICATES

Allow duplicate values in a table definition, providing that the abbreviation is the same. If a value is defined twice with the same abbreviation, it will only be stored once in the table. This is the default option, unless set otherwise in [Global Option - Duplicate Table Entries](#). If duplicate values occur with differing abbreviations, an error 52 (ERR_GRAMMAR_DUPLICATES_IN_TABLE) will occur.

NO_DUPLICATES

Don’t allow duplicate values in a table definition. If duplicates occur, an error 52 (ERR_GRAMMAR_DUPLICATES_IN_TABLE) will occur.

IGNORE_DUPLICATES

Allow duplicate values in the table, even if the abbreviations differ. If two entries have different abbreviations, the first abbreviation will be used, and subsequent entries with the same value will be discarded.

OVERWRITE_DUPLICATES

Allow duplicate values in the table, even if the abbreviations differ. If two entries have different abbreviations, the last abbreviation will be used, and previous entries with the same value will be discarded.

RANDOM_DUPLICATES

Allow duplicate values in the table, even if the abbreviations differ. If two entries have different abbreviations, one abbreviation will be chosen and the others discarded.

Example

```
\TABLE Table1 OVERWRITE_DUPLICATES
MT=MOUNT
MT=MOUNTAIN
PT=POINT
```

In this table definition, we have a duplicate value (MT) with differing abbreviations (MOUNT and MOUNTAIN). Because OVERWRITE_DUPLICATES is specified, the last entry (MT=MOUNTAIN) will be kept in the table and previous ones (MT=MOUT) will be discarded. If IGNORE_DUPLICATES is specified instead, then the first entry (MT=MOUNT) is kept in the table and subsequent ones (MT=MOUNTAIN) discarded.

Fixed length table entries

Entries in [Tables](#) are value/abbreviation pairs which are usually defined in the file using an equal sign. i.e.

```
value=abbreviation
```

Entries in tables can also be in fixed length format, where the first \times number of characters are the value and the remaining characters are the abbreviation. Use the FIXED_LENGTH option on the table definition header line followed by a number (\times) to define fixed length table entries.

Example

```
\TABLE Table1 FIXED_LENGTH 4
2000NSW
3000VIC
```

Equivalent to

```
\TABLE Table1
2000=NSW
3000=VIC
```

Duplicate table definitions

[Tables](#) will be merged to form a single table if they are redefined, that is, the same name is used for two or more table definitions. This is useful when [Including other files](#) to add entries to a table. Note that duplicate table definitions may be disallowed by the [Global Option - Duplicate Tables](#).

RESET

Replace any previous definition of a table with the same name with this table.

IFUSED

Only consider this table if it is a redefinition of a table of the same name that has already been defined. Ignores a table that hasn't already been previously defined. If a table of this name has not yet been defined, this current table definition will be ignored. This option can be used to prevent loading a large table when it is not needed.

Example

```
\TABLE Table1 RESET
```

Grammar definitions

This section defines the [Parsing](#) of the input text. It is usually the heart of the grammar file, and there is only one grammar definition section (if any).

Syntax

```
\GRAMMAR
```

```
token           :   token-definition
                  {commands}
                  :   token-definition
                  {commands}
```

Syntax description

token Name of a token looking for when parsing

token-definition Definition of the token looking for.

commands Optional. Actions to be performed if token is found.

The `\GRAMMAR` heading begins the grammar definition section, and occurs only once. Several tokens may be defined afterwards. Each token may have several [Token definitions](#), each on a separate line beginning with a semicolon. The [Commands](#) are optional. Several commands may be defined for one token, each on a different line, and not on the same line as the token definition. An opening brace is placed before the first command and a closing brace after the last command. The braces may be placed on the same line as the command or the token definition.

Dynamic Linked Library (DLL) definitions

This section defines functions in dynamic linked libraries (DLLs) that can then be used in the commands in this grammar file.

Syntax

```
\DLL DllName LoadErrorAction
      FunctionName Alias ReturnValueOption
```

Syntax description

DllName The name of the DLL file. The full path can be included if the DLL is not in the same directory as the grammar file or the default system DLL directory. Place the DLL name in double quotes if it contains spaces. If this value is not an absolute path, then the DLL will be searched for firstly in the same directory as the grammar file, and then in the DLL search directories defined by the system.

LoadErrorAction Optional parameter defining error handling.

ONLOADERROR FAIL – grammar file will return [Error 97 - ERR GRAMMAR_DLL_CANNOT_LOAD](#) if the DLL cannot be loaded (default).

ONLOADERROR CONTINUE – grammar file does not return an error immediately. Error will be handled when a function defined in this DLL is called in the Commands section, see [DLL Error Specifications](#).

<i>FunctionName</i>	Name of function as it appears in the DLL export table. Several such functions may be defined for each DLL, each on a separate line.
<i>Alias</i>	Optional. How the function is to be known in this grammar file. If it is not provided, <i>FunctionName</i> will be used as the alias. This must be used if the function name itself is not a valid alias because it contains characters besides letter, numbers and the underscore, or if there is more than one function (in different DLLs) with the same name.
<i>ReturnValueOption</i>	Specifies the meaning of the return value of the function, see DLL return value options .

Notes

The functions defined here can be used in [Commands](#). See [DLL Functions](#).

Several such DLLs can be defined, each in a separate section beginning with \DLL.

Error can also be specified, see [DLL Error Specifications](#).

Example (Windows platform)

The following grammar file will return the system and windows directory.

```
\OUTPUT_FIELDS
260      SysDir
260      WinDir
\DLL Kernel32.dll
      GetSystemDirectoryA      ZERO_ERROR
      GetWindowsDirectoryA     ZERO_ERROR

\PRE_PARSE
{
      SysDir = GetSystemDirectoryA("")
      WinDir = GetWindowsDirectoryA("") }
```

Two functions, GetSystemDirectoryA and GetWindowsDirectoryA, are defined to be in Kernel32.dll. Each of these functions returns zero if an error occurs. The prototype of these functions are as required by the standardiser.

See also

[DLL function prototype](#)

[DLL Initialisation function](#)

[DLL Finalisation function](#)

[DLL return value options](#)

[DLL Error Specifications](#)

[DLL Error Actions](#)

[DLL Error Sources](#)

[DLL Example](#)

DLL function prototype

Functions that are defined in the DLL section must have the following prototype.

Syntax

```
int  function-name ( char *s, int maxsize )
```

Syntax description

<i>s</i>	A pointer to a zero-terminated string (LPTSTR). The function should take this passed string as input, and replace it with the result (its output). It should ensure that the result is also zero-terminated.
<i>maxsize</i>	The maximum number of characters that the string can hold (excluding the null-terminator). The function should respect this, and not write more characters to this string than maxsize.
Return value	The function is to return an integer value, indicating error state. See DLL return value options .

Notes

If the DLL function does not have this prototype unpredicted errors could occur, including crashing the Standardiser.

The method of passing parameters and cleaning up the stack is the stdcall convention.

The DLL functions must be multi-thread safe if they are to be used in a multi-threaded environment.

DLL Initialisation function

Optional. For each DLL defined in the \DLL section of the grammar file, an initialisation function may be specified. This initialisation function is specified on a separate line under the \DLL section.

Syntax

```
INITFUNC FunctionName LoadErrorAction ReturnValueOption ErrorAction
```

Syntax description

<i>FunctionName</i>	The name of the function in the DLL that is to be called just after the DLL is loaded.
<i>LoadErrorAction</i>	Optional parameter defining action taken if the function is not found in the DLL. ONLOADERROR FAIL – grammar file will return Error 97 - ERR_GRAMMAR_DLL_CANNOT_LOAD if the function does not exist in the DLL (default). ONLOADERROR CONTINUE – grammar file will not return an error and will continue loading.
<i>ReturnValueOption</i>	Specifies the meaning of the return value of the function, see DLL return value options .
<i>ErrorAction</i>	Specifies what to do if the function returns an error value. Only applicable if a <i>ReturnValueOption</i> is specified. FAIL – grammar file will return an Error 100 - ERR_STAN_DLL_CALL (default) CONTINUE – grammar file will not return an error.

In the error specifications of a function (not of the initialisation function), one can specify special actions to be taken in the event that the initialisation function wasn't found in the DLL, or if it returned an error.

Function Prototype

```
int function-name ()
```

The initialisation function takes no parameters, and returns an integer value, indicating the error state. See [DLL return value options](#).

Example

```

\DLL MyLibrary.dll
    INITFUNC MyInitFunction
    MyFunction

```

This example defines a DLL (*MyLibrary.dll*) having two functions. *MyFunction* can be used later in the Commands section of the grammar file. *MyInitFunction* is called just after the DLL is loaded.

DLL Finalisation function

Optional. For each DLL defined in the \DLL section of the grammar file, a finalisation function may be specified. This finalisation function is specified on a separate line under the \DLL section.

Syntax

```
FINALFUNC FunctionName LoadErrorAction
```

Syntax description

FunctionName The name of the function in the DLL that is to be called just before the DLL is unloaded.

LoadErrorAction Optional parameter defining action taken if the function is not found in the DLL.

ONLOADERROR FAIL – grammar file will return [Error 97 - ERR GRAMMAR_DLL_CANNOT_LOAD](#) if the function does not exist in the DLL (default).

ONLOADERROR CONTINUE – Standardiser will ignore this function and will continue loading.

Function Prototype

```
int function-name ()
```

The finalisation function takes no parameters, and returns an integer value, which is ignored by the standardiser.

DLL return value options

The integer value returned by a DLL function is to indicate whether the function call was successful or not. The error return value option may be one of the following:

NONZERO_ERROR	A non-zero value indicates an error, and zero indicates a successful call. This is the default.
ZERO_ERROR	Zero indicates an error, and a non-zero value indicates a successful call.
NO_ERROR	The return value is ignored. All calls to the function are considered to be successful.

How errors are handled is described in [DLL Error Specifications](#).

DLL Error Specifications

Using error specifications you can specify how to handle errors resulting from DLL function calls.

Syntax

```

\DLL DllName

    FunctionName Alias ErrorSource Action

```

Syntax description

DllName The name of the DLL file.

FunctionName The name of the function in the DLL.

<i>Alias</i>	Optional. How the function is to be known in this grammar file. If it is not provided, <i>FunctionName</i> will be used as the alias.
<i>ErrorSource</i>	One of the DLL Error Sources key words.
<i>Action</i>	One (or two) of the DLL Error Actions .

Notes

Any number of *ErrorSource Action* pairs may be specified.

The TO action may only be used with the error sources of ZERO_ERROR and NONZERO_ERROR, as the other error sources do not return an error code.

If the function return value is to be ignored, specify NO_ERROR on the line, see [DLL return value options](#).

One of ZERO_ERROR, NONZERO_ERROR may be specified without specifying an action. In this case, the *Action* assumed will be the default of FAIL, unless ONERROR is specified.

See [DLL Example](#).

DLL Error Actions

Four actions can be taken when an error occurs in calling a DLL function:

FAIL	The standardiser will return an error.
CONTINUE	The standardiser will ignore the error, no action will be taken.
TO <i>numvar</i>	The error code returned from the function call be placed in the specified numeric variable <i>numvar</i> (see Numeric variable definitions).
RETURN <i>literal</i>	The specified <i>literal</i> will be placed in the result text-operand (see DLL Functions). This literal can either be some text enclosed in double quotes, or the name of a constant (see Constants).

Notes

The TO action and the RETURN action can be used together. Otherwise, all actions are mutually exclusive.

DLL Error Sources

The different sources of error in calling a DLL function are:

ONLOADERROR	The DLL failed to load, or the function was not found in the DLL.
ONINITERROR	The initialisation function (if one was defined) returned an error.
ZERO_ERROR	The function returned zero, indicating an error.
NONZERO_ERROR	The function returned a non-zero value, indicating an error.
ONERROR	Any error.

DLL Example

```

\OUTPUT_FIELDS
100      out
\NUMERIC_VARIABLES
      iErr
\DLL MyLibrary.dll ONLOADERROR CONTINUE
      INITFUNC MyInitFunction NONZERO_ERROR CONTINUE
      MyFunction ONLOADERROR RETURN "LoadError" \
      ONINITERROR TO iErr RETURN "InitError" \
      NONZERO_ERROR TO iErr RETURN "FunctionError"

\PRE_PARSE

```

```
{          out = MyFunction("Input")      }
```

This example defines a DLL *MyLibrary.dll*. When the grammar file loads, it will load *MyLibrary.dll*, and call the function *MyInitFunction*. Then, in the Pre-Parse section, the function *MyFunction* is called, passing to it a buffer containing “Input”, and the integer 100, indicating that the buffer size for the return value is 100. The function *MyFunction* should take the parameter “Input”, process it, place a value into the provided buffer, and return zero. This value should not be more than 100 characters. This value is then placed into the output field *out*.

If *MyLibrary.dll* fails to load or if the function *MyFunction* is not found in the DLL, then “LoadError” will be placed into the output field *out*.

If *MyInitFunction* returns a non-zero value, then this value will be put into the numeric variable *iErr*, and “InitError” will be placed into the output field *out*.

If *MyFunction* returns a non-zero value, then this value will be put into the numeric variable *iErr*, and “FunctionError” will be placed into the output field *out*.

Sub Grammars

This section defines other grammar files that can then be used in the [Commands](#) in this grammar file, see [Sub Grammar Calls](#).

Syntax

```
\SUB_GRAMMARS
    alias file-name
```

Syntax description

<i>alias</i>	The name given to the sub-grammar file in current grammar file. This name must conform to naming rules (only letters, numbers and the underscore).
<i>file-name</i>	The name of the sub-grammar file, including absolute or relative path. If it is not provided, the alias will be used as the file name.

Notes

Several sub-grammars can be defined, each on a separate line under this heading.

If *file-name* is a relative path name (e.g. “MySubGram.grm”), then it will be searched for in the same directory as the original grammar. If it is not found there, it will be searched for in the current directory. If not found there, it will be searched for in the default grammar directory.

If a sub-grammar has itself a sub-grammar, then the directory of the original grammar is no longer used in searching for the second sub-grammar. The directory of the first sub-grammar, the current directory and the default grammar directory are used.

Initialise and Finalise commands

These sections contain commands that performed only once, unlike the [Pre and Post-parse commands](#) that are performed each time data is parsed/standardised.

```
\INITIALISE
```

This section contains commands that are performed only once when the grammar file is loaded or opened. There can be at most one such section. If an error occurs when executing these commands, the FINALISE commands are not executed.

```
\FINALISE
```

This section contains commands that are performed only once before the grammar file is unloaded or closed. There can be at most one such section.

[Global variable definitions](#) can only be assigned to in these sections. The lifetime of these variables is from the time the grammar file is opened until it is closed.

Syntax

```
\INITIALISE  
  
    { commands }  
  
\FINALISE  
  
    { commands }
```

Syntax description

Several commands can be defined in either section, each command on a different line, and not on the same line as the header. An opening brace is placed before the first command and a closing brace after the last command. The braces may be placed on the same line as the command, or the header.

Pre and Post-parse commands

Optional. This optional section contains [Commands](#) performed before and after the parsing.

There can be any number of pre and post-parse sections. If there are more than one, they will be executed one after the other.

Syntax

```
\PRE_PARSE  
  
    { commands }  
  
\POST_PARSE  
  
    { commands }
```

Syntax description

Several commands may be defined in either section, each command on a different line. An opening brace is placed before the first command and a closing brace after the last command. The braces may be placed on the same line as the command, or the header.

Variables

The five types of variables – text, numeric, constant text and global text – must be defined in the following four separate sections before they are used in the grammar file:

VARIABLES	See Text variable definitions
NUMERIC_VARIABLES	See Numeric variable definitions
CONSTANTS	See Constants
GLOBAL_VARIABLES	See Global variable definitions
PARAMETERS	See Parameter definitions

All variables of the same type are normally defined together in the same section, for example, all text variables in the one VARIABLES section, although more than one instance of the same section can be defined, for example, several VARIABLES sections.

If there are no variables of a specific type, the corresponding section need not be defined.

Text variable definitions

This optional section defines text variables used in the commands. Add a list of variables used in the commands under the header line, each on a separate line.

Syntax

```
\VARIABLES
    maximum-size  variable-name
```

Syntax description

maximum-size Optional. No maximum size if not defined.

Example

```
\VARIABLES
50          sLocality
3           sState
```

Defines 2 text variables, of maximum size 50 and 3 respectively.

Notes

Text variables may not be used with commands that require numeric variables.

Numeric variable definitions

This optional section defines numeric variables used in the commands. Add a list of variables used in the commands under the header line, each on a separate line.

Syntax

```
\NUMERIC_VARIABLES
    variable-name
```

Example

```
\NUMERIC_VARIABLES
i
len
```

Defines 2 numeric variables called “i” and “len”.

Notes

Numeric variables may only be used with commands that require numeric variables.

Global variable definitions

This optional section defines global text variables used in the commands. Add a list of variables used in the commands under the header line, each on a separate line.

Syntax

```
\GLOBAL_VARIABLES
    maximum-size  variable-name
```

Syntax description

maximum-size Optional. No maximum size if not defined.

Example

```
\GLOBAL_VARIABLES
50          sLocality
3           sState
```

Defines 2 global text variables, of maximum size 50 and 3 respectively.

Notes

Global variables may only be set (assigned to) in the INITIALISE section and FINALISE section (see [Initialise and Finalise commands](#)). However, they may be accessed in other sections.

Unlike normal variables, these values set in the INITIALISE section remain as long as the grammar file is loaded/open.

Constants

This optional section defines constant text to be used elsewhere in this grammar file. Add a list of constants used in the commands under the header line, each on a separate line.

Syntax

```
\CONSTANTS
    name = value [+ value2] [- value3]
```

Syntax description

<i>name</i>	Name by which the constant can be referred to later in the grammar file.
<i>value</i>	Value of constant, either some text enclosed in double-quotes (Escape sequences may be used), or another constant already defined. Any number of values may be concatenated using the plus sign to form a compound constant and characters can also be removed from a constant using the minus sign.

Example

```
\CONSTANTS
Letters      = "abcdefghijklmnopqrstuvwxyz"
Numbers      = "0123456789"
Consonants   = Letters - "aeiou"
Alphanum     = Letters + Numbers + "_"
```

Notes

Constants defined here may be used anywhere that text is required, such as the [Simple text](#) token type, [Text operands](#) in commands, the [Suggested divider](#), [DELIMITERS](#), [INNER_DELIMITERS](#), and the [WORDOF](#) token types.

Parameter definitions

This optional section defines “Parameters” used in the commands. “Parameters” are text variables whose initial values can be set programmatically.

Syntax

```
\PARAMETERS
    name alias
```

Syntax description

<i>name</i>	The name of the parameter.
<i>alias</i>	Optional. How the parameter is to be known in this grammar file. If it is not provided, <i>name</i> will be used as the alias.

Example

```
\PARAMETERS
Param1
Param2
```

Defines 2 text variables, Param1 & Param2, which can be set programmatically.

Notes

Parameters do not have maximum sizes.

To set the initial value of a parameter, use the [stanPutProperty function](#):

```
stanPutProperty(id, "PARAMETER:<parameter>", "<value>")
```

To set several parameters in one call:

```
stanPutProperty(id, "PARAMETERS", "<p1>=<v1>;<p2>=<v2>...")
```

Each time data is standardised, the parameters will be given their initial values as set programmatically. If the parameter's value is changed during standardising, it will be reset to its initial value before standardising again. If no initial value is given programmatically, then its initial value will be empty, like other variables.

Global Options

A global option line sets the default option for all future lines in the grammar file.

A global option begins with:

```
\GLOBAL_OPTION
```

See Also

[Global Option - Delimiters](#)

[Global Option - Duplicate Tables](#)

[Global Option - Table Case Sensitivity](#)

[Global Option - Duplicate Table Entries](#)

Global Option - Delimiters

This global option sets the default delimiters to be used for the token definitions in the GRAMMAR section.

Syntax

```
\GLOBAL_OPTION DELIMITERS "delimiters"
```

or

```
\DELIMITERS "delimiters"
```

Syntax description

delimiters Characters that will be used as delimiters. Escape sequences except the null character may be used.

Notes

The delimiters specified will be the default delimiters until the next such global qualifier.

Before the first such global qualifier, the default delimiters are all the punctuation characters below, and the space, vertical tab, horizontal tab, formfeed, carriage return and new line characters:

```
`~!@#$%^&*()-_+=\|[]{};:'"<>?,./
```

Example

```
\GLOBAL_OPTION DELIMITERS " , \""
```

This sets the space, comma and double quotes as the default delimiters for grammar definitions to follow.

Global Option - Duplicate Tables

This global option specifies whether [Tables](#) may be redefined. If a table is redefined the entries in the two definitions are merged together. This is useful when [Including other files](#) to add entries to a table. The options (such as CASE_SENSITIVE) for the second table may not change those of the first table.

Syntax

```
\GLOBAL_OPTION ALLOW_DUPLICATE_TABLES
```

or

```
\GLOBAL_OPTION NO_DUPLICATE_TABLES
```

Notes

This option applies to all future tables defined. The default `ALLOW_DUPLICATE_TABLES` applies before the first such global option and if this global option is not set.

Global Option - Table Case Sensitivity

This global options sets the default case sensitivity for future [Tables](#).

Syntax

```
\GLOBAL_OPTION TABLE CASE_SENSITIVE
```

or

```
\GLOBAL_OPTION TABLE CASE_INSENSITIVE
```

or

```
\CASE_INSENSITIVE
```

Notes

The default for tables is to be case-sensitive. This global option changes this default. It will apply for all tables appearing after it, till the end of the file, or till another such global option. An individual table can override the default, see [Table Case Sensitivity](#).

Global Option - Duplicate Table Entries

This global options sets whether future [Tables](#) may contain duplicate entries and if so, how to handle duplicate entries, see [Duplicates in a table](#).

Syntax

```
\GLOBAL_OPTION TABLE DuplicateOption
```

The *DuplicateOption* is one of:

```
NO_DUPLICATES  
ALLOW_DUPLICATES  
IGNORE_DUPLICATES  
OVERWRITE_DUPLICATES  
RANDOM_DUPLICATES
```

Notes

The default `ALLOW_DUPLICATES` allows duplicate entries as long as the abbreviations are the same. This global option changes the default. It will apply to all tables appearing after it, till the end of the file, or till another such global option. An individual table can override the default.

General syntax rules

The following general syntax rules apply to writing grammar files:

- [One statement per line](#)
- [Spaces and tabs](#)
- [Case sensitivity](#)
- [Comments](#)
- [Names](#)
- [Including other files](#)

- [Escape sequences](#)
- [Preprocessor directives](#)
- [Unicode and UTF-8](#)

One statement per line

Most statements in the grammar file need to be on the same line.

Use the **new line character** is used to divide statements.

Use the backslash (“\”) as the last character on the line to continue a statement onto the next line.

Spaces and tabs

White space is used to separate words in grammar files. Use spaces or tabs or a combination of both.

Case sensitivity

The grammar file is **case-sensitive**. This includes table names, token names, variable names, commands, headers etc.

Comments

Anything on a line appearing after a hash character (“#”) is a **comment** and is ignored.

In the [Description](#), [Output fields](#) and [Tables](#) sections, the hash character must be the **first** non-white space character for it to be considered a comment:

Names

Names of tables, tokens, variables etc. are made of letters, digits and the underscore character only. The first character cannot be a digit and accented characters (e.g. “é”) are not allowed in names.

Names have a **length limit** defined in [MAX_NAME](#).

Including other files

Other files may be “included” into the grammar file using the INCLUDE statement.

This enables long tables to be kept in a separate file, or common sections to be kept in one file. The include line may appear anywhere in the grammar file, but must be on its own line.

Syntax

```
\INCLUDE file-name
```

Syntax description

<i>File-name</i>	The name of the file to be included in this position of the grammar file, either an absolute path (e.g. C:\Intech\Table.txt), or path relative to the original grammar file (e.g. Table.txt).
------------------	---

Notes

If *File-name* is a relative path (e.g. Table.txt), it will be searched for in the same directory as the original grammar file. If it doesn’t exist in that directory, it will be searched for in the default grammar directory.

If an included file includes another file, then the second file will be searched for in same directory as the original grammar file, which may not be the same directory as the first included file.

Escape sequences

Whenever a character or series of characters is required in quotes, C-style escape characters may be used.

This enables a sequence of characters to be used to represent special characters. Escape sequences are listed below.

\a	Alert (bell)
----	--------------

`\b` Backspace
`\f` Formfeed
`\n` Newline
`\r` Carriage return
`\t` Horizontal tab
`\v` Vertical tab
`\?` Literal quotation mark
`\'` Single quotation mark
`\"` Double quotation mark
`\\` Backslash
`\xdd` ASCII character in hex notation
`\0` Null character

Preprocessor directives

Similar to C pre-processor directives, lines in the grammar file can be conditionally ignored by using the following:

```
\DEFINE, \UNDEF, \IFDEF, \IFNDEF, \ELSE, \ENDIF
```

Example

```

\DEFINE long
\OUTPUT_FIELDS
\IFDEF long
100          field1
100          field2
100          field3
\ELSE
30           field1
30           field2
30           field3
\ENDIF

```

Explanation

Line	Description
1	Defines a variable “long”
3	Tests for the existence of a variable “long”
4 to 6	Read as normal, since the test was true (“long” had been defined)
8 to 10	Ignored, as they form part of the “else” block
11	Needed to match the \IFDEF on line 3

If Line 1 was removed (and “long” hence not defined), lines 4 to 6 would be ignored, and lines 8 to 10 read as normal.

Notes

`\IFNDEF` (“if not defined”) is the opposite test of `\IFDEF` (“if defined”).

`\UNDEF` will remove the defined variable.

The `\ELSE` is optional – one can just use the `\IFDEF` and `\ENDIF` if desired.

\IFDEFs can be nested. There is not limit (memory permitting) on the number of \DEFINES, or on the depth of nesting.

A variable may be given a value when defined (eg. \DEFINE name John). This value may then be used later in the grammar file instead of a literal.

A variable may be redefined with the same value, another value or no value.

The \IFDEF, \ELSE and \ENDIF must all be in the same file (ie. they cannot span files via the \INCLUDE directive). Variables defined by \DEFINE, however, are accessible to all files.

Lines can be made conditional depending on the version of the IQ Standardiser parsing engine, for example:

```
\IF VERSION > 5.0.25.1
```

The following can also be used: >, >=, <, <=, = or <>

Not all of the numbers in the version need to be specified, and only those specified will be compared.

\ELSEIF can also be used for versions, for example:

```
\ELSEIF VERSION => 5.0.26
\ELSEIF DEF long #else if defined
\ELSEIF NDEF long #else if not defined
```

Unicode and UTF-8

To support Unicode, a grammar file may be in UTF-8 format.

To specify a grammar file as UTF-8, either use a Byte Order Mark (Hex EF BB BF at the beginning of the file), or include this line before any UTF-8 characters:

```
\CHARSET UTF8
```

Unicode characters are allowed in the following places in a grammar file:

- where quotes are used (eg literals)
- TABLE entries
- in \DEFINE statements.

The UTF-8 setting on a file is not assumed for files it includes. So if a UTF-8 file includes another file, the included file is not assumed to be UTF-8 until specified. Similarly, the \CHARSET UTF8 statement in an included file does not affect the including file.

If a file is not UTF-8, the character set is assumed to be Windows-1252 (a superset of ISO 8859-1). This can be explicitly set using this line:

```
\CHARSET WINDOWS_1252
```

For character recognition and case conversion to work correctly, the locale should be set. See [Locale setting](#).

To parse unicode characters, call the new [stanStanW function](#). Calls to the non-Unicode [stanStan function](#) will use the Windows-1252 character set.

Encrypted sections

Parts of a grammar file may be encrypted.

\ENCRYPTED marks the beginning of an encrypted section, and \END_ENCRYPTED marks the end of such a section.

To encrypt a grammar file, use the IQArfUtil utility, menu 12 – Other Utilities.

Parsing

The [Grammar definitions](#) section of a grammar file contains a set of [Token definitions](#), which defines how the input is parsed.

Once the input has been parsed, [Output actions](#) are performed which extract the parsed data.

[Commands](#) may also be applied based on the parsed data.

All together, this makes up how the parser works, see [How the Parser works](#).

Token definitions

Token definitions appear in the [Grammar definitions](#) section of the grammar file, and define how the input is parsed (see [Parsing](#)). Its structure is similar to a formal syntax definition. There are two types of tokens, [Simple](#) and [Composed](#).

Simple tokens

The simple token type is used to specify simple predefined elements in the input, for example:

```
Name : WORD
```

This defines a token called *Name* to be a “WORD”, predefined as any sequence of consecutive letters.

Composed tokens

Simple tokens can be combined to form a composed token, for example:

```
FullName : Name Name
```

This defines *FullName* is being composed of two simple *Name* tokens.

The final token

The last token defined is the full format of the expected input text, for example:

```
\GRAMMAR
Name           : WORD
FullName       : Name Name
               : Name
```

This last defined token “FullName” is what will be searched for in the input text. It is composed of either two “Name”s or a single “Name”.

Simple tokens types

Simple tokens are predefined elements searched for in the input, which are not composed of other tokens. Simple token types are listed below.

[Character](#)

[Simple text](#)

[WORD](#)

[WORD_SIZE_RANGE](#)

[WORDOF](#)

[WORDOF_SIZE_RANGE](#)

[WORD2](#)

[WORDOF2](#)

[WORDOF2_SIZE_RANGE](#)

[ALPHANUM](#)

[ALPHANUM_SIZE_RANGE](#)

[NUMBER](#)

[NUMBER_RANGE](#)

[NUMBER_SIZE_RANGE](#)

[ORDINAL](#)

[TABLE](#)

[TABLE_SINGLE_WORDS](#)

See also

[Token definitions](#)

[Qualifiers of simple types](#)

Character token type

This token type is a single occurrence of a defined character. [Escape sequences](#) may be used.

Example

```
Comma      :  ","
Quote      :  "\""
```

This defines “Comma” to be a comma, and “Quote” to be a double quotation mark.

Simple text token type

Example

```
And        :  "AND"
           :  "&"
```

This simply defines “And” to be either “AND” or “&”.

Notes

The text must be surrounded by quotes.

[Escape sequences](#) may be used.

Matches are case-sensitive.

The [Delimiter before](#) and [Delimiter after](#) can be used to ensure only the whole word is matched.

WORD token type

A WORD is made up of just letters.

Example

```
Name      :  WORD
```

This defines “Name” to be a sequence of letters.

Examples of WORDs

```
Hello
Peter
GOODBYE
```

Examples which are not WORDs

```
Résumé
```

```
Peter1
a_b
GOOD-BYE
```

WORD_SIZE_RANGE token type

A WORD_SIZE_RANGE is a subset of the [WORD](#) type, (i.e. made up of letters only) but with a specified number of letters.

Example

```
US_State      : WORD_SIZE_RANGE 2
Name          : NUMBER_SIZE_RANGE 3 15
```

This defines “US_State” to be a sequence of 2 letters, and “Name” to be a sequence of between 3 and 15 digits inclusive.

Examples of “US_State”

```
NY
ZZ
```

Examples which are not a “US_State”

```
N Y
C.A.
```

Note

A TABLE is recommended for states instead of this token type.

WORDOF token type

A WORDOF is a sequence of characters in a defined list. [Escape sequences](#) may be used.

Example

```
FrenchName    : WORDOF "abcdefghijklmnopqrstuvwxyzéèçàùâêîôûäëïöü"
WebName       : WORDOF "abcdefghijklmnopqrstuvwxyz-."
```

This defines “FrenchName” to be a sequence of lowercase letters, including those with accents, and “WebName” as a sequence of letters, including the dash and the dot.

WORDOF_SIZE_RANGE token type

A WORDOF_SIZE_RANGE is a subset of the [WORDOF](#) type but with a specified number of digits.

Example

```
WebName : WORDOF_SIZE_RANGE "abcdefghijklmnopqrstuvwxyz-." 7 27
```

This defines “WebName” to be a sequence of between 7 and 27 lowercase letters inclusive, including the dash and the dot.

WORD2 token type

A WORD2 is made up of just letters like the [WORD](#) token type, except that an apostrophe (') or a dash (-) is allowed in the word if it is not at the beginning of the word.

Example

```
Name          : WORD2
```

This defines “Name” to be a sequence of letters (allowing the apostrophe and dash in the word).

Examples of WORD2s

```
O'Reily
Peter's
Good-bye
```

Note: in the above case, “Good-bye”, “Good” and “bye” will all be picked up as a WORD2 since they are separated by a dash. However, in the case before “Peter” and “s” will not be picked up.

Examples which are not WORD2s

```
'tis  
-life
```

Note: in the above cases, “tis” and “life” will be picked up as a WORD2, without the punctuation.

WORD2 case qualifiers

The case of the word can be specified by adding one of the following after WORD2: UPPERCASE, LOWERCASE, MIXEDCASE, TITLECASE, NOT UPPERCASE, NOT LOWERCASE, NOT MIXEDCASE or NOT TITLECASE. Title case is a word with the first letter upper-case and all subsequent letters lower-case.

Case qualifier examples

```
UName :      WORD2 UPPERCASE  
LName :      WORD2 LOWERCASE  
SName :      WORD2 NOT MIXEDCASE
```

This defines “UName” to be a WORD2 of upper-case letters only, “LName” of lower-case letters only, and “SName” of same case letters only, ie. all upper-case or all lower-case.

WORDOF2 token type

A WORDOF2 is a sequence of characters from two defined lists. The sequence must contain at least one character from the first list, and none or any number of characters from the second list. [Escape sequences](#) may be used.

Example

```
RealWord : WORDOF2 "aeiou" "bcdfghjklmnpqrstvwxyz"
```

This defines “RealWord” to be a sequence of lowercase letters, with at least one vowel.

“a”, “hello” and “oo” are examples of “RealWord”.

“rythym” would not be an example of “RealWord”.

WORDOF2_SIZE_RANGE token type

A WORDOF2_SIZE_RANGE is the same as the [WORDOF2](#) type but with a specified number of characters.

Example

```
RealWord : WORDOF2_SIZE_RANGE "aeiou" "bcdfghjklmnpqrstvwxyz" 3 5
```

This defines “RealWord” to be a sequence of lowercase letters, with at least one vowel. Its length must be between 3 and 5 inclusive.

ALPHANUM token type

An ALPHANUM is made up of letters and digits.

Example

```
ReservationNo : ALPHANUM
```

This defines “ReservationNo” to be a sequence of letters and/or digits.

Examples of ALPHANUMs

```
1490
```

```
Hello  
H2N5zf4
```

Examples which are not ALPHANUMs

```
ABC_123
```

ALPHANUM_SIZE_RANGE token type

An ALPHANUM_SIZE_RANGE is a subset of the [ALPHANUM](#) type, (i.e. made up of letters and digits only) but with a specified number of characters.

Example

```
ReservationId : ALPHANUM_SIZE_RANGE 8  
Password      : ALPHANUM_SIZE_RANGE 4 10
```

This defines “ReservationId” to be a sequence of 8 letters and/or digits, and “Password” to be a sequence of between 4 and 10 letters and/or digits inclusive.

NUMBER token type

A NUMBER is made up of just digits.

Example

```
StreetNo      : NUMBER
```

This defines “StreetNo” to be a sequence of digits.

Examples of NUMBERs

```
1  
1490  
000
```

Examples which are not NUMBERs

```
1.5  
750 000  
750,000  
-24
```

NUMBER_RANGE token type

A NUMBER_RANGE is a subset of the [NUMBER](#) type, (i.e. made up of digits only) but within a numerical range.

Example

```
PostCode      : NUMBER_RANGE 1 9999
```

This defines “PostCode” to be a sequence of digits with value between 1 and 9999 inclusive.

Examples of “PostCode”

```
2000  
0810  
1
```

Examples which are not a “PostCode”

```
0000  
10000  
4,500
```

NUMBER_SIZE_RANGE token type

A NUMBER_SIZE_RANGE is a subset of the [NUMBER](#) type, (i.e. made up of digits only) but with a specified number of digits.

Example

```
PostCode      : NUMBER_SIZE_RANGE 4
```

```
PhoneNumber : NUMBER_SIZE_RANGE 6 8
```

This defines “PostCode” to be a sequence of 4 digits, and “PhoneNumber” to be a sequence of between 6 and 8 digits inclusive.

Examples of “PhoneNumber”

```
987123
1236987
12349876
```

Examples which are not a “PhoneNumber”

```
987 123
123-6987
```

ORDINAL token type

This token type is a number with a suffix making it an ordinal.

Example

```
Ord           : ORDINAL
Teens         : ORDINAL 11 19
First         : ORDINAL 1
```

This defines “Ord” as any ordinal, “Teens” as an ordinal between 11th, and 19th and “First” as 1st.

Examples of “Ord”

```
1st
2ND
3rd
100th
```

Examples which are not an “Ord”

```
First
21th
100 th
```

Note

The found token will only be the number, without the suffix (eg 1 not 1st).

TABLE token type

A TABLE token type will find any value defined in [Tables](#).

Example

```
\TABLE States
NEW SOUTH WALES=NSW
VICTORIA=VIC

\GRAMMAR
State : TABLE States
```

This defines “State” to be either “NEW SOUTH WALES”, or “VICTORIA” (but not “NSW” nor “VIC”).

Notes

When there are values in the table with spaces, there must be only one space between each word.

If there are other delimiters in the values, it will not find the value. So if the dot is a delimiter (which it is by default), then “N . S . W=NSW”, will not pick up anything. Either ensure that the dot is not a delimiter, or use “N S W=NSW”.

How it works

In the above case “NEW SOUTH WALES” is taken as 3 separate words, and the input is searched for three consecutive [Constants](#), the words “NEW” “SOUTH” “WALES”. A word is anything surrounded by [DELIMITERS](#). So, if the comma and the space are delimiters (which they are by default), then “NEW,SOUTH , WALES” will also match, but “NEWSOUTHWALES” or “NEW2SOUTH3WALES” will not. See also [Qualifiers of TABLE types](#).

TABLE_SINGLE_WORDS token type

This is identical to the [TABLE](#) type, except that table entries can only have one word (e.g. “VICTORIA” but not “NEW SOUTH WALES”).

Example

```
\TABLE States
NSW
VIC
QLD

\GRAMMAR
State : TABLE_SINGLE_WORDS States
```

Composed token type

A composed token type is one made up of previously defined [Simple](#) or composed tokens.

Syntax

```
\GRAMMAR
token : sub-tokens
```

Syntax description

<i>token</i>	The name of this token
<i>sub-token</i>	One or more previously defined tokens

Example

```
\GRAMMAR
Name          : WORD
FullName      : Name Name
```

Here, the composed token “FullName” is defined as two consecutive “Name” tokens.

Examples of “FullName”

```
John Smith
S A
```

See also

[Parsing](#).

[Qualifiers of composed type](#)

Qualifiers of simple types

Optional qualifiers that can be used with [Simple](#) token types are listed below.

- < [Delimiter before](#)
- > [Delimiter after](#)
- \$ [Must be first](#)
- \$ [Must be last](#)
- ! [Look from end](#)

Syntax

```
\GRAMMAR
  token : [$] simple-type [>] [<] [!] [$]
```

Syntax description

Any number of qualifiers may be used with the same definition. Qualifiers may appear in any order, but must be after the simple type, except for the “Must be first” qualifier, which precedes the simple type.

Example

```
\GRAMMAR
Name : WORD $ < !
```

Here, “Name” is defined as a word (letters) that appears at the end of the input text, is preceded by a delimiter, and is searched for from the end of the text.

Qualifier - Delimiter before

This qualifier ensures that the character before the token is a delimiter, or it is the first character in the input.

Symbol

<

Example

```
\GRAMMAR
And : "AND" <
```

Examples of “And”

```
AND
ME AND YOU
ANDREW
```

Assuming that space is a delimiter (which is the default).

Examples which are not an “And”

```
BAND
```

Qualifier - Delimiter after

This qualifier ensures that the character after the token is a delimiter, or it is the last character in the input.

Symbol

>

Example

```
\GRAMMAR
StreetName : WORD >
```

Examples of “StreetName”

```
Main
123 Main, 2000
123Main 2000
```

Assuming that space and comma are delimiters (which is the default).

Examples which are not a “StreetName”

```
123 Main2000
```

Assuming that numbers are not delimiters (which is the default).

Qualifier - Look from end

This qualifier specifies the search for the token starting at the end of the text. This will mean that tokens found closer to the end will be given priority.

Symbol

!

Example

```
\GRAMMAR
PostCode : NUMBER_RANGE 1 9999 !
```

Examples of “PostCode”

```
PO Box 9876 SYDNEY 2001 NSW
```

In this examples the number 2001 is found as a postcode before the number 9876, so will be given priority over it.

Qualifier - Must be first

This qualifier ensures that the token is the first non-delimiter in the input text. It is mutually exclusive with [Look from end](#) and [Must be last](#) qualifiers.

Symbol

\$

Symbol appears before token type.

Example

```
\GRAMMAR
FirstName : $ WORD
```

Examples of “FirstName”

```
Peter Jones
- Peter
```

Assuming that space and dash are delimiters (which is the default).

Examples which are not a “FirstName”

```
123 Main
```

Assuming that numbers are not delimiters (which is the default).

Qualifier - Must be last

This qualifier ensures that the token is the last non-delimiter in the input text.

Must be used with the [Look from end](#) qualifier.

Symbol

\$

Example

```
\GRAMMAR
PostCode : NUMBER_RANGE 1 9999 ! $
```

Examples of “PostCode”

```
MELBOURNE 3000.
```

Assuming that the point is a delimiter (which is the default).

Examples which are not a “PostCode”

```
MELBOURNE 3000 VIC
```

Assuming that letters are not delimiters (which is the default).

Qualifiers of TABLE types

Optional qualifiers that can be used with the [TABLE](#) and [TABLE_SINGLE_WORDS](#) token types are listed below:

- < [Delimiter before](#)
- > [Delimiter after](#)
- \$ [Must be first](#)
- \$ [Must be last](#)
- ! [Look from end](#)
- [DELIMITERS](#)
- A [Tolerance method](#)
- [INNER_DELIMITERS](#)
- [CHAR_SET](#)

Syntax

\GRAMMAR

```
token : [$] TABLE[_SINGLE_WORDS] table-name [![$]] [TOLERANCE tolerance
search-span | TOLERANCE1 tolerance | TOLERANCE2 tolerance | TOLERATE
tolerance-method] [DELIMITERS "delimiters"] [INNER_DELIMITERS
"innerDelimiters"] [CHAR_SET "characters" [<] [>]]
```

Syntax description

The qualifiers must all be on the one line, although a backslash can be used to continue on the next line.

[Escape sequences](#) may be used in the *delimiter* and *innerDelimiter* lists.

Notes

“Delimiter before” and “Delimiter after”, can only be used if CHAR_SET is used.

Unless using [TABLE_SINGLE_WORDS](#), “Delimiter before” cannot be combined with “Look from the end”, and “Delimiter after” can only be used together with “Look from the end”.

Table qualifier - DELIMITERS

This defines the delimiters to use when searching for words that match table entries.

The Standardiser searches for a sequence of characters, then looks them up in the table. The sequence of characters is separated from the rest of the text by delimiters.

If this qualifier is not defined, then the default delimiters will be used.

Examples

If numbers are not delimiters (the default), and the input text has “1ST”, then “1ST” will be looked up in the table, and not “ST”.

If the input has “1 ST”, then both “1” and “ST” and “1 ST” will be looked up.

If numbers are delimiters and the input text has “1ST”, then “ST” will be looked up and not “1ST”.

Example

```
\GRAMMAR
State : TABLE state DELIMITERS " , ./-0123456789"
```

Here, the numbers are defined as delimiters, so if there is no space between the state and the postcode for example, it will still look up the state (eg. "NSW2000").

Table qualifier - INNER_DELIMITERS

This defines characters that are allowed to appear between words, when looking up words in a table.

It has no effect on [TABLE_SINGLE_WORDS](#), as it concerns the characters between multiple-word entries.

Characters in this list of inner delimiters that are not in [DELIMITERS](#), will be ignored, i.e. inner delimiters are a subset of delimiters.

If not defined, then the delimiters defined in [DELIMITERS](#) will be used.

Example

```
\TABLE state
N S W=NSW

\GRAMMAR
State : TABLE state INNER_DELIMITERS " ."
```

Here, "N S W" or "N.S.W." will be found, but not "N,S,W" or "N-S-W", as only the spaces or dots are allowed between the letters, and not commas or hyphens.

Table qualifier - CHAR_SET

This defines the characters that can make up words that are looked for in the table.

If not defined, then all characters that are not in the set of delimiters are used.

Example

```
\TABLE Levels
LEVEL
FLOOR

\GRAMMAR
Level : TABLE Levels CHAR_SET "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Here, only words made up of letters only will be looked up in the table. So with an input of "LEVEL25 135 KING ST", the words looked up in the table would be "LEVEL", "KING" and "ST" (with only "LEVEL" being found). If the character set included numbers, then "LEVEL" would not be looked up, rather "LEVEL25" (and so wouldn't be found).

CHAR_SET and DELIMITERS

The example above would be equivalent of setting the [DELIMITERS](#) to everything besides letters, but the advantage of using **CHAR_SET** is that the [Delimiter before](#) (<) or [Delimiter after](#) (>) qualifiers can be used. If the "Delimiter Before" qualifier is added to the example above:

```
Level : TABLE Levels CHAR_SET "ABCDEFGHIJKLMNOPQRSTUVWXYZ" <
```

then with an input of "2FLOOR LEVEL8", only "LEVEL" would be picked up and not "FLOOR", as LEVEL has a delimiter before it whereas FLOOR does not.

Qualifiers of composed type

Optional qualifiers that can be used with [Composed](#) token types are listed below.

- \$ [Must be first](#)
- \$ [Must be last](#)

- @ [Junk allowed](#)
- ~ [Any order](#)
- ~ [Order digits](#)
- ? [Optional sub-token](#)

DELIMITERS

Syntax

\GRAMMAR

token : [\$] *sub-token-name* [?] [\$][@][~] DELIMITERS "*delimiters*"

Syntax description

Only one sub-token is show above, but there may be many.

Any number of qualifiers may be used with the same definition.

The last three qualifiers may appear in any order.

Example

```
\GRAMMAR
Word           : WORD
Number         : NUMBER
WordNumber     : $ Word Number? $
```

Here, WordNumber will match an input text if it is only a Word or a Word followed by a Number.

Composed qualifier - Must be first

If this qualifier is specified, then no non-delimiters are allowed before the first sub-token.

Symbol

\$

Composed qualifier - Must be last

If this qualifier is specified, then no non-delimiters are allowed after the last sub-token.

Symbol

\$

Composed qualifier - Junk allowed

If this qualifier is specified, then other non-delimiters are allowed in the input text between the sub-tokens. If this qualifier is not set, then only delimiters can appear between sub-tokens.

Symbol

@

Example

```
\GRAMMAR
Word           : WORD
Number         : NUMBER
WordNumber     : $ Word Number @ $
```

Examples of "WordNumber"

Main 125
Main station, platform 125

Since the “Must be first” and “Must be last” qualifiers are set, no non-delimiters are allowed after or before the “Word” and the “Number”, but other text is allowed between them.

Composed qualifier - Optional sub-token

If this qualifier is specified, then the sub-token which it follows, is optional.

Any number of sub-tokens can be defined as optional.

Symbol

?

Example

```
\GRAMMAR
Word           : WORD
Number        : NUMBER
WordNumber    : Word Number? Word?
```

This defines a word, possibly followed by a number, possibly followed by another word.

Examples of “WordNumber”

```
station 125 platform
station platform
station 125
station
```

Composed qualifier - Any order

If this qualifier is specified, then the sub-tokens can appear in any order.

If this qualifier is not set, then they must appear in the same order as in the definition.

Symbol

~

Example

```
\GRAMMAR
Word           : WORD
Number        : NUMBER
WordNumber    : Word Word Number ~
```

Examples of “WordNumber”

```
Main 125 Platform
Main Platform 125
125 Main Platform
```

Composed qualifier - Order digits

After the [Any order](#) qualifier, order digits can be defined. For each sub-token, a digit (0-9) is used to define the acceptable order of the sub-tokens, for example:

~321 means that the three sub-tokens must appear in the input in reverse order

~123 means that the three tokens must appear in order

Example

```
\GRAMMAR
Word           : WORD
Number        : NUMBER
Comma         : ", "
WordNumber    : Word Number Comma ~312
```

This specifies that the order of the sub-tokens of *WordNumber* is 312. In other words, *Word* must be in relative position 3, *Number* in relative position 1, and *Comma* in relative position 2. So a *WordNumber* is a *Number* followed by a *Comma* followed by a *Word*.

Notes

If order digits are specified, they must be specified for all sub-tokens. So if there are three sub-tokens (as in *WordNumber* above), there need to be three digits (*312*).

More than one sub-token can be given the same order digit, meaning that their relative order is not important. So 211 in the above example would mean that *Word* must be last, but *Number* and *Comma* can be in any order.

Order digits 123 would be the equivalent of having neither the [Any order](#) qualifier nor the **Order digits** (except less efficient).

Priority is given in order that the sub-tokens appear on the definition line. So, if a specific order is required, but with a different priority order, then the Any-Order and Order-Digits can be used for this purpose.

Composed qualifier - DELIMITERS

This defines the delimiters to be used for the other [Qualifiers of composed type](#), such as “Must be first”, “Must be last” and “Junk allowed”.

If no delimiters are defined, then the default delimiters will be used. [Escape sequences](#) may be used.

Example

```
\GRAMMAR
Word           : WORD
Number         : NUMBER
WordNumber     : Word Number DELIMITERS "."
```

Examples of “WordNumber”

```
Main125
Main..125
```

Examples which are not a “WordNumber”

```
Main 125
Main--125
```

There are spaces or dashes between the sub-tokens, and neither a space nor a dash is not defined here as a delimiter.

Tolerance methods

When looking up a word in a table, one can specify a tolerance method. This will allow entries in the table that are close matches to be found. Care must be taken not to be too tolerant or even distant matches will be found.

These tolerance methods apply to the [TABLE](#) token type, the [TABLE_SINGLE_WORDS](#) token type and the [LOOKUP](#) command.

The tolerance methods are:

```
TOLERATE DIFFS
TOLERATE DIFFS\_PER\_LEN
TOLERATE DIFFS\_PER\_WORD\_LEN
TOLERATE JARO.
```


The following are predefined tolerance methods, using the above methods:

[TOLERANCE](#)

[TOLERANCE1](#)

[TOLERANCE2](#).

If the tolerance method is omitted then the words must match exactly – there is no tolerance.

DIFFS tolerance method

This is a [Tolerance method](#) that can be used with the [TABLE](#) token type, the [TABLE_SINGLE_WORDS](#) token type and the [LOOKUP](#) command. It allows close matches in a table to be found.

The DIFFS tolerance method defines a close match as one that requires less than a specific number of changes. A change can be adding a character, removing a character, replacing a character or swapping two adjacent characters. This number of changes is also known as the Edit Distance.

The first item found in the table that requires the least number of changes will be chosen as the match.

Syntax

```
TOLERATE DIFFS tolerance [SAME_FIRST] [NO_SWAP] [NO_SPACE | ONLY_SPACE]  
[NO_NUMBERS] [PHONETIC diff]
```

Syntax description

<i>tolerance</i>	The maximum number of character changes allowed.
<i>SAME_FIRST</i>	If specified, then only entries in the table whose first character match are considered.
<i>NO_SWAP</i>	If specified, then swapping of adjacent characters is not allowed. More specifically, swapping two adjacent characters is counted as two changes not one.
<i>NO_SPACE</i>	If specified, then the space character may not be one of the characters changed.
<i>ONLY_SPACE</i>	If specified, then the only difference allowed is a space character.
<i>NO_NUMBERS</i>	If specified, then a number may not be one of the characters changed.
<i>diff</i>	If specified, and the number of differences is <i>diff</i> or more, then the phonetic codes must also match.

Example 1

```
TOLERATE DIFFS 2
```

This will first look for entries in the table that exactly match the text. If none are found, then it will look for entries in the text that differ by only one character. If none are found then it will look for entries in the table that differ by two characters.

Example 2

```
TOLERATE DIFFS 1 SAME_FIRST
```

This will first look for entries in the table that exactly match the text. If none are found, then it will look for entries in the text that have the same first letter as the text, and differ by only one character.

Example 3

```
TOLERATE DIFFS 3 PHONETIC 2
```

This will first look for entries in the table that exactly match the text. If none are found, then it will look for entries in the text that differ by only one character. If none are found then it will look for entries in the table that differ by two characters, provided the phonetic codes of the entry and the text are the same. If none are found then it will look for entries in the table that differ by three characters, provided the phonetic codes of the entry and the text are the same.

Example 4

```
TOLERATE DIFFS 1 ONLY_SPACE
```

Useful for Chinese or Vietnamese localities, which are sometimes written with space and sometimes without. Eg. HA NOI or HANOI.

Example 5

```
TOLERATE DIFFS 1 NO_NUMBERS
```

Prevents matching “District 1” to “District 2”, but allows matching to “Distict 1” (missing “r”).

DIFFS_PER_LEN tolerance method

This is a [Tolerance method](#) that can be used with the [TABLE](#) token type, the [TABLE_SINGLE_WORDS](#) token type and the [LOOKUP](#) command. It allows close matches in a table to be found.

This tolerance method defines a close match as one that requires less than a defined number of changes. A change can be adding a character, removing a character, replacing a character or swapping two adjacent characters. This number of changes is sometimes called the Edit Distance.

DIFFS_PER_LEN is the same as [DIFFS](#), except that the number of differences allowed varies according to the length of the text.

Syntax

```
TOLERATE DIFFS_PER_LEN tolerance [SAME_FIRST] [NO_SWAP] [NO_SPACE |  
ONLY_SPACE] [NO_NUMBERS] [PHONETIC diff]
```

Syntax description

<i>tolerance</i>	A list of space-separated numbers – the lengths of words for each tolerance. The first number is the minimum length of the text for which one difference will be tolerated. The second number is the minimum length of the text for which two differences will be tolerated. And so on. See examples.
<i>SAME_FIRST</i>	If specified, then only entries in the table whose first character match are considered.
<i>NO_SWAP</i>	If specified, then swapping of adjacent characters is not allowed. More specifically, swapping two adjacent characters is counted as two changes not one.
<i>NO_SPACE</i>	If specified, then the space character may not be one of the characters changed.
<i>ONLY_SPACE</i>	If specified, then the only difference allowed is a space character.
<i>NO_NUMBERS</i>	If specified, then a number may not be one of the characters changed.
<i>diff</i>	If specified, and the number of differences is <i>diff</i> or more, then the phonetic codes must also match.

Example 1

```
TOLERATE DIFFS_PER_LEN 5
```

This will first look for entries in the table that exactly match the text. If none are found, and the text has 5 characters or more, then it will look for entries in the table that differ from the text by only one and have at least 5 characters. If the text has less than 5 characters, then only exactly matching entries will be searched for.

Example 2

```
TOLERATE DIFFS_PER_LEN 1 5
```

This will first look for entries in the table that exactly match the text. If none are found, and the text has 1 character or more (which all text does), then it will look for entries in the table that differ from the text by only one. If none are found, and the text has more than 5 characters, it will look for entries in the table that differ from the text by two and that have at least 5 characters.

DIFFS_PER_WORD_LEN tolerance method

This is a [Tolerance method](#) that can be used with the [TABLE](#) token type, the [TABLE_SINGLE_WORDS](#) token type and the [LOOKUP](#) command. It allows close matches in a table to be found.

This tolerance method defines a close match as one that requires less than a defined number of changes. A change can be adding a character, removing a character, replacing a character or swapping two adjacent characters. This number of changes is sometimes called the Edit Distance.

DIFFS_PER_WORD_LEN is the same as [DIFFS_PER_LEN](#), except that the number of differences allowed varies according to the length of individual words in the text and the table, not according to the whole text or table entry. Words are those separated by spaces.

Syntax

```
TOLERATE DIFFS_PER_WORD_LEN tolerance [SAME_FIRST] [NO_SWAP] [NO_SPACE |  
ONLY_SPACE] [NO_NUMBERS] [PHONETIC diff]
```

Syntax description

<i>tolerance</i>	A list of space-separated numbers – the lengths of words for each tolerance. The first number is the minimum length of a word for which one difference will be tolerated. The second number is the minimum length of a word for which two differences will be tolerated. And so on. See examples.
<i>SAME_FIRST</i>	If specified, then only entries in the table whose first character match are considered.
<i>NO_SWAP</i>	If specified, then swapping of adjacent characters is not allowed. More specifically, swapping two adjacent characters is counted as two changes not one.
<i>NO_SPACE</i>	If specified, then the space character may not be one of the characters changed.
<i>ONLY_SPACE</i>	If specified, then the only difference allowed is a space character.
<i>NO_NUMBERS</i>	If specified, then a number may not be one of the characters changed.
<i>diff</i>	If specified, and the number of differences is <i>diff</i> or more, then the phonetic codes must also match.

Example 1

```
TOLERATE DIFFS_PER_WORD_LEN 5
```

This will first look for entries in the table that exactly match the text. If none are found, then it will look for entries in the table for which each word of 5 characters or more differs by no more than one.

Example 2

```
TOLERATE DIFFS_PER_WORD_LEN 1 5
```

This will first look for entries in the table that exactly match the text. If none are found, then it will look for entries in the table for which each word differs by no more than one. If none are found, then it will look for entries in the table for which each word of 5 characters or more differs by no more than two.

JARO tolerance method

This is a [Tolerance method](#) that can be used with the [TABLE](#) token type, the [TABLE_SINGLE_WORDS](#) token type and the [LOOKUP](#) command. It allows close matches in a table to be found.

This tolerance method sorts the table alphabetically. When searching for a close match, the alphabetical position of the text is obtained. Then tables entries near this position are compared for a close match. The Jaro string comparison algorithm is used to determine what a close match is.

Syntax

```
TOLERATE JARO tolerance [SEARCH_SPAN search-span]
```

Syntax description

<i>tolerance</i>	A percentage value, as to how close the text and the table entry need to be. 100 means exact match required, 90 means close match etc.
<i>search-span</i>	(One more than) the number of values in the table, before and after its alphabetical position, that the text will be compared with. If this value is omitted or zero, then all entries in the table with the same first letter will be compared.

Example

```
\TABLE Localities
ALECTOWN
ALEXANDER
ALEXANDRA
ALEXANDRIA
ALFORD

\GRAMMAR
Locality          : TABLE Localities  TOLERATE JARO 90  SEARCH_SPAN 3
```

If the input contained “ALEXANDAR”, then its alphabetical position will be found, i.e. before “ALEXANDER”. Then it will be compared with all those entries close to “ALEXANDER”.

Here, the 2 table entries each side (maximum of 4 altogether) are compared with the text. The first to give a 90% comparison score or better will be chosen as a close match. So with the above table, “ALEXANDAR” will match the table entry “ALEXANDER”.

If the tolerance is set to 95%, then “ALEXANDAR” will not match with “ALEXANDER”, but it will match with “ALEXANDRA”.

TOLERANCE tolerance method

This is a [Tolerance method](#) that can be used with the [TABLE](#) token type, the [TABLE_SINGLE_WORDS](#) token type and the [LOOKUP](#) command. It allows close matches in a table to be found.

It is equivalent to the [JARO](#) tolerance method.

Syntax

```
TOLERANCE tolerance [search-span]
```

Syntax description

<i>tolerance</i>	A percentage value, as to how close the text and the table entry need to be. 100 means exact match required, 90 means close match etc.
<i>search-span</i>	(One more than) the number of values in the table, before and after its alphabetical position, that the text will be compared with. If this value is omitted or zero, then all entries in the table with the same first letter will be compared.

TOLERANCE1 tolerance method

This is a [Tolerance method](#) that can be used with the [TABLE](#) token type, the [TABLE_SINGLE_WORDS](#) token type and the [LOOKUP](#) command. It allows close matches in a table to be found.

If there is no exact match, then any entry in the table starting with the same first letter, and differing by only one character, will be used.

It is equivalent to the [DIFFS_PER_LEN](#) tolerance method, with only one number in the *tolerance* parameter, and with the SAME_FIRST, NO_SPACE and NO_SWAP options.

Syntax

```
token : TABLE table-name TOLERANCE1 tolerance
```

Syntax description

<i>token</i>	The name of the token being defined
<i>table-name</i>	The name of the table. Previously defined with the <code>\TABLE</code> header.
<i>tolerance</i>	The minimum number of characters that both the word in the input and the entry in the table must have, to be considered a match.

Example

```

\TABLE Localities
ALEXANDER
ALMA
ALPHA

\GRAMMAR
Locality          : TABLE Localities TOLERANCE1 5

```

Here, “ALEXANDER” would be found if the input contained any of the following: “ALEXANDAR”, “ALEXANDR”, “ALEXANDEAR”, or “ALIXANDER” as they all only differ by one character.

However, an input of “ELEXANDER” would not find “ALEXANDER”, as the first letter is not the same.

Also, an input “ALEX ANDER” would not find “ALEXANDER”, as the number of words differs.

“ALMA” would only be found on an exact match, as its length is less than 5.

“ALPHA” would not be found on an input of “ALPH”, as its length is less than 5, but would be found on an input of “ALPFA”.

TOLERANCE2 tolerance method

This is a [Tolerance method](#) that can be used with the [TABLE](#) token type, the [TABLE_SINGLE_WORDS](#) token type and the [LOOKUP](#) command. It allows close matches in a table to be found.

If there is no exact match, then any entry in the table starting with the same first letter, and differing by only one character, will be used.

It is identical to [TOLERANCE1](#) with the following leniency in the definition of “differing by only one character”:

1. The swapping of 2 consecutive characters is considered only one difference.
2. The number of words may differ, ie. the differing character may be a space.

It is equivalent to the [DIFFS_PER_LEN](#) tolerance method, with only one number in the *tolerance* value, and with the SAME_FIRST option.

Syntax

```
token : TABLE table-name TOLERANCE2 tolerance
```

Syntax description

<i>token</i>	The name of the token being defined
<i>table-name</i>	The name of the table. Previously defined with the <code>\TABLE</code> header.
<i>tolerance</i>	The minimum number of characters that both the word in the input and the entry in the table must have, to be considered a match.

Example

```

\TABLE Localities
ALEXANDER

\GRAMMAR
Locality          : TABLE Localities TOLERANCE2 5

```

Here, “ALEXANDER” would be found if the input contained any of the following: “ALEXADNER”, “ALEXANDRE”, “ALEX ANDER” or “ALE ANDER”, which is not the case for TOLERANCE1.

Output actions

Output actions are used to extract the parsed data.

Defining

Output actions can appear in the definition of a [Composed](#) token type. They are defined by placing parenthesis after the sub-token definition. The name of the variable or output field to receive the extracted token is put in the parenthesis. For example:

```
FullName : Name(firstName) Name(lastName)
```

Here, *FullName* is defined as two *Name* tokens. The first found *Name* will be placed in the output field called *firstName*, and the second found *Name* will be placed in the output field *lastName*.

See also

[Output actions - full simple example](#)

[Which output actions are performed](#)

[Output actions on composed tokens](#)

[Output actions on table tokens](#)

Output actions - full simple example

```
\OUTPUT_FIELDS
20 firstName
30 lastName

\GRAMMAR
Name : WORD
FullName : Name(firstName) Name(lastName)
```

With an input of “John Smith”, “John” and “Smith” will be both parsed as *Name* tokens. Then, *FullName* will be parsed as two *Name* tokens, “John” and “Smith”. The first *Name* (John) will be placed in the *firstName* output field, and the second *Name* (Smith) will be placed in the *lastName* field.

Which output actions are performed

Only the output actions of the finally parsed token and its sub-tokens are performed. The output actions of those tokens not composing the final token are not performed.

Example

```
\OUTPUT_FIELDS
10 streetNumber
30 streetName
10 postalNumber

\GRAMMAR
Number : NUMBER
PostalNumber : Number(postalNumber)
PostalType : "PO BOX"
Word : WORD
StreetNumber : Number(streetNumber)
StreetName : Word(streetName)
StreetType : "ST"
Address : StreetNumber StreetName StreetType
        : PostalNumber PostalType
```

This grammar looks for an *Address*. An *Address* is defined as either a street address (*StreetNumber*, *StreetName* and *StreetType*) or a postal address (*PostalNumber* and *PostalType*). With an input of “10 MAIN ST 2000”, the grammar will recognise this as a street address, and perform the output actions for the street address – *StreetNumber*, *StreetName*. So the output will be

```
streetNumber=10
streetName=MAIN
```

While parsing for the *PostalNumber* token, both 10 and 2000 will be found. However, since the *PostalNumber* number token isn’t included in the final found token, the output action for *PostalNumber* will not be performed. So, the *postalNumber* output field will not be populated.

Similarly, while parsing for *StreetNumber*, both 10 and 2000 will be found. However, only the output action for *StreetNumber* as 10 will be performed, as it is used in the final token *Address*.

Output actions on composed tokens

If an output action is performed on a composed token type, then all the characters composing this token will be extracted to the destination, including spaces and punctuation between the individual tokens.

Example 1

```
\OUTPUT_FIELDS
40 streetAddress
40 postalAddress

\GRAMMAR
Number           : NUMBER
PostalType       : "PO BOX"
PostalAddress    : PostalType Number
Word            : WORD
StreetType       : "ST"
StreetAddress    : Number Word StreetType
Address          : StreetAddress(streetAddress)
                  : PostalAddress(postalAddress)
```

This grammar looks for an *Address*, which is defined as either a *StreetAddress* or a *PostalAddress*.

With an input of “PO BOX # 10”, the output will be “postalAddress=PO BOX # 10”. Notice that the hash (#) is included in the output. This is because the output action on *Address* will be applied to *PostalAddress*, which will includes what’s between the two sub-tokens *PostalType* and *Number*.

In order to remove the hash from the output, change the grammar to the example 2 below.

Example 2

```
\OUTPUT_FIELDS
40 streetAddress
40 postalAddress

\VARIABLES
temp

\GRAMMAR
Number           : NUMBER
PostalType       : "PO BOX"
PostalAddress    : PostalType(postalAddress) Number(temp)
                  { postalAddress = postalAddress & temp }
Word            : WORD
StreetType       : "ST"
StreetAddress    : Number Word StreetType
Address          : StreetAddress(streetAddress)
                  : PostalAddress
```

In this example, the *PostalType* and *Number* will be extracted separately, and then combined by the command `postalAddress = postalAddress & temp`. This will then not extract the hash, unlike in example 1 above.

Output actions on table tokens

If the sub-token is of a table type, then there are optional second and third output actions. The first output action will place the abbreviation in the chosen destination. The second output action will place the word from the input in the second chosen destination. The third output action will place the item found in the table.

If there is only one output action, it will place the abbreviation in the chosen destination.

An output action on this token (which is composed of the table sub-token) as a sub-token of another token, has only one output action – the input word, not the abbreviation.

Example

```
\OUTPUT_FIELDS
40 out1
40 out2
40 out3
40 address

\TABLE USStateTable
NEW YORK=NY

\GRAMMAR
Postcode      : NUMBER_SIZE_RANGE 5
USState       : TABLE USStateTable TOLERANCE2 3
Address       : USState(out1 out2 out3) Postcode
End           : Address(address)

With an input of "NEW YORC 12345" the output will be
out1=NY
out2=NEW YORC
out3=NEW YORK
address=NEW YORC 12345
```

On the *Address* token definition, the *USState* sub-token has three output actions – *out1*, *out2* & *out3*. The first output destination (*out1*) will receive the abbreviation from the table, “NY” in this case. The second output destination (*out2*) will receive the input that matched to the table entry, “NEW YORC” in this case, with the spelling error. The third output action (*out3*) will receive the word in the table that was matched, “NEW YORK” in this case, with the spelling corrected. If there was only one output action (only one destination), then it would receive the abbreviation – “NY”.

On the *End* token definition, the *Address* sub-token can only take one output destination, as *Address* is not a table token. The value placed in this output action will be directly from the input (“NEW YORC”), not the abbreviation from the table (“NY”) and without spelling correction.

Commands in token definition

When [Composed](#) tokens are found, specific [Commands](#) may also be performed. The commands are performed after the [Output actions](#). See [Parsing](#).

Example

```
FullName      : Name (firstName) Name (lastName)
               { fullName = firstName & lastName }
```

In this example, the two found *Names* are placed into *firstName* and *lastName*.

Then *fullName* is set to be the two names joined together with a space in between.

AS_ABOVE

If the commands to be performed for a token, are the same as the last set of defined commands, the word “AS_ABOVE” can be used to indicate that the previously defined set of commands should be used.

Example

```
FullName      : Name (firstName)  Name (lastName)
               { fullName = firstName & lastName }
               : Title  Name (lastName)
               : Initial (firstName)  Name (lastName)
               { AS_ABOVE }
```

Here, the first and third definitions of *FullName* will have the same commands executed for then.

How the Parser works

The parser processes each token definition from top to bottom. If it is a [Simple](#) type, it looks for the token in the input text. If it is a [Composed](#) type, it looks for the token amongst previously found tokens. All found tokens are then stored in the order in which they were found.

Once the first occurrence of the last token-definition is found, the parser stops looking. This last found token is composed of previously found occurrences of sub-tokens, which may be in turn also composed of previously found sub-tokens. The output actions and commands of all these found sub-tokens of the final token are then performed. The output actions and commands of a token are performed after those of its sub-tokens. This will be designed to produce the desired output.

See [Parsing process example](#)

See also

[Parsing](#)

Parsing process example

The parser scans the input text and previously found tokens, for each token defined. The parser starts with the first token definition, and ends with the last. If the last token is found, then the output actions of it and all its sub-tokens are performed.

Example

```
\OUTPUT_FIELDS
5          title
20         firstName
30         lastName
5          rank
\TABLE titles
MR
MRS
MISS
MS
MISTER=MR
\TABLE ranks
JR
\GRAMMAR
Word      : WORD
Title     : TABLE titles
Name      : Title(title) Word(firstName) Word(lastName)
```

Now take the following line as the input text:

```
MISTER  HARRY JONES, JR.
```

Firstly, the text will be scanned for *Words*. Four will be found – “MISTER” “HARRY”, “JONES” and “JR”.

Then it is scanned for *Titles*. Only one will be found – “MISTER”.

Then the found tokens are scanned for *Names*. Again, only one will be found, a *Title* of “MISTER”, a *Word* of “HARRY” and another *Word* of “JONES”.

Now, the found values will be placed in the output fields. “MR” will go into *title* and not “MISTER”, as the abbreviations are used with table values, “HARRY” will go into *firstName*, and “JONES” will go into *lastName*.

Composition of composed types

Now, if we add on the following lines onto the grammar file:

```
Rank           : TABLE ranks
FullName       : Name Rank(rank)
```

Then, when the *FullName* is found, the output actions of the found *Name* are performed, as above, and then the output actions of *FullName* are performed, i.e., “JR” will be placed into the *rank* output field.

Commands

There are a number of commands that can be in a grammar file, which can manipulate the input text, the output fields, and use variables defined in the grammar file.

The commands can be found in the following sections:

\INITIALISE

These commands will be executed when the grammar file is loaded, see [Initialise and Finalise commands](#).

\PRE_PARSE

These commands will be executed before the text is parsed, see [Pre and Post-parse commands](#).

\GRAMMAR

In the parsing stage in this section. These [Commands](#) will be executed together with the [Output actions](#) of tokens that make up the final token.

\POST_PARSE

These commands will be executed after the text is parsed.

\FINALISE

These commands will be executed when the grammar file is unloaded.

When are parsing stage commands performed?

If the last token is found, then the parsing stage commands begin with it:

Firstly, its output actions are performed. This could include performing first the output action and commands of its sub-tokens.

Then the commands are executed.

Example

```
\OUTPUT_FIELDS
20 firstName
20 lastName
20 order
40 all

\GRAMMAR
Name           : WORD
```

```

Comma          : ", "
FullName       : Name(lastName) Comma Name(firstName)
                { order = "last name first" }
                : Name(firstName) Name(lastName)
                { order = "first name first" }

```

If we have input of “JACKSON, ANDREW”, then “JACKSON” and “ANDREW” will fit the *Name* token. The comma will fit the *Comma* token, and the full input will fit the first *FullName* token definition. The order of output actions and commands will be as follows:

- Output actions and commands of the 1st *Name* token (there aren’t any in this example)
- “JACKSON” is outputted to *lastName*
- Output actions and commands of the *Comma* token (there aren’t any in this example)
- Output actions and commands of the 2nd *Name* token (there aren’t any in this example)
- “ANDREW” is outputted to *lastName*
- “last name first” is copied to *order*

Command syntax

Commands take one of the following formats:

operand = operand, e.g.

```
MyVariable = "HELLO"
```

operand = operand operator operand, e.g.

```
MyVariable = "GOOD" & "BYE"
```

operand = function(operand [,operand [,operand]]), e.g.

```
MyVariable = LEFT("GOOD BYE", 4)
```

IF | WHILE operand comparison operand, e.g.

```
IF MyVariable = "GOOD"
```

ELSE | END IF | WEND

Each command must be on its own line. Use the backslash (“\”) as the last character on a line to join it with the next line.

The opening and closing braces {}, which mark the beginning and end of a group of commands, may be on the same line as a command.

Text operands

These are text operands that can be used in commands.

Variable

[Text variable definitions](#) are defined in the \VARIABLES section.

Each text variable can have its own defined maximum size. If text greater than this maximum is attempted to be placed in the variable, it will be truncated.

Output field

[Output fields](#) are defined in the \OUTPUT_FIELDS section. Each output field can have its own defined maximum size. If anything greater than this maximum is attempted to be placed in the output field, it will be truncated.

Constant

Some text enclosed with quotes " ". [Escape sequences](#) may be used. For example:

```
"abcd"  
". "  
"here is a quote \" character"
```

The input text

INPUT denotes the full input text.

Defined constant

[Constants](#) are defined in the \CONSTANTS section.

Global variable

A global text variable defined in the \GLOBAL_VARIABLES section, see [Global variable definitions](#).

Integer operands

These are the integer/numerical operands that can be used in commands.

Variable

[Numeric variable definitions](#) are defined in the \NUMERIC_VARIABLES section. It can contain only integers.

Constant

Any integer. For example:

```
123  
-456  
0
```

Table operand

The names of [Tables](#) are defined in a \TABLE section.

Command summary

Available commands and their operands are listed below.

Functions

```
Result = LEFT (Source, Count)  
Result = MID(Source, From, Count)  
Result = RIGHT(Source, Count)  
  
Result = INSTR(Source, LookFor [, Start])  
Result = INSTRREV(Source, LookFor [, Start])  
Result = LEN (Source)  
  
Result = UCASE (Source)  
Result = LCASE (Source)  
Result = TCASE (Source)  
Result = PAD\_TCASE (Source)  
  
Result = LTRIM (Source [, Characters])  
Result = RTRIM (Source [, Characters])  
Result = TRIM (Source [, Characters])  
Result = INNERTRIM (Source [, Characters])  
Result = STRIP (Source [, Characters])  
Result = REPLACE (Source, LookFor [, ReplaceWith])
```

Result = [VAL](#) (Text [, DefaultValue])
 Result = [STR](#) (Number)
 Result = [JOIN](#) (SourceArray, ArraySize, Delimiter)
 ResultArray = [SPLIT](#) (Source, ArraySize, Delimiter)
 Result = [SOUNDEX](#) (Source [, Length])
 Result = [RSOUNDEX](#) (Source [, Length])
 Result = [NYSIIS](#) (Source [, Length])
 Result = [METAPHONE](#) (Source [, Length])
 Result = [LOOKUP](#) (Source, Table [, Tolerance])
 Result = [LAT_LONG_DISTANCE](#) (Coord1, Coord2 [, Precision])
 Result = [SUB_INPUT](#) (From, To)
 Result = [SUB_INPUT_FOUND](#) (From, To)
 Result = [ORDER_FOUND](#) (SubTokenNumber)
 Result = [TOKEN_POSITION](#) (SubTokenNumber)
 Result = [TOKEN_LENGTH](#) (SubTokenNumber)

Operators

= ([Assignment](#))
 + [operator \(concatenation\)](#)
 & [operator \(concatenation\)](#)
 + [operator \(addition\)](#)
 - [operator \(subtraction\)](#)
 bitwise [AND](#) operator
 bitwise [OR](#) operator
 bitwise [XOR](#) operator
 bitwise [NOT](#) operator

Conditional statements

[IF ... ELSEIF ... ELSE ... END IF](#)
[WHILE ... WEND](#)
[FOR ... NEXT](#)

DLL Functions

Result = DllFunction (Source)

Sub Grammar Calls

ResultArray = SubGrammar (Source [, ArraySize])

Locale setting

Some commands, such as [LCASE](#), [UCASE](#) and [TCASE](#) depend on a locale. For different locales, the uppercase of words will be different. If no locale is set, then uppercase and lowercase will only apply to letters in the range A-Z and a-z, excluding any accented characters. Other locales (perhaps “iso 8859-1” or “English”) may include accented characters in uppercase and lowercase.

The setting of the locale is done in the registry or configurations file.

Setting the locale to nothing, as in the example below, will use the system default locale.

Window Registry example

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Intech\StanRt]
"Locale"=""
```

Configuration Text File example

```
StanRt/Locale=
```

Operators

(Assignment)

Copies a value.

Syntax

```
Destination = Source
```

Syntax description

Source and Destination must be of the same type. Either both [Text operands](#) or both [Integer operands](#). Destination cannot be a constant.

+ operator (addition)

Syntax

```
Result = Number1 + Number2
```

Syntax description

Result, Number1 and Number2 must all be [Integer operands](#). Result cannot be a constant

- operator (subtraction)

Syntax

```
Result = Number1 - Number2
```

Syntax description

Result, Number1 and Number2 must all be [Integer operands](#). Result cannot be a constant

+ operator (concatenation)

Syntax

```
Result = Text1 + Text2
```

Syntax description

Result, Text1 and Text2 must all be [Text operands](#). Result cannot be a constant

& operator (concatenation)

Concatenates text, placing a space between them.
(Result = Text1 + " " + Text2)

If either text is empty, no space will be added.

Syntax

```
Result = Text1 & Text2
```

Syntax description

Result, Text1 and Text2 must all be [Text operands](#). Result cannot be a constant

Bitwise operators

Performs a bitwise-AND, bitwise-OR, bitwise-exclusive-OR or bitwise-NOT (otherwise known as one's complement).

Syntax

```
Result = Number1 AND Number2
```

```
Result = Number1 OR Number2
```

```
Result = Number1 XOR Number2
```

```
Result = NOT Number1
```

Syntax description

Result, Number1 and Number2 must all be [Integer operands](#). Result cannot be a constant

Combined operators

The bitwise-NOT operator can be combined with other bitwise operators, for example:

```
Result = Number1 AND NOT Number2
```

```
Result = Number1 OR NOT Number2
```

```
Result = Number1 XOR NOT Number2
```

Functions

Function - INNERTRIM

Returns a copy the text with multiple spaces replaced by single spaces.

Syntax

```
Result = INNERTRIM (Source, Characters)
```

Syntax description

Result a non-constant [Text operand](#).

Source a Text operand

Characters an optional Text operand with a list of characters to be considered as space characters.
If omitted, the space character is assumed.

Notes

Any sequence of more than one space character (as defined by the *Characters* parameter) will be replaced by the first occurring space character.

Example

```
S = INNTERTRIM("one    two-= three==", " -=")
```

will return "one two-three=".

Function - INSTR

Returns the position of the first occurrence of some text within some other text.

Syntax

```
Result = INSTR (Source, TextToFind, Start)
```

Syntax description

Result a non-constant [Integer operand](#)

Source a Text operand, the text being searched

TextToFind a Text operand, the sought text

Start an optional integer operand, the position in Source to begin the search.

Notes

Positions are base 1. So if TextToFind occurs at the beginning of Source, then 1 will be returned.

If TextToFind is not found, 0 will be returned.

If TextToFind is empty, Start will be returned.

Example

See [Example](#)

Function - INSTRREV

Returns the position of the last occurrence of some text within some other text.

Syntax

```
Result = INSTRREV (Source, TextToFind, Start)
```

Syntax description

Result	a non-constant Integer operand
Source	a Text operand, the text being searched
TextToFind	a Text operand, the sought text
Start	an optional integer operand, the position in Source to begin the search.

Notes

Positions are base 1. So if the only occurrence of TextToFind occurs at the beginning of Source, then 1 will be returned.

If TextToFind is not found, 0 will be returned.

If TextToFind is empty, 0 will be returned.

Function - JOIN

Combines a set of consecutive variables (or output fields) into one delimited text.

Useful for outputting fields in delimited format.

Syntax

```
Result = JOIN (SourceArray, ArraySize, Delimiter)
```

Syntax description

Result	a non-constant Text operand
SourceArray	an output field or text variable This output field (or variable) is the first element in the array. The variables (or output fields) defined after this one, will be the subsequent elements in the array.
ArraySize	an Integer operand , the number of elements in the array.
Delimiter	a Text operand. The text (usually one character, such as a comma), that is to delimit the array elements

Example

See [Example](#).

Function - LAT_LONG_DISTANCE

Calculates the distance between two points given their latitude and longitude.

Syntax

```
Result = LAT_LONG_DISTANCE (Coords1, Coords2, Precision)
```

Syntax description

Result	a non-constant Text operand .
Coords1	a Text operand, specifying the coordinates of the first point.

Coords2	a Text operand, specifying the coordinates of the second point.
Precision	an optional Integer operand – the number of decimal places to be put in the result. Values between 0 and 6 are valid, with default of 3 used if omitted.

Notes

The coordinates of the two points are to be provided as two numbers separated by another character such as a space or a comma. The first number is the latitude and the second number is the longitude. These numbers are in Degrees without Minutes, Seconds or Directions. For example, 33° 30' S 151° 15' W would be represented by –33.5 151.25.

The result is in kilometres.

Example

```
S = LAT_LONG_DISTANCE("-33.87,151.21", "-37.82,144.97", 0)
```

would give the distance between Sydney and Melbourne “as the crow flies”.

Function - LCASE

Returns a copy of text converted into lowercase.

Syntax

```
Destination = LCASE (Source)
```

Syntax description

Result and Source must be [Text operands](#). Result cannot be a constant

Example

See [Example](#)

Function - LEFT

Returns a specified number of characters from the left (beginning) of some text.

Syntax

```
Result = LEFT(Source, Count)
```

Syntax description

Result	a non-constant Text operand
Source	a Text operand to get the characters from
Count	an Integer operand - the maximum number of characters to take

Example

See [Example](#)

Function - LEN

Returns the number of characters the text.

Syntax

```
Result = LEN (Source)
```

Syntax description

Result	a non-constant Integer operand .
Source	a Text operand

Function - LOOKUP

Looks up a value in a table.

If the value is found, its abbreviation is returned, else empty text is returned.

Syntax

```
Result = LOOKUP(Value, Table, Tolerance)
```

Syntax description

Result	a non-constant Text operand .
Value	a Text operand
Table	the Table operand
Tolerance	an optional Tolerance method with its parameters. These begin with the word TOLERATE.

Example

See [Example](#)

Function - LTRIM

Returns a copy the text without specific leading characters.

Syntax

```
Result = LTRIM (Source, Characters)
```

Syntax description

Result	a non-constant Text operand .
Source	a Text operand
Characters	an optional Text operand with a list of characters. Any character in Characters will be removed from the beginning of the text. If omitted, the space character is assumed.

Example

See [Example](#)

Function - METAPHONE

Returns the METAPHONE code of the text.

Syntax

```
Result = METAPHONE(Source, Length)
```

Syntax description

Result	a non-constant Text operand .
Source	a Text operand
Length	an optional Integer operand – the number of characters the result is to contain. If omitted, the default of 4 is used.

Example

See [Example](#)

Function - MID

Returns a specified number of characters from the middle of some text.

Syntax

```
Result = MID(Source, From, Count)
```

Syntax description

Result	a non-constant Text operand
Source	a Text operand to get the characters from
From	an Integer operand - the starting position in Source to take the characters from. 1 is the first character of the text, 2 the second etc. Ignored if not positive
Count	an optional Integer operand - the maximum number of characters to take. If omitted, then all characters till the end will be taken

Example

See [Example](#)

Function - NYSIIS

Returns the NYSIIS code of the text.

Syntax

```
Result = NYSIIS(Source, Length)
```

Syntax description

Result	a non-constant Text operand .
Source	a Text operand
Length	an optional Integer operand – the number of characters the result is to contain. If omitted, DEFAULT_NYSIIS is used.

Example

See [Example](#)

Function - ORDER_FOUND

Returns the order in which a particular sub-token is found in a composite token.

This command can only appear in the \GRAMMAR section, and only for a [Composed](#) token definition.

Syntax

```
Result = ORDER_FOUND (SubToken)
```

Syntax description

Result	a non-constant Integer operand .
SubToken	an Integer operand with the sub-token number. 1 being the first sub-token, 2 being the second, etc.

Notes

If there are no optional sub-tokens and no “Any-order” qualifiers, then the result of this command is trivial.

Example

See [Example](#).

Function - PAD_TCASE

Returns a copy of the text with spaces inserted to break up words. Words begin with an uppercase letter followed by lowercase letters.

Syntax

```
Result = PAD_TCASE (Source)
```

Syntax description

Result and Source must be [Text operands](#). Result cannot be a constant

Example

```
PAD_TCASE("IntechSolutionsPtyLtd")
```

will return “Intech Solutions Pty Ltd”.

Function - REPLACE

Returns a copy of the text with specific characters replaced by some other text.

Syntax

```
Result = REPLACE (Source, Characters, ReplaceWith)
```

Syntax description

Result	a non-constant Text operand .
Source	a Text operand
Characters	a Text operand containing a list of characters. Any character in Character found in the text will be replaced.
ReplaceWith	an optional Text operand containing the text to replace the found characters with. If omitted, a space is used as default. If empty, all occurrences of the found characters will be replaced with nothing; ie they will be removed.

Example

See [Example](#)

Function - RIGHT

Returns a specified number of characters from the right (end) of some text.

Syntax

```
Result = RIGHT (Source, Count)
```

Syntax description

Result	a non-constant Text operand
Source	a Text operand to get the characters from
Count	an Integer operand - the maximum number of characters to take

Example

See [Example](#)

Function - RSOUNDEX

Returns the soundex code of the text reversed.

Syntax

```
Result = RSOUNDEX (Source, Length)
```

Syntax description

Result	a non-constant Text operand .
Source	a Text operand
Length	an optional Integer operand – the number of characters the result is to contain. If omitted, the default of 4 is used. Note that the original SOUNDEX code has only 4 characters.

Example

See [Example](#)

Function - RTRIM

Returns a copy the text without specific trailing characters.

Syntax

Result = RTRIM (Source, Characters)

Syntax description

Result	a non-constant Text operand .
Source	a Text operand
Characters	an optional Text operand with a list of characters. Any character in Characters will be removed from the end of the text. If omitted, the space character is assumed.

Example

See [Example](#)

Function - SOUNDEX

Returns the soundex code of the text.

Syntax

Result = SOUNDEX (Source, Length)

Syntax description

Result	a non-constant Text operand .
Source	a Text operand
Length	an optional Integer operand – the number of characters the result is to contain. If omitted, the default of 4 is used. Note that the original SOUNDEX code has only 4 characters.

Example

See [Example](#)

Function - SPLIT

Breaks up delimited text into variables or output fields.

Useful for converting a delimited record into a fixed width record.

Syntax

ResultArray = SPLIT (Source, ArraySize, Delimiter)

Syntax description

ResultArray	an output field or text variable This output field (or variable) will receive the first element (the text in Source from the beginning till the first delimiter). The variables (or output fields) defined after this one, will receive subsequent elements in the array.
Source	a Text operand
ArraySize	an Integer operand , the number of elements in the array. (Often, also the number of delimited elements expecting).
Delimiter	a Text operand. The text (usually one character, such as a comma), that delimits the elements in Source

Example

See [Example](#).

Function - STR

Returns a textual representation of a number.

This is needed because the operands of most commands are text and not numeric.

Syntax

```
Result = STR(Source)
```

Syntax description

Result a non-constant [Text operand](#)

Source an [Integer operand](#).

Example

See [Example](#)

Function - STRIP

Returns a copy of the text with specific characters removed.

Syntax

```
Result = STRIP (Source, Characters)
```

Syntax description

Result a non-constant [Text operand](#).

Source a Text operand

Characters an optional Text operand with a list of characters. Any character in Characters will be removed from the text. If omitted, the space character is used as default.

Example

See [Example](#)

Function - SUB_INPUT

Returns a copy of part of the input text, based on the position of the tokens.

This command can only appear in the \GRAMMAR section, and only for a [Composed](#) token definition.

Syntax

```
Result = SUB_INPUT(From, To)
```

Syntax description

Result a non-constant [Text operand](#)

From an [Integer operand](#), describing the position to copy from. 0 is the beginning of the input text; -1 is the beginning of the first token; 1 is the end of the first token; -2 is the beginning of the second token; 2 is the end of the second token. Etc.

To an optional Integer operand, describing the position to copy to. Numbering as with From If omitted, will copy to the end of the input text.

Notes

If specified sub-token is not found (e.g. the sub-token is optional), the beginning of the next sub-token will be used.

If the composed token contains an [Optional sub-token](#), or the sub-token can be in [Any order](#), then consider using [SUB_INPUT_FOUND](#). See its definition for a comparison.

Example

See [Example](#)

Function - SUB_INPUT_FOUND

Returns a copy of part of the input text, based on the position of found tokens.

This command can only appear in the \GRAMMAR section, and only for a [Composed](#) token definition.

Syntax

```
Result = SUB_INPUT_FOUND (From, To)
```

Syntax description

See [SUB_INPUT](#).

SUB_INPUT vs. SUB_INPUT_FOUND

With SUB_INPUT_FOUND, the position is based on the order that the found sub-token appear in the input text.

With SUB_INPUT, the position is based on the order that the sub-tokens are defined in the composed token definition.

See the example below.

If there are no optional sub-tokens and no [Any order](#) qualifiers, the two commands will give the same result, with SUB_INPUT being the preferred choice.

Example

See [Example](#).

Function - TCASE

Returns a copy of the text converted into Title Case.

That is, the first letter of each word is uppercase, and the other letters in the word are lowercase.

Syntax

```
Result = TCASE (Source)
```

Syntax description

Result and Source must be [Text operands](#). Result cannot be a constant.

Example

See [Example](#)

Function - TOKEN_POSITION

Returns the position of the token found in the input text.

This command can only appear in the \GRAMMAR section, and only for a [Composed](#) token definition.

Syntax

```
Result = TOKEN_POSITION (SubToken)
```

Syntax description

Result a non-constant [Integer operand](#).

SubToken an [Integer operand](#), which token (1 is the first token, 2 is the second etc.).

Example

See [Example](#).

Function - TOKEN_LENGTH

Returns the length of the found token.

This command can only appear in the \GRAMMAR section, and only for a [Composed](#) token definition.

Syntax

```
Result = TOKEN_LENGTH (SubToken)
```

Syntax description

Result a non-constant [Integer operand](#).

SubToken an [Integer operand](#), which token (1 is the first token, 2 is the second etc.).

Example

See [Example](#).

Function - TRIM

Returns a copy the text without specific leading and trailing characters.

Syntax

```
Result = TRIM (Source, Characters)
```

Syntax description

Result a non-constant [Text operand](#).

Source a Text operand

Characters an optional Text operand with a list of characters. Any character in Characters will be removed from the beginning and end of the text. If omitted, the space character is assumed.

Example

See [Example](#)

Function - UCASE

Returns a copy of text converted into uppercase.

Syntax

```
Destination = UCASE (Source)
```

Syntax description

Result and Source must be [Text operands](#). Result cannot be a constant.

Example

See [Example](#)

Function - VAL

Converts the text into a number.

If the text represents a number, then the integer value of this number will be returned. If it does not, the default value will be returned.

This command is needed as operands of certain commands need to be numeric.

Syntax

```
Result = VAL (Source, DefaultValue)
```

Syntax description

Result a non-constant [Integer operand](#).

Source a [Text operand](#)
DefaultValue an optional [Integer operand](#). If not provided, zero will be used.

Example

See [Example](#)

Conditional commands

IF ... ELSEIF ... ELSE ... END IF

Conditionally executes a series of commands, depending on the result of a comparison.

Syntax

```
IF Operand1 Comparison Operand2 [THEN]
    Commands
[ELSEIF Operand1 Comparison Operand2 [THEN]
    Commands]
[ELSE
    ElseCommands]
END IF
```

Syntax description

Operand1 either a [Text operand](#) or an [Integer operand](#).
Operand2 an operand of the same type as Operand1. (Either both Text operands or both Integer operands).
Comparison one of <, >, =, <>, <=, >=.
Commands command(s) to be executed if comparison true.
ElseCommands command(s) to be executed if comparison false.

Notes

The word “THEN” is optional.
The ELSEIF section is optional, and there may be any number of them.
The ELSE section is optional.
Textual comparisons are case-sensitive.
Nesting is allowed.
Every IF must have a matching END IF.
An IF block may not span files (using \INCLUDE).

Example

See [Example](#)

WHILE ... WEND

Repeats a series of commands, as long as a comparison is true.

Syntax

```
WHILE Operand1 Comparison Operand2
    Commands
```

WEND

Syntax description

Operand1	either a Text operand or an Integer operand .
Operand2	an operand of the same type as Operand1. (Either both Text operands or both Integer operands).
Comparison	one of <, >, =, <>, <=, >=.
Commands	command(s) to be repeated while the comparison is true.

Notes

Textual comparisons are case-sensitive.

Nesting is allowed.

Every WHILE must have a matching WEND.

A WHILE block may not span files (using \INCLUDE).

Example

See [Example](#)

FOR ... NEXT

Repeats a series of commands a certain number of times.

Syntax

```
FOR LoopVariable = FromValue TO ToValue [STEP StepValue]
```

```
    Commands
```

```
NEXT
```

Syntax description

LoopVariable	a non-constant Integer operand .
FromValue	an Integer operand , the value that the LoopVariable will start with.
ToValue	an Integer operand , the value that the LoopVariable will end with. The LoopVariable may not be used for this operand.
StepValue	an Integer operand , the value to increase the LoopVariable by for each loop. If not provided, then the value of 1 is used. The LoopVariable may not be used for this operand.
Commands	command(s) to be repeated each loop.

Notes

Nesting is allowed.

Every FOR must have a matching NEXT.

StepValue may be negative, in which case the LoopVariable will go down from FromValue to ToValue.

A FOR...NEXT loop may not span files (using \INCLUDE).

Example

See [Example](#)

DLL Functions

DLL function as defined in [Dynamic Linked Library \(DLL\) definitions](#) (see there for more details), can be called as any other functions.

Syntax

```
Result = Function (Source)
```

```
Function (Source)
```

Syntax description

Result	a Text operand
Function	the alias of a previously defined DLL function
Source	a Text operand

Notes:

The contents of Source will be copied to a character buffer. A character pointer (LPTSTR or char*) to this buffer will be passed to the DLL function, together with the maximum allowed length of Result. The contents of this buffer after the call will then be copied to Result.

So if the function does not expect an input, put Source as "".

If you are not interested in the contents of the buffer after the call, use the second syntax above.

Example

See [Example](#).

Sub Grammar Calls

Sub Grammars are defined in the [Sub Grammars](#) section.

They can be called like any other function.

The source text is passed to the sub-grammar, and the output fields of the sub-grammar are placed into an array.

Syntax

```
ResultArray = SubGrammar (Source, ArraySize)
```

Syntax description

ResultArray	an output field or text variable This output field (or variable) will receive the first output field of the sub-grammar. The variables (or output fields) defined after this one, will receive subsequent output fields.
SubGrammar	the alias of a previously defined sub-grammar
Source	a Text operand
ArraySize	an optional Integer operand , the number of elements in the array. If omitted, then the maximum is used, which is the number of variables (or output fields) defined after this one and including this one.

Notes:

If the size of the array is greater than the number of output fields, the extra array elements will remain untouched.

If the size of the array is less than the number of output fields, the extra output fields will be ignored.

Example

See [Example](#).

Command examples

Example - Nysiis, Soundex, RSoundex, Metaphone

```
\OUTPUT_FIELDS
50      name
4       soundex
4       rsoundex
8       nysiis
4       metaphone

\GRAMMAR
Word    : WORD
Final   : Word(name)
        {
            soundex = SOUNDEX(name)
            rsoundex = RSOUNDEX(name)
            nysiis = NYSIIS(name)
            metaphone = METAPHONE(name)
        }
```

The above example will output the word provided, with four phonetic codes of the word – Soundex, Soundex of its reverse, NYSIIS and Metaphone. With an input of Intech, the output will be:

```
name:      Intech
soundex:   I532
rsoundex:  H325
nysiis:    INTAC
metaphone: INTX
```

Example - Left, Right, Mid, INPUT

```
\OUTPUT_FIELDS
20      left
20      right
20      mid

\PRE_PARSE
        {
            left = LEFT (INPUT, 5)
            right = RIGHT (INPUT, 5)
            mid = MID (INPUT, 5, 10)
        }
```

The above example will output the first five characters of the input, the last five, and ten characters starting with the fifth character. With an input of “abcdefghijklmnopqrstuvwxy^z”, the output will be:

```
left:      abcde
right:     vwxyz
mid:       efghijklmn
```

Example - UCase, LCase, TCase

```
\OUTPUT_FIELDS
50      name
50      upper
50      lower
50      title

\GRAMMAR
Word    : WORD
Words   : Word Word? Word? Word?
```

```

Final          : Words (name)
                {
                  upper = UCASE (name)
                  lower = LCASE (name)
                  title = TCASE (name)
                }

```

The above example will output the first 4 words provided, and with in upper, lower and title case. With an input of “INTECH solutions, Pty, LTD”, the output will be:

```

name:          INTECH solutions, Pty, LTD
upper:         INTECH SOLUTIONS, PTY, LTD
lower:         intech solutions, pty, ltd
title:         Intech Solutions, Pty, Ltd

```

Example - Lookup

```

\OUTPUT_FIELDS
3          state
30         capital

\TABLE capital_cities
NSW=SYDNEY
VIC=MELBOURNE
QLD=BRISBANE
\GRAMMAR
State      : WORD
Final      : State (state)
            {
              capital = LOOKUP (state, capital_cities) }

```

The above example will take the first word in the input, and look it up in the table of states. If it is in the table, the capital city of that state will be returned. With an input of “NSW”, the output will be:

```

state:        NSW
capital:      SYDNEY

```

Example - Assignment, Rtrim, Ltrim, Trim, Strip, Replace

```

\OUTPUT_FIELDS
30         input
30         rtrim
30         ltrim
30         trim
30         strip
30         repla

\POST_PARSE
{
  input = INPUT
  rtrim = RTRIM (input, "0123456789")
  ltrim = LTRIM (input, "0123456789")
  trim = TRIM(input, "0123456789")
  strip = STRIP (input, "0123456789")
  repla = REPLACE (input, "0123456789")
}

```

The above example will return the input, with the numbers removed from the end of the input, from the beginning, everywhere, and with leaving spaces when removed, respectively. With an input of “123abc456def7890”, the output will be:

```

input: 123abc456def7890
rtrim: 123abc456def

```

```
ltrim: abc456def7890
trim : abc456def
strip: abcdef
repla: (3 spaces) abc (3 spaces) def
```

Example - Concatenation

```
\OUTPUT_FIELDS
30      fullName
30      oneWord

\VARIABLES
sFirstName
sMiddleName
sLastName

\GRAMMAR
Word      : WORD
Name      : Word(sFirstName) Word(sMiddleName) Word(sLastName)
{
    fullName = sFirstName & sMiddleName
    fullName = fullName & sLastName
    oneWord = sFirstName + sMiddleName
    oneWord = oneWord + sLastName
}
```

The above example will take the first three words of the input, and return them with one space in between each word, and as one long combined word. With an input of “John - Howard - Smith”, the output will be:

```
fullName : John Howard Smith
oneWord  : JohnHowardSmith
```

Example - While, Subtraction, Val, Str

```
\OUTPUT_FIELDS
30      out

\VARIABLES
s

\NUMERIC_VARIABLES
i
\POST_PARSE {
    i = VAL(INPUT)
    WHILE i > 0
        s = STR(i)
        out = out & s
        i = i - 1
    WEND
}
```

The above example will take the first number in the input, and count down to 1 from it. E.g, with an input of “9”, the output will be:

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

Example - For ... Next, Val, Str

```
\OUTPUT_FIELDS
30      out

\NUMERIC_VARIABLES
```

```

i

\ VARIABLES
s

\ POST_PARSE {
    i = VAL (INPUT)
    FOR i = i TO 1 STEP -1
        s = STR (i)
        out = out & s
    NEXT
}

```

The above example will take the first number in the input, and count down to 1 from it. E.g, with an input of “9”, the output will be:

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

Example - If ... Elseif ... Else ... End If

```

\ OUTPUT_FIELDS
30          out

\ VARIABLES
s1
s2
\ GRAMMAR
Word          : WORD
TwoWords      : Word(s1) Word(s2)

\ POST_PARSE {
    IF s1 = s2
        out = "The same"
    ELSEIF s1 < s2
        out = "First one before Second"
    ELSE
        out = "Second one before First"
    END IF
}

```

The above example will compare the first two words in the input, and output an appropriate comment. E.g, with an input of “hello everybody”, the output will be:

```
Second one before First
```

Example - InStr, Left, Mid

```

\ OUTPUT_FIELDS
30          before
30          after

\ NUMERIC_VARIABLES
i
\ POST_PARSE {
    i = INSTR (INPUT, ",")
    IF i > 0 THEN
        i = i - 1
        before = LEFT (INPUT, i)
        i = i + 2
        after = MID (INPUT, i)
    END IF
}

```

The above example will find the position of the first comma in the input. The output will be the input before the comma and the input after the comma. With an input of “hello,world”, the output will be:

```
Before: hello
After : world
```

Example - SubInput

```
\OUTPUT_FIELDS
30          before
30          in
30          after

\GRAMMAR
Quote: ""
Quotes: Quote Quote @
#           @ means that something else is allowed between the Quotes
#           {   before = SUB_INPUT (0, -1)
#               # from the beginning of the input till the first Quote
#               in = SUB_INPUT (1, -2)
#               # from after the first Quote, till the second Quote
#               after = SUB_INPUT (2)
#               # from after the second Quote, till the end of the
input
#           }
```

The above example will extract what’s before the quotation, what’s in the quotation and what’s after it. With an input of “This is the 'best' standardiser”, the output will be:

```
Before      :This is the
In          :best
After       : standardiser
```

Example - SubInputFound

```
\OUTPUT_FIELDS
30          before
30          in
30          after

\GRAMMAR
Slash       : "/"
Backslash   : "\"
Slashes     : Slash Backslash ~ @
#           @ means that something else is allowed between the slashed
#           ~ means that the slashes are not necessarily in that
order.
#           {   before = SUB_INPUT_FOUND (0, -1)
#               # from the beginning of the input till the first slash
#               in = SUB_INPUT_FOUND (1, -2)
#               # from after the first slash, till the second slash
#               after = SUB_INPUT_FOUND (2)
#               # from after the second slash, till the end of the
input
#           }
```

The above example will extract what’s before the slashes, what’s between the slashes and what’s after it. With an input of “This is the \best/ standardiser”, the output will be:

```
Before      :This is the
In          :best
```



```
After          : standardiser
```

If the SUB_INPUT command had been used instead, then the “1” would always refer to the *Slash* and the “2” to the *Backslash*. So with the about input of “This is the \best/ standardiser”, the output would be:

```
Before          : This is the \best
In              :
After           : best/ standardiser
```

Example - Token_Position, Token_Length

```
\OUTPUT_FIELDS
10          position
10          length

\nUMERIC_VARIABLES
i
\GRAMMAR
Number      : NUMBER
Final       : Number
            { i = TOKEN_POSITION(1)
              position = STR(i)
              i = TOKEN_LENGTH(1)
              length = STR(i)
            }
```

The above example will give the position and length of the first number in the input. With an input of “Here's a number: 9876”, the output will be:

```
position: 18
length  : 4
```

Example - DLL Function

```
\OUTPUT_FIELDS
50          sysdir

\DLL kernel32.dll
GetSystemDirectoryA ZERO_ERROR

\PRE_PARSE {
            sysdir = GetSystemDirectoryA ("") }
```

The above example will return the systems directory.

Example - Sub Grammar call

```
\OUTPUT_FIELDS
50          out1
50          out2
50          out3
50          out4

\SUB_GRAMMARS
Sub         Grammar2.grm

\PRE_PARSE {
            IF INPUT = "" THEN
                out1 = "empty"
            ELSE
                out1 = Sub(INPUT, 4)
            END IF
```

```
}
```

The above example assumes that there is another grammar file (used as a sub-grammar) called “Grammar2.grm” that returns 4 output fields.

The main grammar file will return “empty” in its first output field if there is a blank input, else it will return what the sub-grammar returns. It does this by passing its input to the sub-grammar, and filling its own 4 output fields with the output fields of the sub-grammar.

Example - Join

```
\OUTPUT_FIELDS
50                                out

\VARIABLES
title
first_name
last_name

\GRAMMAR
Word                               : WORD
FullName                           : Word(title) Word(first_name) Word(last_name)
                                   { out = JOIN(title, 3, ",") }
```

The above example will take the first 3 words of the input, and return them, separated by commas. With an input of “Mr. John Citizen”, the output would be

Mr,John,Citizen

Example - Split

```
\OUTPUT_FIELDS
4                                title
15                               first_name
20                               last_name

\PRE_PARSE {
                                title = SPLIT (INPUT, 3, ",") }
```

The above example expects the input to be the title, first name and last name, divided by commas. It will extract these elements, and return them. With an input of “Mr,John,Citizen”, the output would be:

```
title           : Mr
first_name      : John
last_name       : Citizen
```

Example - OrderFound

```
\OUTPUT_FIELDS
2                                word_position
2                                number_position
2                                dash_position

\NUMERIC_VARIABLES
i

\GRAMMAR
Word  : WORD
Number: NUMBER
Dash  : "-"
End   : Word Number Dash? ~
      { i = ORDER_FOUND(1)
        word_position = STR(i)
```

```
i = ORDER_FOUND(2)
number_position = STR(i)
i = ORDER_FOUND(3)
dash_position = STR(i) }
```

The above example expects the input to contain a word, a number and possibly a dash. It will tell you in what order these 3 where in the input. With an input of “1 – Hello”, the output would be:

```
word_position      : 3
number_position    : 1
dash_position      : 2
```

Reference

Limits

There are several limits imposed on the grammar file, to allow speed and efficiency. If some of these limits are too restricting for a particular need, please contact your supplier for a version with the limit set higher.

MAX_SUBTOKENS

The maximum number of sub-tokens that a composed token definition can have.

Current Value

20

Example of passing the limit

```
\GRAMMAR
W                : WORD
Words21          : W W W W W W W W W W W W W W W W W W W W W
```

MAX_TOKENS_FOUND

The maximum number of tokens allowed to be found

Current Value

1000

Example of passing the limit

```
\GRAMMAR
Word            : WORD
ThreeWords      : Word Word Word @~
Result         : ThreeWords
```

This grammar will first find all words. Then, it will find all combinations of 3 *Words*, in any order. If the input contains 11 words, then there will be 11 *Word* tokens found, plus 11x10x9=990 *ThreeWords* tokens found. This will go over the limit.

MAX_LINE

The maximum number of characters allowed on each line in the grammar file.

Current Value

1500

MAX_NAME

The maximum length of a token name, table name, output field name, function name alias or variable name.

Current Value

30

Example of passing the limit

```
\TABLE This_Table_Name_Is_Just_Too_Long
```

MAX_QUOTES

The maximum number of characters allowed between quotes.

This might affect the length of number of characters in a [WORDOF](#) token type, or the length of a literal, if it is not otherwise limited by another limit.

Current Value

256

MAX_NYSIIS

The maximum number of characters used as input for calculating the NYSIIS. This value is amply enough.

Current Value

50

MAX_NESTING

The maximum number of IF, WHILE and FOR nests allowed.

Current Value

20

MAX_DLL_PATH

The maximum length allowed for the path of a DLL defined in the \DLL section, see [Dynamic Linked Library \(DLL\) definitions](#)

Current Value

255

MAX_DESCRIPTION

The maximum length allowed for the description of the grammar file.

Current Value

255

MAX_FIELD_DESCRIPTION_NEW

The maximum length allowed for the description of an output field.

Current Value

255

Old value

Previous versions had a maximum value of 50. This value still applies to the `Get_OutputFieldDescriptions` function.

MAX_SUBGRAMMAR_NAME

The maximum length of sub-grammar name that can be defined in the [Sub Grammars](#) section.

Current Value

255

DEFAULT_NYSIIS

The default length of the NYSIIS coding, if a length is not specified.

Current Value

8

Error messages

The Standardiser will return an error number if one has occurred. When [Debugging with StanConsole](#) to test the grammar file, the line where the error occurred will be displayed.

See [IQStanSDK.doc:TestStan](#)

Error 1 - ERR_GRAMMAR_LINE_TOO_LONG

The length of the grammar line is greater than [MAX_LINE](#).

Error 2 - ERR_GRAMMAR_NO_TABLE_NAME

A table is defined without a name, e.g.

```
\TABLE
```

A table is referenced to in a token definition with no name, e.g.

```
titles          : TABLE
```

Error 3 - ERR_GRAMMAR_NAME_TOO_LONG

The name of a table, output field, function alias, token or sub-token is greater than [MAX_NAME](#).

Error 4 - ERR_GRAMMAR_MISSING_HEADER

A line not within any section. Sections begin with a header such as \GRAMMAR, \OUTPUT_FIELDS, \TABLE, \PRE_PARSE, \POST_PARSE, \VARIABLES, \NUMERIC_VARIABLES. Sections end with the beginning of another section or with a global qualifier such as \DELIMITERS or \CASE_SENSITIVE.

Error 6 - ERR_GRAMMAR_NO_WORD

Expecting an entry in a table before the abbreviation, e.g.

```
\TABLE tableName  
=abbreviation
```

Error 8 - ERR_GRAMMAR_NO_ABBR

Expecting an abbreviation after the table entry and the equals sign "=", e.g. after "entry" below

```
\TABLE tableName  
entry=
```

Error 9 - ERR_GRAMMAR_NO_LENGTH

Missing, zero or negative length of an output field, e.g. before the "fieldOne" and "fieldTwo" below

```
\OUTPUT_FIELDS  
fieldOne  
-1          fieldTwo
```

Error 10 - ERR_GRAMMAR_NO_OUTFIELD_NAME

Missing name of an output field, e.g. after the number 10 below

```
\OUTPUT_FIELDS  
10
```

Error 12 - ERR_GRAMMAR_NO_COLON

Expecting a colon ":" before a token definition, e.g. before "WORD" in below

```
\GRAMMAR
```

Name WORD

Error 14 - ERR_GRAMMAR_DUP_TABLE_DEF

Two tables given the same name, eg.

```
\TABLE cities
NEW YORK
\TABLE cities
WASHINGTON
```

Error 15 - ERR_GRAMMAR_DONT_KNOW_TABLE

Expecting a table name, but no table of this name previously defined, e.g., if there is no table called *cities*, then two of these lines below would give errors

```
\GRAMMAR
City                      : TABLE cities
                          { paris = LOOKUP ("PARIS", cities) }
```

Error 16 - ERR_GRAMMAR_NO_TOKENDEF_NAME

Expecting the name of a token, before the colon and its definition, unless it is another definition of the previous token. E.g.

```
\GRAMMAR
                          : WORD
```

Error 17 - ERR_GRAMMAR_MAX_LESSTHAN_MIN

In the definition of a token which has a range, the first number was greater than the second. It should be the reverse, e.g.

```
\GRAMMAR
postCode                 : NUMBER_RANGE 9999 1
```

Error 18 - ERR_GRAMMAR_NO_RANGE

In the definition of a token which has a range, a number is expected after the word RANGE, e.g.

```
\GRAMMAR
postCode                 : NUMBER_RANGE
telephone                : NUMBER_RANGE eight nine
```

Error 19 - ERR_GRAMMAR_NONPOSITIVE_RANGE

In the definition of a token which has a size range, the size cannot be zero or negative, e.g.

```
\GRAMMAR
postCode                 : NUMBER_SIZE_RANGE 0 5
```

Error 20 - ERR_GRAMMAR_NO_SUBTOKENDEFS

After the colon in a token definition, expecting the name of a sub-token or a simple type, e.g.

```
\GRAMMAR
postCode                 : 2000
```

This should be replaced with

```
postCode                 : "2000"
```

Error 21 - ERR_GRAMMAR_DONT_KNOW_TOKENDEF

Expecting a simple type or the name of a sub-token, but cannot find it. Perhaps it is misspelled, e.g.

```
\GRAMMAR
Name : WORD
```

This should be replaced with

```
Name : WORD
```

Error 22 - ERR_GRAMMAR_TOOMANY_SUBTOKENS

A composed token can only have [MAX_SUBTOKENS](#) number of sub-tokens.

Error 23 - ERR_GRAMMAR_NO_OUTPUTFIELD

Expecting an output field for the sub-token, in the parentheses e.g.

```
\GRAMMAR
FullName          : Name ()
```

Error 24 - ERR_GRAMMAR_DONTKNOW_DESTINATION

The given destination is not a pre-defined output field or text variable. Perhaps it is misspelled, e.g.

```
\OUTPUT_FIELDS
10          firstName
\GRAMMAR
Namze       : WORD
FullName    : Name (FirstName)
```

Here, the output field is misspelled, as it should begin with a lowercase f.

Error 25 - ERR_GRAMMAR_MISSING_BRACKET

After the output action, there must be a closing parenthesis, e.g.

```
\GRAMMAR
Name          : WORD
FullName      : Name (FirstName
```

Error 26 - ERR_GRAMMAR_SYNTAX_ERROR

This error occurs if there is extra text on the line in the grammar file, after the line has already been fully understood. Can often occur, if the line's syntax is incorrect.

Error 27 - ERR_GRAMMAR_NO_QUOTES

Expecting an opening quotation mark, such as after the token type WORDOF, or after DELIMITERS, e.g.

```
\DELIMITERS ,.-
\GRAMMAR
Hex : WORDOF 01213456789abcdefABCDEF
```

Error 50 - ERR_STAN_TOOMANY_TOKENS_FOUND

While standardising the input, more than [MAX_TOKENS_FOUND](#) were found. Try changing the grammar definitions, so that the result can be achieved with finding less tokens.

Error 52 - ERR_GRAMMAR_DUPLICATES_IN_TABLE

Duplicate words in a table, e.g.

```
\TABLE states
NEW YORK
NEW YORK
```

Error 53 - ERR_GRAMMAR_MUTUALLY_EXCLUSIVE

Cannot use *Must be first* and *Look from end* qualifiers in the same definition (with the exception of with a [TABLE_SINGLE_WORDS](#) token), e.g.

```
\GRAMMAR
Title : $ TABLE Titles !
```

Error 54 - ERR_GRAMMAR_OPTION_NOT_AVAILABLE

1. Cannot use *Must be last* qualifier, unless the *Look from end* qualifier is used in the same definition (with the exception of with a [TABLE_SINGLE_WORDS](#) token), e.g.

```
\GRAMMAR
Title : TABLE Titles $
```

2. With a table token, cannot use *Delimiter before* or *Delimiter after*, unless using the [CHAR_SET](#) qualifier, e.g.

```
\GRAMMAR
Title : TABLE Titles <
```

Error 55 - ERR_GRAMMAR_DUP_TOKENDEF

Cannot give two token definitions the same name, e.g.

```
\GRAMMAR
Name          : WORD
Name          : WORDOF "abc"
```

Error 56 - ERR_GRAMMAR_NO_COMMAND

Expecting a command, e.g.

```
\PRE_PARSE { 5 }
```

Error 57 - ERR_GRAMMAR_DONTKNOW_COMMAND

Expecting a command, perhaps it is misspelled, e.g.

```
\PRE_PARSE { INPUT = UCISE(INPUT) }
```

should be replaced with

```
\PRE_PARSE { INPUT = UCASE(INPUT) }
```

Error 58 - ERR_GRAMMAR_DONTKNOW_STR_OPERAND

Expecting a text operand – a text variable, an output field, a constant or the word INPUT. Perhaps it is misspelled, e.g.

```
\OUTPUT_FIELDS
5          postCode
\PRE_PARSE { pcode = "2000" }
```

Here, *pcode* should be replaced with *postCode*.

Error 59 - ERR_GRAMMAR_DONTKNOW_NUM_OPERAND

Expecting a numeric operand – a numeric variable or a constant. Perhaps it is misspelled, e.g.

```
\NUMERIC_VARIABLES
n1
\POST_PARSE { IF n > 0
```

Here, *n* should be replaced with *n1*.

Error 60 - ERR_GRAMMAR_TOOLONG_QUOTES

The number of characters between the quotation marks is greater than [MAX_QUOTES](#).

Error 61 - ERR_STAN_OUTPUT_TOO_LONG

When calling the standardiser, the length specified for the output, is not long enough to hold the output.

Error 63 - ERR_GRAMMAR_NO_BEGIN_COMMANDS

Missing opening brace after PRE_PARSE, POST_PARSE, INITIALISE or FINALISE.

Error 64 - ERR_GRAMMAR_NO_END_COMMANDS

Grammar file ended without the closing brace at the end of a command section.

Or, no closing brace after { **AS_ABOVE**

Error 65 - ERR_GRAMMAR_DUPLICATE_VARIABLE

The names of output fields and text variables need to be unique. If not, this error is produced. Similarly, if the names of the numeric variables are not unique.

Error 66 - ERR_GRAMMAR_DUPLICATE_PROCESS

More than one set of commands is defined for one token definition.

Error 69 - ERR_GRAMMAR_NO_ENDIF

IF command without matching END IF command.

Error 70 - ERR_GRAMMAR_TOOMUCH_NESTING

IF, WHILE or FOR nesting greater than [MAX_NESTING](#)

Error 71 - ERR_GRAMMAR_ELSE_WITHOUT_IF

ELSE command without matching IF command.

Error 72 - ERR_GRAMMAR_ENDIF_WITHOUT_IF

END IF command without matching IF command.

Error 73 - ERR_GRAMMAR_COMMAND_WITHOUT_TOKEN

A command section (beginning with the opening brace) in the \GRAMMAR section without a token definition

Error 74 - ERR_OPEN_GRAMMAR_FILE

Cannot open the grammar file, or an included file.

Error 77 - ERR_GRAMMAR_ASABOVE_WITHOUT_ABOVE

The commands for a token definition are { AS_ABOVE }, yet there is no previous token definition with commands defined with it.

Error 81 - ERR_GRAMMAR_ENDWHILE_WITHOUT_WHILE

An WEND command found within the matching WHILE command.

Error 83 - ERR_GRAMMAR_DOUBLE_SPACING

A table entry has more than one space between words. All entries of multiple words, must have their words separated with one and only one space.

Error 84 - ERR_GRAMMAR_MISSING_ORDER_DIGIT

When specifying some order in an Any-Order clause (beginning with the tilde), not all sub-tokens were given an order number. E.g.:

```
Full_Address      : Pobox Street Locality ~12
```

Here, only two were given an order, instead of all three.

Error 85 - ERR_GRAMMAR_TOOMANY_ORDER_DIGITS

When specifying some order in an Any-Order clause (beginning with the tilde), there were more [Order digits](#) than sub-tokens. E.g.:

```
Full_Address      : Pobox Street Locality ~1234
```

Here, only four were given an order, instead of only three.

Error 86 - ERR_GRAMMAR_INVALID_SUBINPUT_USE

The SUB_INPUT command was used not on a composed token type.

This can happen if this command is used for a simple type or in the pre parse or post parse sections.

If a simple type uses AS_ABOVE, and the previous token definition uses SUB_INPUT, the error will also result.

Error 87 - ERR_GRAMMAR_NO_VARIABLE_NAME

The name of the variable is missing in the \VARIABLES section. E.g. after the 10 here:

```
\VARIABLES
10
```

Error 88 - ERR_GRAMMAR_INVALID_LENGTH

A variable is defined with zero or negative length. E.g.

```
\VARIABLES
0 myVariable
```

Error 89 - ERR_GRAMMAR_NO_DLL_PATH

The name of the DLL is not provided in a “DLL” header. E.g.

```
/DLL
```

Error 90 - ERR_GRAMMAR_DLL_TOO_LONG

The path name of the DLL define with the \DLL header, is longer than [MAX_DLL_PATH](#).

Error 94 - ERR_GRAMMAR_DUPLICATE_FUNCTION

More than one DLL function has the same alias.

Note: If aliases are not used, then the alias is the function name itself.

Error 95 - ERR_GRAMMAR_INVALID_AS_ALIAS

The DLL function name is not valid to be used as an alias here. For example, it contains characters other than letters, numbers and the underscore.

Give it a valid alias.

Error 96 - ERR_STAN_DLL_CALL_EXCEPTION

The DLL function call threw an exception.

Error 97 - ERR_GRAMMAR_DLL_CANNOT_LOAD

The DLL cannot be loaded.

Ensure that the path provided in the \DLL header is correct.

Error 98 - ERR_GRAMMAR_DLL_FUNCTION_INVALID

The function in the DLL cannot be found.

Ensure that it exists in the DLL.

Error 99 - ERR_GRAMMAR_DLL_FREE

A windows error occurred while unloading a DLL.

Error 100 - ERR_STAN_DLL_CALL

While standardising, a DLL function returned an error.

See [Dynamic Linked Library \(DLL\) definitions](#) for how to define what return values are to be considered as an error.

Error 101 - ERR_GRAMMAR_DONTKNOW_ARRAY_OPERAND

Expecting a variable or output field.

Ensure it is correctly spelled.

Error 102 - ERR_GRAMMAR_ARRAY_OVERBOUND

The number of elements in the array is too high, e.g.

```
\VARIABLES
```

```

result
var1
var2
var3

\PRE_PARSE {
MAKE_DELIMITED result var1 4 "," }

```

Here, the maximum size of the array beginning with “var1” is 3 (var1, var2 & var3). Hence a value of 4 is too high.

NOTE: If the array size is a numeric variable, then it cannot be tested when reading the grammar file. If its value is greater than the maximum number of elements, then the maximum number of elements will be used instead.

Error 103 - ERR_GRAMMAR_EXPECTING_DELIMITER

In the [Output fields](#) section, the word “DELIMITER” was included to indicate that the output should be delimited, but the given delimiter was invalid.

For example, the ASCII number provided was out of range – below 1 or above 255. Or, there wasn’t just one character enclosed in double quotes.

Error 104 - ERR_GRAMMAR_INVALID_TOLERANCE

In a [TABLE](#) or [TABLE_SINGLE_WORDS](#) token definition:

a [TOLERANCE](#) was defined with an invalid tolerance value (not between 0 and 100 exclusive).

OR

a [TOLERANCE1](#) or [TOLERANCE2](#) was defined with an invalid tolerance value (less than 1).

Error 105 - ERR_GRAMMAR_INVALID_SEARCHSPAN

In a [TABLE](#) or [TABLE_SINGLE_WORDS](#) token definition, a [TOLERANCE](#) was defined with an invalid search span value (negative).

Error 106 - ERR_GRAMMAR_INVALID_HEX_IN_QUOTES

Within some quotes, a hex escape sequence was used (“\x” – see Escape sequences), with an invalid hex number..

Hex numbers must have two digits, each either a number from 0-9 or a letter from a-f (or A-F).

Error 107 - ERR_GRAMMAR_INVALID_ESCAPE_CHAR

A backslash was found with quotes, not followed by a valid escape sequence (see Escape sequences).

To use backslash itself, double it – “\\”.

Error 108 - ERR_GRAMMAR_NULL_IN_QUOTES

The null character (“\0” – see Escape sequences) is not allowed in quotes, unless only one character is expected.

Error 109 - ERR_GRAMMAR_MISSING_CLOSING_QUOTES

Closing quotes are missing. Eg.

```
\DELIMITERS ", . #
```

Error 110 - ERR_NOT_OPEN

An attempt was made to use the process before it was fully initialised/opened.

Error 111 - ERR_GRAMMAR_EMPTY_QUOTES

A token was defined as an empty string. Eg.

```
\GRAMMAR
Empty : ""
```

Error 113 - ERR_OUT_OF_MEMORY

Out of memory error.

Error 120 - ERR_GRAMMAR_DONTKNOW_OPERAND

Expecting an operand. Check operand's spelling.

In a command line, after "IF", "WHILE", "IF Operand Operator", "Operand1 = Function (" or "Operand1 = Operand2 Operator" expecting an operand.

Error 121 - ERR_GRAMMAR_MISSING_OPERATOR

Expecting a comparison operator – one of "<", ">", "<=", ">=", "<>", "=".

In a command line, after "IF Operand1 " or "WHERE Operand1 " expecting an operator.

Error 122 - ERR_GRAMMAR_INVALID_OPERAND_TYPE

Operand type invalid. Eg. using a text operand where a numeric one should be used.

Error 123 - ERR_GRAMMAR_INVALID_OPERATOR

Comparison operator invalid. Expecting one of "<", ">", "<=", ">=", "<>", "=".

Error 124 - ERR_GRAMMAR_END_WITHOUT_IF

Expecting the word "IF" to follow the word "END" (an "END IF" command to close an IF group).

Error 125 - ERR_GRAMMAR_INVALID_COMMAND_LINE

The first item in a command line is unrecognisable.

Expecting an operand, or one of these words "IF", "ELSE", "WHILE", "END IF" or "WEND".

Error 126 - ERR_GRAMMAR_EXPECTING_EQUAL_SIGN

In a command line, an operand was found but no equal sign.

Expecting a command of the form "Operand =..."

Error 127 - ERR_GRAMMAR_EXPECTING_PARENTHESSES

In a command line, after a function, an open parenthesis is expected.

Expecting a command in the form "Operand1 = Function (..."

Error 128 - ERR_GRAMMAR_MISSING_PARENTHESIS

The closing parenthesis is missing in a command.

Expecting a command in the form "Operand1 = Function (Operands)"

Error 129 - ERR_GRAMMAR_EXPECTING_COMMA

A comma must separate the parameters of a function.

Expecting a command in the form "Operand1 = Function (Operand2, Operand3...)"

Error 130 - ERR_GRAMMAR_TOOMANY_PARAMETERS

More than 3 parameters in a function.

Eg. "Operand1 = Function(Operand2, Operand3, Operand4, Operand5)"

Error 131 - ERR_GRAMMAR_TOOFEW_PARAMETERS

Specific function requires more parameters than provided.

Eg, the MID function requires at least 2 parameters. "Operand1 = MID(Operand2)" will give this error.

Error 132 - ERR_GRAMMAR_NOT_LVALUE

Attempting to assign to a constant. Eg

```
1 = VAL(INPUT)
```

Error 133 - ERR_GRAMMAR_EXPECTING_OPERATOR

Expecting an operator ("+", "-", or "&").

After "Operand1 = Operand2" expecting either the end of the line, or an operator.

Error 134 - ERR_GRAMMAR_DUPLICATE_CONSTANTS

A constant was defined twice, e.g.

```
\CONSTANTS  
c1="abc"  
c1="def"
```

Error 135 - ERR_GRAMMAR_CONSTANT_TOO_LONG

A constant defined longer than the maximum length (defined in [MAX_QUOTES](#)).

Error 136 - ERR_GRAMMAR_EXPECTING_CONSTANT

Expecting a constant or literal text.

Error 137 - ERR_GRAMMAR_EXPECTING_DIVIDER

In the [Suggested divider](#) clause, a divider must follow it having only one character.

Error 138 - ERR_GRAMMAR_QUALIFIER_NOT_1_CHAR

The Text Qualifier in the OUTPUT_FIELDS section must be only 1 in length.

Error 139 - ERR_GRAMMAR_CIRCULAR_RECURSION

Circular recursive sub-grammars (using the [Sub Grammars](#) section).

Error 141 - ERR_GRAMMAR_SUBGRAMMAR_TOO_LONG

Grammar name defined in the [Sub Grammars](#) section has more than [MAX_SUBGRAMMAR_NAME](#) characters.

Error 142 - ERR_LICENCE_NOTFOUND

Licence file not found.

Ensure the licence file is present in the [Home Directory](#).

Error 143 - ERR_LICENCE_INVALID

Licence file invalid.

Error 144 - ERR_LICENCE_EXPIRED

Licence file expired.

Error 145 - ERR_LICENCE_DOESNT_CONFORM

Licence not valid for this computer.

Error 146 - ERR_LICENCE_OVER_CLIENT_LIMIT

Number of clients surpasses that specified in the licence.

Error 147 - ERR_GRAMMAR_QUALIFIER_IS_DELIMITER

In the [Output fields](#) section, the same character was specified as the delimiter and the text-qualifier.

Error 149 - ERR_GRAMMAR_PRE_MISSING_NAME

A `\DEFINE` statement (see [Preprocessor directives](#)) without a variable name following it.

Error 150 - ERR_GRAMMAR_PRE_ENDIF_WITHOUT_IF

A `\ENDIF` statement (see [Preprocessor directives](#)) without a matching `\IFDEF` or `\IFNDEF` statement.

Error 151 - ERR_GRAMMAR_PRE_ELSE_WITHOUT_IF

An `\ELSE` statement (see [Preprocessor directives](#)) without a matching `\IFDEF` or `\IFNDEF` statement.

Error 152 - ERR_GRAMMAR_SUBINPUT_TO_INPUT

INPUT cannot be the destination for a `SUB_INPUT` or `SUB_INPUT_FOUND` command. Eg.

```
{ INPUT = SUB_INPUT(0) }
```

Error 156 - ERR_GRAMMAR_ELSE_AFTER_ELSE

Two `ELSE` statements in an `IF ... ELSEIF ... ELSE ... END IF` block.

Error 157 - ERR_GRAMMAR_EXPECTING_TO

Missing the word `TO` in an `FOR ... NEXT` statement.

Error 158 - ERR_GRAMMAR_INVALID_FOR

LoopVariable is the same as the ToVariable or StepVariable in an `FOR ... NEXT` statement.

Error 159 - ERR_GRAMMAR_NEXT_WITHOUT_FOR

`NEXT` statement without a matching `FOR` statement (see [FOR ... NEXT](#)).

Error 160 - ERR_GRAMMAR_EXPECTING_FUNCTION

Expecting a function name after the word `INITFUNC` or `FINALFUNC` (see [Dynamic Linked Library \(DLL\) definitions](#)).

Error 161 - ERR_GRAMMAR_EXPECTING_FAIL_OR_CONT

Expecting one of the [DLL Error Actions](#) `FAIL` or `CONTINUE`. See [DLL Error Specifications](#).

Error 162 - ERR_GRAMMAR_DUPLICATE_FINALFUNC

Two finalisation functions are defined with `FINALFUNC` for the same DLL (see [Dynamic Linked Library \(DLL\) definitions](#)).

Error 163 - ERR_GRAMMAR_DUPLICATE_INITFUNC

Two initialisation functions are defined with `INITFUNC` for the same DLL (see [Dynamic Linked Library \(DLL\) definitions](#)).

Error 164 - ERR_GRAMMAR_EXPECTING_NUMVAR

Expecting a numeric variable after the word `TO` in a [DLL Error Actions](#). See [DLL Error Specifications](#).

Error 165 - ERR_GRAMMAR_EXPECTING_LITERAL

Expecting a literal after the word `RETURN` in a [DLL Error Actions](#). See [DLL Error Specifications](#).

Error 166 - ERR_GRAMMAR_EXPECTING_ERR_ACTION

Expecting one of the [DLL Error Actions](#). See [DLL Error Specifications](#).

Error 167 - ERR_GRAMMAR_DUPLICATE_INITIALISE

More than one `INITIALISE` sections were defined (see [Initialise and Finalise commands](#)).

Error 168 - ERR_GRAMMAR_DUPLICATE_FINALISE

More than one FINALISE sections were defined (see [Initialise and Finalise commands](#)).

Error 169 - ERR_GRAMMAR_SET_GLOBAL_VARIABLE

Attempted to assign a value to a global variable outside of the INITIALISE or FINALISE sections (see [Global variable definitions](#)).

Error 170 - ERR_LICENCE_CORRUPT

Corrupt licence file (iStan.lic).

Error 171 - ERR_LICENCE_OPEN_REGISTRY

On Windows: Error in opening the registry.

On Unix: Error in opening the Intech configuration file.

Error 172 - ERR_LICENCE_REGISTRY_KEY_NOT_FOUN

On Windows: The HomeDir entry wasn't found in the registry.

On UNIX: The HomeDir entry wasn't found in the Intech configuration file.

Error 174 - ERR_GRAMMAR_NOT_SUPPORTED

Feature not supported.

This can occur if the word TABLE is followed by quotes in the GRAMMAR section.

Errors 180-189**Error 181 - ERR_GRAMMAR_EXPECTING_CASE_QUAL**

Expecting a case qualifier (UPPERCASE, LOWERCASE, MIXEDCASE or TITLECASE) after the word NOT. See [WORD2](#).

Error 182 - ERR_GRAMMAR_EXPECTING_TOLERATE

Expecting a [Tolerance method](#) – beginning with the word TOLERATE.

Error 183 - ERR_GRAMMAR_DONTKNOW_PRE_IF

After \IF or \ELSEIF, expecting DEF, NDEF or VERSION. See [Preprocessor directives](#).

Error 184 - ERR_GRAMMAR_DLL_CALL

The initialisation function of a DLL returned an error. See [DLL Initialisation function](#).

Error 185 - ERR_GRAMMAR_TOLERANCE_NOTSUPPORTED

Invalid [Tolerance method](#).

Error 188 - ERR_FUNCTIONALITY_NOTSUPPORTED

Functionality not supported.

Error 189 - ERR_GRAMMAR_EXPECTING_NUMBER

Expecting a number after the FIXED_LENGTH Table Option. See [Fixed length table entries](#).

Errors 190-199**Error 190 - ERR_GRAMMAR_RECURSIVE_TOKENDEF**

A sub-token of a [Composed](#) token type is the token itself, for example:

```
\GRAMMAR
Token1: Token1
```

Error 191 - ERR_GRAMMAR_NO_SUBROUTINE_NAME

SUBROUTINE definition without a name, for example

`\SUBROUTINE`

Error 192 - ERR_GRAMMAR_DUPLICATE_SUBROUTINE

Two (or more) SUBROUTINEs with the same name.

Error 193 - ERR_GRAMMAR_NO_PARAMETER_NAME

SUBROUTINE parameter without a name.

Error 194 - ERR_GRAMMAR_EXIT_SUB_NOT_IN_SUB

EXIT_SUB command not within a SUBROUTINE.

Error 195 - ERR_GRAMMAR_INVALID_CHARACTER

Invalid UTF-8 character sequence. See [Unicode and UTF-8](#). Or Unicode character cannot be converted to single-byte (Windows-1252) character, where only single byte-characters allowed.

Error 196 - ERR_GRAMMAR_UNKNOWN_CHARSET

Character-set specified by \CHARSET not recognised. See [Unicode and UTF-8](#).

Error 197 - ERR_GRAMMAR_UNICODE_FILE

Attempting to read a grammar file that is Unicode encoded (UCS-2). I.e. with a Unicode Byte Order Mark (Hex FE FF or FF FE at the beginning of the file).

Error 198 - ERR_GRAMMAR_INVALID_ENCRYPTED_CHAR

Invalid character for an \ENCRYPTED section of a grammar file.

Error 1001 - ERR_TOO_MANY_GRAMMAR_FILES

The maximum number of grammar files allowed to be concurrently opened was exceeded. See [Max Grammars](#)

Error 1100 - ERR_STAN32_TOOMANY_STANS

Occurs when using the [Stan Libraries](#). Too many calls have been made to the Open function, without calls to the Close function. See StanLib Max ID's Configuration Setting.

Error 1101 - ERR_STAN32_INVALID_ARG

Occurs when a call to a function in the [Stan Libraries](#) is made with an invalid parameter (eg. a NULL pointer).

Error 1102 - ERR_STAN32_INVALID_ID

Occurs when a call to a function in the [Stan Libraries](#) is made with an invalid Id. Only Ids returned from an Open function call are valid, and become invalid after a call to the Close function.

Error 1106 - ERR_STAN32_INVALID_PROPERTY

Occurs when calling the PutProperties function in the [Stan Libraries](#) with an invalid property name.

Error 1107 - ERR_STAN32_NOTINITIALIZED

The [Stan Libraries](#) haven't been correctly initialised by the operating system.

Error 1108 - ERR_STAN32_NOT_IMPLEMENTED

Functionality not available via sockets. This option is not available as a client to a socket server.

Errors 4200 to 4299

The following errors can occur as a client to a socket server:

4200 - Socket call error.

4201 – Set socket option error.

4202 – Socket bind error.
4203 – Socket listen error.
4204 – Socket connect error.
4205 – Unknown socket host.
4206 – Unknown socket service.

Errors 4500 to 4599

The following errors can occur as a client to a socket server:

4500 – ERR_SOCSVR_CALL_NOT_RECOGNISED
4501 – ERR_SOCSVR_ACCEPT
4502 – ERR_SOCSVR_INPUT_TOO_LONG
4550 – ERR_SOC_READLINE
4551 – ERR_SOC_SEND
4555 – ERR_SOC_RESPONSE_TOO_LONG
4556 – ERR_SOC_BROKEN

Errors 11200 to 11399

Invalid data in [Encrypted sections](#), or error in decrypting.

Glossary

ABS

Australian Bureau of Statistics – Australian government body which provides the national statistical service including Census data. See www.abs.gov.au

AMAS

Address Matching Approval System - the Australia Post program for approval of software which demonstrate the ability to accurately match addresses with those in the PAF.

ARF

Address Reference File – a source data set that can be used for Rapid Address entry and address validation, for example, the PAF.

Barcode

An address barcode generated from the address DPID. It is represented by a sequence of digits that is used to produce the physical barcode printed on the mail item.

BSP

Barcode Sort Plan – a number from 1 to 54 is assigned to each postcode used to sort mail for Barcoded PreSort mail discounts.

CD

Collection District – the second smallest Australian Bureau of Statistics geographical area, represented by a unique seven digit code. These are the smallest Census collection and processing units, which serve as building blocks for aggregating statistics info to larger Census geographic areas.

DPID

Delivery Point Identifier – a unique 8-digit number used to barcode addresses which has been randomly allocated by Australia Post to each postal address in Australia.

FCC

Format Control Code – a two digit number comprising 4 bars that identifies the type of barcode. An invalid FCC will cause a mail article to be rejected.

GIF

Geographical Information File – additional data that relates to an address for a particular ARF.

G-NAF

Geocoded National Address File – the authoritative geocoded address index for Australia, containing state, suburb, street, number and coordinate reference or geocode for 12.5 million Australian street addresses. It is managed by PSMA.

GroupDID

Group Delivery Identifier – an 8-digit number used to barcode addresses which has been assigned to some streets in Australia where DPID values have not been assigned to individual points.

LocalityDID

Locality Delivery Identifier – an 8-digit number used to barcode addresses which has been assigned to some localities in Australia where GroupDID values have not been assigned to streets.

MB

Mesh Block – the smallest geographic region for which statistical data is collected by the Australian Bureau of Statistics and Statistics New Zealand, providing the building block for larger regions. There are about 340,000 mesh blocks covering the whole of Australia, each containing about 30-60 households in residential and agricultural regions.

NPSP

National PreSort Plan – a three-digit number used for sorting bulk mail lodgements.

NSP

Alternative abbreviation for National PreSort Plan (NPSP).

PAF

Postal Address File – a file containing all postal delivery addresses in Australia, each address record consisting of a DPID and the delivery address details in a correct address format. It is managed by Australia Post, updated quarterly, and only supplied with approved AMAS software.

Phantom Primary Point

A Primary Point address that is not a deliverable address in its own rights, and requires Secondary Information.

Phonetic Algorithm

A method of encoding words, mostly by the sound of consonants, so that words with the same sound have the same code.

Soundex – encodes words as the first letter of the word followed by three digits according to the type of sound. For example, b, p and v are encoded as 1, l is encoded as 4, so pill, pall and pile are encoded as P400 and bill, bell and ball as B400.

NYSIIS – (New York State Identification and Intelligence System), more recent and accurate than Soundex, uses a variable length code. For example, pill, pall and pile are encoded as PAL, bill, bell and ball as BAL, belly and billy as BALY, and bellweather as BALWATAR.

Metaphone – uses a larger set of rules than Soundex to generate a variable length code.

Primary Point

An address that has a house/street number, but no Secondary Information.

Real Primary Point

A Primary Point address that is a deliverable address in its own rights, even without any Secondary Information.

SendRight

The a New Zealand Post equivalent of the Australia Post AMAS program.

Secondary Information

The following components of an address: Flat/unit type (e.g. UNIT), flat/unit number, level type (e.g. FLOOR), level number, and house/street number suffix (e.g. A in “1A Main St”).

SLA

Statistical Local Area – an Australian Bureau of Statistics area comprising all or part of a Local Government Area, consisting of one or more Collection Districts (CDs).

SOA

Statement of Accuracy – a New Zealand Post electronic certificate showing the percentage of valid matches between a customer mailing list and the New Zealand PAF.

SOAP

Simple Object Access Protocol.

Synonym locality

An alternative name for a locality.

Unacceptable synonym

A Synonym locality that may not be used on the mail piece. The real locality name must be used instead.

Valid synonym

A Synonym locality that may be used on the mail piece. The real locality name may be used instead.

XML

Extensible Markup Language.

Index

- operator 426

!

! (look from end) 407

#

(comment) 396

\$

\$ (must be first) 407, 410

\$ (must be last) 407, 410

&

& operator 426

?

? (optional sub-token) 411

@

@ (junk allowed) 410

\

\ (line continue) 396

\ escape sequences 396

~

~ (any order) 411

~ (order digits) 411

+

+ operator 426

<

< (delimiter before) 406

=

= (Assignment) 426

>

> (delimiter after) 406

A

AbbreviateLevel property 134

AbbreviateStreet property 134

ABSGeocode 364, 365

ACCENT_INSENSITIVE 382

ACCENT_SENSITIVE 382

Address

Enhancer Grammar 253

Field Grammar 254

Address ID

finding 97

Address line 86

Address Matching Summary Report 60, 68, 288, 290

Address record

Australian format 222

New Zealand format 227

Addresses

abbreviations 101, 230

accepting 70, 72, 80, 85, 268

Australian 253, 254, 360

Australian format 222

colours 102

components 111, 112

copying 99

copying to clipboard 109

DPID 97

email 352

expanding 101, 103

fields 59, 60, 62, 264

final 224

finding 70, 72, 80, 268

fixing 61, 70, 262, 268

format 62, 111, 112, 264

formatting 109, 253, 254, 353, 359, 360

found 96

- Geocoded 97, 105, 109
 - invalid 61
 - map display 97
 - New Zealand 368, 369, 370
 - New Zealand format 227
 - passing between applications 99
 - pasting 99
 - postcodes 259
 - processed 65, 75
 - processing 69, 76, 268
 - searching 86, 89, 93
 - statistics 78
 - validating 360
 - writing to file 109
- AddressFields property 216
- AddressFieldsDelimited property 217
- AddressFilter property 134
- AddressLine property 217
- ALLOW_DUPLICATE_TABLES 394
- ALLOW_DUPLICATES 383
- AllowWithinRange property 134
- ALPHANUM_SIZE_RANGE 403
- AMAS
 - certified date 24
 - compliance 66
- AND operator 426
- APLHANUM 402
- ARF
 - description 229
 - directory 22, 240
 - files 54, 229
 - G-NAF 247
 - installation 54
 - managing 44
 - path 240
 - removal 55
 - selecting 98, 108
 - testing 44

- ARFUtil
 - menus 44
 - options 44
 - overview 44
 - using 44
- AS_ABOVE 421
- Assignment 426
- Australian
 - aaddress record 222
 - address fields 59
- Autoload 242
 - Grammar files 319, 322
 - SOAP Server 242
 - Socket Server 241
- automatic
 - loading grammar files 24

B

- backslash 396
- Barcode 59
 - decoding 352
 - description 221
 - encoding 222
 - error 187
 - GenerateBarcode method 217
 - generating 255, 259
 - IQRapidGenerateBarcode 213
 - printing 222
- Batch
 - StanConsole 324
 - Standardiser 277
- BatchStan
 - SOAP Server function 336
- BeginPolygonSearch
 - PAF Library function 149
- Bitwise operators 426
- BSP 59, 66
- BuildBarcode

- PAF Library function 150
- SOAP Server function 191

Buttons

- Rapid Address 85

C

CallPafFromStan

- PAF Library function 151

- case sensitivity 188, 331, 382, 395, 396, 435, 436
 - of tokens 402

- CASE_INSENSITIVE 382

- CASE_SENSITIVE 382

- ChangeARF method 129

ChangeDatabase

- RapidJNI function 206

character

- ASCII 113

- at 410

- backslash 396

- carriage return 188, 287, 331

- delimiter 409

- dollar 407, 410

- exclamation mark 407

- greater than 406

- hash 396

- less than 406

- new line 188, 287, 331, 396

- Null 396

- punctuation 394, 396

- question mark 411

- space 396

- special 113

- tab 188, 331, 396

- tilde 411

- token type 400

- CHARSET 398

Cleanup

- RapidJNI function 207

- clear screen 85

- Clipboard 109

Close

- Stan Library function 308

- Close method 298

ClosePaf

- PAF Library function 152

ClosePafId

- PAF Library function 152

Codes

- Amended Flag 251

- Bit 251

- Correct Flag 250

- Error Flag 249

- Flag 249

- GIF Flag 252

colours

- of addresses 102

- COM 49, 50

- ComAuditFile 322

Command line

- Standardiser 277

Commands

- examples 440

- finalise 390

- initialise 390

- location 422

- operators 426

- parsing 422

- pre/post parse 391

- summary 424

- syntax 423

- token definitions 420

- comments 396

- ComplianceType property 135

- Composed token type 405

- output actions 419

- concatenation 442

conditional operators 437

Configuration

AMAS certified date 24

ARF directory 22

creating 107

deleting 107

Easy Post 24

editing 107

grammar files directory 23

home directory 22

loading grammar files 24

maximum grammar files 23, 51

overview 48

PAF directory 23

Rapid Address 83, 99, 106, 115

Stan Libraries 321

Unix 22, 52, 238

Windows 22, 52, 238

Constant 424

CONSTANTS 393

ConvertHouseNumRange property 135

CustInfoBarcode

PAF Library function 152

Customer 222

D

data

input 280, 283, 284

OLE DB 280

output 280, 281

passthrough 287

Database

Microsoft SQL Server 28

Oracle 26

DCOM 49, 52

debugging 377

DefaultDirectory property 301

DefaultProcesses property 302

DEFINE 397

DefineSpec method 340

delimited

text 428, 433

DELIMITED 380

Delimiter property 135

delimiters 412, 433

field 287

inner 409

table 408

DELIMITERS 394, 408, 412

DESCRIPTION 379

Description property 298

DIFFS 413

DIFFS_PER_LEN 414

DIFFS_PER_WORD_LEN 415

directory

ARF 22

grammar files 23

home 22

PAF 23

display

fields 268

Divider 380

DLL 385

error actions 389

error sources 389

error specifications 388

example 445

finalisation function 388

function prototype 386

initialisation function 387

iPaf32 120, 147

iqgiss 120

IQRapidForm32 208

IQxp 28

RapidJNI 201

RapidSvr 120, 121

- return value 388
- Stan32 296, 305, 325
- StanB 297
- DontKeepLoaded
 - Stan Library function 315
- DontKeepLoaded method 301
- DPID 87
 - finding 97, 255, 256, 259
 - missing 61
- DpidToAddress
 - method 127
 - PAF Library function 153
 - RapidJNI function 204
 - SOAP Server function 193
- DpidToAddressId
 - PAF Library function 153
- DpidToBarcode
 - method 127
 - PAF Library function 154
 - RapidJNI function 204
- Duplicate
 - table entries 383, 395
 - tables 394

E

- Easy Post
 - Configuration 24
 - exiting 62
 - input files 63
 - options 64, 65, 66, 68, 75
 - output files 65, 75
 - overview 58
 - processing 69, 76
 - reporting 60
 - starting 62
 - tabs 61, 74
 - workflow 59
- ELSE 397, 437

- ELSEIF 397, 437
- EmbedSynonym property 135
- Encoding
 - N 222
- ENCRYPTED 398
- END IF 437
- END_ENCRYPTED 398
- ENDIF 397
- Enum
 - IQAddressFields (IQRapidForm) 218
 - IQBarcodeFormatEnum (IQRapidForm) 219
 - IQFormatOptionsEnum (IQRapidForm) 220
 - IQFormOptionsEnum (IQRapidForm) 220
 - IQUserChoiceEnum (IQRapidForm) 221
- Error codes
 - PAF functions 179
 - RapidSvr 121
 - Stan Libraries 306
- errors
 - DLL 388, 389
 - Standardiser 449
- Escape sequences 396
- Event
 - OnClose (IQRapidForm) 221
- Example
 - case change 441
 - concatenation 442
 - conditional operators 443
 - debug output 377
 - DLL function 390, 445
 - grammar files 375
 - join 446
 - lookup 441
 - loops 442, 443
 - numeric operators 441
 - of commands 440
 - output actions 418
 - parsing 421

- phonetic algorithm commands 440
- priorities 376
- string operators 440, 444
- sub grammar 446

exiting

- Easy Post 62
- Rapid Address 83
- Standardiser 278

ExpandAbbreviations property 135

ExpandGifLists property 136

ExpandLevelType property 136

ExpandStreetSuffix property 136

ExpandStreetType property 137

ExpandUnitType property 137

F

FieldDelimiter property 137

Fields

- address 59, 60, 62, 264
- delimiter 287
- display 268
- input 62, 264, 286, 287, 380
- output 59, 60, 62, 266, 287, 380, 423
- PAF 266
- search 267
- selecting 62, 264, 266, 267, 268

File

- data 54
- grammar see Grammar file
- including 396
- input 62, 63, 263, 264
- licence 53
- log 275
- output 65, 75, 263, 285
- project 275
- report 288
- separate mail 66
- spec 275

Stan Spec 343

Standardiser 275

text format 64, 265, 285

FileAcknowledge

SOAP Server function 200

Final Address 224

FINALISE 390

FindStatus

RapidJNI function 205

FindStatus property 129

Fix Address

input files 264

overview 262

starting 262

tabs 262

using 262

FIXED_LENGTH 384

FixLastLine

PAF Library function 154

FixLastLineId

PAF Library function 154

flag

Amended 225

Final Address 224

modify 161

Status 225

Flag codes

amended 251

bit 251

correct 250

error 249

GIF 252

Flat/Unit type abbreviations 236

Floor/Level type abbreviations 237

FOR 438

ForceCloseAll

PAF Library function 155

format

input files 63, 64, 264, 265

output files 64, 65, 265

FormatAddress

method (IQRapidForm) 217

method (Rapid) 126

PAF Library function 155

RapidJNI function 204

SOAP Server function 193

FormatAddressId

PAF Library function 155

FormatAddressS

PAF Library function 156

FormOptions property 217

functions

DLL 385

grammar file 427

IQRapidForm 215

PAF Libraries 147

Stan Library 305, 307

StanJNI 325

G

GenerateBarcode

PAF Library function 157

GenerateBarcode method 217

geocoding 252, 254, 257, 364, 365, 428

reliability 247

Geographical information 252

defining fields 114

displaying 87

finding 97, 109

GeographicalCodes property 137

Get_DefaultDirectory

Stan Library function 316

Get_DefaultProcesses (Stan Libraries) 317

Get_Description

Stan Library function 309

Get_FindStatus

PAF Library function 162

Get_Licence

Stan Library function 317

Get_OutputDelimiter

Stan Library function 310

Get_OutputFieldCount

Stan Library function 312

Get_OutputFieldDescriptions property (Stan Libraries) 313

Get_OutputFieldNames

Stan Library function 312

Get_OutputFieldSizes property (Stan Libraries) 313

Get_OutputFormatType

Stan Library function 310

Get_OutputQualifier

Stan Library function 311

Get_ParsedInput

PAF Library function 165

Get_Properties

Stan Library function 317

Get_SuggestedDivider property (Stan Libraries) 314

GetAddress

PAF Library function 158

RapidJNI function 202

GetAddress method 121

GetAddresses

RapidJNI function 203

SOAP Server function 193

GetAddresses method 124

GetAddressesHint

SOAP Server function 195

GetAddressHint

PAF Library function 159

GetAddressHintId

PAF Library function 159

GetAddressId

PAF Library function 158

GetDpid
 PAF Library function 160

GetDpidId
 PAF Library function 160

GetDpidNonAmas
 PAF Library function 161

GetDpidNonAmasId
 PAF Library function 161

GetErrorDescription
 PAF Library function 161

GetErrorMessage
 Stan Library function 321

GetFindStatusId
 PAF Library function 162

GetGrammarInfo
 SOAP Server function 338

GetGroupRecordCount
 PAF Library function 162

GetGroupRecordCountId
 PAF Library function 162

GetHintAddress
 RapidJNI function 202

GetHintAddress method 122

GetHintAddresses
 RapidJNI function 203

GetHintAddresses method 125

GetLocalityRecordCount
 PAF Library function 162

GetLocalityRecordCountId
 PAF Library function 162

GetNextAddress
 PAF Library function 163

GetNextAddressHint
 PAF Library function 163

GetNextAddressId
 PAF Library function 163

GetNextAddressInPolygon
 PAF Library function 164

GetNextPopupHint
 PAF Library function 164

GetNextPopupHintId
 PAF Library function 164

GetOutputFieldDescription property (Stan Libraries) 314

GetParsedInputId
 PAF Library function 165

GetPointRecordCount
 PAF Library function 165

GetPointRecordCountId
 PAF Library function 165

GetPopupHint
 PAF Library function 165
 RapidJNI function 207

GetPopupHint method 125

GetPopupHintId
 PAF Library function 165

GetProperties
 PAF Library function 167
 SOAP Server function 196

GetPropertiesId
 PAF Library function 167

GetSearchStatistics
 PAF Library function 167
 SOAP Server function 201

GetSearchStatisticsTimeLimit property 138

GetSynonymRecordCount
 PAF Library function 169

GetSynonymRecordCountId
 PAF Library function 169

GIF
 specifying 109

GIF files
 loading 241

GifHouseNumberRange property 138

GLOBAL_OPTION 394

GLOBAL_VARIABLES 392

G-NAF 247

Grammar

 _RapidFormat 253

 ABSGeocode 364

 AccountNames 348

 AddressEnhancer 253

 AddressField 254

 AdminGeocode 365

 AliasNames 350

 ArfGeocode 254

 Ausphone 351

 Company 351

 CustInfoBarcode 255

 DecodeBarcodeScanner 352

 Dpid3Pass 255

 DpidDefault 256

 DpidDefault3Pass 256

 DpidGeocode 257

 DpidToBarcode 259

 Email 352

 FixLastLine2 353

 IndNames 353

 IndNames3a 354

 NzAddress 368

 NzAddressTwoArfsCNAR 371

 NzDefault 370

 NzValidate 369

 Organisation3 356

 OrgOrIndividual 357

 Phone 358

 Phone2 358

 PostcodeToSortPlans 259

 RemoveAddress 359

 RuleInList 360

 RuleInValueRange 360

 RuleIsValidAddress 360

 Soundex 361

GRAMMAR 385

Grammar files

 _RapidFormat 101

 autoload 51, 319, 322

 automatically loading 24

 categories 245

 commands 422

 customising 248

 debugging 377

 default directory 23, 239, 322

 definitions 385

 dependencies 247

 example 375

 geocoding 254, 257, 364, 365

 geocoding reliability 247

 installation 245

 KeepLoaded 322

 limits 447

 loading 304, 306

 location 23, 239, 322

 maximum loaded 23, 51

 overview 244

 passing address data to 108

 running in batch mode 324

 running in interactive mode 324

 running in project mode 325

 structure 379

 summary 245

 syntax 395

 testing 324, 377

 tokens 399

 unloading 304, 306

 updating 304, 306

 writing 375

GrammarName

 Stan Library function 309

GrammarName property 298

GrammarPath 239, 322

H

Hiding

- Rapid Address 101

Home directory 22, 239

Hot keys

- Rapid Address 83, 108, 113

HouseNumResolveLocality property 138

I

ID line 87

IF 437

IF VERSION 397

IFDEF 397

IFNDEF 397

IFUSED 384

IGNORE_DUPLICATES 383

INCLUDE 396

Individuals

- formatting 348, 357

INITIALISE 390

INNER_DELIMITERS 409

INNERTRIM 427

input

- divider 380

- fields 63, 264, 286

- file 62, 263

INPUT 424

Installation

- Grammar files 245

- IQ Office 18

- IQ Rapid Address 207

- IQRapidForm 214

- PAF Files 54, 229

- PAF Libraries 147

- RapidSvr 51, 120

- SOAP Server 51, 190, 333

- Socket Server 51, 187, 330

- Stan Libraries 296

- StanB 297

- StanConsole 297

- Standardiser SDK 296

- StanRt 51, 296

INSTR 427

INSTRREV 428

Intech

- home directory 239

Integer operands 424

iPaf32.dll 147

IQ Office

- installation 18

- releases 30

- versions 30

IQ Rapid Address 207

- installation 207

IQAddressFieldsEnum 218

IQBarcodeFormatEnum 219

IQFormatOptionsEnum 220

IQFormOptionsEnum 220

iqpafjni 201

IQRapidForm

- functions 215

- installation 214

- operation 215

- overview 214

IQRapidForm32

- data types 209

- operation 209

- overview 208

IQRapidFormatAddress 213

IQRapidGenerateBarcode 213

IQRapidGetAddressField 211

IQRapidGetAddressFieldsDelimited 210

IQRapidGetAddressLine 211

IQRapidGetFormOptions 212

IQRapidGetLocalityLine 211

IQRapidGetUsersChoice 212

IQRapidSetFormOptions 212

IQRapidShow 210

IQRapidShowModal 209

IQRapidUnload 214

IQSoapSvr 190, 333

IQSocSvr 187, 330

iqstan 325

iqstanjni 325

IQUserChoiceEnum 221

IsSpecValid method 340

J

JARO 415

Java 201

Java Class

 RapidJNI 201

 StanJNI 325

JOIN 428

K

KeepLoaded

 Grammar files 322

 Stan Library function 315

KeepLoaded method 301

Keys

 Rapid Address 85, 100, 112, 113

L

LAT_LONG_DISTANCE 428

LatLongSearchRange property 138

LCASE 429

LEFT 429

LEN 429

Libraries

 PAF 147

 Stan 305

Licence 53

 files 53

Licence property 302, 343

Limits 447

ListAllPoints property 138

ListAlternateStreetNames property 139

ListDeeperIfOnly1Found property 139

ListDeeperOnBestMatch property 139

ListDeliveryIDParent property 139

ListSuffixesSeparate property 140

ListSynonyms property 140

Loading

 ARF 228, 241, 242

 GIF files 241

 Grammar files 304, 306

Locale 323, 425

localities

 synonym 224

Locality

 line 86

 name 86

 search 92

 synonymn 93

LocalityLine property 217

Lodgement Information Report 60, 288

log file

 RapidSvr 241

 Standardiser 275

Logical operators 426

LoneNumberIsPostal property 140

LOOKUP 429

loops 437, 438

Lowercase 429

LOWERCASE 402

LTRIM 430

M

Manager object 301

Manifest Report 60, 66, 68, 288, 289

MaxAddressesReturned property 140

MaxGrammars 323

- maximum values 447
- memory management 228
- menus 84
- Metaphone 361
- METAPHONE 430
- Method
 - AddressFields (IQRapidForm) 216
 - ChangeARF (Rapid) 129
 - Close (StanRt) 298
 - DefineSpec (StanB) 340
 - DontKeepLoaded (StanRt) 301
 - DpidToAddress (Rapid) 127
 - DpidToBarcode (Rapid) 127
 - FormatAddress (IQRapidForm) 217
 - FormatAddress (Rapid) 126
 - GenerateBarcode (IQRapidForm) 217
 - GetAddress (Rapid) 121
 - GetAddresses (Rapid) 124
 - GetHintAddress (Rapid) 122
 - GetHintAddresses (Rapid) 125
 - GetPopupHint (Rapid) 125
 - IsSpecValid (StanB) 340
 - KeepLoaded (StanRt) 301
 - Open (StanRt) 297
 - PostcodeToBsp (Rapid) 128
 - PostcodeToNpsp (Rapid) 128
 - Run (StanB) 341
 - Show (IQRapidForm) 216
 - ShowModal (IQRapidForm) 215
 - Stan (StanRt) 298
 - Update (StanRt) 303
 - ValidateFullAddress (Rapid) 128
- Microsoft SQL Server
 - database integration 28
- MID 430
- MIXEDCASE 402

N

- Names 396
 - formatting 350, 353, 354
- new line character 396
- New Zealand
 - address fields 60
 - address record 227
- NEXT 438
- NO_DUPLICATE_TABLES 394
- NO_DUPLICATES 383
- NOT operator 426
- NUMBER 403
- NUMBER_RANGE 403
- NUMBER_SIZE_RANGE 403
- Numbers
 - calculating 360
- NUMERIC_VARIABLES 392
- Numerical operands 424
- NYSIIS 361, 431
- NzAddressTwoArfsCNAR 371

O

- Object
 - Manager (StanRt) 301
 - Rapid 121
 - StanB 339
 - StanRt 297
- OLE DB
 - output data 280, 281
- OnClose event 221
- OnlyReturnGeocodes property 141
- Open
 - Stan Library function 307
- Open method 297
- OpenArfId
 - PAF Library function 169
- OpenPaf
 - PAF Library function 170

- OpenPafId
 - PAF Library function 170
- OpenPafMem
 - PAF Library function 171
- OpenPafMemId
 - PAF Library function 171
- Operands
 - integer 424
 - table 424
 - text 423
- Operators 426
 - assignment 426
 - bitwise 426
 - logical 426
- Options 64
 - AMAS 66
 - file 65, 75
 - general 66
 - global 394
 - report 68
- OR operator 426
- Oracle
 - database integration 26
- Order digits 411
- ORDER_FOUND 431
- ORDINAL 404
- Organisations
 - formatting 348, 351, 356, 357
- output
 - actions 418
 - fields 59, 60, 62, 266, 287, 423
 - file 65, 75, 263
- OUTPUT_FIELDS 380
- OutputDelimiter property 299
- OutputFieldCount property 299
- OutputFieldDescriptions property 300
- OutputFieldNames property 299
- OutputFields property 343

- OutputFieldSizes property 300, 343
- OutputFormatType property 299
- OutputQualifier property 299
- OutputType property 141
- OVERWRITE_DUPLICATES 383

P

- PAD_TCASE 431
- PAF 188
 - abbreviations 230
 - directory 23, 239
 - fields 266
 - managing 44
 - searching 70, 72, 80, 268
 - testing 44
- PAF Files
 - description 229
 - installation 54, 229
 - loading 228
 - removal 55
- PAF Libraries
 - description 147
 - error codes 179
 - functions 147
 - installation 147
 - loading 23, 240
 - Socket Client 187
- pafBeginPolygonSearch 149
- pafBuildBarcode 150
- pafCallPafFromStan 151
- pafClosePaf 152
- pafClosePafId 152
- pafCustInfoBarcode 152
- pafDpidToAddress 153
- pafDpidToAddressId 153
- pafDpidToBarcode 154
- pafFixLastLine 154
- pafFixLastLineId 154

pafForceCloseAll 155
 pafFormatAddress 155
 pafFormatAddressId 155
 pafFormatAddressS 156
 pafGenerateBarcode 157
 pafGet_FindStatus 162
 pafGet_ParsedInput 165
 pafGetAddress 158
 pafGetAddressHint 159
 pafGetAddressHintId 159
 pafGetAddressId 158
 pafGetDpid 160
 pafGetDpidId 160
 pafGetDpidNonAmas 161
 pafGetDpidNonAmasId 161
 pafGetErrorDescription 161
 pafGetFindStatusId 162
 pafGetGroupRecordCount 162
 pafGetGroupRecordCountId 162
 pafGetLocalityRecordCount 162
 pafGetLocalityRecordCountId 162
 pafGetNextAddress 163
 pafGetNextAddressHint 163
 pafGetNextAddressId 163
 pafGetNextAddressInPolygon 164
 pafGetNextPopupHint 164
 pafGetNextPopupHintId 164
 pafGetParsedInputId 165
 pafGetPointRecordCount 165
 pafGetPointRecordCountId 165
 pafGetPopupHint 165
 pafGetPopupHintId 165
 pafGetProperties 167
 pafGetPropertiesId 167
 pafGetSearchStatistics 167
 pafGetSynonymRecordCount 169
 pafGetSynonymRecordCountId 169
 pafOpenArfId 169
 pafOpenPaf 170
 pafOpenPafId 170
 pafOpenPafMem 171
 pafOpenPafMemId 171
 pafPostcodeToBsp 172
 pafPostcodeToNpsp 172
 pafReadBorderRecord 176
 pafReadBorderRecordId 176
 pafReadGroupRecord 173
 pafReadGroupRecordId 173
 pafReadLocalityRecord 173
 pafReadLocalityRecordId 174
 pafReadPointRecord 174
 pafReadPointRecordId 174
 pafReadSynonymRecord 175
 pafReadSynonymRecordId 175
 pafSetGeographicalCodesId 177
 pafSetProperties 177
 pafSetPropertiesId 177
 pafValidateFullAddress 178
 pafValidateFullAddressId 178
 pafValidatePartialAddressId 179
 PARAMETERS 393
 ParsedInput
 RapidJNI function 206
 ParsedInput property 129
 Parsing 399
 commands 422
 example 421
 operation 421
 passthrough data 287
 Path
 relative 291
 Phantom Primary Point 96
 Phone numbers
 formatting 351, 358
 phonetic algorithm
 generating 361

- phonetic code 430, 431, 432, 433
- point
 - primary 224
- PolygonLongLatOrder property 141
- PolygonSearch
 - SOAP Server function 200
- PolygonSearchFilter property 141
- POST_PARSE 391
- Postal delivery type abbreviations 237
- Postcodes
 - formatting 259
- PostcodeToBarcode
 - RapidJNI function 205
- PostcodeToBsp
 - PAF Library function 172
- PostcodeToBsp method 128
- PostcodeToNpsp
 - PAF Library function 172
 - RapidJNI function 205
- PostcodeToNpsp method 128
- PRE_PARSE 391
- Primary Point 96, 224
- Priorities 376
- Process
 - adding 288
 - deleting 287
 - output 380
- Projects
 - managing 278
- Properties
 - AbbreviateLevel (Rapid) 134
 - AbbreviateStreet (Rapid) 134
 - AddressFieldsDelimited (IQRapidForm) 217
 - AddressFilter (Rapid) 134
 - AddressLine (IQRapidForm) 217
 - AllowWithinRange (Rapid) 134
 - ComplianceType (Rapid) 135
 - ConvertHouseNumRange (Rapid) 135
 - default values 238
 - DefaultDirectory (StanRt) 301
 - DefaultProcesses (StanRt) 302
 - Delimiter (Rapid) 135
 - Description (StanRt) 298
 - EmbedSynonym (Rapid) 135
 - ExpandAbbreviations (Rapid) 135
 - ExpandGifLists (Rapid) 136
 - ExpandLevelType (Rapid) 136
 - ExpandStreetSuffix (Rapid) 136
 - ExpandStreetType (Rapid) 137
 - ExpandUnitType (Rapid) 137
 - FieldDelimiter (Rapid) 137
 - FindStatus (Rapid) 129
 - FormOptions (IQRapidForm) 217
 - GeographicalCodes (Rapid) 137
 - GetSearchStatisticsTimeLimit (Rapid) 138
 - GifHouseNumberRange (Rapid) 138
 - GrammarName (StanRt) 298
 - HouseNumResolveLocality (Rapid) 138
 - LatLongSearchRange (Rapid) 138
 - Licence (StanB) 343
 - ListAllPoints (Rapid) 138
 - ListAlternateStreetNames (Rapid) 139
 - ListDeeperIfOnly1Found (Rapid) 139
 - ListDeeperOnBestMatch (Rapid) 139
 - ListDeliveryIDParent (Rapid) 139
 - ListSuffixesSeparate (Rapid) 140
 - ListSynonyms (Rapid) 140
 - LocalityLine (IQRapidForm) 217
 - LoneNumberIsPostal (Rapid) 140
 - MaxAddressesReturned (Rapid) 140
 - OnlyReturnGeocodes (Rapid) 141
 - OutputDelimiter (StanRt) 299
 - OutputFieldCount (StanRt) 299
 - OutputFieldDescriptions (StanRt) 300
 - OutputFieldNames (StanRt) 299
 - OutputFields (StanB) 343

- OutputFieldSizes (StanB) 343
- OutputFieldSizes (StanRt) 300
- OutputFormatType (StanRt) 299
- OutputQualifier (StanRt) 299
- OutputType (Rapid) 141
- ParsedInput (Rapid) 129
- PolygonLongLatOrder (Rapid) 141
- PolygonSearchFilter (Rapid) 141
- Property (IQRapidForm) 221
- Property (RapidSvr) 130
- Property (StanB) 343
- Property (StanRt) 302
- PutDidInDpid (Rapid) 142
- RapidCodeDump (Rapid) 142
- RapidJNI function 203
- Read/Write Properties (StanRt) 318
- RecordDelimiter (Rapid) 142
- ResolveGroupAmbiguity (Rapid) 142
- ReturnFormattedAddress (Rapid) 143
- ReturnFormattedAddressOptions (Rapid) 143
- SearchBorderLocalities (Rapid) 143
- SearchHouseNumAsOther (Rapid) 144
- SearchPostcodes (Rapid) 144
- SearchStateOnly (Rapid) 144
- SpecFile 340
- SuggestedDivider (StanRt) 300
- SwitchAlternateStreetPosition (Rapid) 144
- SwitchSynonymLocalityPosition (Rapid) 145
- TcpHost (Rapid) 145
- TimeLimit (Rapid) 145
- TitleCaseFields (Rapid) 145
- UseFindStatus (Rapid) 146
- UsersChoice (IQRapidForm) 221
- UseSockets (Rapid) 146
- ValidateAddressOptions (Rapid) 146
- Properties property
 - IQRapidForm: 221
 - RapidSvr 130

- StanB 343
- StanRt 302
- Property
 - Licence (StanRt) 302
- Put_DefaultDirectory
 - Stan Library function 316
- Put_OutputDelimiter
 - Stan Library function 311
- Put_OutputFormatType
 - Stan Library function 310
- Put_OutputQualifier
 - Stan Library function 312
- Put_Properties
 - Stan Library function 318
- PutDidInDpid property 142

Q

- Qualifiers 405, 408, 409

R

- RANDOM_DUPLICATES 383

- Rapid Address

- and other applications 99
- buttons 85
- components 118
- customising 253
- defining options 99, 106
- exiting 83
- hiding 101
- hot key 83, 108, 113
- key 113
- keys 85, 100, 112, 113
- menus 84
- overview 82, 118
- passing address data to Standardiser 108
- restoring 83, 101
- starting 83
- Window 84

- Rapid object 121

- RapidCodeDump property 142
- RapidJNI 201
- RapidSvr 120
 - auditing log 241
 - configuration 49
 - debugging 241
 - error codes 121
 - installation 51, 120
 - permissions 52
 - registration 120
- ReadBorderRecord
 - PAF Library function 176
- ReadBorderRecordId
 - PAF Library function 176
- ReadGroupRecord
 - PAF Library function 173
- ReadGroupRecordId
 - PAF Library function 173
- ReadLocalityRecord
 - PAF Library function 173
- ReadLocalityRecordId
 - PAF Library function 174
- ReadPointRecord
 - PAF Library function 174
- ReadPointRecordId
 - PAF Library function 174
- ReadSynonymRecord
 - PAF Library function 175
- ReadSynonymRecordId
 - PAF Library function 175
- RecordDelimiter property 142
- Registration
 - RapidSvr 120
- Registry settings 22, 52
 - ARF directory 240
 - ARF path 240
 - Grammar files directory 239
 - home directory 239
- IQ Rapid Address 208
- IQRapidForm 214
- PAF directory 239
- Rapid Server 238
- Stan Libraries 321
- Relative path 291
- releases
 - IQ Office 30
- reliability of geocoding 247
- Removal
 - PAF Files 55
 - services 55
- REPLACE 432
- Report
 - Address Matching Summary 60, 68, 288, 290
 - creating 69, 70
 - generating 69, 70
 - Lodgement Information 60, 288
 - Manifest 60, 66, 68, 288, 289
 - options 68
 - Standardiser 288, 289
 - viewing 69, 70
- RESET 384
- ResolveGroupAmbiguity property 142
- restoring
 - Rapid Address 83
- Restoring
 - Rapid Address 101
- ReturnFormattedAddress property 143
- ReturnFormattedAddressOptions property 143
- RIGHT 432
- RSOUNDEX 432
- RTRIM 433
- Run method 341
 - stored procedure 342
 - text file 341

S

Screen

- Standardiser 279

search

- fields 267

Search

- address 86, 89

- Address ID 97

- automatically 89, 103, 106

- building name 91

- DPID 97

- Geocode 105

- GIF 97, 109

- hint list 104

- institution 91

- locality 92, 104

- lot number 90

- postal address 91

- property 94

- results 88

- state 103

- street 89, 93

- timeout 104

- unit or level 90, 95

SearchBorderLocalities property 143

SearchHouseNumAsOther property 144

SearchPostcodes property 144

SearchStateOnly property 144

Secondary information 96

Server

- SOAP 51, 190, 333

- Socket 51, 187, 330

Service

- RapidSvr 51, 55, 120

- SOAP 190, 333

- StanRt 51, 55, 296

Set Properties

- RapidJNI function 204

SetGeographicalCodesId

- PAF Library function 177

SetProperties

- PAF Library function 177

SetPropertiesId 177

Setup

- starting 48

Show method 216

ShowModal method 215

Simple token type 399

SOA

- creating 61, 78

- description 61

- viewing 78, 79

SOAP Server 51, 190, 333

- Autoload 242

- BatchStan function 336

- BuildBarcode function 191

- calling 191, 334

- DpidToAddress function 193

- FileAcknowledge function 200

- FormatAddress function 193

- GetAddresses function 193

- GetAddressesHint function 195

- GetGrammarInfo function 338

- GetProperties function 196

- GetSearchStatistics function 201

- installation 51, 190, 333

- PolygonSearch function 200

- Standardise function 334

- ValidateAddressWithOptions function 198

- ValidateFullAddress function 197

Socket 188

Socket Client 49, 187

Socket Server 51, 187, 318, 323, 330

- Autoload 241

- calling 188, 331

- installation 51, 187, 330

- PAF library example 190
- Stan library example 332
- Stan library functions 331
- Stan Library requests 318, 323
- sorting 66
- sorting output 290
- Soundex 361
- SOUNDEX 433
- spaces 396
- Spec File
 - Standardiser 286
- SpecFile property 340
- SPLIT 433
- SQL
 - address validation sample code 28
 - batch validation script 28
 - registering Oracle library 27
- Stan
 - Stan Library function 308
- Stan libraries
 - Socket Server functions 331
- Stan Libraries
 - configuration 321
 - data types 307
 - description 305
 - error codes 306
 - functions 305, 307
 - registry entries 321
- Stan method 298
- Stan Spec File 343
- Stan32 305
- StanB 339
 - object 339
 - overview 339
 - Spec File 343
- stanClose 308
- StanConsole 324, 377
 - batch mode 324
 - installation 297
 - interactive mode 324
- Standardise
 - SOAP Server function 334
- Standardiser
 - batch mode 277
 - components 294
 - configuration 50
 - error numbers 449
 - exiting 278
 - files 275
 - main screen 279
 - OLE DB output data 281
 - process 288
 - projects 278
 - reporting 288, 289
 - Spec File 286
 - starting 277
 - stored procedures 282
 - text data format 285
 - text input data 283, 284
 - text output data 285
- Standardiser SDK
 - installation 296
- stanDescription 309
- stanDontKeepLoaded 315
- stanGet_DefaultDirectory 316
- stanGet_DefaultProcesses 317
- stanGet_Licence function 317
- stanGet_OutputDelimiter 310
- stanGet_OutputFieldCount function 312
- stanGet_OutputFieldDescriptions 313
- stanGet_OutputFieldNames 312
- stanGet_OutputFieldSizes 313
- stanGet_OutputFormatType 310
- stanGet_OutputQualifier 311
- stanGet_Properties 317
- stanGet_SuggestedDivider 314

- stanGetErrorMessage function 321
- stanGetOutputFieldDescription 314
- stanGetProperty 319
- stanGrammarName 309
- StanJNI 325
 - functions 325
- stanKeepLoaded 315
- stanOpen 307
- stanPut_DefaultDirectory 316
- stanPut_OutputDelimiter 311
- stanPut_OutputFormatType 310
- stanPut_OutputQualifier 312
- stanPut_Properties 318
- stanPutProperty 319
- StanRt 305
 - description 297
 - installation 51, 296
 - objects 297
 - permissions 52
 - Stan Libraries 296
 - StanB 297
- StanRt object 297
- stanStan 308
- stanStanW 308
- stanUpdate 320
- starting
 - Easy Post 62
 - Fix Address 262
 - Rapid Address 83
 - Setup 48
 - Standardiser 277
- state abbreviations 230
- Stored procedures 282, 342
- STR 434
- street suffix abbreviations 237
- street type abbreviations 231
- STRIP 434
- Sub Grammars 390, 446

- SUB_GRAMMARS 390
- SUB_INPUT 434
- SUB_INPUT_FOUND 435
- SUGGESTED_DIVIDER 380
- SuggestedDivider property 300
- SwitchAlternateStreetPosition property 144
- SwitchSynonymLocalityPosition property 145
- Synonym locality 93, 224
- Syntax
 - of commands: 423
 - of grammar files 395

T

- TABLE 382, 404
- Table operand 424
- TABLE_SINGLE_WORDS 405
- Tables 382
 - duplicates 394
 - output actions 420
- tabs 396
- TCASE 435
- TCP
 - host setting 241
 - port setting 241
- TcpHost 318, 323
- TcpHost property 145
- TcpPort 318, 323
- Telephone numbers
 - formatting 351, 358
- Testing
 - Grammar files 324
- text data
 - Standardiser 283, 284, 285
- Text operands 423
- Text-file description 341
- THEN 437
- Time limit
 - address searching 104

TimeLimit property 145
Tips 376
Title Case 435
TITLECASE 402
TitleCaseFields property 145
Token types
 ALPHANUM 402
 ALPHANUM_SIZE_RANGE 403
 character 400
 Composed 405
 NUMBER 403
 NUMBER_RANGE 403
 NUMBER_SIZE_RANGE 403
 ORDINAL 404
 simple 399
 TABLE 404
 TABLE_SINGLE_WORDS 405
 WORD 400
 WORD_SIZE_RANGE 401
 WORD2 401
 WORDOF 401
 WORDOF_SIZE_RANGE 401
 WORDOF2 402
 WORDOF2_SIZE_RANGE 402
TOKEN_LENGTH 436
TOKEN_POSITION 435
Tokens 399
 qualifiers 405, 409
Tolerance
 DIFFS method 413
 DIFFS_PER_LEN method 414
 DIFFS_PER_WORD_LEN method 415
 JARO method 415
 methods 412
 TOLERANCE method 416
 TOLERANCE1 method 416
 TOLERANCE2 method 417
TOLERANCE 416

TOLERANCE1 416
TOLERANCE2 417
TOLERATE 412
trays 66
TRIM 436

U

UCASE 436
UNDEF 397
Unicode 398
Unix
 configuration 22, 52
Unloading
 Grammar files 304, 306
Update
 Stan Library function 320
Update method 303
Updating
 Grammar files 304, 306
Uppercase 436
UPPERCASE 402
UseFindStatus property 146
UsersChoice property 221
UseSockets property 146
UTF8 398

V

VAL 436
ValidateAddressOptions property 146
ValidateAddressWithOptions
 SOAP Server function 198
ValidateFullAddress
 PAF Library function 178
 RapidJNI function 206
 SOAP Server function 197
ValidateFullAddress method 128
ValidateFullAddressId
 PAF Library function 178
ValidatePartialAddressId

PAF Library function 179

Values

tables of 382

Variables

defining 397

global 392

numeric 392, 424

parameters 393

redefining 397

text 391, 423

types 391

VARIABLES 391

versions

IQ Office 30

W

WEND 437

WHILE 437

Window

Rapid Address 84

Windows

configuration 22, 52

WORD 400

WORD_SIZE_RANGE 401

WORD2 401

WORDOF 401

WORDOF_SIZE_RANGE 401

WORDOF2 402

WORDOF2_SIZE_RANGE 402

Words

formatting 361

Workflow

Standardiser 276

X

XOR operator 426