
QuickAddress Pro Web

Technical Reference Guide

Common Classes

QAS Ltd.

Disclaimer

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

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

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

Copyright

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

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

Microsoft, Word, and Windows are trademarks of Microsoft Corporation.

© QAS Ltd. 2007

Integration Source Code

Copyright © 2007 QAS Ltd.

Permission is hereby granted, free of charge, to any person obtaining a copy of the QAS integration source code (the "Software"), to deal in the Software, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

1. The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

2. The Software may only be used in conjunction with the QAS Licensed Programs and Data.
3. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

For resolutions to common issues, answers to Frequently Asked Questions, and hints and tips for using our products, visit the QAS Support Site at:

support.qas.com

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

To request metered clicks, email **clicks@qas.com**.

QAS Support Website:
support.qas.com

QAS Global Website:
www.qas.com

UK

QAS Ltd
George West House
2-3 Clapham Common North Side
LONDON
SW4 0QL
UNITED KINGDOM

Tel: +44 (0) 20 7498 7777
Fax: +44 (0) 20 7498 0303

Technical Support

Tel: +44 (0) 20 7498 7788
E-mail: uk.support@qas.com

France

QAS
38 avenue des Champs Elysees
75008 PARIS
FRANCE

Tel: +33 (0) 1 70 39 43 20
Fax: +33 (0) 1 70 39 43 21

Technical Support

Tel: +33 (0) 1 70 39 43 43
E-mail: fr.support@qas.com

USA and Canada (Eastern Standard Time)

QAS
1 Memorial Dr Ste 800
Cambridge MA 02142-1362
USA

Tel: +1 888 322 6201
Fax: +1 888 882 7082

Technical Support

Tel: +1 888 712 3332
E-mail: us.support@qas.com

USA and Canada (Pacific Standard Time)

QAS
221 Main St Ste 1310
San Francisco CA 94105-1931
USA

Tel: +1 888 882 7203
Fax: +1 888 882 7204

Technical Support

Tel: +1 888 712 3332
E-mail: us.support@qas.com

Australia

QAS Pty Ltd
L 23 111 Pacific Highway
NORTH SYDNEY NSW 2060
AUSTRALIA

Tel: +61 (0) 2 9922 4422
Fax: +61 (0) 2 9922 3522

Technical Support

Tel: +61 (0) 2 9922 4422
E-mail: au.support@qas.com

Singapore

QAS
80 Raffles Place
35-00 UOB Plaza 1
Singapore 048624

Tel: +65 6248 4771
Fax: +65 6248 4531

Technical Support

Tel: +65 6248 4771
E-mail: sg.support@qas.com

Netherlands

QAS Nederland
Gustav Mahlerplein 60
1082 MA AMSTERDAM
THE NETHERLANDS

Tel: +31 (0) 20 504 0040
Fax: +31 (0) 20 504 0044

Technical Support

Tel: +44 (0) 20 7498 7788
E-mail: nl.support@qas.com

For all other countries

QAS Ltd
George West House
2-3 Clapham Common North Side
LONDON
SW4 0QL
UNITED KINGDOM

Tel: +44 (0) 20 7498 7777
Fax: +44 (0) 20 7498 0303

Technical Support

Tel: +44 (0) 20 7498 7788
E-mail: support@qas.com

Metered Clicks

E-mail: clicks@qas.com

QAS Global Website

www.qas.com

QAS Support Website

support.qas.com

Version 6.00, Revision 2

Contents

Overview.....	1
What Is In This Guide?	1
Conventions Used In This Manual.....	2
Product Documentation	3
Technical Support.....	5
Web.....	5
Email Or Telephone	5
Professional Services.....	6
Where Next?	6
Common Classes.....	7
Overview	7
Search Process (Capture Of An Address).....	8
Search Process (Verification)	11
Constants.....	13
Constants Referring To Search Engine Types.....	13
Constants Referring To Search Engine Intensity	15
Constants For Referring To Prompt Sets.....	16
QuickAddress Class	18
Using The QuickAddress Class	18
Search Methods	21
Status Methods	28
Settings Methods	32
AddressLine Class.....	36
Constants	36
Methods	38
Properties.....	40
DataSet Class.....	41

Methods	41
ExampleAddress Class	43
Methods	43
FormattedAddress Class	45
Methods	45
Layout Class	47
Methods	47
LicensedSet Class	49
Constants	49
Methods	52
Picklist Class	56
Methods	56
PicklistItem Class	62
Methods	62
CanSearch Class	69
Methods	69
PromptLine Class	71
Methods	71
PromptSet Class	73
Constants	73
Methods	73
SearchResult Class	75
Constants	75
Methods	77
BulkSearchResult Class	79
Methods	79
Properties	80
BulkSearchItem Class	80
Constants	80
Methods	81
Properties	81
Engine Behaviour	83
Single Line Engine	84
Hierarchical Mode	85
Flattened Mode	90

Typedown Engine	94
Hierarchical Mode	96
Special Picklist Items	99
Verification Engine	102
Using The Verification Engine	102
Keyfinder Engine	108
Flattened Mode	108
Prompt Sets	110
Hierarchical Picklists	112
Appendix A: Differences Between Common Classes	117
Overview	117
Glossary Of Terms.....	125
Index	137

Overview

QuickAddress Pro Web provides a solution for address capture and verification in Web and intranet environments. If you are an Internet or intranet developer, Pro Web provides the ability to capture a full, correct address for our supported data mappings as quickly as possible and with minimal interaction required from your users.

In order to maximise the ease of integrating this product, it is supplied with the following interfaces:

.NET	Windows only.
PHP	Windows and UNIX.
Java	Windows and UNIX.

To facilitate easy integration, QAS also supply detailed C#.NET, Java, and PHP sample code, which embodies what we consider to be best practice to produce ASP.NET, JSP and PHP pages. Additional sample code is available from the Support website at <http://support.qas.com/proweb>.

What Is In This Guide?

The three Pro Web Technical Reference Guides contain the following information:

Pro Web Technical Reference Guide - Common Classes

This guide contains details of how to use the common classes. The common classes are provided as an interface for those integrators who want to use Pro Web functionality without using the WSDL/SOAP interface or any of the other solutions.

Pro Web Technical Reference Guide - WSDL

This guide contains details of how to use the WSDL document. The WSDL document is provided for integrators who want to communicate with the QAS Server, but do not want to base their integration on the standard QAS Integration code supplied with the product.

Pro Web Technical Reference Guide - C API

This guide contains details of how to use the C API, a multithreaded Application Programming Interface (API) which enables the functionality of Pro Web to be used in systems and environments that do not support the Simple Object Access Protocol (SOAP) or Extensible Markup Language (XML).

See Product Documentation on page 3, for a full list of the other documentation available for Pro Web.

Conventions Used In This Manual

Example	Convention
<code>getPrompts (int)</code>	Sample code, method prototypes pseudo code and settings look like this.
<code>[]</code>	Arrays are represented by square brackets.
<i>viHandle</i>	Parameter names are shown in italics (when not part of sample code).
Press Enter	Key presses are shown in bold .
Press Cursor up	(Enter is the same as Return or Carriage return .)
UIStartup	API functions appear in bold type (when not part of sample code).



Text in this format indicates additional information.

Product Documentation

This section provides a comprehensive list of the documentation supplied with this product, and information about where it can be located.

Pro Web QuickStart Guide

The Pro Web QuickStart Guide provides an overview of the Pro Web documentation and how to get started with Pro Web. It describes the various available options and tells you where you can obtain further information.

The Pro Web QuickStart Guide is printed as a card and accompanies each copy of Pro Web that you have purchased. It can also be accessed from the index.htm file, which is located in the Docs folder on the Installation CD-ROM.

Pro Web Installation And Administration Guide

The Pro Web Installation And Administration Guide provides detailed information about installing Pro Web on Windows or UNIX. It also provides detailed information about administering and configuring Pro Web using the Administration Console and Configuration Editor. In addition, it provides a list of configuration settings.

The Pro Web Installation And Administration Guide is supplied as a printed bound manual with each copy of Pro Web purchased, and can also be accessed via the index.htm file, which is located in the Docs folder on the Installation CD-ROM.

Pro Web Integration Guide

The Pro Web Integration Guide provides information for integrators who want to use the QAS sample code as it is shipped, or adapt it for their own use.

The Pro Web Integration Guide is supplied as a printed bound manual with each copy of Pro Web purchased, and can also be accessed via the index.htm file, which is located in the Docs folder on the Installation CD-ROM.

Pro Web Technical Reference Guide

The Pro Web Technical Reference Guides are supplied as online PDF manuals, and can also be accessed via the index.htm file, which is located in the \Docs folder on the Installation CD-ROM.

There are three Pro Web Technical Reference Guides:

- Pro Web Technical Reference Guide - Common Classes;
- Pro Web Technical Reference Guide - WSDL;
- Pro Web Technical Reference Guide - C API.

Configuration Editor Online Help

For Windows users, the Configuration Editor Help file is supplied with your copy of QuickAddress Pro Web. This file can be accessed by:

- pressing **F1** when using the Configuration Editor;
- selecting **Contents and Index** from the Help menu on the Configuration Editor;
- pressing **Ctrl+F1** to access context-sensitive help for the function that is currently active.

Administrator Console Online Help

For Windows users, the Administrator Console Help file is supplied with your copy of QuickAddress Pro Web. This file can be accessed by:

- pressing **F1** when using the product;
- selecting **Contents and Index** from the Help menu on the Administrator Console user interface.

Data Guide

A Data Guide is supplied with each dataset. This guide provides dataset-specific information and search tips for each dataset. This should be used in conjunction with the other documentation supplied with this product.

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

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

Upgrade Guide

The Upgrade Guide is stored on the product CD in electronic format only and details the new features in this version. It also highlights key integration and compatibility issues of which you should be aware, and provides a step-by-step guide to upgrading an existing installation.

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

Previous versions of Pro Web included a Readme text file. This is no longer included in the product documentation. However, the information previously contained in the Readme file can now be found in "Appendix B: Utilities" of the Pro Web Installation And Administration Guide.

Technical Support

In addition to the documentation supplied with each QAS product, QAS also provide three forms of Technical Support:

Web

If you encounter problems using Pro Web which are not answered in the product documentation, please visit the QAS Support Zone Web site at:

<http://support.qas.com/proweb>

This Web site is dedicated to Pro Web. It contains answers to many frequently-asked questions, a searchable knowledge base, downloads, and hints and tips.

Email Or Telephone

If you cannot locate the required information on <http://support.qas.com>, QAS Technical Support may be contacted via e-mail or telephone. The Technical Support contact details for each local QAS office can be found at the beginning of this manual.

In order that your request can be dealt with as efficiently as possible, please have your QAS account reference number (provided on your despatch note) and the version number of the software that you are using to hand.

Professional Services

QAS Professional Services are available to help with the initial setting up of QuickAddress Pro Web. For more information on Professional Services, and to book the service, contact your QAS Account Manager.

Where Next?

Depending on how you want to integrate Pro Web, you should read the copy of the Technical Reference Guide that meets your needs:

- **Pro Web Technical Reference Guide - Common Classes.** This contains details of how integrators can use common classes to integrate Pro Web. The classes are termed "common" because the layer is designed to be as consistent as possible across all the languages for which QAS have provided integration code. This manual is available as the `webrefcc.pdf` file.
- **Pro Web Technical Reference Guide - C API.** This contains details of how integrators can use the C API (a multithreaded Application Programming Interface) to enable Pro Web functionality to be used in systems and environments that do not support the Simple Object Access Protocol (SOAP) or Extensible Markup Language (XML). This manual is available as the `webrefca.pdf` file.
- **Pro Web Technical Reference Guide - WSDL.** This contains details of how integrators can use an XML/SOAP methodology on which to base their integration. This manual is available as the `webrefws.pdf` file.

Each manual contains this same, shared introduction and a chapter explaining the engine behaviour of the Single Line, Typedown, Verification, and Keyfinder engines.

Each manual can be found as a PDF file in the \Docs folder on the Pro Web CD-ROM.

Common Classes

Overview

The Common classes are provided as an interface for those integrators who want to use Pro Web functionality via a layer that is as consistent as possible across all the languages for which QAS have provided integration code.

For information about the differences between the languages, see Appendix A: Differences Between Common Classes on page 117.

The main class of interest is the `QuickAddress` class (see page 18), through which all operations are performed.

The other classes are supporting classes; they are thin wrappers over the equivalent data structures defined in the WSDL interface (see the Pro Web Technical Reference Guide - WSDL for details of the WSDL interface).

Search Process (Capture Of An Address)

Typically, the process of searching with the Single Line, Typedown or Keyfinder engine follows the format outlined as follows (where the methods referred to are of the `QuickAddress` class):

1. Pre-check.

An optional check on the validity of a search against the intended engine/data mapping/layout combination. (Refer to the **canSearch** method on page 22).



QAS recommend that this check is carried out if metered data mappings are being used.

2. Initial search.

This returns a `SearchResult` object, from which a `Picklist` object can be obtained. The `Picklist` object contains a list of `PicklistItem` objects that can be displayed. Each item can either be expanded, formatted into a final address or may be merely informational. The **search** method (page 23) can be used for the initial search.

For example, a Single Line search in Java:

```
String[] asSearch = new String[]{ "7 old town", "clapham"
};
QuickAddress searchService = new QuickAddress("http://
myhost:2021");
searchService.setEngineType(QuickAddress.SINGLELINE);
// set the layout as we can get a formatted address
returned
SearchResult result = searchService.search("GBR",
asSearch, PromptSet.DEFAULT, "( QAS Standard Layout )");
{
    FormattedAddress address = result.getAddress();
    // do something with the address
}
```

Typedown and Keyfinder searches work in the same way. For examples of different types of searches, see "Appendix B: Searching With Pro Web" in the Pro Web Integration Guide.

3. Handling picklists and monikers.

The calling code then displays the picklist items to the user. Depending on the choice made by the user, one of the following will occur:

- **stepIn** to a particular item, which results in another picklist (for example, stepping into a street name will result in a picklist of premises). This hierarchical picklist behaviour is the default. If a single flattened picklist is required, the `flatten` flag must be set before the initial search is performed. For more information about hierarchical and flattened picklist modes of behaviour, see Engine Behaviour on page 83.
- **refine** the picklist: that is, narrow down the current picklist to one with a subset of items (for example, refining with the text "high" on a picklist of street names will return another picklist containing only those streets starting with "high").

The **refine** method is also called in certain special cases, namely where the item is an unresolvable range (certain countries) or a Phantom Primary Point (AUS only). Refer to page 99 for a discussion of these concepts.

- Format a final address using the **getFormattedAddress** method (see page 10).

A call to **stepIn**, **refine** or **getFormattedAddress** requires a moniker from either the `Picklist` or a `PicklistItem` returned by a previous call. These monikers encode the information required to recreate a particular picklist or picklist item, and are the means by which any integration navigates the search process. Refer to the "Using Pro Web" section of the Pro Web Integration Guide for more information.

For example, a step-in in Java:

```
// user selected picklist item
PicklistItem item =
originalPicklist.getItems()[selectedIndex];
QuickAddress searchService = new QuickAddress("http:
myhost:2021");
// ask the service to step in to item
Picklist picklist =
searchService.stepIn(item.getMoniker());
```

And a refine:

```
// moniker that created current picklist
String sMoniker;
// ...
QuickAddress searchService = new QuickAddress("http://
myhost:2021");
// ask the service to refine the picklist
Picklist picklist = searchService.refine(sMoniker,
"high");
```

4. Format the final address.

Once a final address picklist item has been identified, the **getFormattedAddress** method will apply a layout to the item, returning a **FormattedAddress** object that contains the final, formatted address.

For example, in Java:

```
PicklistItem item =
originalPicklist.getItems()[selectedIndex];
if ( item.isFullAddress() )
{
// only format if it is a final address
QuickAddress searchService = new QuickAddress("http://
myhost:2021");
// ask the service to format the address represented by
the item
FormattedAddress address =
searchService.getFormattedAddress ("( QAS Standard
Layout )", item.getMoniker() );
// do something with the formatted address
}
```

Search Process (Verification)

The Verification engine is designed so that only minimal interaction, or none at all, is required from the user. This is the choice of the integrator. The user should enter their whole address in the same format that they would write it on an envelope, and the entire address is submitted to the engine.

Picklists are only returned by verification searches where the verification level is less than **Verified**. See the "Web: Address Verification" section of the Pro Web Integration Guide for more information about verification levels. In such cases, the integrator may decide to offer the user a picklist and proceed in a similar fashion to the address capture process (page 8), or simply to ignore the picklist and use the address as originally entered (in order to minimise user interaction).

The **canSearch** method may be called for the Verification engine, but it is unlikely that this extra step in the workflow will be useful: the page can simply perform a search and process the results as required.

The typical search process for verification is as follows:

1. Initial search.

Call the **search** method, which returns a `SearchResult` object that may contain a `FormattedAddress` object and/or a `Picklist` object, as well as the `VerifyLevel`. If picklists are not being handled, the integrator may simply check that the `VerifyLevel` is high enough before using the `FormattedAddress`.

For example, in Java:

```
String [] asSearch = new String[]{ "q a s ltd", "7 old  
town", "clapham", "london"};  
QuickAddress searchService = new QuickAddress("http://  
myhost:2021");  
searchService.setEngineType(QuickAddress.VERIFICATION);  
// set the layout as we can get a formatted address  
returned  
SearchResult result = searchService.search("GBR",  
asSearch, PromptSet.DEFAULT, "( QAS Standard Layout )");  
String sVerifyLevel = result.getVerifyLevel();  
if ( sVerifyLevel.equals(SearchResult.Verified) )  
{  
    FormattedAddress address = result.getAddress();  
    // do something with the address  
}
```

2. OPTIONAL: If required, you can handle the picklist (refer to Handling picklists and monikers. on page 9).



QAS recommend that Flattened mode is used if picklists are needed in a verification search, but this is not a requirement.

3. OPTIONAL: If required, you can format the final address (refer to Format the final address. on page 10).

Constants

This section lists the constants used as arguments in address capture and verification operations. There are other constants which are specific to certain classes; these are listed with the relevant class.

Constants Referring To Search Engine Types

See Engine Behaviour on page 83 for more information about the four search engines.

SINGLELINE

This specifies the Single Line engine type.

VERIFICATION

This specifies the Verification engine type.

TYPEDOWN

This specifies the Typedown engine type.

KEYFINDER

This specifies the Keyfinder engine type.

Prototypes and Usage Examples

Java Prototype (QuickAddress class):

```
public static final String SINGLELINE =  
    EngineEnumType._Singleline;  
public static final String VERIFICATION =  
    EngineEnumType._Verification;  
public static final String TYPEDOWN =  
    EngineEnumType._Typedown;  
public static final String KEYFINDER=  
    EngineEnumType._Keyfinder;
```

Java Usage Example:

```
searchService.setEngineType(QuickAddress.SINGLELINE);
```

PHP Prototype (qaddress.inc):

```
define( "QAS_SINGLELINE_ENGINE", "Singleline" );  
define( "QAS_VERIFICATION_ENGINE", "Verification" );  
define( "QAS_TYPEDOWN_ENGINE", "Typedown" );  
define( "QAS_KEYFINDER_ENGINE", "Keyfinder" );
```

PHP Usage Example:

```
$searchService->setEngineType( QAS_SINGLELINE_ENGINE );
```

C#.NET Prototype (QuickAddress class):

```
public enum EngineTypes  
{  
    Singleline = EngineEnumType.Singleline,  
    Verification = EngineEnumType.Verification,  
    Typedown = EngineEnumType.Typedown,  
    Keyfinder = EngineEnumType.Keyfinder  
}
```

C#.NET Usage Example:

```
searchService.Engine = QuickAddress.EngineTypes.Singleline;
```


Constants Referring To Search Engine Intensity

EXACT

This specifies Exact engine search intensity. This does not allow many mistakes in the search term, but is the fastest.

CLOSE

This specifies Close engine search intensity. This allows some mistakes in the search term, and is the default setting.

EXTENSIVE

This specifies Extensive engine search intensity. This allows many mistakes in the search term, and takes the longest.

Prototypes and Usage Examples

Java Prototype (QuickAddress class):

```
public static final String EXACT =  
    EngineIntensityType._Exact;  
public static final String CLOSE =  
    EngineIntensityType._Close;  
public static final String EXTENSIVE =  
    EngineIntensityType._Extensive;
```

Java Usage Example:

```
searchService.setEngineIntensity(QuickAddress.CLOSE);
```

PHP Prototype (qaddress.inc):

```
define( "QAS_EXACT_SEARCHING", "Exact" );  
define( "QAS_CLOSE_SEARCHING", "Close" );  
define( "QAS_EXTENSIVE_SEARCHING", "Extensive" );
```

PHP Usage Example:

```
$searchService->setEngineIntensity( QAS_CLOSE_SEARCHING );
```

C#.NET Prototype (QuickAddress class):

```
public enum SearchingIntensityLevels
{
    Exact = EngineIntensityType.Exact,
    Close = EngineIntensityType.Close,
    Extensive = EngineIntensityType.Extensive
}
```

C#.NET Usage Example:

```
searchService.SearchingIntensity =
QuickAddress.SearchingIntensityLevels.Close;
```

Constants For Referring To Prompt Sets

ONELINE

This refers to the prompt set where all search text must be submitted upon a single line.

DEFAULT

This refers to the default prompt set for the engine.

GENERIC

This refers to the general data-independent prompt set.

OPTIMAL

This refers to the prompt set that requires the minimum possible amount of search text to perform a search.

ALTERNATE

This refers to an extended country-specific set for where the information required for an optimal search is not available.

ALTERNATE2

This refers to a different alternative prompt set (United States only).

Prototypes and Usage Examples

Java Prototype (PromptSet class):

```
public static final String ONELINE = PromptSetType._OneLine;
public static final String DEFAULT = PromptSetType._Default;
public static final String GENERIC = PromptSetType._Generic;
public static final String OPTIMAL = PromptSetType._Optimal;
public static final String ALTERNATE =
    PromptSetType._Alternate;
public static final String ALTERNATE2 =
    PromptSetType._Alternate2;
```

Java Usage Example:

```
searchresult = searchService.search (sDataId, asSearch,
    PromptSet.ONELINE);
```

PHP Prototype (qaddress.inc):

```
define( "QAS_ONELINE_PROMPT", "OneLine" );
define( "QAS_DEFAULT_PROMPT", "Default" );
define( "QAS_GENERIC_PROMPT", "Generic" );
define( "QAS_OPTIMAL_PROMPT", "Optimal" );
define( "QAS_ALTERNATE_PROMPT", "Alternate" );
define( "QAS_ALTERNATE2_PROMPT", "Alternate2" );
```

PHP Usage Example:

```
$searchresult = $searchService->search( $sDataId, $asSearch,
    QAS_ONELINE_PROMPT );
```

C#.NET Prototype (PromptSet class):

```
public enum Types
{
    OneLine = PromptSetType.OneLine,
    Default = PromptSetType.Default,
    Generic = PromptSetType.Generic,
    Optimal = PromptSetType.Optimal,
    Alternate = PromptSetType.Alternate,
    Alternate2 = PromptSetType.Alternate2,
}
```

C#.NET Usage Example:

```
Searchresult = searchService.Search(sDataID, asSearch,
    PromptSet.Types.OneLine);
```

QuickAddress Class

This class is a facade into Pro Web and provides the main functionality of the package. It performs SOAP operations in a stateless manner, but maintains some state between construction and subsequent method calls, to allow for optional settings (for example, timeout). An instance of this class is not intended to be preserved across pages.

Using The QuickAddress Class

The intended usage idiom is:

1. Construct a `QuickAddress` object.
2. Set optional settings.
3. Call methods (for example, **search**) until user interaction is required.
4. Discard object.

This approach should be used instead of storing a reference to a single `QuickAddress` object in the calling code (for example, you should not pass this type of reference from one page to another).

The methods of the `QuickAddress` class generally take String parameters and return objects (or arrays of objects) of the wrapper classes.



Arrays are represented by square brackets [].

Where information returned in a wrapper object is required by a subsequent method, a "getter" method must be called on the returned object. This is demonstrated by the way monikers are passed into the **stepIn**, **refine** and **getFormattedAddress** methods in the following examples.

The following methods are available:

Method	Page
Constructor Constructs an instance of the ProWeb service endpoint URL	page 21
canSearch Checks that the combination of data mapping, engine and layout are valid to search with.	page 22
search Submits an initial search to the server.	page 23
BulkSearch Submits a batch of addresses to be verified by the server.	page 24
refine Performs a picklist refinement.	page 25
stepIn Performs a step-in to a picklist.	page 26
getFormattedAddress Retrieves the final, formatted address.	page 27
getAllDataSets Retrieves a list of available data mappings.	page 28
getDataMapDetail Retrieves a list of available datasets, additional datasets and DataPlus sets which are defined within a data mapping.	page 28
getAllLayouts Retrieves a list of layouts.	page 29
getExampleAddresses Returns an array of formatted sample addresses.	page 30
getLicenceInfo Retrieves a list of installed data resources.	page 31
getSystemInfo Retrieves diagnostic system information text from the server.	page 31
setConfigFile Sets the server configuration file.	page 32

Method	Page
setConfigSection Sets the server configuration section.	page 33
setEngineIntensity Sets the engine intensity.	page 33
setEngineType Sets the engine to use in the initial search.	page 34
setFlatten Determines whether the results will be flattened to a single picklist.	page 34
setThreshold Controls the number of items displayed in a picklist.	page 35
setTimeout Sets the search timeout period.	page 35

Search Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

Constructor

This method constructs an instance using the ProWeb service endpoint URL.

Prototype

```
public QuickAddress (String sEndpointURL)
                    throws Exception
```

Arguments

<i>sEndpointURL</i>	String specifying the URL of the ProWeb SOAP service.
---------------------	---

canSearch

This method checks that the combination of data mapping, engine and layout are valid to search with.



QAS recommend that the engine is set before this method is called. Otherwise, canSearch will not be able to check whether the engine is available.

This method will return **False** if you try to use a data mapping which is not installed or which does not have a valid licence, or if a layout has not been defined.

In addition, some data mappings are available for the Single Line and Typedown engines and not for the Verification or Keyfinder engines. Layouts can only be defined for specific data mappings and so may not always be valid for others.

If you are using a metered licence (see the "Installation" section of the Pro Web Installation And Administration Guide), and the value of the meter is less than or equal to zero, this function will return qaerr_CCNOUNITS. See the "Administrator Console" section of the Pro Web Installation And Administration Guide for more information about what happens when the value of a meter goes below zero.

Prototype

```
public CanSearch canSearch (String sDataID,  
                             String sLayoutName)  
                             throws Exception
```

Arguments

<i>sDataID</i>	ID of the data mapping to be searched with.
<i>sLayoutName</i>	Layout to be used in the final address.

Return Values

CanSearch	Contains the availability of searching, and any reasons for searching being unavailable. See page 69, for more details about the CanSearch class.
-----------	---

search

This method is the first stage of the address capture process and is used to submit an initial search to the server. A search must be performed against a specific data mapping and engine combination.

A search can return:

- A picklist of results with match information.
- A formatted final address (VERIFICATION only).
- A verified level (VERIFICATION only). See the "Web: Address Verification " section of the Pro Web Integration Guide.

The behaviour of a search depends on the choice of engine and engine options. The layout name is only necessary when you are performing a VERIFICATION search as the search result could be a `FormattedAddress`. Other engine types return only a `Picklist`.

Prototype

```
public SearchResult search (String sDataId,  
                           String[] asSearch,  
                           String sPromptSet,  
                           String sLayoutName)  
                           throws Exception
```

Arguments

<i>sDataId</i>	ID of the data mapping to be searched on.
<i>asSearch</i>	Array of search terms.
<i>sPromptSet</i>	Name of the prompt set used for these search terms. See page 73.
<i>sLayoutName</i>	Name of the layout (optional).

Return Values

<code>SearchResult</code>	See page 75 for information about the <code>SearchResult</code> class.
---------------------------	--

BulkSearch

This method is used to submit a batch of addresses to be verified by the server for a specific data mapping.

A bulk search returns:

- An array of BulkSearchItems, each of which has the original search terms, verification level and, if verified, the formatted address as a FormattedAddress object.

Prototype

```
public BulkSearchResult BulkSearch (String sDataId,  
                                   String[] asSearch,  
                                   String sPromptSet,  
                                   String sLayoutName)  
                                   throws Exception
```

Arguments

<i>sDataId</i>	ID of the data mapping to be searched on.
<i>asSearch</i>	Array of search addresses. Each array element corresponds to a single search.
<i>sPromptSet</i>	Name of the prompt set used for these search terms. See page 73.



*For the **BulkSearch** method, this is redundant as the Verification engine is always used.*

<i>sLayoutName</i>	Name of the layout (optional).
--------------------	--------------------------------

Return Values

BulkSearchResult	See page 79 for information about the BulkSearchResult class.
------------------	---

refine

This method performs a picklist refinement, after an initial search has been performed. If the user enters text to filter the current picklist, this method returns a list of results that is a subset of the original picklist.

For example, a picklist that contains a set of streets can be refined using the text "ba" to generate a new picklist that only contains entries beginning with the letters "ba".

Prototype

```
public Picklist refine (String sMoniker,  
                        String sRefinementText)  
    throws Exception
```

Arguments

<i>sRefinementText</i>	The text used to refine.
<i>sMoniker</i>	The Search Point Moniker of the picklist (item) that is being refined. See the "Using Pro Web" section of the Pro Web Integration Guide.

Return Values

Picklist	containing the results of the refinement. See Picklist Class on page 56.
----------	--

stepIn

This method performs a step-in to a picklist. This is used after an initial search has been performed, when the user selects an entry that can be expanded into elements "beneath" it.

For example, a picklist item that represents a street can often be stepped into so that a picklist of the premises "beneath" the street are displayed.

Prototype

```
public Picklist stepIn (String sMoniker)
                        throws Exception
```

Arguments

<i>sMoniker</i>	The Search Point Moniker of the picklist (item) being stepped into. See the "Using Pro Web" section of the Pro Web Integration Guide.
-----------------	---

Return Values

Picklist	containing the results of the step-in. See Picklist Class on page 56.
----------	---

getFormattedAddress

This method retrieves the final formatted address. This method formats a picklist item using its moniker in order to obtain the final formatted address. This is when the user selects a `PicklistItem` for which `isFullAddress()` returns `true`. The formatting is performed using a specified layout.

Prototype

```
public FormattedAddress getFormattedAddress (String
sLayoutName,
                                           String sMoniker)
                                           throws Exception
```

Arguments

<i>sMoniker</i>	Search Point Moniker of the address item to be formatted. See the "Using Pro Web" section of the Pro Web Integration Guide.
<i>sLayoutName</i>	Layout name that specifies which layout should be used to format the final address.

Return Values

FormattedAddress	Final formatted address. See <code>FormattedAddress</code> Class on page 45.
------------------	--

Status Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP, .NET and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

getAllDataSets

This method retrieves a list of available data mappings that can be searched against using Pro Web. For a list of installed data resources and their respective status, the **getLicenseInfo** method can be used (page 31).

Prototype

```
public DataSet[] getAllDataSets ()  
                                throws Exception
```

Return Values

DataSet[]	An array of data mappings, containing the data ID and name of each. See DataSet Class on page 41.
------------	---

getDataMapDetail

This method retrieves a list of available datasets, additional datasets and DataPlus sets which are defined within a data mapping. It retrieves any relevant status information such as days until data and licence expiry, and data version.

Prototype

```
public LicensedSet[] getDataMapDetail (string sDataId)  
                                throws Exception
```

getAllLayouts

This method retrieves a list of layouts that have been configured within the server configuration file, and that can be used to format address results.

A list of layouts may be useful for situations where the integration enables the user to select which layout to use to format an address result.

If only one layout is ever used within an integration, it would be more common to code the layout name explicitly.

Prototype

```
public Layout[] getAllLayouts (String sDataId)
                               throws Exception
```

Arguments

sDataId ID of the data mapping whose layouts are required.

Return Values

Layout[] An array of layouts, containing the name and a comment on each. See Layout Class on page 47.

getExampleAddresses

This method returns an array of formatted sample addresses. This is commonly used to see how a given layout works with different styles of address.

Prototype

```
public ExampleAddress[] getExampleAddresses (String sDataId,  
                                             String  
sLayoutName)  
                                             throws Exception
```

Arguments

<i>sDataId</i>	ID of the data mapping for which examples are required.
<i>sLayoutName</i>	Name of the layout for formatting.

Return Values

ExampleAddress[]	An array of example addresses, containing a formatted address and a comment. See ExampleAddress Class on page 43.
-------------------	---

getLicenceInfo

This method retrieves a list of installed data resources, and any relevant status information such as days until data and licence expiry, and data version. This method returns information about all datasets, DataPlus files and other third-party datasets.

This method is designed for the integrator to access diagnostic information regarding the data resources and is not suitable for display to web users. If you want to obtain a list of datasets that can be searched against, use the **getAllDataSets** method instead.

Prototype

```
public LicensedSet[] getLicenceInfo ()  
                                throws Exception
```

Return Values

LicensedSet[] An array of licence sets. See LicensedSet Class on page 49.

getSystemInfo

This method retrieves system (diagnostic) information text from the server; for example, file paths to configuration files and other resources, the version number and other internal settings.

Prototype

```
public String[] getSystemInfo ()  
                throws Exception
```

Return Values

String[] An array of strings, each of which contains a (tab-separated) name/value pair of system information.

Settings Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

setConfigFile

This method sets the server configuration file from which settings are read. If this is not set, the default file qawserve.ini will be used. The same configuration settings should generally be used for all SOAP actions to ensure consistent behaviour.

Prototype

```
public void setConfigFile (String sFile)
```

Arguments

sFile Configuration file name.



The file name specified in the method will be interpreted relative to the home directory of the program. For Windows, the home directory will be the Windows directory (unless you have explicitly specified another directory).

For example, an sFile of myfile.ini will be expected to be at C:\WINNT\myfile.ini

setConfigSection

This method sets the server configuration section from which settings are read. If this is not set, the default section [QADefault] will be used. The same configuration settings should generally be used for all SOAP actions to ensure consistent behaviour.

Prototype

```
public void setConfigSection (String sSection)
```

Arguments

sSection Configuration section name.

setEngineIntensity

This method sets engine intensity; that is, how hard the search engine will work to obtain a match. Higher intensity values may yield more results than lower intensity values, but will also result in longer search times. If this is not called before **search**, the server will use the value from the configuration file. If this value is not set, it will default to CLOSE.

Prototype

```
public void setEngineIntensity (String sEngineIntensity)
```

Arguments

sEngineIntensity One of the available engine intensity constants. See Constants Referring To Search Engine Intensity on page 15.

setEngineType

This method sets the engine type to be used in the initial search. This is either SINGLELINE, TYPEDOWN, VERIFICATION, or KEYFINDER. If this is not called before **search**, then the default value is used (SINGLELINE).

Prototype

```
public void setEngineType (String sEngineType)
```

Arguments

sEngineType One of the available engine constants. See Constants Referring To Search Engine Types on page 13.

setFlatten

This method defines whether the search results will be 'flattened' to a single picklist of deliverable results, or returned as hierarchical picklists of results that can be stepped into. See Engine Behaviour on page 83.

Flattened picklists are simpler for untrained users to navigate (i.e. Web site users), although can potentially lead to larger number of results. Hierarchical picklists are more suited to users who are trained in using the product (i.e. intranet users), and produce smaller picklists with results that can be stepped into.



If you are using metered licences, the picklist should always be flattened, as "per-click" licences only work with flattened picklists. For more information about metered licences, see "Metered Licences" in the Installation And Administration Guide.

If this method is not called before **search**, the default value of False is used, resulting in hierarchical picklists.

Prototype

```
public void setFlatten (boolean bFlatten)
```

Arguments

bFlatten Indicates whether the picklist should be flattened (containing only deliverable addresses) or should be hierarchical.

setThreshold

This method sets the threshold that is used to control the number of items displayed in a picklist, following a refinement or a step-in. Where the number of matches exceeds the threshold, an informational picklist result will be returned, advising the user to refine the picklist further.

Due to the algorithms used with returning result picklists, this value should only be used as an approximation. It should not be assumed that a larger number of picklist items will never be returned, or a lesser number of results will never be treated as if they exceeded the threshold.

If this threshold is not set before **refine** or **stepIn**, the server will use the value from the configuration file. If this value is not set, it will default to 25.

Prototype

```
public void setThreshold (int iThreshold)
```

Arguments

<i>iThreshold</i>	The value for the threshold. The maximum value for this setting is 750.
-------------------	---

setTimeout

This method sets the search timeout period; that is, the time threshold in milliseconds that will cause a search to abort if exceeded. If this method is not called before **search**, **stepIn** or **refine**, the server will use the value from the configuration file. If this value is not set, it will default to 10,000 milliseconds.

Prototype

```
public void setTimeout (int iTimeout)
```

Arguments

<i>iTimeout</i>	The value for the timeout period in milliseconds. 0 signifies that searches will not timeout.
-----------------	---

AddressLine Class

A wrapper class that encapsulates data associated with an address line, namely the formatted text itself plus various flags.

Constants

ADDRESS

This indicates that there are address elements on this line.

ANCILLARY

This indicates that there is additional data on this line.

DATAPLUS

This indicates that there is DataPlus information on this line.

NAME

This indicates that there is Name information on this line.

NONE

This indicates that there was no content specified for this line.

Prototypes and Usage Examples

Java Prototype (AddressLine class):

```
public static final String NONE = LineContentType._None;
public static final String ADDRESS =
LineContentType._Address;
public static final String NAME = LineContentType._Name;
public static final String ANCILLARY =
LineContentType._Ancillary;
public static final String DATAPLUS =
LineContentType._DataPlus;
```

Java Usage Example:

```
if (aAddressLines[i].getLineType() == AddressLine.DATAPLUS)
...
```

PHP Prototype (directly from proweb.wsdl):

```
<xs:simpleType name="LineContentType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="None" />
    <xs:enumeration value="Address" />
    <xs:enumeration value="Name" />
    <xs:enumeration value="Ancillary" />
    <xs:enumeration value="DataPlus" />
  </xs:restriction>
</xs:simpleType>
```

PHP Usage Example:

```
if ( $tLine->LineContent == "DataPlus" ) ...
```

C#.NET Prototype (AddressLine class):

```
public enum Types
{
    None = LineContentType.None,
    Address = LineContentType.Address,
    Name = LineContentType.Name,
    Ancillary = LineContentType.Ancillary,
    DataPlus = LineContentType.DataPlus
}
```

C#.NET Usage Example:

```
if (aAddressLines[i].LineType == AddressLine.Types.DataPlus)
...
```

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 38
getLabel Returns a text label that describes the contents of a line.	page 39
getLine Returns the final formatted address line.	page 39
getLineType Returns the line type (for example, NAME).	page 39
isOverflow Indicates that not all the address line could not fit on this line, and that the text had to overflow onto other lines.	page 39
isTruncated Indicates that the address line was too short to fit all of the formatted address, so the address was cut short.	page 40

Constructor

Constructs an instance from a SOAP layer object.

Prototype

```
public Constructor (com.gas.proweb.soap.AddressLineType t)
```


getLabel

Returns a text label which describes the contents of the line; for example, "Town". Line labels may be blank if multiple elements are fixed to a single line in the layout.

Prototype

```
public String getLabel ()
```

getLine

Returns the final formatted address line.

Prototype

```
public String getLine ()
```

getLineType

Returns the line type which is one of the constants on page 36; for example, NAME.

Prototype

```
public String getLineType ()
```

isOverflow

Flag that indicates that some address elements could not be formatted upon this line, and so had to overflow onto other lines. In order to avoid this, configure the layout to ensure that there are enough lines, and sufficient line width, to accommodate the elements you are returning. Configuring layouts is discussed in the "Configuration Settings" section of the Pro Web Installation And Administration Guide.

Prototype

```
public boolean isOverflow ()
```

isTruncated

Flag that indicates that this address line was too short to fit all of the formatted address, and so truncation has occurred. In order to avoid this, configure the layout to ensure that there are enough lines, and sufficient line width, to accommodate the elements you are returning. Configuring layouts is discussed in the "Configuration Settings" section of the Pro Web Installation And Administration Guide.

Prototype

```
public boolean isTruncated ()
```

Properties

DataplusGroup

This lists the multiple DataPlus values that are configured on one address line. If you have two or more DataPlus sets configured on one line, each set has its own DataplusGroup section.

The DataplusGroup consists of one or more DataplusGroupItems, and each DataplusGroupItem contains one DataPlus value.

The following example shows an extract of an AddressLine SOAP message returned by the server when both Gas and Electricity meter numbers are configured to appear on the same address line:

```
<qas:AddressLine LineContent="DataPlus" Truncated="True">
  <qas:Label>Gas and Electricity Meter Numbers</qas:Label>
  <qas:Line>2408630034|2408630036,1508436932|1508436943</qas:Line>
    <qas:DaplusGroup GroupName="GBRGAS">
      <qas:DaplusGroupItem>2408630034</qas:DaplusGroupItem>
      <qas:DaplusGroupItem>2408630036</qas:DaplusGroupItem>
    </qas:DaplusGroup>
    <qas:DaplusGroup GroupName="GBRELC">
      <qas:DaplusGroupItem>1508436932</qas:DaplusGroupItem>
      <qas:DaplusGroupItem>1508436943</qas:DaplusGroupItem>
    </qas:DaplusGroup>
</qas:AddressLine>
```

DataSet Class

A wrapper class that encapsulates the name and ID of a data mapping that may be searched upon by Pro Web.

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP, .NET and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 41
createArray Creates an array from the SOAP layer array.	page 42
getName Returns the name of the data mapping, as text, for display to the user.	page 42
getID Returns the data mapping identifier.	page 42
findByID Returns the data mapping identifier that relates to an entered string.	page 42

Constructor

Constructs an instance from a SOAP layer object.

Prototype

```
public DataSet(com.gas.proweb.soap.QADataset d)
```

createArray

Creates an array from the SOAP layer array.

Prototype

```
public static DataSet [] createArray  
(com.gas.proweb.soap.QADataset[] aDatasets)
```

getName

Returns the name of the data mapping (as text) that can be displayed to users. For example, "Australia". This string is defined within the `DataMappings` server configuration setting (see the "Configuration Settings" section of the Pro Web Installation And Administration Guide).

Prototype

```
public String getName ()
```

getID

Returns the data mapping identifier. This is the string that should be passed into QuickAddress searching methods. For example, "AUS".

Prototype

```
public String getID ()
```

findByID

Returns the data mapping identifier that relates to the search term that the user specifies. This is the string that should be passed into QuickAddress searching methods. For example, "AUS". This then returns the `DataSet` object for that country.

Prototype

```
public static DataSet findByID (DataSet[], array, string sID)
```

ExampleAddress Class

A wrapper class that encapsulates example address data, consisting of a sample `FormattedAddress` and a comment. This is commonly used to see how a given layout works with different styles of address.

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 43
createArray Creates an array from the SOAP layer array.	page 44
getComment Returns a text description of the example address.	page 44
getAddress Returns the formatted example address.	page 44

Constructor

Constructs an instance from a SOAP layer object.

Prototype

```
public ExampleAddress (com.gas.proweb.soap.QAExampleAddress a)
```

createArray

Creates an array from a SOAP layer array.

Prototype

```
public static ExampleAddress[] createArray  
(com.gas.proweb.soap.QAExampleAddress[] aAddresses)
```

getComment

Returns a text description of the example address. For example, "A typical residential address".

Prototype

```
public String getComment ()
```

getAddress

Returns the formatted example address. See FormattedAddress Class on page 45.

Prototype

```
public FormattedAddress getAddress ()
```

FormattedAddress Class

A wrapper class that encapsulates data associated with a final formatted address, namely an array of `AddressLines` plus various flags.

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 45
getAddressLines Returns an array of individual formatted address lines.	page 46
isOverflow Indicates that not all the address elements could be fitted in the final address.	page 46
isTruncated Indicates that some address elements could not fit completely on the line and were cut short.	page 46

Constructor

Constructs an instance from a SOAP layer object.

Prototype

```
public FormattedAddress (com.qas.proweb.soap.QAAddressType t)
```

getAddressLines

Returns an array of individual formatted address lines.

Prototype

```
public AddressLine[] getAddressLines ()
```

isOverflow

Flag that indicates that some address elements could not fit in the final address and these elements are not displayed (even partially). In order to avoid this, configure the layout to ensure that there are enough lines, and sufficient line width, to accommodate the elements you are returning. Configuring layouts is discussed in the "Configuration Settings" section of the Pro Web Installation And Administration Guide.

Prototype

```
public boolean isOverflow ()
```

isTruncated

Flag that indicates that some address elements could not completely fit on the address line and were cut short. In order to avoid this, configure the layout to ensure that there are enough lines, and sufficient line width, to accommodate the elements you are returning. Configuring layouts is discussed in the "Configuration Settings" section of the Pro Web Installation And Administration Guide.

Prototype

```
public boolean isTruncated ()
```


Layout Class

A wrapper class that encapsulates data that identifies a layout to be used in formatting a final address. The layouts that are relevant for a particular data mapping are defined in the server configuration file.

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 47
createArray Creates an array from the SOAP layer array.	page 48
getName Returns the name of a layout	page 48
getComment Returns a description of the layout.	page 48
findByName Enables the name of a layout to be searched for.	page 48

Constructor

Constructs an instance from a SOAP layer object.

Prototype

```
public Layout(com.qas.proweb.soap.QALayout l)
```

createArray

Creates an array from a SOAP layer array.

Prototype

```
public static Layout[] createArray  
(com.gas.proweb.soap.QALayout[] aLayouts)
```

getName

Returns the name of a layout. This is the layout name that should be passed to the **getFormattedAddress** method to format a picklist item. See page 27.

Prototype

```
public String getName ()
```

getComment

Returns a description of the layout.

Prototype

```
public String getComment ()
```

findByName

Enables the name of a layout to be searched for. This then returns the `Layout` object for that layout.

Prototype

```
public static Layout findByName (Layout[], aLayouts, String  
sLayoutName)
```

LicensedSet Class

A wrapper class that provides detailed status information about installed data resources, such as days until data and licence expiry, and data version. This method returns information about all datasets, DataPlus files and other third-party datasets.

This class is designed for the integrator to obtain diagnostic information regarding the data resources and is not designed for display to web users.

See the "Installation" section of the Pro Web Installation And Administration Guide for more information about data / licence expiry.

Constants

Constants For Warning Levels

NONE

This indicates that there are no warnings relating to the data.

DATA_EXPIRING

This indicates that the data is close to its expiry date. The number of days before this warning is returned can be controlled using the NotifyDataWarning configuration setting. See the "Configuration Settings" section of the Pro Web Installation And Administration Guide.

LICENCE_EXPIRING

This indicates that the licence that controls the usage of the data is close to its expiry date. The number of days before this warning is returned can be controlled using the NotifyLicenceWarning configuration setting. See the "Configuration Settings" section of the Pro Web Installation And Administration Guide.

CLICKS_LOW

This indicates that the number of available clicks for the data have run out and that overdraft clicks are being used. See the "Administrator Console" section of the Pro Web Installation And Administration Guide for more information about clicks and the overdraft facility.

EVALUATION

This indicates that an evaluation licence is being used for the data.

NO_CLICKS

This indicates that there are no more clicks remaining which can be used to search against the data, so the data cannot be used.

DATA_EXPIRED

This indicates that the data has passed its expiry date and so cannot be used.

EVAL_LICENCE_EXPIRED

This indicates that the evaluation licence which controls the use of this data has expired.

FULL_LICENCE_EXPIRED

This indicates that the full (non-evaluation) licence which controls the use of this data has expired.

LICENCE_NOT_FOUND

This indicates that the product is unable to locate a licence for this data, and so it cannot be used.

DATA_UNREADABLE

This indicates that this data cannot be opened or read, and so is unusable.

Prototypes and Usage Examples

Java Prototype (LicensedSet class):

```
public static final String NONE;
public static final String DATA_EXPIRING;
public static final String LICENCE_EXPIRING;
public static final String CLICKS_LOW
public static final String EVALUATION
public static final String NO_CLICKS
public static final String DATA_EXPIRED;
public static final String EVAL_LICENCE_EXPIRED;
public static final String FULL_LICENCE_EXPIRED;
public static final String LICENCE_NOT_FOUND;
public static final String DATA_UNREADABLE;
```

Java Usage Example:

```
if (!info.getWarningLevel().equals(LicensedSet.NONE))
```

PHP Prototype (directly from proweb.wsdl):

```
<xs:simpleType name="LicenceWarningLevel">
  <xs:restriction base="xs:string">
    <xs:enumeration value="None" />
    <xs:enumeration value="DataExpiring" />
    <xs:enumeration value="LicenceExpiring" />
    <xs:enumeration value="ClicksLow" />
    <xs:enumeration value="Evaluation" />
    <xs:enumeration value="NoClicks" />
    <xs:enumeration value="DataExpired" />
    <xs:enumeration value="EvalLicenceExpired" />
    <xs:enumeration value="FullLicenceExpired" />
    <xs:enumeration value="LicenceNotFound" />
    <xs:enumeration value="DataUnreadable" />
  </xs:restriction>
</xs:simpleType>
```

PHP Usage Example:

```
if ( $tInfo->WarningLevel != "None") ...
```

C#.NET Prototype (LicensedSet class):

```
public enum WarningLevels
{
    None,
    DataExpiring,
    LicenceExpiring,
    ClicksLow
    Evaluation
    NoClicks
    DataExpired,
    EvalLicenceExpired,
    FullLicenceExpired,
    LicenceNotFound,
    DataUnreadable
}
```

C#.NET Usage Example:

```
if (info.WarningLevel != LicensedSet.WarningLevels.None) ...
```

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object	page 53
createArray Creates an array from the SOAP layer array.	page 53
getID Returns a short identifier for the data resource.	page 54
getDescription Returns a text description of the data resource.	page 54
getCopyright Returns a text description of the copyright message for the data.	page 54

Method	Page
getVersion Returns a text description of the data version.	page 54
getBaseCountry Returns a text description of the base country.	page 54
getStatus Returns a text description of the data status.	page 55
getServer Returns the name of the server where the data is situated.	page 55
getWarningLevel Returns the warning level for the data resource.	page 55
getDaysLeft Returns the number of days remaining until the data becomes unusable.	page 55
getDataDaysLeft Returns the number of days before the data resource expires.	page 55
getLicenceDaysLeft Returns the number of days remaining before the licence that controls the data resource expires.	page 56

Constructor

Constructs an instance from a SOAP layer object.

Prototype

```
public LicensedSet (com.qas.proweb.soap.QALicensedSet s)
```

createArray

Creates an array from a SOAP layer array.

Prototype

```
public static LicensedSet[] createArray  
(com.qas.proweb.soap.QALicenceInfo info)
```

getID

Returns a short identifier for the data resource. For example "AUSMOS" for the Australian Mosaic DataPlus set.

Prototype

```
public String getID ()
```

getDescription

Returns a text description of the data resource. For example, "Australia MOSAIC code" for the Australian Mosaic DataPlus set.

Prototype

```
public String getDescription ()
```

getCopyright

Returns a text copyright message for the data.

Prototype

```
public String getCopyright ()
```

getVersion

Returns a text description of the data version. For example, "01 April 2006".

Prototype

```
public String getVersion ()
```

getBaseCountry

Returns a short identifier for the base country (corresponding to a data mapping ID).

Prototype

```
public String getBaseCountry ()
```


getStatus

Returns a text description of the data status, such as licence and expiry information.

Prototype

```
public String getStatus ()
```

getServer

Returns the name of the server where the data is situated.

Prototype

```
public String getServer ()
```

getWarningLevel

Returns the warning level for the data resource. See the warning level constants on page 49.

Prototype

```
public String getWarningLevel ()
```

getDaysLeft

Returns the number of days remaining before the data is unusable. This is the lesser of the values obtained from the **getDataDaysLeft** and **getLicenceDaysLeft** methods.

Prototype

```
public int getDaysLeft ()
```

getDataDaysLeft

Returns the number of days remaining before the data resource expires.

Prototype

```
public int getDataDaysLeft ()
```

getLicenceDaysLeft

Returns the number of days remaining before the licence that controls the data resource expires.

Prototype

```
public int getLicenceDaysLeft ()
```

Picklist Class

This is a wrapper class that encapsulates picklist data. This class defines a set of result items and properties that may be returned at certain stages in the search process. The picklist will be sorted in the order in which it is strongly advised to display the entries.

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 58
getTotal Returns the total number of matches.	page 58
getMoniker Returns the full picklist moniker.	page 58
getPrompt Returns a short text prompt for display to the user.	page 58
getItems Returns the array of picklist items.	page 59

Method	Page
isAutoStepinSafe Indicates that a match to a single non-deliverable result can be stepped into.	page 59
isAutoStepinPastClose Indicates that a match to a single non-deliverable result can be auto-stepped into.	page 59
isAutoStepinSingle Indicates whether the picklist contains a single, non-informational result that can be stepped into.	page 59
isAutoFormatSafe Indicates a match to a single deliverable result.	page 60
isAutoFormatPastClose Indicates a match to a single deliverable result, but also other lesser matches.	page 60
isAutoFormatSingle Indicates that a picklist contains a single final address item.	page 60
isLargePotential Indicates that a search has potentially returned too many items to display in a picklist.	page 60
isMaxMatches Indicates that the number of matches returned from a search has exceeded the maximum allowed.	page 61
isMoreOtherMatches Indicates that the number of matches returned from a search has exceeded the picklist threshold, and that there are other matches that can be displayed.	page 61
isOverThreshold Indicates that the number of matches returned from a search has exceeded the picklist threshold.	page 61
isTimeout Indicates that the search exceeded the specified timeout value.	page 61

Constructor

Constructs an instance from a SOAP layer object.

Prototype

```
Picklist(com.gas.proweb.soap.QAPicklistType p)
```

getTotal

Returns the total number of matches (not including informational prompts). This number should only be used for display purposes and should not be assumed as the size of the returned picklist, which will often contain informational items and which is restricted by a threshold.

Prototype

```
public int getTotal ()
```

getMoniker

Returns the full picklist moniker, which is the moniker that can be used to recreate the given picklist. Full picklist monikers are typically used when refining a picklist using search text to filter the results (see page 25).

Prototype

```
public String getMoniker ()
```

getPrompt

Returns a short text prompt that may be displayed to the user in an interactive scenario. This indicates the address element that they should enter. For example "Enter building number/name or organisation".

Prototype

```
public String getPrompt ()
```

getItems

Returns the `PicklistItem` array. Each picklist item represents a separate result, and has associated information.

Prototype

```
public PicklistItem[] getItems ()
```

isAutoStepinSafe

Indicates a match to a single non-deliverable result that can be stepped into. This should only be checked after an initial search, and not once user interaction has occurred. QAS recommend that the integration should automatically step into the result, without user interaction. This aids address capture efficiency. Stepping into address results is performed using the **stepIn** method (see page 26).

Prototype

```
public boolean isAutoStepinSafe ()
```

isAutoStepinPastClose

Indicates a match to a single non-deliverable result that can be auto-stepped into, but also indicates that there are other lesser matches. Refer to Auto Step-In on page 88 for detailed information about this. Stepping into address results is performed using the **stepIn** method of the `QuickAddress` class.

Prototype

```
public boolean isAutoStepinPastClose ()
```

isAutoStepinSingle

Indicates whether the picklist contains a single, non-informational result that can be stepped into. Refer to Auto Step-In on page 88 for detailed information about this. Stepping into address results is performed using the **stepIn** method of the `QuickAddress` class.

Prototype

```
public boolean isAutoStepinSingle ()
```

isAutoFormatSafe

Indicates a match to a single deliverable result. Refer to Auto Step-In on page 88 for detailed information about this. Formatting address results is performed using the **getFormattedAddress** method of the `QuickAddress` class.

Prototype

```
public boolean isAutoFormatSafe ()
```

isAutoFormatPastClose

Indicates a match to a single deliverable result, but also to other lesser matches. Refer to Auto Step-In on page 88 for information about auto-formatting. Formatting address results is performed using the **getFormattedAddress** method of the `QuickAddress` class.

Prototype

```
public boolean isAutoFormatPastClose ()
```

isAutoFormatSingle

Indicates that the picklist contains a single final address item. Refer to Auto Step-In on page 88 for information about auto-formatting. Formatting address results is performed using the **getFormattedAddress** method of the `QuickAddress` class.

Prototype

```
public boolean isAutoFormatSingle ()
```

isLargePotential

Indicates that the search has potentially returned too many items to display in a picklist. This signifies that further refinement is required before a picklist containing all of the results can be returned. Picklist refinement is performed using the **refine** method of the `QuickAddress` class.

Prototype

```
public boolean isLargePotential ()
```

isMaxMatches

Indicates that the number of matches returned from a search has exceeded the maximum allowed. This signifies that the search was too broad to match, and that a new search should be performed with more information specified.

For search tips, refer to Appendix C of the Pro Web Integration Guide.

Prototype

```
public boolean isMaxMatches ()
```

isMoreOtherMatches

Indicates that the number of results returned from a search has exceeded the picklist threshold, and therefore, that there are additional other matches that could be displayed. A subset of the matches has been returned in the picklist, and an informational picklist item that enables the user to access the others.

Prototype

```
public boolean isMoreOtherMatches ()
```

isOverThreshold

Indicates that the number of results returned from a search has exceeded the picklist threshold and only a single informational picklist item has been returned. This enables the user to access the full results.

Prototype

```
public boolean isOverThreshold ()
```

isTimeout

Indicates that the search exceeded the specified timeout value. If this is unexpected for the given search, this could either signify that the timeout is set too low, or that the search was too broad. For search tips, refer to Appendix C of the Pro Web Integration Guide.

Prototype

```
public boolean isTimeout ()
```

PicklistItem Class

This is a wrapper class that encapsulates the data associated with one item in a picklist, which may be related to an address item or to an informational item.

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 63
getText This string is displayed to the user for the picklist text.	page 64
getPostcode Returns the postcode for display.	page 64
getScore Returns the percentage score of this item.	page 64
getMoniker Returns the moniker representing the item.	page 64
getPartialAddress Returns the full address details which have been captured so far.	page 65
isFullAddress Indicates whether this item represents a full deliverable address.	page 65
canStep Indicates whether this item can be stepped into to produce a new picklist with more detail.	page 65
isAliasMatch Indicates whether a picklist item has been produced by a match to an alias of an item.	page 65

Method	Page
isPostcodeRecoded Indicates that a postcode was searched on.	page 66
isIncompleteAddress Indicates that the picklist item does not contain all the information required to make it a deliverable address.	page 66
isUnresolvableRange Indicates that the picklist item is a range of premises which cannot be expanded.	page 66
isCrossBorderMatch Indicates whether this item represents a nearby area, outside the strict initial boundaries of the search.	page 67
isDummyPOBox Indicates whether this item is a dummy PO Box.	page 67
isName Indicates that the picklist item contains names information.	page 67
isInformation Indicates that the picklist item is an informational item.	page 68
isWarnInformation Indicates that the picklist item is a warning informational item.	page 68
isPhantomPrimaryPoint Indicates that the picklist item is a phantom primary point.	page 69
isMultiples Indicates that the picklist item represents multiple addresses.	page 69

Constructor

Constructs an instance from a SOAP layer object.

Prototype

```
public PicklistItem(com.qas.proweb.soap.PicklistEntryType t)
```

getText

This is the string that must be displayed to the user for the picklist text. This should be used in combination with the **getPostcode** method. The main picklist text and postcode have been separated to ease integration formatting for display.

Prototype

```
public String getText ()
```

getPostcode

This returns the postcode for display, and should be used in conjunction with the **getText** method for the picklist text.

This string is for display purposes only, and may be returned blank if no postcode is associated with the picklist string.

Prototype

```
public String getPostcode ()
```

getScore

Returns the percentage score of this item; 0 if not applicable. This can be displayed to the user as a guide of the quality of match produced.

Prototype

```
public int getScore ()
```

getMoniker

Returns the moniker representing the item. The moniker can be used to perform actions upon the picklist item, such as stepping in (using the **stepIn** method: see page 26) or formatting the final address (using the **getFormattedAddress** method: see page 27).

Prototype

```
public String getMoniker ()
```

getPartialAddress

Returns the full address details which have been captured so far. The partial address item is partially formatted into a single string, which may be useful for display in user interactive environments. QAS do not recommend that this string is displayed as the picklist text; the **getText** method can be used for this purpose (see page 64).

Prototype

```
public String getPartialAddress ()
```

isFullAddress

Indicates whether this item represents a full deliverable address. If the user selects this picklist item, then it should be formatted into a final address, signifying the end of the address capture process. Formatting picklist items is performed using the **getFormattedAddress** method (see page 27).

Prototype

```
public boolean isFullAddress ()
```

canStep

Indicates whether the item may be stepped into to produce a new picklist with more detail. For example, a picklist item that represents a street can be stepped into to produce a new picklist of premises within the street. Stepping into picklist items is performed using the **stepIn** method (see page 26). This method only applies to hierarchical picklists; see page 112.

Prototype

```
public boolean canStep ()
```

isAliasMatch

Indicates whether the picklist item has been produced by a match to an alias of the item. This is to enable the integrator to display the picklist item with a different icon to a non-alias match. See Appendix C of the Pro Web Integration Guide for more information about alias matching.

Prototype

```
public boolean isAliasMatch ()
```

isPostcodeRecoded

Indicates that a postcode was searched on, and that the match was made to a newer recoded version of the postcode. See Appendix C of the Pro Web Integration Guide for more information.

Prototype

```
public boolean isPostcodeRecoded ()
```

isIncompleteAddress

Indicates that the picklist item does not contain all of the information required to make it a deliverable address. This is commonly returned when searching within data mappings that do not contain premises information.

The integration should prompt the user for additional address information that will be passed to the **refine** method of the `QuickAddress` class in order to obtain a deliverable address. See Special Picklist Items on page 99 for more information.

Prototype

```
public boolean isIncompleteAddress ()
```

isUnresolvableRange

Indicates that the picklist item is a range of premises which cannot be expanded, due to a lack of information about the possible elements within the range.

The integration should prompt the user to provide additional premises information that will be passed to the **refine** method of the `QuickAddress` class in order to obtain a deliverable address. See Special Picklist Items on page 99 for more information.

Prototype

```
public boolean isUnresolvableRange ()
```

isCrossBorderMatch

Indicates whether this item represents a nearby area, outside the strict initial boundaries of the search. CrossBorderMatches are commonly returned from bordering locality matches in the AUS data, as specified by AMAS certification.

Prototype

```
public boolean isCrossBorderMatch ()
```

isDummyPOBox

Indicates whether this item is the dummy PO Box, which represents the set of PO Boxes. If this item is stepped into, the resulting picklist will contain all PO Boxes relevant to the search term. This enables the integrator to display the picklist item with a different icon to a normal match, if they so choose. This method only applies to hierarchical picklists; see page 112.

Prototype

```
public boolean isDummyPOBox ()
```

isName

Indicates that the picklist item contains names information. This is only relevant when searching within data mappings that contain Names information, such as GBR together with GBR Names. It enables the integrator to display the picklist item with a different icon to a non-name match.

Prototype

```
public boolean isName ()
```

isInformation

Indicates that the picklist item is an informational item. This means that the entry does not correspond to any particular address item. Informational picklist items are designed to help the user to complete the address capture process, and must be displayed in the picklist. See Informational Picklist Items on page 114. This enables the integrator to display the picklist item with a different icon to a non-informational entry.

This only has significance when searching if **setFlatten**(false) has been called (*QuickAddress* class).

Prototype

```
public boolean isInformation ()
```

isWarnInformation

Indicates whether this picklist item is a warning informational item. Unlike a normal informational entry, this will usually be displayed in the picklist when the user should be warned of a situation, such as a search yielding no matches. This attribute enables the integrator to display the picklist item with a different icon to a non-informational entry. This only has significance when searching when **setFlatten**(false) has been called (*QuickAddress* class). See Engine Behaviour on page 83.

Prototype

```
public boolean isWarnInformation ()
```

isPhantomPrimaryPoint

Indicates whether this picklist item is a phantom primary point. This is only relevant when using the AUS data. A phantom primary point is a premises which is non-deliverable unless the user enters further secondary information. This secondary information may or may not be in the actual data. The user must enter this sub-premises information in order to complete a final address match.

If the user selects a picklist item with this attribute, then the integration must prompt the user for the sub-premises information and pass this to the **refine** method of the `QuickAddress` class in order to get a deliverable address. See Special Picklist Items on page 99 for more information.

Prototype

```
public boolean isPhantomPrimaryPoint ()
```

isMultiples

Indicates that the picklist item represents multiple addresses, merged into a single entry. This element is solely to enable the integrator to display the picklist item with a different icon than a non-multiple entry.

Prototype

```
public boolean isMultiples ()
```

CanSearch Class

A wrapper class that encapsulates the results of a **CanSearch** operation. Such results include the availability of searching, and, if searching is not available, the reasons why searching is not available.

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 70
isOK Returns whether searching is available for the specified data mapping/engine/layout combination.	page 70
getMessage Returns the reason that searching has been reported as being unavailable.	page 70

Constructor

Constructs an instance from SOAP layer object.

Prototype

```
public CanSearch (com.qas.proweb.soap.QASearchOk cs)
```

isOk

Returns whether searching is available for the specified country/engine/layout combination.

Prototype

```
public boolean IsOk ()
```

getMessage

Returns the reason that searching has been reported as unavailable. For example, "Dataset cannot be used".

Prototype

```
public String getMessage()
```


PromptLine Class

A wrapper class that encapsulates information about a prompt set line. An ordered list (array) of prompt lines make up a prompt set.

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 71
getPrompt Returns a hint as to what should be entered into the input field.	page 72
getExample Returns an example of which address element can be entered into the prompt line.	page 72
getSuggestedInputLength Returns the minimum length of the input field.	page 72

Constructor

Constructs an instance from SOAP layer object.

Prototype

```
public PromptLine (com.qas.proweb.soap.PromptLine t)
```

getPrompt

Returns a text hint about what should be entered into the input field associated with this prompt line. For example, "Building number or name".

Prototype

```
public String getPrompt ()
```

getExample

Returns an example of which address element can be entered into the prompt line, to aid the user. For example, "Flat A".

Prototype

```
public String getExample ()
```

getSuggestedInputLength

Returns the minimum length of the input field.

Prototype

```
public int getSuggestedInputLength ()
```

PromptSet Class

A wrapper class that encapsulates data representing a prompt set. A prompt set consists of an array of prompt set lines. See page 110 for more information about prompt sets.

Constants

See page 16 for information about constants for referring to prompt sets.

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 73
getLines Returns the array of prompt lines that comprise the prompt set.	page 74
getLinePrompts Returns a string of prompts from the prompt line array.	page 74
isSearchingDynamic Returns whether dynamic search submission is recommended.	page 74

Constructor

Constructs an instance from SOAP layer object.

Prototype

```
public PromptSet (com.gas.proweb.soap.QAPromptSet t)
    throws Exception
```

getLines

Returns the array of prompt lines that make up this prompt set. Each prompt set line should correspond to a separate input field for the user to enter their search terms.

Prototype

```
public PromptLine[] getLines ()
```

getLinePrompts

Returns a String[] of prompts (from the prompt line array).

Prototype

```
public String[] getLinePrompts ()
```

isSearchingDynamic

Returns whether a search can be submitted as a dynamic search. See Typedown Engine on page 94, for details about dynamic searching.

Prototype

```
public boolean isSearchingDynamic ()
```

SearchResult Class

A wrapper class that encapsulates data that may be returned by an initial search: a Picklist and/or FormattedAddress and VerifyLevel (for verification searches).

Constants

Constants for verify levels

None

This indicates that there was no verification upon the search.

Verified

This indicates that the address that was searched upon has been matched to a single deliverable address.

InteractionRequired

This indicates that the address that was searched upon has been matched to a single deliverable address, although user interaction is recommended to confirm that it is correct.

PremisesPartial

This indicates that the address that was searched upon could not be matched to a complete deliverable result, and instead has been matched to a partially-complete address at premises level. This implies that there is also a picklist associated with the partial address.

StreetPartial

The address that was searched upon could not be matched to a complete deliverable result, and instead has been matched to a partially-complete address at street level. This implies that there is also a picklist associated with the partial address.

Multiple

This indicates that the address searched could not be matched to a single deliverable result, and instead has matched equally to more than one result.

See the "Web: Address Verification " section of the Pro Web Integration Guide for more information about verify levels.

Prototypes and Usage Examples

Java Prototype (SearchResult class):

```
public static final String None = VerifyLevelType._None;
public static final String Verified =
VerifyLevelType._Verified;
public static final String InteractionRequired =
VerifyLevelType._InteractionRequired;
public static final String PremisesPartial =
VerifyLevelType._PremisesPartial;
public static final String StreetPartial =
VerifyLevelType._StreetPartial;
public static final String Multiple =
VerifyLevelType._Multiple;
```

Java Usage Example:

```
if
(m_Result.getVerifyLevel().equals(SearchResult.Verified))
...
```

PHP Prototype (directly from proweb.wsdl):

```
<xs:simpleType name="VerifyLevelType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="None" />
    <xs:enumeration value="Verified" />
    <xs:enumeration value="InteractionRequired" />
    <xs:enumeration value="PremisesPartial" />
    <xs:enumeration value="StreetPartial" />
    <xs:enumeration value="Multiple" />
  </xs:restriction>
</xs:simpleType>
```

PHP Usage Example:

```
if ( $result->sVerifyLevel == "Verified" ) ...
```

C#.NET Prototype (SearchResult class):

```
public enum VerificationLevels
{
    None = VerifyLevelType.None,
    Verified = VerifyLevelType.Verified,
    InteractionRequired = VerifyLevelType.InteractionRequired,
    PremisesPartial = VerifyLevelType.PremisesPartial,
    StreetPartial = VerifyLevelType.StreetPartial,
    MultipleMatch = VerifyLevelType.Multiple
}
```

C#.NET Usage Example:

```
if (m_SearchResult.VerifyLevel ==
    SearchResult.VerificationLevels.Verified) ...
```

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 78
getAddress (Verification Only) Returns the address.	page 78
getPicklist Returns the picklist.	page 78
getVerifyLevel (Verification Only) Returns the verify level.	page 78

Constructor

Constructs an instance from a SOAP layer object.

Prototype

```
SearchResult (com.qas.proweb.soap.QASearchResult sr)
```

getAddress

(Verification only). Returns the address; this will be null unless the Verification engine returns a single formatted address.

Prototype

```
public FormattedAddress getAddress ()
```

getPicklist

Returns the picklist; this may be null, depending upon the engine and number of results. Picklists are returned when searching with the Single Line or Typedown engines, and may be returned using the Verification or Keyfinder engines.

Prototype

```
public Picklist getPickslislist ()
```

getVerifyLevel

(Verification only). Returns the verify level, which specifies how well the search has been matched.

Prototype

```
public String getVerifyLevel ()
```


BulkSearchResult Class

A wrapper class that encapsulates data that is returned by the bulk verification search.

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 79
BulkSearchItem (Verification Only) Returns a specific BulkSearchItem, returns nothing if the iIndex is out of range.	page 79

Constructor

Constructs an instance from a SOAP layer object.

BulkSearchItem

Returns a specific BulkSearchItem. Returns nothing (null) if the iIndex is out of range.

Prototype

```
public BulkSearchItem (Integer iIndex)
```

Properties

Count	The number of BulkSearchItems contained in this BulkSearchResult object.
BulkSearchItems	Returns the array of BulkSearchItems.

BulkSearchItem Class

A wrapper class that encapsulates the elements contained in a `BulkSearchResult` object and consists of the original address as input, the verification level and, if verified, a formatted address object.

Constants

Constants for verify levels

None

This indicates that there was no verification upon the search.

Verified

This indicates that the address that was searched upon has been matched to a single deliverable address.

InteractionRequired

This indicates that the address that was searched upon has been matched to a single deliverable address, although user interaction is recommended to confirm that it is correct.

PremisesPartial

This indicates that the address that was searched upon could not be matched to a complete deliverable result, and instead has been matched to a partially-complete address at premises level. This implies that there is also a picklist associated with the partial address.

StreetPartial

The address that was searched upon could not be matched to a complete deliverable result, and instead has been matched to a partially-complete address at street level. This implies that there is also a picklist associated with the partial address.

Multiple

This indicates that the address searched upon could not be matched to a single deliverable result, and instead has matched equally to more than one result.

See the "Web: Address Verification " section of the Pro Web Integration Guide for more information about verify levels.

Methods

There are minor differences between the methods when different languages are used: this chapter refers only to JSP. The differences between JSP and .NET, and PHP are listed in Appendix A: Differences Between Common Classes on page 117.

The following methods are available:

Method	Page
Constructor Constructs an instance from a SOAP layer object.	page 81

Constructor

Constructs an instance from a SOAP layer object.

Properties

Address	A formatted address, or null if the address was not verified.
VerifyLevel	Contains the verification level.
InputAddress	A string representing the address as originally specified.

Engine Behaviour

QuickAddress Pro Web provides four distinct search engines that can be implemented to facilitate the address capture process. Integrations are usually based on one of these engines.

The available engines are:

- Single Line
- Typedown
- Verification
- Keyfinder

These four engines are designed to be used for different methods of address capture, and the choice of engine will depend on the way in which you want to implement the address capture process.



There are some restrictions around the Verification and Keyfinder engines. See the "Web: Address Verification" and "Web: Key Search" sections of the Pro Web Integration Guide for details.

The differences between these engines are discussed in the following sections:

- Single Line Engine on page 84
- Typedown Engine on page 94
- Verification Engine on page 102
- Keyfinder Engine on page 108

Single Line Engine

The Single Line engine is designed so that the user can enter minimal information in order to produce a list of matching and close-matching addresses from which they can select the required one.

The Single Line engine works in the opposite way to the Typedown Engine (see page 94), in that the user enters address elements, starting with the most specific (for example, a house number and/or a street name) and then moves on to more general elements (for example, a town or postcode in the United Kingdom). When the user has entered all the information, they should start the search manually.

The search term entered can contain elements such as wildcards.

A prompt set can be used to guide the user as to what information should be entered at each stage. Ideally, the user will always enter information that generates the smallest number of matches. This makes the search more efficient, and makes it easier for the user to select the final address. Which prompt set should be used depends on whether the Flatten engine mode is set to **Yes** or **No**. This is discussed in the following relevant sections.

The Single Line engine is demonstrated within the "Web: Address Capture" and all "Intranet: Rapid Addressing" shipping scenarios. See the "Web: Address Capture" and "Intranet: Rapid Addressing - Single Line" sections of the Pro Web Integration Guide for details.

The Single Line scenario for address capture commonly requires user interaction after the initial search has been submitted, in order to select a matching address from the returned set of picklist items. The manner in which the engine will return the picklist is determined by one of two modes:

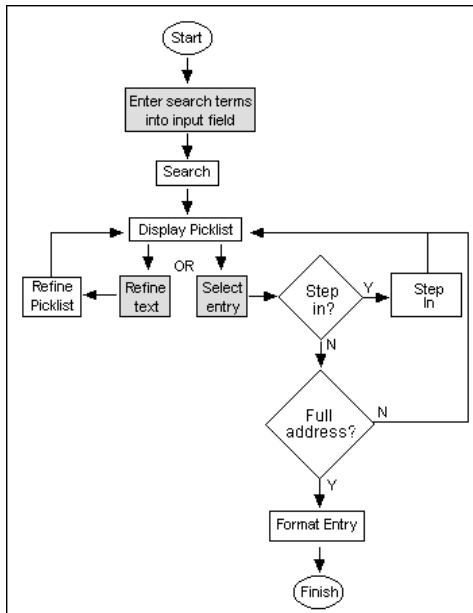
- Hierarchical mode
- Flattened mode.

Setting the mode is controlled through the **Flatten** engine option. This does not affect the way in which the initial search is performed, but does affect the number and type of picklist items that are returned.

Hierarchical Mode

The hierarchical mode of the Single Line engine is available in all Intranet: Rapid Addressing scenarios. See the "Intranet: Rapid Addressing - Single Line" and "Intranet: Rapid Addressing - Standard" sections of the Pro Web Integration Guide.

The following diagram demonstrates the flow of searching using this mode:



In this diagram, the shaded boxes represent user actions; all other boxes are integration actions.

This diagram illustrates that address capture using the hierarchical mode of the Single Line engine requires interaction with the user (picklist refinement and step-ins).

The user therefore has more control over the address capture process, but this scenario requires the following conditions:

- Users who are trained upon or familiar with the address capture process;
- A responsive connection between the client and the integration.

This scenario is therefore more suitable for internal applications such as an Intranet site than for public Web site address capture.

The flattened mode of the Single Line engine, and the Verification engine are suitable for public Web site address capture, as discussed in the following sections.

Searches that are performed using the Single Line engine in hierarchical mode will create a picklist which may have one or more of the following properties:

- Hierarchical picklist items which can be 'stepped into' to produce new picklists;
- Picklist items that may contain 'informational' picklist items (see page 114);
- Picklists that will commonly contain fewer items than if the search was performed with the flattened mode, due to their hierarchical nature.

The hierarchical mode of the Single Line engine is designed to allow users to enter any search terms that they feel are appropriate. QAS recommend, therefore, that they use the **OneLine** prompt set (see page 111). Due to the hierarchical nature of the picklist returned, and the fact that the user can step into entries to obtain more detail, there is not a requirement to use the more restrictive prompt sets (see page 110) to minimise picklist size.

Example of Hierarchical Mode Searching

This example shows how hierarchical mode Single Line searching can be used in a situation where a customer on a Web site is encouraged to enter their address details. However, the customer does not enter a premises number, so the hierarchical mode generates a picklist of possible addresses.

The example uses 329 Werona Kingston Road, Kooroocheang, VIC from the Australian data.

1. Select the Intranet: Rapid Addressing - Single Line scenario.
2. Select Australia from the **Country** drop-down box.

3. Type **Werona Kingston Road, Kooroocheang** in the Search field.

Enter Your Search

Please select a country or datamap from the list below, and enter your search terms.

Country or Datamap	<input type="text" value="Australia"/>	
Search	<input type="text" value="Werona Kingston Road, Kooroocheang"/>	<input type="button" value="Search"/>

The next step is to drill-down to the right address.

4. Click the **Search** button.

A list of possible addresses is displayed.

Select from Address Picklist

Enter text to refine your search, or select one of the addresses below.

<input type="button" value="New"/>	<input type="button" value="Back"/>	Enter selection: <input type="text"/>	<input type="button" value="Search"/>
------------------------------------	-------------------------------------	---------------------------------------	---------------------------------------

Searching on... 'Werona Kingston Road, Kooroocheang'

	Werona Kingston Road, KOOROOCHANG VIC	100%
	Werona Kingston Road, SMEATON VIC	100%

The possible addresses are shown in the format of an hierarchical picklist. This requires further selection to drill down to the required address (see Example of Flattened Mode Searching on page 91, for a method that does not require further drilling down).

5. Click on "Werona Kingston Road, Kooroocheang" to expand the picklist entry.

Select from Address Picklist

Enter text to refine your search, or select one of the addresses below.

<input type="button" value="New"/>	<input type="button" value="Back"/>	Enter building number/name: <input type="text"/>	<input type="button" value="Search"/>
------------------------------------	-------------------------------------	--	---------------------------------------

Searching on... 'Werona Kingston Road, Kooroocheang'

Werona Kingston Road, KOOROOCHANG VIC 100%

- [329](#)
- [340](#)
- [356](#)

6. Select "329".

The selected address is returned.

Confirm your address

Please confirm your address below.

	329 Werona Kingston Rd
	KOOROOCHANG VIC 3364
Country	Australia

This completes the address capture process.

< Back	Accept
--------	--------

7. Click the **Accept** button to confirm the address.

Auto Step-In

Auto step-in functionality is designed to make the address capture process more efficient for users of the Single Line engine with hierarchical picklists, by reducing the number of interactive steps that the user has to perform to capture an address.

When a search has been performed by the engine, the picklist that is returned contains many properties and flags, some of which correspond to the auto step-in functionality. These should only be checked after an initial search, and not after the point where a picklist has been displayed to the user for interaction.

There are two sets of relevant flags:

- Auto-step flags;
- Auto-format flags.

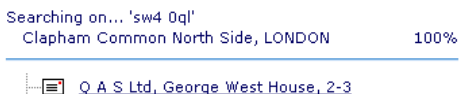
When auto-step flags are returned from an address search, QAS suggest that the Integration code performs a **stepin** action on the first picklist item.

When auto-format flags are returned from an address search, QAS suggest that the Integration code performs a **format** action on the first picklist item.

For example, if the user searches against the GBR data using the search term "SW4 0QL", the engine will return a picklist which contains the **autostepsafe** flag, and a single 100% match, as shown in the following screenshot:



As this is an exact match to the search term that has been entered, and as the picklist contains the **autostepsafe** flag, QAS suggest that the Integration code automatically steps into the first result (an exact match to the search term), without displaying the picklist to the user. This produces the following:



As the previous picklist contains the **autoformatsafe** flag, QAS suggest that the Integration code automatically formats the first result to produce only the final address rather than the picklist.

This is shown in the following screenshot:

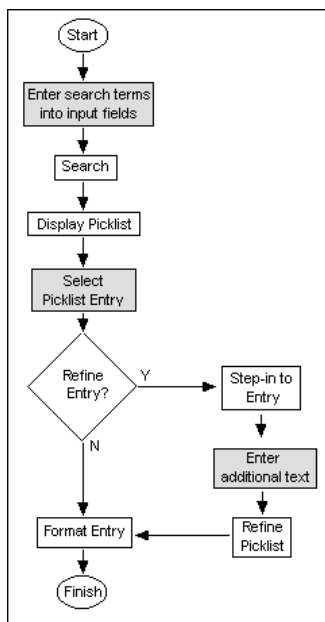
Please confirm your address below.

	<input type="text" value="Q A S Ltd, George West House"/>
	<input type="text" value="2-3 Clapham Common North Side"/>
Town	<input type="text" value="LONDON"/>
County	<input type="text"/>
Postcode	<input type="text" value="SW4 0QL"/>
Country	<input type="text" value="United Kingdom"/>

Flattened Mode

The flattened mode of the Single Line engine is available from the Web: Address Capture scenario. See the "Web: Address Capture" section of the Pro Web Integration Guide. The flattened mode is only available if it has previously been specified using the **Flatten** engine option (which is the default for the Web: Address Capture scenario).

The following diagram demonstrates the flow of searching in this scenario:



* Some special matches will require a refinement step; see page 99.

In this diagram, the shaded boxes represent user actions; all others are integration actions.

The flattened mode of the Single Line engine is designed to be used within an environment that has the following attributes:

- Users who are untrained and unfamiliar with the address capture concept;
- An unresponsive connection between the client and the integration; for example, a slow modem internet connection;

This scenario is therefore suitable for public Web site address capture;

The Verification engine is also suitable for this environment, but has a different style of searching. See page 102 for details.

Searches that are performed using the flattened mode of the Single Line engine create a picklist with the following properties:

- Except for certain special cases (see Special Picklist Items on page 99), picklist items cannot be refined on;
- Picklists will not contain any 'informational' picklist items other than 'No Matches' (see page 114);
- Picklists returned using the flattened mode may contain more items than if the search was performed with the hierarchical mode. This is especially true if the search is not restricted using one of the recommended prompt sets (see page 110).

The flattened mode of the Single Line engine is designed to specify the search terms in a restrictive fashion, in order to generate the smallest number of matches to select from. This is done by using one of the following prompt sets:

- Optimal
- Alternative
- Alternative 2 (USA only).

Example of Flattened Mode Searching

This example shows how flattened mode searching can be used in a situation where a customer on a Web site is encouraged to enter their address details. However, the customer does not enter a premises number, so the flattened mode generates a non-expandable picklist of possible addresses.

The example uses 329 Werona Kingston Road, Kooroocheang, VIC from the Australian data.

1. Select the Web: Address Capture scenario.
2. Select Australia from the **Country** drop-down box.
3. Click the **Next** button.
4. Click the **If you do not know the postal/ZIP code then click here** link.

5. Type the following in the relevant Search fields:

- **Werona Kingston Road** in the **Street** Search field.
- **Kooroocheang** in the **Locality** Search field.

Enter the address elements requested below.

Building number or PO Box	<input type="text"/>	(eg. 5/58)
Street	<input type="text" value="Werona Kingston Road"/>	(eg. Balfour)
Locality	<input type="text" value="Kooroocheang"/>	(eg. Kensington)

We will now find your address from the information you have entered above.

< Back

Next >

6. Click the **Next** button.

A list of possible addresses is displayed.

Select your address

Select one of the following addresses that matched your selection.

- ☐ [329 Werona Kingston Road, KOOROOCHANG VIC](#) 3364
- ☐ [340 Werona Kingston Road, KOOROOCHANG VIC](#) 3364
- ☐ [356 Werona Kingston Road, KOOROOCHANG VIC](#) 3364
- ☐ [328 Werona Kingston Road, SMEATON VIC](#) 3364
- ☐ [1309 Werona Kingston Road, SMEATON VIC](#) 3364

The next step is to confirm your address.

< Back

Next >

The possible addresses are shown in the format of a flattened picklist. This does not require further drilling down to the required address. Instead, the user can select the required address direct from this list (see Example of Hierarchical Mode Searching on page 86, for a method that requires further drilling down).

7. Select "329 Werona Kingston Road".

8. Click the **Next** button.

The selected address is returned.

Confirm your address

Please confirm your address below.

	329 Werona Kingston Rd
	KOOROOCHANG VIC 3364
Country	Australia

This completes the address capture process.

< Back	Accept
--------	--------

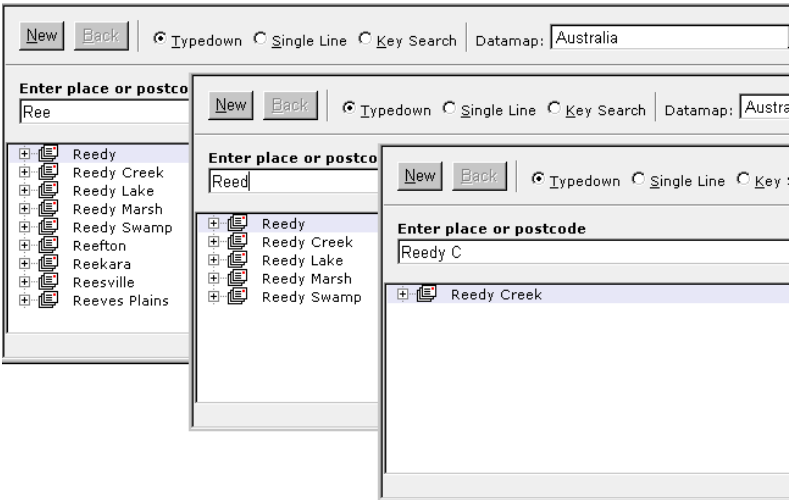
9. Click the **Accept** button to confirm the address.

Typedown Engine

The Typedown engine is designed to allow the user to drill down through a series of picklists to select the required address. This is the fastest method of searching for address data in Pro Web. Typedown is only available from the Intranet: Rapid Addressing - Standard scenario.

When a user enters text into the search field, the picklist is updated a short period after the user stops typing (this period is currently 300 milliseconds for the Intranet: Rapid Addressing - Standard scenario). This means that a picklist is generated as the user types. The more characters that the user enters, the more refinement is shown in the picklist. This 'dynamic refinement' enables searches to be initiated without the user having to type 'Enter' or click the **Search** button.

For example, using the Australia data, if the user could enter the characters "Re". The picklist is updated as the the user continues to type more characters, as shown.



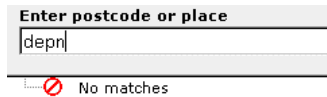
Prompt Sets

Prompt sets are used to guide the user as to what information should be entered at each stage. Ideally, the user will always enter information that generates the smallest number of matches. This makes the search more efficient, and makes it easier for the user to select the final address.

Typedown and Fuzzy Matching

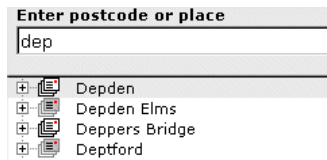
Typedown searching does not have fuzzy matching capability. Therefore, if you misspell an address element when searching for an address in Typedown, the engine will not be able to find it.

For example, with United Kingdom data, if the user types the letters "deprn" in the search box, the Typedown search engine cannot find any address elements beginning with that combination of letters. The following message is returned in the picklist area:



The screenshot shows a search box with the title "Enter postcode or place". Inside the box, the text "deprn" is entered. Below the box, there is a message "No matches" preceded by a red circle with a diagonal line through it, indicating a failed search.

However, deleting just one character of the text with the **Backspace** key returns a choice of places beginning with "dep", and the user can continue searching.

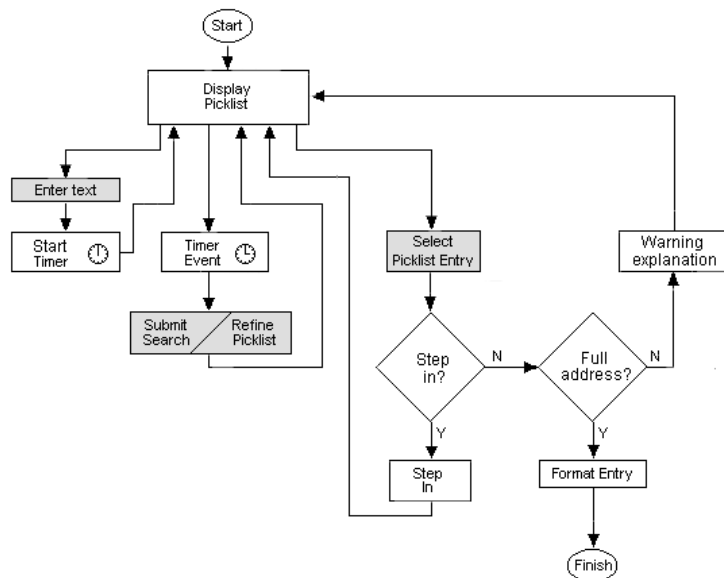


The screenshot shows the same search box with the title "Enter postcode or place". The text "dep" is entered. Below the box, a list of suggestions is displayed, each preceded by a small icon of a house with a red location pin. The suggestions are: "Depden", "Depden Elms", "Deppers Bridge", and "Deptford".

More information can be found in the "Intranet: Rapid Addressing - Standard" section of the Pro Web Integration Guide.

Hierarchical Mode

The Typedown method of address capture usually requires user interaction after the initial search has been submitted, in order to select a final address from the returned set of picklist items. The manner in which the engine will return the picklist is determined by a hierarchical mode of searching.



In this diagram, the shaded boxes represent user actions; all other boxes are integration actions.



The Typedown engine only searches using a hierarchical mode, unlike the Single Line engine that can use hierarchical or flattened modes of searching.

The diagram illustrates that address capture using the Typedown engine requires interaction with the user (picklist refinement and step-ins).

The user therefore has more control over the address capture process, but this scenario requires the following attributes:

- Users who are trained upon or familiar with the address capture process;
- A responsive connection between the client and the integration.

This scenario is therefore more suitable for internal applications such as an Intranet site rather than for public facing Internet applications.

Searches that are performed using the Typedown engine will create a picklist which may have one or more of the following properties:

- Hierarchical picklist items which can be 'stepped into' to produce new picklists;
- Picklist items that may contain 'informational' picklist items (see page 114);
- Picklists will update as the user enters more characters in the search field.

Typedown generally involves two, three or four stages. When searching for a United Kingdom address, users should first enter the postcode, county, town or locality. The Typedown engine usually returns a picklist choice after three or four characters, from which the user can select a place name.

Example of Hierarchical Searching

This example shows how hierarchical mode Typedown searching is used when an intranet user needs to enter address details. However, the customer does not enter a premises number, so the hierarchical mode generates a picklist of possible addresses.

The example uses 329 Werona Kingston Road, Kooroocheang, VIC from the Australian data.

1. Select the Intranet: Rapid Addressing - Standard scenario.
2. Click the **QuickAddress** button.
3. Ensure that the **Typedown** mode is selected.
4. Select Australia from the **Country** drop-down box.
5. Type **Kooroocheang** in the Search field.

The picklist is automatically updated as you type to show the list of possible address matches.

New Back | Typedown Single Line | Database: Australia

Enter place or postcode

Kooroocheang Select

+ Kooroocheang

The possible addresses are shown in the format of an hierarchical picklist. This requires further selection to drill down to the required address (see Example of Flattened Mode Searching on page 91, for a method that does not require further drilling down).

- Click "Kooroocheang" to expand the picklist entry.

Enter street name or PO Box type

Select

Kooroocheang

+ Continue typing (or select to show all matches)

Pro Web prompts you to enter a street name.

- Type **Werona Kingston Road** in the Search field.

The picklist is updated.

Enter street name or PO Box type

Select

Kooroocheang

+ Werona Kingston Road, KOOROOCHANG

+ Werona Kingston Road, SMEATON

- Click on the "Werona Kingston Road, Kooroocheang" entry to expand the picklist further.

Enter building number/name

Select

Kooroocheang

Werona Kingston Road, KOOROOCHANG

329

340

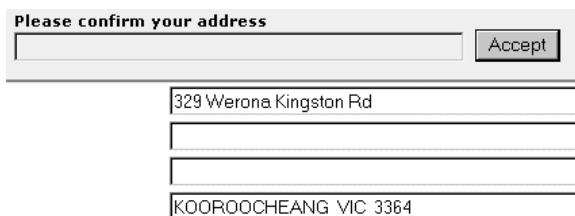
356

A list of available addresses is shown.

- Select "329".

The selected address is returned to the address edit screen.

10. Click **Accept** to confirm the address.



The screenshot shows a dialog box titled "Please confirm your address". It contains a text input field with the address "329 Werona Kingston Rd" and an "Accept" button. Below the input field, there are three more empty text input fields, and the address "KOOROOCHANG VIC 3364" is displayed at the bottom.

11. Click the **Accept** button to confirm the address.

Auto Step-In

This flag is designed to make searches more efficient. For information about this, see page 88.

Special Picklist Items

There are three types of picklist item that must be treated as special cases in Typedown and Single Line searches. These are:

- unresolvable ranges.
- Phantom Primary Points.
- incomplete addresses.

See Dealing with Special Picklist Items on page 101, for information about dealing with these types of picklist item.

Unresolvable ranges

An unresolvable range is a picklist item that represents a range of premises, but where there is not enough information within the data to resolve the entry into a list of premises.

These are very common when searching against the USA data, although they also exist within other data and must be handled appropriately.

For example, search using Single Line against the USA data using the optimal prompt set with the address:

Street Address: Arch St

Zip Code: 02110-14ND

This returns a page containing a text box that prompts the user to enter a premises within the following range:

2...78 Arch St, Boston MA [even]

This is an unresolvable range, meaning that there is no available data to determine which possible even values between 2 and 78 are valid, and which are invalid. The user therefore has to specify the premises number that will resolve this picklist item, so that a single address can be generated from the range.

Phantom Primary Points

A Phantom Primary Point (PPP) is specific to AUS data. A phantom primary point is a premises which is non-deliverable unless the user enters further secondary information. This secondary information may or may not be in the actual data. The user must enter this sub-premises information in order to complete a final address match.

For example, search using Single Line against the AUS data using the optimal prompt set with the address:

Building number or PO Box: 44

Street: Miller St

Postcode: 2060

This returns a picklist where the first entry is:

44 Miller Street, NORTH SYDNEY NSW

This is marked as a PPP, which means that if this picklist item is selected, the integration must prompt the user for additional sub-premises information.

Incomplete Addresses

An incomplete address is an address that is not deliverable due to missing premises information within the data. This therefore requires the user to provide additional premises information so that the address is deliverable. Incomplete addresses are found in various datasets, such as the DEU data.

For example, search using Single Line against the DEU data using the optimal prompt set with the address:

Street: Feldburg

Building number or name:

Postcode: 50181

This returns a picklist with the following single entry:

Feldburg, BEDBURG 50181

This is marked as an incomplete address, which means that if this picklist item is selected, the integration must prompt the user for additional sub-premises information.

Dealing with Special Picklist Items

The following steps should be taken when a picklist item is flagged as an unresolvable range, Phantom Primary Point or incomplete address:

1. The entry should be stepped into, in the same manner as with hierarchical picklists (see page 112).
2. The premises information submitted by the user should then be used to refine the resulting picklist.
3. The picklist should contain a picklist item at the first position that does not have the original flag, but instead a **FullAddress** flag.
4. This flag can be used to format the address.

See the Pro Web Technical Reference Guide - Common Classes and the Pro Web Technical Reference Guide - WSDL for a description of these flags.

Verification Engine

The Verification engine is designed so that only minimal interaction, or none at all, is required from the user. The amount of user interaction required is the choice of the integrator. The user enters their whole address in the same way that it would be written on an envelope, and the entire address is submitted to the engine.

The engine returns a verification level which corresponds to the degree of confidence in the returned (and reformatted) match. For addresses that could not be matched with confidence, it is up to the integrator whether to return a prompt for more user interaction, or whether to return the address as it was entered.

The Verification engine with full user interaction available is demonstrated within the "Web: Address Verification" scenario (see the "Web: Address Verification" section of the Pro Web Integration Guide).

Unlike the other engines, the verification scenario does not require prompt sets to be used to constrain the manner in which the search term is submitted; instead, it expects a complete address to be submitted. The prompt set that should be used for searching with the Verification engine is the **Default** prompt set which does not apply any restrictions.

Customised restrictions upon specific lines can be implemented using the configuration setting `InputLineCount` (see the "Configuration Settings" section of the Pro Web Installation And Administration Guide). These will apply to the Verification engine only. This is not a requirement for searching, although the integrator may want to place restrictions on specific input lines.

Using The Verification Engine

There are three ways that the Verification engine can be used within an address capture environment:

- No user interaction after submitting the search;
- Minimal user interaction after submitting the search;
- Full user interaction after submitting the search.

The Verification engine will return one of the following six **VerifyLevels** for a search, each of which may be treated differently by the integrator depending upon the amount of user interaction that is required:

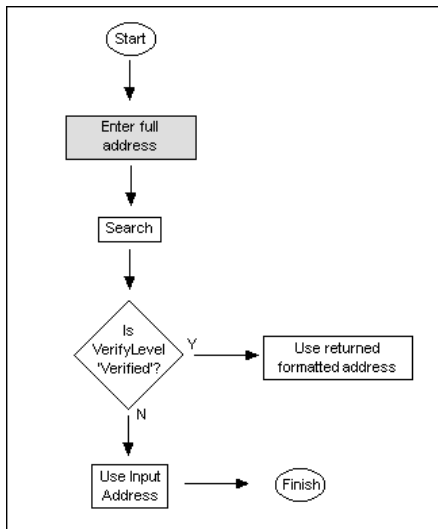
- None;
- Verified;
- InteractionRequired;
- PremisesPartial;
- StreetPartial;
- Multiple.

See the "Web: Address Verification" section of the Pro Web Integration Guide for more information about each level.

No User Interaction

The Verification engine may be used with no user interaction required after an initial search has been submitted. This allows an integrator to hide the fact that there is any address management occurring within an application from the user.

This method of using the Verification engine means that addresses that are entered will be simply verified as being correct or not verified as being correct

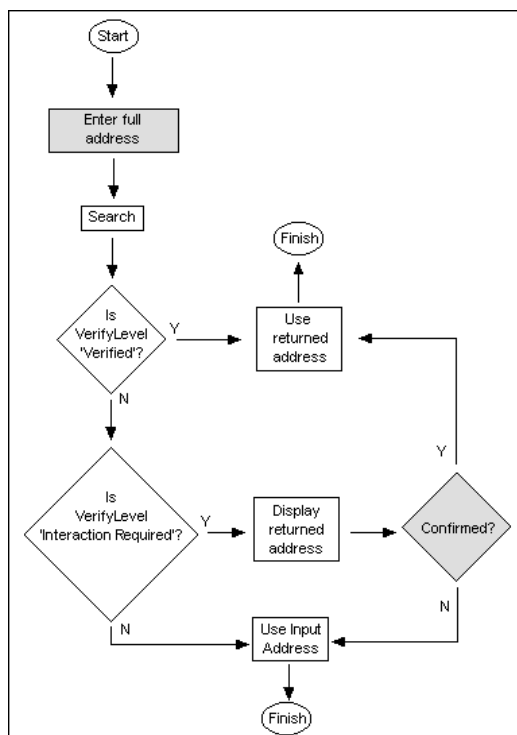


If a search results in a **VerifyLevel** of **Verified**, then the final formatted address that is returned from the Search call can be used as the captured address.

If a search results in any other type of **VerifyLevel**, then the original address as entered by the user must be used as the captured address.

Minimal User Interaction

The Verification engine may be used with minimal user interaction after an initial search has been submitted. This does not require the display or use of picklists (as with the Single Line and Typedown engines), but may require a single confirmation by the user if the Verification engine is not highly confident in the match.



This method of using the Verification engine means that addresses that are entered will either be:

- Verified as being correct;
- Verified as being correct, after user confirmation;
- Not verified as being correct.

If a search results in a **VerifyLevel** of **Verified**, then the final formatted address as returned from the Search call can be used as the captured address.

If a search results in a **VerifyLevel** of **InteractionRequired**, then the final formatted address as returned from the Search call can be displayed to the user for confirmation. If confirmation is given, then this can be used as the captured address. If confirmation is not given, then the original address as entered by the user can be used as the captured address.

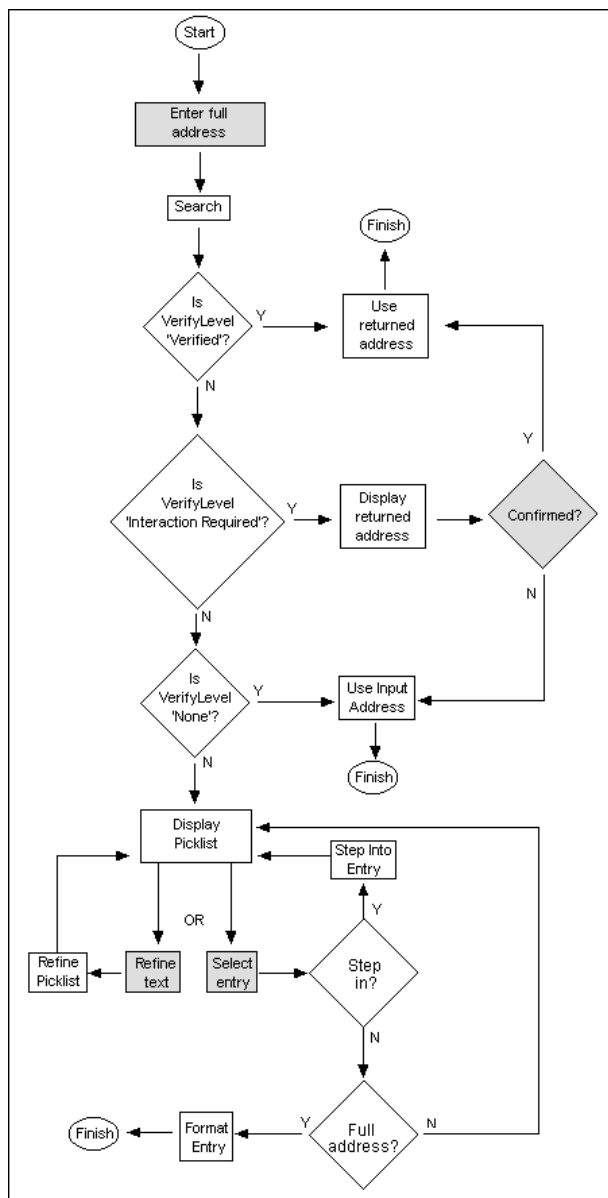
If a search results in any other type of **VerifyLevel**, then the original address as entered by the user must be used as the captured address.

Full User Interaction

The Verification engine may be used with full user interaction after an initial search has been submitted. This will require the display and use of picklists for searches that could not locate a single deliverable match.

This method of using the Verification engine means that addresses that are entered will either be:

- Verified as being correct;
- Verified as being correct, after user confirmation;
- Matched to one or more close addresses, which the user can interactively traverse;
- Not verified.



If a search results in a **VerifyLevel** of **Verified**, then the final formatted address as returned from the **Search** call can be used as the captured address.

If a search results in a **VerifyLevel** of **InteractionRequired**, then the final formatted address as returned from the Search call can be displayed to the user for confirmation. If confirmation is given, then this can be used as the captured address. If confirmation is not given, then the original address as entered by the user can be used as the captured address.

If a search results in a **VerifyLevel** of either **StreetPartial**, **PremisesPartial** or **Multiple**, then the picklist returned can be displayed to the user for use. Picklists are handled in the same manner as with the Single Line engine, and can similarly function in either hierarchical or flattened mode. For more information about using picklists, refer to Hierarchical Picklists on page 112.

If a search results in a **VerifyLevel** of **None**, then the original address as entered by the user must be used as the captured address.

Keyfinder Engine

The Keyfinder engine is designed to enable the user to enter a key search term to produce the correct address. The Keyfinder engine is only available for use with certain data mappings which include datasets that contain a logical reverse search key.

The Keyfinder engine works in a similar way to the Single Line Engine (see page 84), in that the user enters the key search term, and the engine returns the address that matches the search term. If the key search term matches multiple addresses, a drop-down list of addresses is produced, from the user can select the required one. When the user has entered the key search term, they should start the search manually.

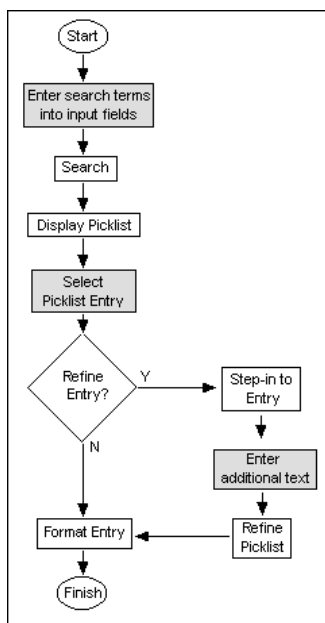
The Keyfinder engine is demonstrated within the "Web: Key Search" scenario. See the "Web: Key Search" section of the Pro Web Integration Guide for details.

The Key Search scenario normally requires minimal user interaction, as in most cases the key search term only matches one address. However, in some cases it is possible that the key search term matches more than one address, in which case the engine will return a flattened picklist.

Flattened Mode

The Keyfinder engine picklists are always returned in a flattened mode.

The following diagram demonstrates the flow of searching in this mode:



* Some special matches will require a refinement step; see page 99.

In this diagram, the shaded boxes represent user actions; all others are integration actions.

Searches that are performed using the Keyfinder engine create a flattened picklist with the following properties:

- Except for certain special cases (see Special Picklist Items on page 99), picklist items cannot be refined on;
- Picklists will not contain any 'informational' picklist items other than 'No Matches' (see page 114).

Prompt Sets

Prompt sets are designed primarily to constrain search terms for the Single Line and Typedown engines, in order to aid users to capture addresses quickly and easily. With this engine combination, the integration should query the chosen prompt set detail for the number of input lines and their prompt text, and display this to the user in the corresponding number of input fields.

For example, the **Optimal** prompt set for the **GBR** dataset has two input lines defined:

Building number or name
Postcode

The initial search input form should therefore look like this (for Web: Address Capture):

Enter your address

Enter the address elements requested below.

Building number or name	<input type="text"/>	(eg. 12)
Postcode	<input type="text"/>	(eg. BS40 5SJ)

However, every initial search that is submitted to one of the engines must have one of the prompt sets defined. The available prompt sets are as follows:

Default

This prompt set is designed to be used with the Verification engine.

This is an unconstrained prompt set that can accept one or many text field inputs with any address element in any field. The only proviso is that the elements should be in the order that a user would expect to enter them; for example, using UK data, the premises number must appear before the postcode.

All text fields are treated the same way in search methods: if you call a method to obtain information about the Default prompt set, unless otherwise specified, it will return a value of zero for the number of prompt set lines, together with the empty prompt set. No text prompts or examples are provided.

When using the Verification engine, the configuration setting `InputLineCount` (see the "Configuration Settings" section of the Pro Web Installation And Administration Guide) can optionally be used to constrain the Default prompt set.

Optimal

This prompt set is designed to be used with the Single Line engine in Flattened mode.

This prompt set defines the minimum number of fields that users must complete, in order to return the required address.

This is specific to the active dataset. For example, when using USA data, the user will only need to fill in two prompt set lines to return an address; "street address" and "ZIP Code". When using UK data, the required prompt set lines are "building number or name" and "postcode".

Alternative

This prompt set is designed to be used with the Single Line engine in Flattened mode.

This is an extended country-specific prompt set. It is designed for cases where the user does not have the information required to fill in all fields requested in **Optimal** (such as when they cannot remember their postcode).

Generic

This prompt set is designed to be used with the Single Line engine in Flattened mode.

This is a standard prompt set that can be used across all countries. It has four fields: "building number or name", "street", "town/ city", and "postcode". This is useful in situations where the user must complete the address fields at the same time as or before specifying the country.

OneLine

This prompt set is designed to be used with the Single Line engine in hierarchical mode (i.e., the Flatten engine option set to False) or the Typedown engine.

This prompt set specifies a single unconstrained input line that will accept any address elements.

Hierarchical Picklists

Hierarchical picklists are those that have picklist items that represent multiple addresses, and can be 'stepped into' to generate a new picklist with address elements at a greater level of detail.

For example, using the GBR data and the OneLine prompt set, with the Single Line engine in hierarchical mode, enter the following text:

sw1a 2a?


The following picklist is returned:

Searching on... 'sw1a 2a?'

	Downing Street, LONDON	100%
	King Charles Street, LONDON	100%
	Whitehall, LONDON	100%
	Old Admiralty Building, Whitehall, LONDON	SW1A 2AF 100%

The two picklist items each represent multiple addresses, and each can be stepped into. If you call the method to **Stepln** to the first picklist item, a new picklist is generated:

Searching on... 'sw1a 2a?'

Downing Street, LONDON		100%
	Foreign & Commonwealth Office	SW1A 2AL
	Government Chief Whips Office, 9	SW1A 2AG
	Prime Minister & First Lord of the Treasury, 10	SW1A 2AA
	Chancellor of the Exchequer, 11	SW1A 2AB
	Government Chief Whips Office, 12	SW1A 2AD

Hierarchical picklists are designed to allow easy use of picklists, and to reduce the number of picklist items by combining multiple similar entries into a single entry that can be stepped into. However, this does mean that this mode requires more user interaction than the flattened equivalent, and so is more suited to intranet and application usage rather than to address capture upon a public Web site.

Picklist Refinement



Picklist refinement is the process of using some text to filter the current picklist to only contain entries that match. This must be used when searching with hierarchical picklists, as they will often contain too many picklist items to display, and refinement is necessary before an entry can be selected.

For example, using the GBR data and the OneLine prompt set, with the Single Line engine in hierarchical mode, enter the following text:

Baker St, London

The following picklist is returned:



Searching on... 'Baker St, London'

  [Baker Street, LONDON](#) 100%

If you step into the single picklist item, a new picklist is generated:

Searching on... 'Baker St, London'

Baker Street, LONDON 100%

  [Continue typing \(or select to show all matches\)](#)

This informational prompt signifies that there are too many picklist items (all of the premises in the street) to display in the picklist. The user therefore needs to refine the picklist to generate fewer entries so that they can select one.

To search for "Flat 1", refine using the text "1" and click **Search**, which generates the following picklist:

Searching on... 'Baker St, London'		
Baker Street, LONDON		100%
	Gala Barracuda Casino, 1	W1U 8ED
	Manors & Co, 1a	W1U 8ED
	Peter Black Footwear & Accessories Ltd, 1b	W1U 8ED
	1e	W1U 8ED
	Habib Bank A G Zurich, 2	W1U 3DS
	3	W1U 8EE
	Mizoni, 3b	W1U 8EE
	Henry Ltd, 3c-3d	W1U 8ED

You can then easily select the required picklist item.

Informational Picklist Items

Informational picklist items are commonly displayed when searching using hierarchical picklists. If you are using flattened picklists, then informational entries will never be returned.

An informational picklist item is one that does not represent an address, but rather is displayed in the picklist to assist users. For example:

- No matches
- Continue typing for more (or select to show all matches)
- Continue typing (too many matches)
- Address not recognised (click to accept)
- <PO Box>


Informational picklist items are essential to guide address capture with hierarchical picklists. The integrator must treat informational picklist items in exactly the same manner as normal address entries, except that they may be displayed differently, such as with a different colour picklist text.

To clarify the current state of a search, Pro Web provides informational prompts in the picklist area. For example, at the beginning of a search the picklist area will contain an informational prompt similar to:

 Continue typing (too many matches)

If you type the letters **cre** into the search box, the informational prompt will change to:

 Continue typing (or select to show all matches)

You can continue typing to narrow down the search. Alternatively, you can click the **Select** button, press **Enter**, double-click on the informational prompt or click on the  sign to show all matches.

Depending on the status of a search, Pro Web may display other intuitive informational prompts.

Informational picklist items could be Formatted (as signified by the **FullAddress** flag) or Stepped Into (as signified by the **CanStep** flag).

Appendix A: Differences Between Common Classes

Overview

The Common classes are provided as an interface for those integrators who want to use Pro Web functionality via a layer that is as consistent as possible across all the languages for which QAS have provided integration code.

There are minor differences between the common classes, depending on the language used. These differences are described in the following sections.

QuickAddress Object Search Methods		
JSP	.NET	PHP
canSearch	CanSearch	canSearch
search	Search	search
refine	Refine	refine
stepIn	StepIn	stepIn
getFormattedAddress	GetFormattedAddress	getFormattedAddress

QuickAddress Object Status Methods		
JSP	.NET	PHP
getAllDataSets	GetAllDataSets	getAllDataSets
getDataMapDetail	GetDataMapDetail	getDataMapDetail
getAllLayouts	GetAllLayouts	getAllLayouts
getExampleAddresses	GetExampleAddresses	getExampleAddresses
getLicenceInfo	GetLicenceInfo	getLicenceInfo
getSystemInfo	GetSystemInfo	getSystemInfo

QuickAddress Object Settings Methods		
JSP	.NET	PHP
setConfigFile	ConfigFile	setConfigFile
setConfigSection	ConfigSection	setConfigSection
setEngineIntensity	SearchingIntensity	setEngineIntensity
setEngineType	Engine	setEngineType
setFlatten	Flatten	setFlatten
setThreshold	Threshold	setThreshold
setTimeout	Timeout	setTimeout

AddressLine Object		
JSP	.NET	PHP
getLabel	Label (property)	Label (property)
getLine	Line (property)	Line (property)
getLineType	LineType (property)	N/A
isOverFlow	IsOverFlow (property)	N/A
isTruncated	IsTruncated (property)	N/A

BulkSearchItem Object		
JSP	.NET	PHP
getInputAddress	InputAddress (property)	sInputAddress (property)
getAddress	Address (property)	address (property)
getVerifyLevel	VerifyLevel (property)	sVerifyLevel (property)

BulkSearchResult Object		
JSP	.NET	PHP
getBulkSearchItems	BulkSearchItems (property)	bulkSearchItems (property)
getBulkSearchItem	BulkSearchItem (property)	N/A
N/A	Count (property)	sCount (property)
getErrorMessage	ErrorMessage	N/A
getErrorCode	ErrorCode	errorCode (property)

CanSearch Object		
JSP	.NET	PHP
isOk	IsOk	IsOk (property)
getMessage	ErrorMessage	ErrorMessage (property)
	Not supported	ErrorCode (property)

Dataset Object		
JSP	.NET	PHP
getName	Name (property)	Name (property)
getID	ID (property)	ID (property)
findByID	FindByID (property)	N/A

ExampleAddress Object		
JSP	.NET	PHP
getComment	Comment (property)	asComment (property array)
getAddress	AddressLine (array property)	atAddress (property array)

FormattedAddress Object		
JSP	.NET	PHP
getAddressLines	AddressLine (array property)	atAddressLines (property array)
N/A	Length (property)	N/A
isOverFlow	IsOverflow (property)	bOverflow (property)
isTruncated	IsTruncated (property)	bTruncated (property)

Layout Object		
JSP	.NET	PHP
getName	Name (property)	Layouts are held and returned in an array
getComment	Comment (property)	
findByName	FindByName (property)	

Licensedset Object		
JSP	.NET	PHP
getID	ID (property)	ID (property)
getCopyright	Copyright (property)	Copyright (property)
getVersion	Version (property)	Version (property)
getBaseCountry	BaseCountry (property)	BaseCountry (property)
getStatus	Status (property)	Status (property)
getDescription	Description (property)	Description (property)
getServer	Server (property)	Server (property)
getWarningLevel	WarningLevel	WarningLevel (property)
getDaysLeft	DaysLeft	DaysLeft (property)
getDataDaysLeft	DataDaysLeft	DataDaysLeft (property)
getLicenseDays Left	LicenseDays Left	LicenseDaysLeft (property)

Picklist Object		
JSP	.NET	PHP
getTotal	Total (property)	iTotal (property)
getMoniker	Moniker (property)	sPicklistMoniker (property)
getPrompt	Prompt (property)	sPrompt (property)
getItems	Items (property)	atItems (property array)
N/A	Length (property)	N/A
isAutoStepinSafe	IsAutoStepinSafe (property)	bAutoStepinSafe (property)
isAutoStepinPastClose	IsAutoStepinPastClose (property)	bAutoStepinPastClose (property)
isAutoStepinSingle	IsAutoStepinSingle (property)	isAutoStepinSingle
isAutoFormatSafe	IsAutoFormatSafe (property)	bAutoFormatSafe (property)
isAutoFormatPastClose	IsAutoFormatPastClose (property)	bAutoFormatPastClose (property)
isAutoFormat Single	IsAutoFormatSingle (property)	isAutoFormatSingle
isLargePotential	IsLargePotential (property)	bLargePotential (property)
isMaxMatches	IsMaxMatches (property)	isMaxMatches (property)
isMoreOther Matches	IsMoreOtherMatches (property)	bMoreOtherMatches (property)
isOverThreshold	IsOverThreshold (property)	bOverThreshold (property)
isTimeOut	IsTimeOut (property)	isTimeout (property)

PicklistItem Object		
JSP	.NET	PHP
getText	Text (property)	N/A
getPostcode	Postcode (property)	Postcode (property)
getScore	Score (property)	Score (property)
N/A	ScoreAsString (property)	N/A
getMoniker	Moniker (property)	Moniker (property)
getPartialAddress	PartialAddress (property)	PartialAddress (property)
isFullAddress	IsFullAddress (property)	FullAddress (property)
canStep	CanStep (property)	CanStep (property)
isAliasMatch	IsAliasMatch (property)	AliasMatch (property)
isPostcodeRecoded	IsPostcodeRecoded (property)	PostcodeRecoded (property)
isIncompleteAddress	IsIncompleteAddress (property)	N/A
isUnresolvableRange	IsUnresolvableRange (property)	UnresolveableRange (property)
isCrossBorderMatch	IsCrossBorderMatch (property)	CrossBorderMatch (property)
isDummyPOBox	IsDummyPOBox (property)	DummyPOBox (property)
isName	IsName (property)	Name (property)
isInformation	IsInformation (property)	Information (property)
isWarnInformation	IsWarnInformation (property)	WarnInformation (property)
isPhantomPrimaryPoint	IsPhantomPrimaryPoint (property)	PhantomPrimaryPoint (property)
isMultiples	IsMultipleAddresses (property)	Multiples (property)

PromptLine Object		
JSP	.NET	PHP
getPrompt	Prompt (property)	Prompt (property)
getExample	Example (property)	Example (property)
getSuggestInputLength	SuggestInputLength (property)	SuggestInputLength (property)

PromptSet Object		
JSP	.NET	PHP
PromptLines (property array)	PromptLines (property array)	atLines (property array)
PromptSet	PromptSet	PromptSet
getLinePrompts	GetLinePrompts	N/A

SearchResult Object		
JSP	.NET	PHP
getAddress	Address (property)	address
getPicklist	Picklist (property)	picklist
getVerifyLevel	VerifyLevel (property)	sVerifyLevel

Glossary Of Terms

Additional Dataset

Additional datasets are available with some *Datasets* to enhance the data. They cannot be used without the *Base Dataset* they are associated with, and only some datasets support additional datasets.

For example, the Names additional dataset is available for USA and GBR data only.

Address Elements

The fields that comprise an address. Each *Dataset* consists of a set of address elements specific to that country.

For example, for United Kingdom data, these fields include building number, thoroughfare, town, and postcode.

Address Layout

The format of an address, arranged with the *Address Elements* in an order specific to the convention of the country.

Layouts can be changed from within the *Configuration Editor* to reflect the needs of the user.

Administrator Console

A tool shipped with the Windows version of the Pro Web server. It enables administrators to configure the QAS server, perform live data updates, list and disconnect clients, manage *Counters* and monitor network traffic.

Alias Matching

Aliases are unofficial alternative *Address Elements* used to match information entered by the user.

For example, if the user entered a locality that was recorded as out of date or not required, then the address would be replaced by the official, postally-correct version from the relevant dataset. See also *Alternative Names*.

Alternative Names

A street, town/city or a suburb may have a different name to the official postal address. This alternative will be returned if it has been supplied in the search and if the appropriate submitted element has been fixed in the address layout.

API

Application programming interface.

Primarily used in reference to a development version of QAS products, which can be integrated directly within third party applications.

Base Dataset

The base dataset is the main address *Dataset* in a *Data Mapping*, against which a combination of associated *Additional Datasets* can be specified.

Bordering Localities

(Australian address data only)

When users search for a street, they may not know the correct postal locality in which the street is situated. In these cases, the search engine also looks for the specified street in all of the localities that border the input locality and the input postcodes.

Bordering Localities are very similar to *Suburb Adjacencies* (New Zealand address data only).

Bulk Searching

The bulk search process performs address verification on one or more addresses. Unlike the normal *Verification* search process, the bulk search process consists of a single step: each address is either verified and returns a formatted address, or is not verified.

Clicks

The unit of measurement for metered *Licences*.

Pro Web supports metered licences which decrement when a search results in a match. More than one click may be decremented for searches that return a high number of matches, depending on the *Dataset* you are using. Metered licences are only compatible with Internet integrations.

Command Line Management Utility

An administration tool that is shipped with Windows and UNIX versions of QAS products. It enables administrators to perform various tasks on clients and on the server.

Configuration Editor

A Windows-only interface to which provides access to some of the functionality available in the *Configuration Files*. You can use the Configuration Editor to create, edit and manage *Address Layouts*, to specify details of Pro's user interface, and to manage *Datasets* and *Data Mappings*.

Configuration Files

Both Windows and UNIX users can use the configuration files (*qawserve.ini* and *qaworld.ini*) to configure the QAS server, to specify which *Datasets* are installed, to configure search options and results settings and to set output address formats.

Counters

A display, visible from the *Administrator Console*, that shows the number of *Clicks* remaining on each of your metered *Licences*.

Datamap

See *Data Mapping*.

DataPlus

Extra information that can be returned with each matched address.

You can determine which information you want to be returned by configuring DataPlus through the Configuration Editor.

The available DataPlus information depends on the *Dataset* being used. For example, for United States data such items could include Country Code or a Postnet barcode.

Dataset

A collection of proprietary data files, containing address, business and/or names information, that are shipped to customers on a data CD-ROM.

For example, the AUS CD-ROM contains the Australian address dataset, while the GBR Names CD-ROM contains the United Kingdom address dataset, together with GBR Names data.

Optional extra data is available for some datasets, in the form of *DataPlus* sets or *Additional Datasets*.

Data Expiry

If you have data installed which is nearing expiry, the Dataset Expiry Notification screen will be displayed when you first open the *Configuration Editor*. This screen will warn you if product or data *Licences* have expired or are due to expire. You can also use the *Configuration Files* to configure the server to return expiry notification emails to a specified email address.

Data Guide

A reference guide that is supplied with each *Dataset* that you purchase. It provides dataset-specific information and search tips for each dataset.

Data Identifier

A unique three character alpha-numeric code that defines a *Dataset* or a *Data Mapping*.

Data Mapping

A combination of a *Base Dataset* and any number of *Additional Datasets* which are relevant to that dataset. Each data mapping has a unique *Data Identifier* and name.

Data mappings can be defined using the *Configuration Editor*, or with the `DataMappings` configuration setting in the `qawserve.ini` file.

For example, a data mapping that consists of a base dataset and no additional datasets could be AUS. A data mapping that consists of a base dataset and multiple additional datasets could be GBR With Gas And Electricity.

Deliverable Address

The address as recognised by the postal services for a specific country.

Dynamic Refinement

The process whereby you type characters into the search field and the display of *Picklist* results updates as you type.

This is applicable to *Typedown* searching, and to *Single Line* searching using Pro Web (via the *Standard Scenario*).

Engine

The underlying functionality behind the search and verification operations in QuickAddress products.

There are four search engines available in Pro Web:

- *Single Line* engine
- *Typedown* engine
- *Verification* engine
- *Keyfinder* engine

For information about how these engines work, see Engine Behaviour on page 83.

Flattened Mode

A search mode designed to produce simple *Picklists* from which users can easily select the correct address.

This mode also requires search terms to be entered in a restrictive fashion, so it is useful for public Web site applications, where less user interaction is needed than the *Hierarchical Mode*.

Fuzzy Matching

A non-exact match, based upon related words to those in the query.

For example, a search for "Partley St" in the Australia *Dataset* may return "Bartley St", "Hartley St" and several other possibilities.

Hierarchical Mode

A search mode designed to facilitate the use of *Picklists* by combining multiple similar items into a single item that can be *Stepped Into*.

Users can enter any terms that they feel are appropriate, so this mode is designed for more experienced users than the *Flattened Mode*. This mode is useful for intranet and application usage.

Ini File

A common name for the *Configuration Files* qawserve.ini and qaworld.ini.

Integration Pages

Sample pages of code which are provided with Pro Web to enable integrators to integrate the code into their own applications.

For Pro Web, the following integration pages are available:

- Java/JSP
- C#.NET
- PHP

ISO Code

See *Data Identifier*.

Keyfinder Engine

The Keyfinder engine is designed to enable the user to enter a *Key Search* term to produce the correct address. The Keyfinder engine is only available for use with certain *Datasets* that contain a logical reverse search key.

Key Search

Key Searching allows the user to search on key elements of the data.

Key searching can only be used with certain *Datasets* that contain a logical reverse search key; for example, United Kingdom With Gas data contains a Meter Number (MPRN) that can be searched on.

Licences

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

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

Metered licences are also available for Pro Web, which offer an alternative pricing scheme for certain kinds of address searching.

Locality

See *Bordering Locality*.

Overdraft

If a meter value for a metered *Licence* is less than or equal to zero, it is said to be overdrawn.

Pro Web has an overdraft facility of 50 *Clicks* that should only be used as an emergency measure for searches in progress.

Partial Matching

An address that was searched upon which could not be matched to a complete deliverable result and so was matched to a partially-complete address.

Picklist

A list of the matches returned by a search, together with the percentage confidence for each match, from which a user may either select an item or *Step Into a Range*.

PNRL

Postally Non-Required Locality.

A PNRL is a locality address element that is not required for the address to be deliverable.

Returned addresses do not include PNRLs unless they are added during the search.

For example, a Single Line search on: "2-3 Clapham Common Northside, London" will return the following address:

QAS Ltd
George West House
2-3 Clapham Common North Side
London
SW4 0QL

A Typedown search on the elements "Clapham > Clapham Common North Side > 2" will return the following address, including the PNRL "Clapham":

QAS Ltd
2-3 Clapham Common North Side
Clapham
London
SW4 0QL

PPP

Phantom Primary Point

These are specific to Australian addresses. A phantom primary point is a premises which is a non-deliverable address, unless the user enters further secondary information. This secondary information may or may not be in the actual data. The user must enter this sub-premises information to complete a final address match.

Prompt Set

Prompt sets are designed to manage the way that users enter addresses in Pro Web. An address can be submitted as one or as many text fields, and prompt sets aid the search engine by detailing which *Address Elements* can be entered in each field.

Range

A single *Picklist* item that encompasses a number of premises or sub-premises. Odd and even premises are split into separate ranges.

A range can be refined to a single premises or sub-premises.

Rapid Addressing

There are three Rapid Addressing *Scenarios* that ship with Pro Web, and which are designed to provide *Typedown*, *Single Line* and *Bulk Searching*.

Furthermore, the Rapid Addressing Scenarios that enable Typedown searching allow the *Dynamic Refinement* of *Picklists* in an intranet environment.

Scenario

Six sample scenarios are included with this release of Pro Web. These represent the most common uses of the product, although they do not have to be followed exactly.

Integrators can add or remove functionality from each one as appropriate, or merge functions from different scenarios.

The six scenarios are:

- Web: Address Capture
- Web: Address Verification

- Web: Key Search
- Intranet: Rapid Addressing - Single Line
- Intranet: Rapid Addressing - Standard
- Intranet: Rapid Addressing - Bulk Processing

Single Line

Single Line searching requires a user to enter two or three *Address Elements*, each separated by a comma, in the order that they would appear on an envelope. For example, the street name followed by the town.

Single Line searches can use a variety of techniques to return the correct address from incomplete or misspelled information.

For example, if Pro Web was used to verify addresses from coupons where the handwriting might be illegible, it would be better to use Single Line searching than *Typedown* searching.

SPM

Search Point Moniker

The process of searching for addresses in a *Stateless* environment (such as Pro Web and Mainframe) is based around SPMs. A moniker is a name that is used to uniquely identify an object. Pro Web uses text string monikers to represent a position in a search.

An SPM is used to store the state of a search that is in progress; each *Picklist* item therefore has an SPM as well as visible text. When a user steps into a picklist item, the corresponding SPM is sent from the client to the server in order to continue with the search and generate a resulting picklist or formatted address.

Stateless

The ability to perform a search and display *Picklists* without needing to maintain and track the progress of the search within the search engine (i.e., the server). In theory, a stateless server can deal with an unlimited number of clients.

Step Into

The action of expanding a *Picklist* item to view all the address details contained within it. For example, you can 'step into' a street to pinpoint the house numbers contained within it, or you can step into a range of addresses to find a specific premises.

Suburb Adjacencies

(New Zealand address data only)

These are very similar to *Bordering Localities* (Australian address data only).

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

Test Harness

The test harness is a simple application, based on the C API methods, which can be used to verify that you have installed the product correctly, and to demonstrate a subset of the product functionality.

Typedown

Typedown searching requires a user to enter one of the most general *Address Element* first, and once that has been found, to enter more specific parts of the address.

This mode of searching does not require a user to activate the search, but starts searching as soon as the first character is typed. Therefore, the more characters that are entered, the smaller the *Picklist* that will be returned.

For example, if an operator is taking address details over the phone, they can enter the caller's postcode and then, if required, they can search for the correct street and building number.

When a user is certain of the address information, Typedown is the more useful search option than *Single Line* searching.

See also *Dynamic Refinement*.

Unresolvable Range

An unresolvable range is a *Picklist* item that represents a *Range* of premises, but where there is not enough information within the data to resolve the item into a list of *Deliverable Addresses*.

These are very common when searching within USA address data although they also exist in other address *Datasets*.

Verification

The *Verification Scenario* in the Pro Web product is designed to verify an address once it has been entered in full into a web form.

The *Verification Engine* returns a verification level which corresponds to the degree of confidence in the returned (and potentially reformatted) match. If an accurate match could not be found, the integrator can choose whether to return the address as it was entered by the user, or to return a prompt demanding more user interaction.

Wildcard

If you are carrying out a *Single Line* search, you can use wildcards to replace one or more missing letters in your address information. There are two wildcards available:

- Question mark wildcard (?)

This replaces a single character in an address or postcode.

- Asterisk wildcard (*)

This replaces any number of characters at the end of an address element.

You can use a combination of wildcards in a single search line. Refer to the *Data Guide* that ships with your data for searching tips which include wildcards.

Index

Symbols

.NET

interface 1

A

Address constant 36

Address Line class 36

GetLabel method 39

GetLine method 39

GetLineType method 39

IsOverflow method 39

IsTruncated method 40

Administrator Console

help file 4

Alternative prompt set 111

Ancillary constant 36

ASP.NET integration pages 1

Auto step-in

Single Line 88

Typedown 99

Autoformatsafe flag 89

Autostepsafe flag 88

B

BulkSearch method 24

BulkSearchItem class 80

Constructor method 81

BulkSearchResult class 79

BulkSearchItem method 79

Constructor method 79

C

C#.NET sample code 1

CanSearch class 69

GetMessage method 70

IsOk method 70

CanSearch method 22

CanStep flag 115

CanStep method 65

Clicks_Low constant 49

Client/Server help file 4

Common classes

Address Line class 36

BulkSearchItem class 80

BulkSearchResult class 79

CanSearch class 69

constants 13

DataSet class 41

ExampleAddress class 43

FormattedAddress class 45

Layout class 47

LicensedSet class 49

- overview 7, 117
- Picklist class 56
- PromptLine class 71
- PromptSet class 73
- QuickAddress class 7, 117
- search (address capture) 8
- search (verification) 11
- SearchResult class 75, 79–80
- Single Line engine 8
- Typedown engine 8
- Verification engine 11
- Configuration Editor
 - help file 4
- Constants 13
 - Address 36
 - Address Line class 36
 - Alternative 16
 - Alternative2 16
 - Alternative3 16
 - Ancillary 36
 - Clicks_Low 49
 - Close 15
 - Data_Expired 50
 - Data_Expiring 49
 - Data_Unreadable 50
 - DataPlus 36
 - Default 16
 - Eval_Licence_Expired 50
 - Evaluation 50
 - Exact 15
 - Extensive 15
 - Full_Licence_Expired 50
 - Generic 16
 - InteractionRequired 75, 80
 - Keyfinder 13
 - Licence_Expiring 49
 - Licence_Not_Found 50
 - LicensedSet class 49
 - Multiple 76, 81
 - Name 36
 - No_Clicks 50
 - None 36, 49, 75, 80
 - Optimal 16
 - PremisesPartial 75, 80
 - prompt sets 16
 - PromptSet class 73
 - search engine intensity 15
 - search engine types 13
 - SearchResult class 75, 80
 - Single Line engine 13
 - StreetPartial 75, 81
 - Typedown 13
 - USA 16
 - Verification engine 13
 - Verified 75, 80
 - verify levels 75, 80
 - warning levels 49
 - warning levelsConstants
 - LicensedSet class 49
- Constructor method
 - BulkSearchItem class 81
 - BulkSearchResult class 79
 - DataSet class 41
 - ExampleAddress class 43
 - FormattedAddress class 45
 - Layout class 47
 - LicensedSet class 53
 - Picklist class 56
 - PromptLine class 71
 - PromptSet class 73

- QuickAddress class 21
- SearchResult class 75, 78–81
- Country-specific documentation 4
- CreateArray method
 - DataSet class 42
 - ExampleAddress class 44
 - Layout class 48
 - LicensedSet class 53

D

- Data documentation 4
- Data Guide 4
- Data_Expired constant 50
- Data_Expiring constant 49
- Data_Unreadable constant 50
- DataPlus
 - constant 36
- DataSet class 41
 - Constructor method 41
 - CreateArray method 42
 - FindByID method 42
 - GetID method 42
 - GetName method 42
- Default prompt set 110
- Dummy PO box 67
- Dynamic picklists 94

E

- Engine behaviour 83
 - differences 84, 108
- Eval_Licence_Expired constant 50
- Evaluation
 - constant 50

- ExampleAddress class 43
 - Constructor method 43
 - CreateArray method 44
 - GetAddress method 44
 - GetComment method 44

F

- FindByID method 42
- FindByName method 48
- Flags
 - auto format 88
 - auto step 88
 - canstep 115
 - fulladdress 115
- Flatten option 84
- Flattened mode
 - Keyfinder 108
 - prompt sets 91
 - Single Line 90
 - Single Line example 91
- Format action 88
- FormattedAddress class 45
 - Constructor method 45
 - GetAddressLines method 46
 - IsOverflow method 46
 - IsTruncated method 46
- Full_Licence_Expired constant 50
- FullAddress flag 115
- Fuzzy matching
 - not in Typedown 95

G

- Generic prompt set 111

GetAddress method 44, 78
GetAddressLines method 46
GetAllDataSets method 28
GetAllLayouts method 29
GetBaseCountry method 54
GetComment method 44, 48
GetCopyright method 54
GetDataDaysLeft method 55
GetDataMapDetail method 28
GetDaysLeft method 55
GetDescription method 54
GetExample method 72
GetExampleAddresses method 30
GetFormattedAddress method 27
GetID method 42, 54
GetItems method 59
GetLabel method 39
GetLicenceDaysLeft method 56
GetLicenceInfo method 31
GetLine method 39
GetLinePrompts method 74
GetLines method 74
GetLineType method 39
GetMessage method 70
GetMoniker method 58, 64
GetName method 42, 48
GetPartialAddress method 65
GetPicklist method 78
GetPostcode method 64
GetPrompt method 58, 72
GetScore method 64
GetServer method 55
GetStatus method 55
GetSuggestedInputLength method 72
GetSystemInfo method 31

GetText method 64
GetTotal method 58
GetVerifyLevel method 78
GetVersion method 54
GetWarningLevel method 55

H

Help files

- Administrator Console 4
- Client/Server 4
- Configuration Editor 4

Hierarchical mode

- Single Line 85
- Single Line example 86
- Typedown 96
- Typedown example 97

Hierarchical picklists

- description 112

I

Informational picklist items 114

InputLineCount setting 102

Integration Guide 3

Integration pages

- ASP.NET 1
- JSP 1
- PHP 1

InteractionRequired 107

- constant 75, 80

Interfaces

- .NET 1
- Java 1
- PHP 1

- IsAliasMatch method 65
- IsAutoFormatPastClose method 60
- IsAutoFormatSafe method 60
- IsAutoFormatSingle method 60
- IsAutoStepinPastClose method 59
- IsAutoStepinSafe method 59
- IsAutoStepinSingle method 59
- IsCrossBorderMatch method 67
- IsDummyPOBox method 67
- IsFullAddress method 65
- IsIncompleteAddress method 66
- IsInformation method 68
- IsLargePotential method 60
- IsMaxMatches method 61
- IsMoreOtherMatches method 61
- IsMultiples method 69
- IsName method 67
- IsOk method 70
- IsOverflow method 39, 46
- IsOverThreshold method 61
- IsPhantomPrimaryPoint method 69
- IsPostcodeRecoded method 66
- IsSearchingDynamic method 74
- IsTimeout method 61
- IsTruncated method 40, 46
- IsUnresolvableRange method 66
- IsWarnInformation method 68

J

- Java
 - sample code 1
- JSP integration pages 1

K

- Key Search
 - description 108
- Key search
 - flattened mode 109
- Keyfinder engine
 - constants 13

L

- Layout class 47
 - Constructor method 47
 - CreateArray method 48
 - FindByName method 48
 - GetComment method 48
 - GetName method 48
- Licence_Expiring constant 49
- Licence_Not_Found constant 50
- LicensedSet class 49
 - constants 49
 - Constructor method 53
 - CreateArray method 53
 - GetBaseCountry method 54
 - GetCopyright method 54
 - GetDataDaysLeft method 55
 - GetDaysLeft method 55
 - GetDescription method 54
 - GetID method 54
 - GetLicenceDaysLeft method 56
 - GetServer method 55
 - GetStatus method 55
 - GetVersion method 54
 - GetWarningLevel method 55

M

Multiple 107
 constant 76, 81

N

Name constant 36
No matches 95
No_Clicks constant 50
None
 constant 36
 verify level constant 75, 80
 warning level constant 49

O

OneLine prompt set 111
Optimal prompt set 111

P

Phantom Primary Points 99
PHP
 integration pages 1
Picklist class 56
 Constructor 56
 GetItems method 59
 GetMoniker method 58
 GetPrompt method 58
 GetTotal method 58
 IsAutoFormatPastClose method 60
 IsAutoFormatSafe method 60
 IsAutoFormatSingle method 60
 IsAutoStepinPastClose method 59

IsAutoStepinSafe method 59
IsAutoStepinSingle method 59
IsLargePotential method 60
IsMaxMatches method 61
IsMoreOtherMatches method 61
IsOverThreshold method 61
IsTimeout method 61

Picklist refinement 113

PicklistItem class

 CanStep method 65
 GetMoniker method 64
 GetPartialAddress method 65
 GetPostcode method 64
 GetScore method 64
 GetText method 64
 IsAliasMatch method 65
 IsCrossBorderMatch method 67
 IsDummyPOBox method 67
 IsFullAddress method 65
 IsIncompleteAddress method 66
 IsInformation method 68
 IsMultiples method 69
 IsName method 67
 IsPhantomPrimaryPoint method 69
 IsPostcodeRecoded method 66
 IsUnresolvableRange method 66
 IsWarnInformation method 68

Picklists

 dealing with 101
 dynamic 94
 informational 114
 properties in Single Line 86
 special cases 99

PremisesPartial 107
 constant 75, 80

Prompt set types

- alternative 111
- default 102, 110
- generic 111
- online 111
- optimal 110–111

Prompt sets

- Alternative 16
- Alternative2 16
- Alternative3 16
- constants 16
- Default 16
- description 110
- Generic 16
- Optimal 16
- Single Line 91
- Typedown 94
- USA-specific 16
- Verification 102

PromptLine class 71

- Constructor 71
- GetExample method 72
- GetPrompt method 72
- GetSuggestedInputLength method 72

PromptSet class 73

- constants 73
- Constructor method 73
- GetLinePrompts method 74
- GetLines method 74
- IsSearchingDynamic method 74

Q

QuickAddress class 7, 18, 117

BulkSearch 24

- CanSearch method 22
- Constructor method 21
- GetAllDataSets method 28
- GetAllLayouts method 29
- GetDataMapDetail method 28
- GetExampleAddresses method 30
- GetFormattedAddress method 27
- GetLicenceInfo method 31
- GetSystemInfo method 31
- Refine method 25
- Search method 23–24
- SetConfigFile method 32
- SetConfigSection method 33
- SetEngineIntensity method 33
- SetEngineType method 34
- SetFlatten method 34
- SetThreshold method 35
- SetTimeout method 35
- StepIn method 26

QuickStart Guide 3

R

- Readme file 5
- Refine method 25
- Refining picklists 113

S

Sample code

C#.NET 1

Java 1

Search engine

close intensity 15

- exact intensity 15
- extensive intensity 15
- intensity 15
- types 83
- Search method 23–24
- SearchResult class 75, 79–80
 - constants 75, 80
 - Constructor method 75, 78–81
 - GetAddress method 78
 - GetPicklist method 78
 - GetVerifyLevel method 78
- SetConfigFile method 32
- SetConfigSection method 33
- SetEngineIntensity method 33
- SetEngineType method 34
- SetFlatten method 34
- SetThreshold method 35
- SetTimeout method 35
- Single Line
 - description 84
 - flattened mode 90
 - hierarchical mode 85
- Single Line engine
 - common classes 8
 - constants 13
- Stepin action 88
- StepIn method 26
- StreetPartial 107
 - constant 75, 81
- Support Zone 5

T

- Technical Support 5
- Timer delay 94

- Typedown
 - description 94
 - hierarchical mode 96
 - timer delay 94
- Typedown engine
 - common classes 8
 - constants 13

U

- Unresolvable ranges 99
- Upgrade Guide 5

V

- Verification
 - description 102
 - full user interaction 105
 - minimal user interaction 104
 - no user interaction 103
- Verification engine
 - Common classes 11
 - constants 13
- Verified 106
 - constant 75, 80
- Verify level constants 75, 80
- VerifyLevel 103
 - InteractionRequired 107
 - Multiple 107
 - None 107
 - PremisesPartial 107
 - StreetPartial 107
 - Verified 106

W

Warning level constants 49

Wildcards

 Single Line 84

Windows

 interfaces 1

