

BlackBerry Software Development Kit

Version 2.5

Remote Address Lookup API Reference Guide

BlackBerry Software Development Kit 2.5 Remote Address Lookup API Reference Guide
Last modified: 7 June 2002

Part number: PDF-04803-001

At the time of publication, this documentation complies with RIM Wireless Handheld version 2.5.

© 2002 Research In Motion Limited. All Rights Reserved. The BlackBerry and RIM families of related marks, images and symbols are the exclusive properties of Research In Motion Limited. RIM, Research In Motion, 'Always On, Always Connected', the "envelope in motion" symbol and the BlackBerry logo are registered with the U.S. Patent and Trademark Office and may be pending or registered in other countries. All other brands, product names, company names, trademarks and service marks are the properties of their respective owners.

The handheld and/or associated software are protected by copyright, international treaties and various patents, including one or more of the following U.S. patents: 6,278,442; 6,271,605; 6,219,694; 6,075,470; 6,073,318; D445,428; D433,460; D416,256. Other patents are registered or pending in various countries around the world. Visit www.rim.net/patents.shtml for a current listing of applicable patents.

While every effort has been made to ensure technical accuracy, information in this document is subject to change without notice and does not represent a commitment on the part of Research In Motion Limited, or any of its subsidiaries, affiliates, agents, licensors, or resellers.

Research In Motion Limited
295 Phillip Street
Waterloo, ON N2L 3W8
Canada

Published in Canada

Contents

About this guide	5
Related documentation	6
 Remote Address Lookup API Reference	 7
Using the Remote Address Lookup API.....	7
iDirectoryLookupClient	9
Creating a client object	9
Adding the query object to the client	9
Functions.....	10
iDirectoryLookupQuery	11
Creating a query object	11
Setting the callback function	11
Functions.....	11
DirectoryLookupClientCallback	24
Creating the callback	24
Functions.....	25
iAddressLookup	26
Structures	26
Functions.....	27
iDirectoryLookupAddress.....	30
Functions.....	30
iDirectoryLookupManager.....	31
Functions.....	31
Error codes.....	34
 Index of functions	 35
 Index	 37

About this guide

The Remote Address Lookup application programming interface (API) enables users to search and retrieve Address Book entries from a company or third party directory. The Remote Address Lookup on the RIM Wireless Handheld functions within the Message Compose window, but with this API, developers can enable other applications to interact with a remote directory, and enable all applications to execute queries on other types of directories (such as on the Internet).

The RIM implementation of the Remote Address Lookup is comprised of several components:

- Address Resolution Module (AddressRM.DLL) performs the wireless search, through the SecureTransport. The ARM utilizes the Address Lookup service book.
- By means of the Address Lookup Protocol (ALP), the BlackBerry Enterprise Server connects with the Global Address List (GAL) to create a MAPI session.

Both of these components (or equivalents) must be accessible by the handheld. Developers can create their own ARM and lookup protocol, depending on the required function of the search. A simple alternative usage of the API would be to perform a search of an Internet site and return the results to the handheld.

This API Reference Guide documents how to perform remote queries using the Address Resolution Module (ARM) provided with the BlackBerry SDK. For information about creating and implementing your own ARM and lookup protocol, please contact us at sdk@rim.net.

Related documentation

Before you use this guide, you should be familiar with the following documentation. These other resources can help you develop C++ applications for the RIM Wireless Handheld.

- *BlackBerry SDK 2.5 Developer Guide*

The *BlackBerry SDK Developer Guide* explains how to use the BlackBerry SDK, with tutorials and sample code to demonstrate how to write handheld applications. For additional information, visit the BlackBerry Developer Zone at <http://www.blackberry.net/developers>.

- *System Utilities API Reference Guide*

Remote Address Lookup search queries are objects instantiated by the Registrar. You can use the Registrar to manage object lifetime and allocate string memory. You can view or download the *System Utilities API Reference Guide* at <http://developers.rim.net>.

Chapter 1

Remote Address Lookup API Reference

The RIM Wireless Handheld 3.5 enables you to perform remote queries of a Microsoft Exchange enterprise server Address Book. The Remote Address Lookup API enables developers to mimic this functionality, as well as create applications that are able to query Exchange Global Address Lists (GALs).

The Remote Address Lookup API consists of several components. These enable you to instantiate and manage remote search queries on both clients and services, request and retrieve query results, and add query results to the handheld's Address Book.

Using the Remote Address Lookup API

You can implement the Remote Address Lookup API in several ways, but the most common usage would follow a procedure similar to the following:

1. Instantiate the `iDirectoryLookupClientPtr` interface, which retrieves an instance of a client object. Create the client object.

```
iDirectoryLookupClientPtr myClient;  
myClient.create;
```
2. Instantiate the `iDirectoryLookupQueryPtr`, which retrieves an instance of a query object. Create the query object.

```
iDirectoryLookupQueryPtr myQuery;
```

Chapter 1: Remote Address Lookup API Reference

```
myQuery.create;
```

3. Using `iDirectoryLookupQuery` functions, specify properties of the query as necessary.

```
myQuery.setDesiredMatchSortOrder();  
myQuery.setDesiredNumberOfResults();  
etc.
```

4. Implement the `DirectoryLookupClientCallback` interface to create a callback for the query. Cast the query object to an `iDirectoryLookupQueryAddressBookView` object and pass it into the `ViewSearchResults` method on `iAddressLookup`.

5. Set the callback for the query.

```
MyCallback myCallbackObject;  
myQuery.SetCallback(myCallbackObject);
```

6. Add the query to the client object. The query is executed.

```
myClient->iDirectoryLookupClient::AddQuery(myQuery);
```

7. When the query is ready to be returned, the query callback calls `LookupSearchResultsReady`.

```
LookupSearchResultsReady(myQuery);
```

8. Call `ViewSearchResults` to display the results and provide the user with an interface to manipulate them.

```
myQuery.ViewSearchResults;
```

The classes and interfaces in this API reference guide are organized in a similar manner.

Interfaces/Classes	Page	Header file	Description
<code>iDirectoryLookupClient</code>	9	<code>iDirectoryLookupClient.h</code>	Instantiates client objects and initiates remote queries.
<code>iDirectoryLookupQuery</code>	11	<code>iDirectoryLookupQuery.h</code>	Instantiates query objects and specifies search criteria and query properties.
<code>DirectoryLookupClientCallback</code>	24	<code>iDirectoryLookupQuery.h</code>	Creates a callback class for a query.

Interfaces/Classes	Page	Header file	Description
iAddressLookup	26	iAddressLookup.h	Displays query results in an Address Book view.
iDirectoryLookupAddress	30	iDirectoryLookupAddress.h	Accesses an individual address field from the set of query results.
iDirectoryLookupManager	31	iDirectoryLookupManager.h	Responsible for the registration of multiple lookup client and service providers.

iDirectoryLookupClient

iDirectoryLookupClient is a Registrar interface for initiating remote directory queries. It is implemented by the Address Resolution Module (ARM) to perform a wireless search.

All requests are generated from the handheld. The Remote Address Lookup API does not currently allow the remote directory to push unsolicited results to a handheld.

The iDirectoryLookupClient interface has two primary purposes, creating the client object and adding query objects to the client.

Creating a client object

Before you can begin to perform directory searches, a client object must be created, by instantiating the iDirectoryLookupClientPtr interface. iDirectoryLookupClientPtr is an interface pointer wrapper class used to simplify object instantiation and lifetime management.

```
iDirectoryLookupClientPtr myClient;
myClient.create;
```

Adding the query object to the client

To initiate a search, you must add a query object to the client. Once a query object has been added to the client, the search is initiated.

```
myClient->AddQuery(myQuery);
```

In the above example, myQuery is a query object instantiated by implementing the iDirectoryLookupQueryPtr interface.

Refer to "iDirectoryLookupQuery" on page 11 for more information on implementing iDirectoryLookupQueryPtr to create a query object.

Functions

The following functions are list alphabetically.

AddQuery	10
GetQuery	10
RemoveQuery	10

AddQuery

Registers an instance of a query object.

```
virtual IMETHOD AddQuery(iDirectoryLookupQueryPtr & query) = 0
```

Parameters query A pointer to the query to be initiated.

Description AddQuery initiates a search on the handheld. Query objects are instantiated by implementing iDirectoryLookupQueryPtr.

AddQuery adds a query to the client object.

GetQuery

Retrieves the reference to an instance of a query object.

```
virtual iDirectoryLookupQuery * ICALLTYPE GetQuery(  
    DirectoryLookupQueryID query_id) = 0
```

Parameters query_id The ID of the query to retrieve.

Description GetQuery retrieves the reference to a query in progress.

Each query is assigned a unique identifier upon instantiation. GetQuery returns this unique identifier.

RemoveQuery

Removes an instance of a query object.

```
virtual IMETHOD RemoveQuery(iDirectoryLookupQuery & query) = 0
```

Parameters query A pointer to the query object to destroy.

Description RemoveQuery destroys a search on the handheld. This query is no longer accessible. When the last reference to a query is released, its database of search results is deleted.

RemoveQuery removes a query from the client object.

iDirectoryLookupQuery

The query session itself is implemented in the `iDirectoryLookupQuery` interface. It is used to create the query object, and to set specific search properties for query.

Creating a query object

1. After creating a client object, you must create a query object which contains the search criteria and data. Instantiate the `iDirectoryLookupQueryPtr` interface. `iDirectoryLookupQueryPtr` is an interface pointer wrapper class used to simplify object instantiation and lifetime management.

```
iDirectoryLookupQueryPtr myQuery;
myQuery.create;
```

2. Once a query object has been created, you can specify the search criteria for the query, by calling any `iDirectoryLookupQuery` function, such as `SetDesiredFields`, `SetDesiredMatchSortOrder`, and so on.

Setting the callback function

After implementing `DirectoryLookupClientCallback` to create a callback class for the query, create a callback and then call `setCallback` to set it to this query object.

```
MyCallback myCallbackObject;
myQuery.SetCallback(myCallbackObject);
```

Refer to "DirectoryLookupClientCallback" on page 24 for more information on creating a callback class.

Functions

The following functions are listed alphabetically.

AddDirectoryLookupAddress	12
DisableUserNotify	12
EnableUserNotify	12
GetDesiredFields	13
GetDesiredMatchSortOrder	13
GetDesiredNumberOfResults	13
GetErrorCode	14
GetErrorString	14
GetNumberOfAvailableResults	14
GetNumberOfResults	15
GetNumberOfStoredResults	15
GetOffsetIntoResults	15
GetQueryID	16
GetQueryStr	16
GetResolvedAddress	16
GetResolvedAddressType	16

Chapter 1: Remote Address Lookup API Reference

GetSearchResult	16
GetSearchStatus	17
NewDirectoryLookupAddress	17
SetCallback	17
SetClient	18
SetDesiredFields	18
SetDesiredMatchSortOrder	19
SetDesiredNumberOfResults	20
SetErrorCode	20
SetErrorString	20
SetNumberOfAvailableResults	21
SetNumberOfResults	21
SetNumberOfStoredResults	21
SetOffsetIntoResults	21
SetQueryID	22
SetQueryStr	22
SetResolvedAddressType	22
SetResolvedAddress	23
SetSearchStatus	23

AddDirectoryLookupAddress

Appends additional results to the existing set of results.

```
virtual IMETHOD AddDirectoryLookupAddress(  
    iDirectoryLookupAddressPtr & pDirectoryLookupAddress) = 0
```

Parameters	pDirectoryLookupAddress	A reference to an address entry that is part of the search results.
-------------------	--------------------------------	---

DisableUserNotify

Disables user notification when a query is complete.

```
virtual IMETHOD DisableUserNotify() = 0
```

Description	Notification consists of a status message popup window stating that the query is complete.
--------------------	--

EnableUserNotify

Enables user notification when a query is complete.

```
virtual IMETHOD EnableUserNotify() = 0
```

Description	Notification consists of a status message popup window stating that the query is complete.
--------------------	--

GetDesiredFields

Retrieves the fields in which to search.

```
virtual IMETHOD GetDesiredFields(
    iBuffer * FieldList) = 0
```

Parameters `FieldList` A pointer to the list of fields to search.

Returns The list of fields to search for this query.

Description If no fields are specified, all Address Book fields are queried.

Each search criteria list must consist of one or more address fields, where each field is indicated by a byte from `addressLookupFieldType.h`. If more than one search criteria is provided, each search criteria must be separated by the NULL (0x00) character.

See `SetDesiredFields` for a list of fields specified in `addressLookupFieldType.h`.

`GetDesiredFields` has no effect in the current Address Book implementation, since all fields are returned by default.

GetDesiredMatchSortOrder

Retrieves the required sort order of the query results.

```
virtual IMETHOD GetDesiredMatchSortOrder(
    iBuffer * MatchSortOrderString) = 0
```

Parameters `MatchSortOrderString` A pointer to the query sort order.

Description Each request must contain an octet string that specifies how to perform the search and how to sort the lookup results. The octet string must consist of at least one search criteria, but may contain two or more search criteria. The sort order string consists of these octets, separated by the NULL (0x00) character. For example:

```
0x21 0x00 (Company)
0x39 0x38 0x00 (Last Name, First Name)
0x38 0x39 0x00 (First Name, Last Name)
0x01 (Email)
```

The order of these searches prescribes the required sort order of the results, and the order in which to return the search results to the handheld.

Returns The sort order of the query.

GetDesiredNumberOfResults

Retrieves the number of results that the user wants displayed.

```
virtual IMETHOD GetDesiredNumberOfResults(
```

Chapter 1: Remote Address Lookup API Reference

```
unsigned int * DesiredNumberOfResults) = 0
```

Parameters `DesiredNumberOfResults` A pointer to the desired number of query results.

Description If `DesiredNumberOfResults` is set to `ALL_LOOKUP_RESULTS_DESIRED (0)`, it can be assumed that all results can be displayed to the user. The number of results actually returned in the query response is dependant on the number of available results on the server.

GetErrorCode

Returns the error code for the current operation.

```
virtual IMETHOD GetErrorCode(unsigned int * ErrorCode) = 0
```

Parameters `ErrorCode` A pointer to the error code for the current operation.

Description The possible error codes for a directory query are defined by the `DirectoryLookupQueryErrorCodes` enumeration. Refer to the error codes section for a complete list of possible error codes and strings.

Returns The error code for the current operation.

GetErrorString

Returns the string for the error code of the current operation.

```
virtual IMETHOD GetErrorString(iStr * ErrorString) = 0
```

Parameters `ErrorString` A pointer to the string for the current operation's error code.

Description The possible error codes for a directory query are defined by the `DirectoryLookupQueryErrorCodes` enumeration. Refer to the error codes section for a complete list of possible error codes and strings.

Returns The error string (not NULL-terminated) for the current operation.

GetNumberOfAvailableResults

Retrieves the total number of results available on the server.

```
virtual IMETHOD GetNumberOfAvailableResults(  
    unsigned int * NumberOfAvailableResults) = 0
```

Parameters `NumberOfAvailableResults` A pointer to the number of available results.

Description `GetNumberOfAvailableResults` retrieves the total number of results potentially available to download to the handheld.

Returns The total number of results.

GetNumberOfResults

Retrieves the number of query results in the response.

```
virtual IMETHOD GetNumberOfResults(unsigned int * NumberOfResults) = 0
```

Parameters `NumberOfResults` A pointer to the number of query results.

Description `GetNumberOfResults` retrieves the number of results actually returned to the handheld, regardless of the number the user requested (if a higher number was requested.)

`NumberOfResults` is equal to or less than `NumberOfAvailableResults`.

Returns The number of query results in the response.

GetNumberOfStoredResults

Retrieves the number of returned results.

```
virtual IMETHOD GetNumberOfStoredResults(
    unsigned int * NumberOfStoredResults) = 0
```

Parameters `NumberOfStoredResults` A pointer to the number of results stored on the handheld.

Description `GetNumberOfStoredResults` retrieves the number of results that currently have been retrieved onto the handheld. This function can be used to determine the offset for the next fetch; deletions of search results have no effect on this total.

Returns The number of results stored initially and saved on the handheld.

GetOffsetIntoResults

Retrieves the offset into the results from which to request additional results.

```
virtual IMETHOD GetOffsetIntoResults(unsigned int * OffsetIntoResults) = 0
```

Parameters `OffsetIntoResults` A pointer to the offset into the search results.

Description `GetOffsetIntoResults` retrieves the position in the results array from which to request more results.

If `OffsetIntoResults` is not present, the offset is assumed to be 0.

Chapter 1: Remote Address Lookup API Reference

Returns The offset into the search results.

GetQueryID

Retrieves the ID of the query.

```
virtual DirectoryLookupQueryID GetQueryID() const = 0
```

Description A unique query ID is assigned to a query by GetQueryUID() when the iDirectoryLookupQuery object is instantiated.

Returns The ID of the query.

GetQueryStr

Gets the query string.

```
virtual const char * GetQueryStr() = 0
```

Description The query string is the text for which to search the GAL for.

GetResolvedAddress

Gets the desired match from the search results.

```
virtual iDirectoryLookupAddress * GetResolvedAddress() = 0
```

Returns The resolved address is the query result selected by the user. (For example, Email, PIN, etc.)

GetResolvedAddressType

Gets the field type of the desired match from the search results.

```
virtual DbFieldTag ICALLTYPE GetResolvedAddressType() = 0
```

Returns The resolved address field type of the query result selected by the user.

GetSearchResult

Retrieves the result of the query at the specified index.

```
virtual iDirectoryLookupAddress * GetSearchResult(unsigned int index) = 0
```

Parameters index The location from which to return the search result.

Description GetSearchResult returns a pointer to an iDirectoryLookupAddress object.

Returns The search result at the specified location.

GetSearchStatus

Retrieves the status of the current search.

```
virtual DirectoryLookupSearchStatus GetSearchStatus() const = 0
```

Description The status of a search is defined by the DirectoryLookupSearchStatus enumeration.

Status	Description
LOOKUP_STATUS_INVALID_QUERY_ID	An invalid query ID is assigned.
LOOKUP_STATUS_IN_PROGRESS	The query operation is in progress.
LOOKUP_STATUS_NOT_STARTED	The query operation has not yet been started .
LOOKUP_STATUS_SUCCESS	The query completed successfully, with 0 or more results.
LOOKUP_STATUS_RESOLVED	One of the results returned by the query was selected for use (this status is set by the application).
LOOKUP_STATUS_FAILED	The query operation failed to complete.
LOOKUP_STATUS_CANCELLED	The query was cancelled by the application.

Returns The status of the current search.

NewDirectoryLookupAddress

Creates a new address containing a search result.

```
virtual IMETHOD NewDirectoryLookupAddress(
    iDirectoryLookupAddressPtr & pDirectoryLookupAddress) = 0
```

Parameters pDirectoryLookupAddress A reference to the address.

Description The address is saved as a record in the query's database of search results.

SetCallback

Specifies a callback listener for notification of a completed query operation.

```
virtual IMETHOD SetCallback(DirectoryLookupClientCallback * listener) = 0
```

Parameters listener A pointer to the callback listener.

Chapter 1: Remote Address Lookup API Reference

Description A callback is created by implementing `DirectoryLookupClientCallback`.

SetClient

Specifies the service which performs the search, for this query object.

```
virtual IMETHOD SetClient(idirectoryLookupClient * client) = 0
```

Parameters `client` A pointer to the client over which the query is being performed.

Description `SetClient` prevents a query from being destroyed before it is removed by the client application (a reference pointer to the service is maintained.)

SetDesiredFields

Specifies the desired Address fields to search.

```
virtual IMETHOD SetDesiredFields(  
    const unsigned char * FieldListString,  
    unsigned int FieldListStringLength) = 0
```

Parameters `FieldListString` A pointer to the list of fields to search.

`FieldListStringLength` The length of the list of fields.

Description If no fields are specified, all Address Book fields will be queried.

`SetDesiredFields` has no effect in the RIM implementation; all Address Fields are queried by default.

Each request may contain an octet string that specifies how to perform the search. The octet string must consist of at least one search criteria, but may contain two or more search criteria.

Each search criteria list must consist of one or more address fields, where each field is indicated by a byte from `addressLookupFieldType.h`. If more than one search criteria is provided, each search criteria must be separated by the NULL (0x00) character.

Address field	Octet (Decimal)
EMAIL	0x01 (01)
FAX	0x03 (03)
WORK_PHONE	0x06 (06)
HOME_PHONE	0x07 (07)

Address field	Octet (Decimal)
MOBILE_PHONE	0x08 (08)
PAGER	0x09 (09)
COMPANY	0x21 (33)
ADDRESS_LINE1	0x23 (35)
ADDRESS_LINE2	0x24 (36)
ADDRESS_LINE3	0x25 (37)
ADDRESS_CITY	0x26 (38)
ADDRESS_STATE	0x27 (39)
ADDRESS_POSTAL_CODE	0x28 (40)
ADDRESS_COUNTRY	0x29 (41)
TITLE	0x2A (42)
SALUTATION	0x37 (55)
NAME_FIRST	0x38 (56)
NAME_LAST	0x39 (57)

SetDesiredMatchSortOrder

Sets the required sort order of the query results.

```
virtual IMETHOD SetDesiredMatchSortOrder(
    const unsigned char * MatchSortOrderString,
    unsigned int MatchSortOrderStringLength) = 0
```

Parameters	MatchSortOrderString	A pointer to the query sort order. This value is not NULL-terminated.
	MatchSortOrderStringLength	The length of the sort order string.

Description Each request must contain an octet string that specifies how to sort the lookup results. The octet string must consist of at least one search criteria, but may contain two or more search criteria. For example:

Desired matching and sort order:

Chapter 1: Remote Address Lookup API Reference

0x21 0x00 (Company)
0x39 0x38 0x00 (Last Name, First Name)
0x38 0x39 0x00 (First Name, Last Name)
0x01 (Email)

In the above example, the query will sort the search results first according to matches in the Company field, and so on. The order of these searches prescribes the required sort order of the results, and the order in which to return the search results to the handheld.

SetDesiredNumberOfResults

Sets the number of results requested by the user.

```
virtual IMETHOD SetDesiredNumberOfResults(  
    unsigned int DesiredNumberOfResults) = 0
```

Parameters DesiredNumberOfResults The number of results requested by the user.

Description The desired number of results actually returned in the response is limited by the number of available results on the server (and any server restrictions on traffic size, etc.).

If DesiredNumberOfResults == 0, then all query results are desired.

SetErrorCode

Sets the error code for the current operation.

```
virtual IMETHOD SetErrorCode(unsigned int ErrorCode) = 0
```

Parameters ErrorCode The error code to set.

Description The possible error codes for a directory query are defined by the DirectoryLookupQueryErrorCodes enumeration. Refer to the error codes section for a complete list of possible error codes.

SetErrorCode is called by the ARM.

SetErrorString

Set the string for the error code of the current operation.

```
virtual IMETHOD SetErrorString(const char * ErrorString) = 0
```

Parameters ErrorString A pointer to the error string to set.

Description The possible error codes for a directory query are defined by the DirectoryLookupQueryErrorCodes enumeration.

SetErrorString is called by the ARM.

SetNumberOfAvailableResults

Specifies the total number of results available to the handheld from the server.

```
virtual IMETHOD SetNumberOfAvailableResults(
    unsigned int NumberOfAvailableResults) = 0
```

Parameters `NumberOfAvailableResults` The number of available results.

Description `SetNumberOfAvailableResults` specifies the total number of results potentially available to view on the handheld.

`SetNumberOfAvailableResults` is called by the ARM.

SetNumberOfResults

Specifies the number of query results in the response.

```
virtual IMETHOD SetNumberOfResults(unsigned int NumberOfResults) = 0
```

Parameters `NumberOfResults` The number of results to return.

Description `SetNumberOfResults` specifies the number of results actually returned to the handheld, and may differ from the number the user requested (if a higher number was requested.)

`NumberOfResults` must be equal to or less than `NumberOfAvailableResults`.

`SetNumberOfResults` is called by the ARM.

SetNumberOfStoredResults

Specifies the stored number of returned results.

```
virtual IMETHOD SetNumberOfStoredResults(
    unsigned int NumberOfStoredResults) = 0
```

Parameters `NumberOfStoredResults` The number of stored results.

Description `SetNumberOfStoredResults` specifies the number of results that currently have been retrieved onto the handheld.

`SetNumberOfStoredResults` is called by the client.

SetOffsetIntoResults

Specifies the offset into the results from which more results can be retrieved.

Chapter 1: Remote Address Lookup API Reference

```
virtual IMETHOD SetOffsetIntoResults(unsigned int OffsetIntoResults) = 0
```

Parameters `OffsetIntoResults` The offset into the search results.

Description `SetOffsetIntoResults` specifies the position in the results array from which to request more.

If `OffsetIntoResults` is not specified, the offset is assumed to be 0.

SetQueryID

Specifies the unique ID for this query.

```
virtual IMETHOD SetQueryID(DirectoryLookupQueryID query_id) = 0
```

Parameters `query_id` The ID for this query.

Description A query ID is necessary to retrieve a reference to an instance of a query from the lookup service object.

`SetQueryID` is called by the client to re-use a query object. For example, if the user saved a query and then requested to use it again, the query ID would serve as a reference to the query.

SetQueryStr

Specifies the string on which to perform a search.

```
virtual IMETHOD SetQueryStr(const char * query_str) = 0
```

Parameters `query_str` The query string.

Description `SetQueryStr` is called by the client.

The query string is the text for which to search the GAL for.

SetResolvedAddressType

Specifies the field type of the desired match from the query results, as selected by the user.

```
virtual IMETHOD SetResolvedAddressType(DbFieldTag type) = 0
```

Parameters `type` The Address Book field type of the desired match.

Description `SetResolvedAddressType` is called by the client.

SetResolvedAddress

Specifies the desired match from the query results, as selected by the user.

```
virtual IMETHOD SetResolvedAddress(
    iDirectoryLookupAddress * resolved_addr) = 0
```

Parameters resolved_addr A pointer to the desired match (address entry).

Description SetResolvedAddress is automatically called by iAddressLookup::ViewSearchResults. SetResolvedAddress is called by the client.

SetSearchStatus

Sets the status of the current search.

```
virtual IMETHOD SetSearchStatus(DirectoryLookupSearchStatus status) = 0
```

Parameters status The status to set for this search.

Description The possible status of a search is defined by the DirectoryLookupSearchStatus enumeration.

Status	Set By	Description
LOOKUP_STATUS_INVALID_QUERY_ID	-	The query does not exist; an invalid query ID was requested.
LOOKUP_STATUS_IN_PROGRESS	ARM	The query operation is in progress.
LOOKUP_STATUS_NOT_STARTED	-	The query operation has not been started.
LOOKUP_STATUS_SUCCESS	ARM	The query completed successfully, with 0 or more results.
LOOKUP_STATUS_RESOLVED	Client	One of the results returned by the query was selected (this status is set by the application).
LOOKUP_STATUS_FAILED	ARM	The query operation failed to complete.
LOOKUP_STATUS_CANCELLED	Client	The query was cancelled by the application.

DirectoryLookupClientCallback

Remote directory queries are asynchronous; in other words, a query isn't a single continuous operation. It is better understood as a client-side operation followed by a server-side operation that may or may not occur immediately following the client-side operation.

The client initiates the query and sends the request to the server. Any number of factors can contribute to the server being unable to immediately perform the search and return the results, including network traffic and availability. In order to minimize excessive network latency in which the client would be idly waiting for the server response, the connection is ended and it becomes the server's responsibility to re-establish it when the query is ready.

To direct the server how to re-establish the connection, the client includes a callback function in its query, which contains `LookupSearchResultsReady()`, a function that notifies the client when the query is complete.

Creating the callback

1. Implement the `DirectoryLookupClientCallback` to create a callback class for the query.

```
DirectoryLookupClientCallback myQuery
```

2. After implementing `DirectoryLookupClientCallback`, cast the query object to a `iDirectoryLookupQueryAddressBookView` object (which contains the database views for the query results and is a pointer to the Address Book) and pass it into the `ViewSearchResults` method in `iAddressLookup`.

`iDirectoryLookupQueryAddressBookView` is an interface for displaying query results in the Address Book. `iAddressLookup` enables users to manage the search results (such as, adding them to the Address Book, or composing an email with them, etc.)

```
class MyCallback: public DirectoryLookupClientCallback
{
    virtual bool LookupSearchResultsReady(iDirectoryLookupQuery * query)
    {
        // cast the query object to a
        // iDirectoryLookupQueryAddressBookView object and pass it
        // into iAddressLookup::ViewSearchResults
    }
};
```

Refer to "iDirectoryLookupQuery" on page 11 for more information on setting the callback to a query object.

Functions

DirectoryLookupClientCallback contains only the LookupSearchResultsReady function.

LookupSearchResultsReady

Notifies the client when the server has completed the query.

```
virtual bool LookupSearchResultsReady(iDirectoryLookupQuery * query) = 0;
```

Parameters query A pointer to the query object.

Returns True if successful; false otherwise.

Description LookupSearchResultsReady must cast the query object to an iDirectoryLookupQueryAddressBookView object and then call iAddressLookup::ViewSearchResults.

iAddressLookup

The iAddressLookup interface is used to display the results of the query in the address list window. It interfaces with the handheld's Address Book.

Structures

iAddressLookup contains the following structures.

LookupCommandParameters	26
LookupResultViewParameters	26

LookupCommandParameters

The LookupCommandParameters structure initiates a directory query from within the address list window.

```
struct LookupCommandParameters {
    bool lookup_allowed;
    char * query_str;
    const unsigned int * query_str_len;

    LookupCommandParameters(
        bool in_lookup_allowed = false,
        char * in_query_str = NULL,
        const unsigned int * in_query_str_len = NULL) :

        lookup_allowed(in_lookup_allowed),
        query_str(in_query_str),
        query_str_len(in_query_str_len)
    {}
};
```

Field	Description
lookup_allowed	Boolean determining if remote queries are allowed by the user. If lookup_allowed is set to true, the user is allowed to initiate a new remote directory query from within the Address Book.
query_str	The query string.
query_str_length	The length of the query string.

LookupResultViewParameters

The LookupResultViewParameters structure processes search results from searches performed within the address list window.

```
struct LookupResultViewParameters {
```

```

AddressDatabase * database;
AddressView * result_view;
AddressView * working_list_view;
AddressIndexSearch * search_view;
unsigned int num_stored_results;
unsigned int num_available_results;

LookupResultViewParameters(
    AddressDatabase * in_database = NULL,
    AddressView * in_result_view = NULL,
    AddressView * in_working_list_view = NULL,
    AddressIndexSearch * in_search_view = NULL,
    unsigned int in_num_stored_results = 0,
    unsigned int in_num_available_results = 0) :

    database(in_database),
    result_view(in_result_view),
    working_list_view(in_working_list_view),
    search_view(in_search_view),
    num_stored_results(in_num_stored_results),
    num_available_results(in_num_available_results)
{}
};

```

Field	Description
database	The database containing the search results.
result_view	The database view of the search results.
working_list_view	The database view of the search results contained by a local search.
search_view	The database indexed search view.
num_stored_results	The number of query results stored.
num_available_results	The number of query results available on the server.

Functions

The following functions are listed alphabetically.

AddLookupResultToDatabase	27
ViewSearchResults	28

AddLookupResultToDatabase

Adds the resolved query result to the Address Book.

Chapter 1: Remote Address Lookup API Reference

```
virtual bool AddLookupResultToDatabase(  
    iDirectoryLookupAddressPtr & lookup_addr,  
    bool check_for_duplicates = true) = 0
```

Parameters	lookup_addr	A pointer to the query result to be added.
	check_for_duplicates	True if the function should check for duplicate entries in the Address Book while adding new entries; false otherwise. If check_for_duplicates is set to true, and duplicates entries is disabled in the handheld's Device Options, the duplicate address will not be added to the address book.

Returns True if the result was saved to the Address Book; false otherwise.

Description Once the user has selected (resolved) a query result, AddLookupResultToDatabase can be used to add the query result to the Address Book. The result might not be saved to the Address Book for a number of reasons, for example, if checking for duplicates is enabled, and the result was a duplicate.

ViewSearchResults

Displays the search results in the address list window.

```
virtual int ViewSearchResults(bool lookup_allowed,  
    iDirectoryLookupQueryAddressBookViewPtr & lookup_query_view_if,  
    unsigned int comm_method_supported,  
    AddressFieldType & comm_method_chosen,  
    char * lookup_str = NULL,  
    const unsigned int * lookup_str_len = NULL,  
    unsigned int num_stored_results = 0,  
    unsigned int num_available_results = 0) = 0
```

Parameters	lookup_allowed	True if the user is allowed to start a new query when viewing search results.
	lookup_query_view_if	A reference to the lookup query to be viewed.
	comm_method_supported	Communication methods supported.
	comm_method_chosen	Communication method selected by the user.
	lookup_str	A buffer for the new lookup query string (for a new query initiated by the user).

<code>lookup_str_len</code>	Length of the query string buffer.
<code>num_stored_results</code>	Number of query results stored on the handheld.
<code>num_available_results</code>	Total number of query results available on the remote server.

Returns One of the following `eSelectMethodUsingWithLookupReturnCode` results:

Return code	Description
CANCEL	The user cancelled the selection.
SELECT	The user selected an address entry.
GET_MORE_LOOKUP_RESULTS	The user requested the next set of results be retrieved.
GET_ALL_LOOKUP_RESULTS	The user requested that all results be retrieved.
DELETE_LOOKUP	The user deleted the query.
START_LOOKUP	The user initiated a new query.

Description Communication methods (email, phone, fax, etc.) are specified in `addressfieldtype.h`.

If a new query is initiated from the lookup results view, the new query string is copied into the `lookup_str` parameter. The new parameter must point to a buffer in the caller, with the length specified by `lookup_str_len`.

iDirectoryLookupAddress

iDirectoryLookupAddress is the interface for accessing an individual address entry from the set of query results. It works in conjunction with iDirectoryLookupQuery to implement the query session.

In the RIM Address Book implementation each iDirectoryLookupQuery object maintains its own database, and each iDirectoryLookupAddress comprises a separate record in this database.

Functions

The following functions are listed alphabetically.

ReadField	30
SetField	30

ReadField

Reads an Address Book field from the query result.

```
virtual const char * ReadField(DbFieldTag type, char * buffer,
    unsigned int buffer_len) const = 0
```

Parameters	type	The Address Book field type. It must correspond to a field defined in addresslookupfieldtype.h.
	buffer	The buffer into which the field should be written.
	buffer_len	The length of the buffer.

Returns NULL if the lookup result cannot be found.

Description ReadField copies the requested field into the buffer.

SetField

Assigns an Address Book field type to a query result.

```
virtual IMETHOD SetField(DbFieldTag type, const char * buffer,
    unsigned int buffer_len) = 0
```

Parameters	type	The Address Book field type to set this result as.
	buffer	The buffer into which the field should be written.
	buffer_len	The length of the buffer.

iDirectoryLookupManager

The iDirectoryLookupManager interface is responsible for the registration of multiple lookup client and service providers. There can only be one instance of a lookup manager.

When you instantiate a client object by implementing iDirectoryLookupClientPtr, you request the default service type. It is possible to create your own Address Resolution Module, which would require that clients using it be instantiated with its own service type. If you implement your own service, you would need to instantiate each client object manually, or create a pointer wrapper interface similar to iDirectoryLookupClientPtr to request the new service type by default.

For information about creating and implementing your own ARM and lookup protocol, please contact us at sdk@rim.net.

The DirectoryLookupServiceType enumeration defines the possible service types:

Service type (Code)	Description
LOOKUP_SERVICE_NOT_SELECTED (-1)	The client doesn't specify the service type.
LOOKUP_SERVICE_INVALID (0)	The Lookup Manager's service map may return NULL on queries.
LOOKUP_SERVICE_EXCHANGE_GAL (1)	Service is a Global Address List (Exchange) directory.
LOOKUP_SERVICE_USER (128+)	User-defined service type (must have a value of 128 or higher).

Functions

The following functions are listed alphabetically.

FindLookupService	31
GetQueryUID	32
RegisterLookupService	32
UnregisterLookupService	33

FindLookupService

Registers the client interface for initiating remote directory queries.

```
virtual IMETHOD FindLookupService(ClassId & class_id,
    DirectoryLookupServiceType service_type =
    LOOKUP_SERVICE_NOT_SELECTED) = 0
```

Chapter 1: Remote Address Lookup API Reference

Parameters	<code>service_type</code>	The query service type.
	<code>class_id</code>	The class ID of the first lookup service implementation watching the <code>service_type</code> . Note: If multiple services of the same type are registered, then only the first entry is matched and returned. If <code>service_type == LOOKUP_SERVICE_NOT_SELECTED</code> , then the first service of any type is returned; if none are found, <code>IRESULT_FAIL</code> is returned.
Description	When you instantiate <code>iDirectoryLookupClientPtr</code> , <code>FindLookupService</code> is called internally, specifying an Exchange GAL as the service type. If you are implementing your own Address Resolution Module, you would have to call <code>FindLookupService</code> yourself, and specify the <code>service_type</code> . If you do not specify a lookup service, the first available service is used.	

GetQueryUID

Retrieves a unique query identifier to assign to a new `iDirectoryLookupQuery` object.

```
virtual DirectoryLookupQueryID ICALLTYPE GetQueryUID() = 0
```

Returns The unique identifier for a query.

Description When you instantiate `iDirectoryLookupQueryPtr`, `GetQueryUID` is called internally, assigning a unique identifier to the query object.

RegisterLookupService

Registers an Address Resolution Module for initiating remote directory queries.

```
virtual IMETHOD RegisterLookupService(  
    DirectoryLookupServiceType service_type, ClassId class_id) = 0
```

Parameters	<code>service_type</code>	The query service type.
	<code>class_id</code>	The class ID of the lookup service implementation to be registered.

Returns Success if the lookup service is successfully registered.

Description In the RIM implementation, the **Search** option does not appear in the Address Book application on the handheld unless a valid ARM with a `LOOKUP_SERVICE_GAL` `service_type` has been registered.

If you are implementing your own Address Resolution Module, you would have to call `RegisterLookupService` to register the ARM with the handheld.

UnregisterLookupService

Unregisters the ARM so that remote directory queries can no longer be performed.

```
virtual IMETHOD UnregisterLookupService(  
    DirectoryLookupServiceType service_type, ClassId class_id) = 0
```

- Parameters**
- | | |
|---------------------------|---|
| <code>service_type</code> | The query service type. |
| <code>class_id</code> | The class ID of the lookup service implementation to be unregistered. |
- Returns** Success if the lookup service is successfully unregistered.
- Description** You can only call `UnregisterLookupService` in reference to an implementation of your own Address Resolution Module. You cannot unregister the default ARM.

Error codes

The following error codes pertain to the RIM implementation of the Remote Address Lookup.

Error code from 0-127 are detected on the ARM. Error codes 128 and greater are issued by the BlackBerry Enterprise Server and propagated to the device. You can use the EventLogger and EventViewer to diagnose Remote Address Lookup errors.

The following is a complete list of all error codes.

Error Code	String	Description
0	LOOKUP_ERROR_DEVICE_START	An error occurred on the handheld during initiation of the query.
	LOOKUP_ERROR_UNKNOWN	An unknown error occurred.
	LOOKUP_ERROR_OUT_OF_MEMORY	The system ran out of memory before the operation could be completed.
	LOOKUP_ERROR_NO_SERVICE_BOOK	The operation could not be performed because no supporting service book could be found on the handheld (such as the ALP Transport service book).
	LOOKUP_ERROR_NO_SECURETRANSPORT	The operation could not be performed because the SecureTransport layer could not be found.
	LOOKUP_ERROR_SECURETRANSPORT_ERROR	The operation could not be performed because of an error sending datagram through the SecureTransport layer.
128 and higher	LOOKUP_ERROR_REMOTE_START	An error occurred on the remote service during initiation of the query.

Index of functions

D

DirectoryLookupClientCallback
 LookupSearchResultsReady(), 25

I

iAddressLookup
 AddLookupResultToDatabase(), 27
 ViewSearchResults(), 28
 iDirectoryLookupAddress
 ReadField(), 30
 SetField(), 30
 iDirectoryLookupClient
 AddQuery(), 10
 GetQuery(), 10
 RemoveQuery(), 10
 iDirectoryLookupManager
 GetQueryUID(), 32
 RegisterLookupClient(), 31
 RegisterLookupService(), 32
 UnregisterLookupService(), 33
 iDirectoryLookupQuery
 AddDirectoryLookupAddress(), 12
 DisableUserNotify(), 12
 EnableUserNotify(), 12
 GetDesiredFields(), 13
 GetDesiredMatchSortOrder(), 13
 GetDesiredNumberOfResults(), 13
 GetErrorCode(), 14

GetErrorString(), 14
 GetNumberOfAvailableResults(), 14
 GetNumberOfResults(), 15
 GetNumberOfStoresResults(), 15
 GetOffsetIntoResults(), 15
 GetQueryID(), 16
 GetQueryStr(), 16
 GetResolvedAddress(), 16
 GetResolvedAddressType(), 16
 GetSearchResults(), 16
 GetSearchStatus(), 17
 NewDirectoryLookupAddress(), 17
 SetCallback(), 17
 SetClient(), 18
 SetDesiredFields(), 18
 SetDesiredMatchSortOrder(), 19
 SetDesiredNumberOfResults(), 20
 SetErrorCode(), 20
 SetErrorString(), 20
 SetNumberOfAvailableResults(), 21
 SetNumberOfResults(), 21
 SetNumberOfStoredResults(), 21
 SetOffsetIntoResults(), 21
 SetQueryID(), 22
 SetQueryStr(), 22
 SetResolvedAddress(), 23
 SetResolvedAddressType(), 22
 SetSearchStatus(), 23

Index of functions

Index

A

- adding
 - desired match, 27
 - resolved query, 27
- address field values, 18
- ARM
 - registering, 32
 - unregistering, 33

C

- callback
 - about, 24
 - creating, 24
 - setting, 11, 17
- client
 - adding a query to, 9
 - creating, 9
- communication methods, 29

D

- destroying, query object, 10
- disabling, notification, 12

E

- enabling, notification, 12
- error
 - code
 - ARM, 34
 - complete list, 34
 - retrieving, 14
 - setting, 20
 - string
 - retrieving, 14
 - setting, 20

G

- Global Address List, 31

I

- instantiating, query object, 10

L

- lookup service
 - registering, 32
 - unregistering, 33

N

- notification, disabling, 12
- notification, enabling, 12

O

- offset
 - not specified, 15
 - retrieving, 15
 - setting, 21

Q

- query ID
 - assigning a new, 32
 - retrieving, 16
 - setting, 22

R

- reference to a query object, 10
- registering
 - ARM, 32
 - client interface, 31
 - lookup service, 32
 - query object, 10
- resolved address
 - setting, 23

Index

resolved address type

setting, 22

results

appending additional results to, 12

displaying, 28

number in response, 15, 21

total number available, 14, 21

retrieving

available results, 14

desired fields, 13

desired match, 16

desired match sort order, 13

desired match type, 16

desired number of results, 13

error code, 14

error string, 14

field from query result, 30

number of stored results, 15

offset, 15

query ID, 16

query object reference, 10

query result, 16

resolved address, 16

resolved address type, 16

results in a response, 15

search status, 17

RIM implementation, 18, 30, 32

S

search status

retrieving, 17

setting, 23

service type, 31, 33

setting

available results, 21

desired fields, 18

desired match, 23

desired match field type, 22

desired match sort order, 19

desired number of results, 20

error code, 20

error string, 20

number of stored results, 21

offset, 21

query completion callback function, 17

query content, 22

query ID, 22

query string, 22

resolved address, 23

resolved address field type, 22

results in a response, 21

search status, 23

unique ID, 22

U

unregistering

ARM, 33

lookup service, 33

query object, 10



© 2002 Research In Motion Limited
Published in Canada