BlackBerry Software Development Kit

Version 2.5

Address Book API Reference Guide

BlackBerry Software Development Kit Version 2.5 Address Book API Reference Guide
Last revised 06 May 2002

Part number PDF-04633-001

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

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

Published in Canada

Contents

About this guide

This guide describes the Address Book application programming interface (API), which is part of the extended API set for the RIM Wireless Handheld. The Address Book database in the RIM Wireless Handheld maintains a list of contacts. Using the functions of the Address Book API, your application can add, delete, edit and send and receive contact information, and can send messages to contacts.

Related documentation

Before using 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 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.`

- *Database API Reference Guide*

  If your application needs to create its own database, refer to the *Database API Reference Guide*.

- *Remote Address Lookup API Reference Guide*

  Using the Remote Address Lookup API, your applications can access and record contact information from an enterprise server. The RIM implementation of the Remote Address Lookup interfaces with the Address Book application.

- `README.txt`

  The `README.txt` file is installed with the BlackBerry Software Development Kit (SDK). It provides information on any known issues and workarounds, as well as last-minute documentation updates and release notes.

Chapter 1
Address Book API Reference

The Address Book API is a set of interfaces that access the Address Book database. (The Address Book application creates and maintains the Address Book database.) Using these calls, an application can search the Address Book database by contact name, create new records, add fields to a record, and edit the data in the fields.

The Address Book application on the RIM Wireless Handheld is an example of an application that uses the Address Book API. The *Handheld User Guide* explains how to use the aplication to select addresses and search for addresses.

To use the following Address Book API functions in your application, you must include `<address.h>` in your code.

Functions

The following functions are listed alphabetically.

The following group addressing functions can be used if the network provider makes the service available. Group addresses must be created using the RIM Address Book application. To use group address functions in your application, you must include `<AddressGroupAPI.h>` in your code.

add_address_field_type

Enables an application to add or modify an address field type.

```
bool add_address_field_type(
    AddressFieldType field_type,
    AbFlag command_flags = UPDATE_FIELD_FLAGS,
    AbFlag field_flags = FIELD_DEFAULT,
    const char * field_name = NULL,
    const char * field_label = NULL,
    int max_edit_size = DEFAULT_MAX_EDIT_SIZE)
```

Parameters

`field_type`	The type of the address field.
`command_flags`	Characteristics to be added or updated for this field type; refer to the table below.
`flags`	If set, the characteristics to be associated with this field type; refer to the table below.
`field_name`	If set, the name(s) to be used for this field type.
`field_label`	If set, the label(s) to be used for this field type.
`max_edit_size`	If set, the maximum number of characters that can be placed in this type of field.

Returns True if the operation is successful; false otherwise.

Description add_address_field_type updates or creates the field type for the address book records. The Address Book provides a number of predefined field types. Applications can also add their own field types, and/or modify the characteristics of any address field type as required.

The following table displays a list of possible command flags:

Flag	Action
CREATE_FIELD	Set this flag to create a new field type. If it is set, all other parameters must be provided; that is, it is equivalent to: SET_FIELD_FLAGS | SET_FIELD_NAME | SET_FIELD_LABEL | SET_FIELD_EDIT_SIZE If it is not set, then this function will update the given field type using the other command flags.
SET_FIELD_FLAGS	Set this flag to set the field characteristics for the given field type. If this flag is not set, the current field flags are retained.
UPDATE_FIELD_FLAGS	Set this flag to add new characteristics to the given field type. The current characteristics are retained, and the given characteristics are OR'ed with them.
SET_FIELD_EDIT_SIZE	Set this flag to use the value specified in max_edit_size when you edit a field of this type.
SET_FIELD_NAME	Set this flag to point to a NULL-terminated string (or strings) which is returned by calls to get_field_type_name for this field type.
SET_FIELD_LABEL	Set this flag to point to a NULL-terminated string (or strings) that is returned by calls to get_field_type_label for this field type.

Note: SET_FIELD_FLAGS and UPDATE_FIELD_FLAGS should not be set in the same function call. If either one is set, the flags parameter is used to set or update the field characteristics.

Field labels are used when displaying a field and its contents. The format is generally fieldLabel field_data.

In the following example, the field label is Email: Email: johndoe@rim.net.

Field names are typically used in menus. A typical format is Action Field_name.

In the following example, "Email" and "PIN" are field names:

```
Use Email
Use PIN
```

The following table displays a list of possible field characteristic flags:

Field	Description
`FIELD_COMM_METHOD`	Set this flag to specify the field as a communication method.
`FIELD_DEFAULT`	Set this flag to specify the default field characteristics, equal to FIELD_TEXT | FIELD_EDITABLE.
`FIELD_EDITABLE`	Set this flag to specify that the field is editable by the user.
`FIELD_LOCAL`	Set this flag to specify that the field should not be backed up, or sent from the handheld.
`FIELD_MULTIPLE`	Set this flag to specify that the field can occur more than once in a single address record (for example, Email).
`FIELD_NEW_CREATE`	Set this flag to specify that the field is created for each new address record.
`FIELD_REQUIRED`	Set this flag to specify that this field must exist in all address records.
`FIELD_SORTABLE`	Set this flag to specify that this field can be used for sorting address records.
`FIELD_TEXT`	Set this flag to specify that this field can be displayed to the user.
`FIELD_UNIQUE`	Set this flag to specify that this field contains unique data. Each field of this type in the Address Book must contain different data.

add_validation_callback

Adds a callback to determine if data is valid for a specific field type.

```
void add_validation_callback(AddressFieldType field_type,
    bool(*callback_verify)(const char * data,
        AddressFieldType field_type)
```

Parameters

`field_type`	The address field to affect.
`(*callback_verify) (const char * data, AddressFieldType field_type)`	A pointer to the callback function.

Description Before a callback can be added, `register_address_book_update` must be called.

This function enables you to validate user input into Address Book fields that you create.

delete_address_record

Removes an address record from the database.

```
bool delete_address_record(AddressHandle address_handle)
```

Parameters

`address_handle`	Handle for the address record to remove.

Returns True if the record is deleted successfully; false if the record deletion fails or if the address handle is invalid.

edit_address_data

Edits data in an address field.

```
bool edit_address_data(
    AddressHandle address_handle,
    AddressFieldHandle field_handle = DB_INVALID_FIELD_HANDLE,
    AddressFieldType field_type = INVALID_FIELD_TYPE)
```

Parameters

`address_handle`	Handle for the address record.
`field_handle`	If set, handle for address field that is to have initial edit focus. If set to `DB_INVALID_FIELD_HANDLE`, the first editable field in the address record has the initial edit focus. **Note:** `field_type` must be set to the type associated with this field handle.
`field_type`	Type of field in the address record that is to have initial edit focus. If set to `INVALID_FIELD_TYPE`, the first editable field in the address record has the initial edit focus.

Returns True if the address record could be edited; false otherwise.

Description `edit_address_data` enables the user to modify the editable data in the address record. The field, if specified by `field_type` or `field_handle`, is given initial edit focus. If no field is specified, then the first editable field (typically the **First Name** field) has initial edit focus.

find_address_handle

Retrieves a handle for a specific address record.

```
AddressHandle find_address_handle(
    AddressFieldType field_type,
    byte * data_ptr, int data_length,
    AddressHandle * field_handle = NULL)
```

Parameters

`field_type`	Type of data pointed at by `data_ptr`.
`data_ptr`	Pointer to the data to be matched.
`data_length`	Length of the data. If set to `DB_TEXT`, then `data_ptr` points to a NULL-terminated string.
`field_handle`	If set, pointer to a data location in which the handle of the matching field in the address record is to be placed.

Returns Handle of the first address record that has a field of the given field type and field data that matches the given data. The data lengths must match, but the comparison is case insensitive.

Returns `DB_NULL_HANDLE` if:

- `field_type` is invalid
- `data_ptr` is NULL or empty (“”)
- no address record fields with the given field type match a given data string

If the `field_handle` parameter is provided, the handle of the matching field is also returned.

Description `find_address_handle` searches for the first address record that contains a field of `field_type` that matches the data pointed at by `data_ptr`.

Example: A Message application can use this function to determine the friendly name of an incoming message sender. In this case, the email or PIN information is matched against the current contacts and their **Email** or **PIN** fields. For example, the following code could be used to find the name ‘John Doe’ in the Address Book:

```
char * raw_name ="johndoe@rim.net"; // Sender Email address
char friendly_name[40]; // Name of Sender

AddressHandle adh = find_address_handle(EMAIL,
    (unsigned char *) raw_name, DB_TEXT);

if ( adh != DB_NULL_HANDLE)
{
    get_name( adh, (char *) friendly_name, 40 );
}
```

get_address_book_change_count

Retrieves the number of changes since the last handheld reset.

```
int get_address_book_change_count()
```

Returns The number of changes made to the address book database since the last handheld reset.

get_address_field_data

Retrieves data from the address book.

```
const byte * get_address_field_data(
     AddressHandle address_handle,
     AddressFieldHandle field_handle,
     int * data_length = NULL)
```

Parameters

`address_handle`	Handle for the address record.
`field_handle`	Handle for field within the address record.
`data_length`	If set, a location in which the data length of the field is to be placed.

Returns A pointer to the data for the given field within the address record. It returns NULL if either the `address_handle` or `field_handle` is invalid. If specified, the length of the data is placed in `data_length`.

Note: If `address_handle` is equal to `ONESHOT_HANDLE`, the value of `field_handle` is ignored.

Description If `field_handle` is NULL, `get_address_field_data` looks for the first instance of the field type. If `field_handle` is not NULL, `get_address_field_data` looks for the next instance of the field type.

get_address_field_handle

Retrieves a handle to an address field.

```
AddressFieldHandle get_address_field_handle(
    AddressHandle address_handle,
    AddressFieldType field_type,
    AddressFieldHandle * field_handle = NULL)
```

Parameters

`address_handle`	Handle for the address record within the Address Book.
`field_type`	Type of the address field.
`field_handle`	If set, the handle of the previous field in the address record.

Returns If a field is found in the address record with the given field type, then the handle of this field is returned. Otherwise, `DB_INVALID_FIELD_HANDLE` is returned.

Description By setting the `field_handle` parameter to NULL, this function returns the first field whose type matches the `field_type` parameter, for the given address record. If the `field_handle` parameter is a valid field handle, then this function returns a handle to the next field in the address record whose type matches `field_type`. When no more matching fields of the given type can be found in the given address record, `DB_INVALID_FIELD_HANDLE` is returned.

get_address_field_type

Retrieves the type of a field within an address record.

```
AddressFieldType get_address_field_type(
    AddressHandle address_handle,
    AddressFieldHandle field_handle)
```

Parameters

`address_handle`	Handle for the address record.
`field_handle`	Handle for the field within the address record.

Returns The type of data in the field. `INVALID_FIELD_TYPE` is returned if the `address_handle` or `field_handle` is invalid.

Description This function can be used to get the type of data associated with a given field in an address record.

get_address_handle_from_UIN

Retrieves an address handle from a database record ID. This is a group addressing function.

```
bool get_address_handle_from_UIN(
    DbRecordId UIN,
    AddressHandle & adHandle)
```

Parameters

`UIN`	Unique identifier number (database record ID).
`adHandle`	Storage location for resulting address handle.

Returns True if successful; false otherwise.

get_address_match_count

Retrieves the number of contacts with names that match the given search string.

```
int get_address_match_count(
    const char * search_string,
    AddressHandle * address_handle = NULL)
```

Parameters		
	`search_string`	String of characters that is to be matched.
	`address_handle`	If set, points to a data location in which the handle of the first address record matching the search string is to be placed.

Returns The number of address records matching the given search string.

Description `get_address_match_count` displays the number of matched addressees.

get_address_method_count

Retrieves the number of communication methods available for the given contact (address record).

```
int get_address_method_count(
    AddressHandle address_handle,
    AddressFieldHandle * field_handle = NULL,
    AddressFieldType this_field_type = INVALID_FIELD_TYPE)
```

Parameters		
	`address_handle`	Handle for the address record.
	`field_handle`	If set, the location in which the handle for the first field of the given field type will be placed.
	`this_field_type`	Communication method field type to be counted.

Returns The number of send paths (communication methods) that the given contact has. If no type is specified, then the count of all send methods for the contact is returned.

If the contact has only one matching communication method then `field_handle`, if set, it points to the handle of the field within the address record.

Description `get_address_method_count` provides the number of address communication methods.

get_default_method

Retrieves the default communication method for a contact.

```
bool get_default_method(
    AddressHandle address_handle,
    AddressFieldHandle * field_handle,
    AddressFieldType * field_type = NULL)
```

Parameters

`address_handle`	Handle for the address record.
`field_handle`	If set, the location in which the field handle will be placed.
`field_type`	If set, the location in which the field type will be placed.

Returns True if the default method for the given contact could be found; false otherwise.

Description The field representing the default communication method is returned in the `field_handle` parameter.

The default communication method is defined as the last communication method that was selected for a given contact found using a call to `select_method_using`.

get_field_type_name

Retrieves the display string associated with the given field type.

```
const char * get_field_type_name(
    AddressFieldType field_type,
    int offset = 0)
```

Parameters

`field_type`	The type of address field.
`offset`	Offset within the field type name array. The `offset` parameter can be used to select from two or more names for the same field type. For example, the name `field_type` has two names: First and Last. You could use the following code to get the label of the Last name field: `const char * last_name = get_field_type_name (NAME, 1);`

Returns A displayable NULL-terminated string for a given field type.

Description This function returns “Email” in the following example:

```
const char * email_label=get_field_type(EMAIL);
```

get_group_member_info

Retrieves information on a recipient in a group address.

```
bool get_group_member_info(
    AddressHandle group_handle,
    int index,
    AddressHandle * address_handle,
    char ** field_data)
```

Parameters

`group_handle`	Handle of group address.
`index`	Index (0-based) of the recipient about whom information is wanted; the last entry is at `get_num_of_members_in_group(group_handle) - 1.`
`address_handle`	Storage for the handle that corresponds to the address of the recipient, specified by `index.`
`field_data`	Storage for the email address string of the address of the recipient, specified by `index.`

Returns True if successful; false if the data could not be retrieved.

This is a group addressing function.

get_name

Retrieves the name associated with a given address record.

```
bool get_name(
    AddressHandle address_handle,
    char * buffer,
    int length)
```

Parameters

`address_handle`	Handle for the address record.
`buffer`	Location of the data buffer.
`length`	Length of the data buffer.

Returns True if the `address_handle` points to a valid address record and the name can be copied to the buffer; false otherwise.

Description `get_name` has a return value of the name for the given address. A string that contains the name is copied into the `buffer` parameter and its length is returned in the `length` parameter.

get_num_of_members_in_group

Retrieves the number of recipients in group address.

```
int get_num_of_members_in_group(AddressHandle adHandle)
```

Parameters

`adHandle`	Handle of the group address to query.

Returns Number of recipients for the group address, or -1 if `adHandle` is invalid or not a group address.

This is a group addressing function.

get_ordinal_offset_for_field_handle

Retrieves an ordinal number for field handle in an address.

```
byte get_ordinal_offset_for_field_handle(
    AddressHandle adHandle,
    AddressFieldHandle fh,
    AddressFieldtype FieldType = EMAIL)
```

Parameters

`adHandle`	Handle of the address to query.
`fh`	Field handle.
`FieldType`	The type of field.

Returns A 1-indexed byte value representing the ordinal of the field handle `fh`. If `fh` is the first instance of a field of type `FieldType`, the function returns 1. If it is the second, 2, and so on. If there are no instances of `FieldType`, it returns 0.

get_uin_from_address_handle

Retrieves the UIN (database record ID) associated with an address handle.

```
DbRecordId get_uin_from_address_handle(AddressHandle address_handle)
```

Parameters

`address_handle`	Address handle of a group address.

Returns Database record ID of the address handle, or `DB_NULL_RECORD_ID` if the `DbRecordHandle` cannot be found.

This is a group addressing function.

is_group_address

Determines whether an address handle corresponds to a group address.

```
bool is_group_address(AddressHandle adHandle)
```

Parameters

`adHandle`	Handle of the address to query.

Returns True if `adHandle` corresponds to a group address; false otherwise.

This is a group addressing function.

register_address_book_update

Registers with the Address Book to receive callbacks when address book data changes.

```
bool register_address_book_update(void (*callback_function)(void))
```

Parameters

`(*callback_function)(void)`	A pointer to a callback function that accepts no parameters and has a return value of void.

Returns False if the callback function pointer is NULL; true otherwise.

Description To avoid blocking the calling thread, the callback function should generate a separate thread to process address book changes. The `get_address_book_change_count` function can be used to determine if any changes have been made since the last handheld reset.

save_address_data

Saves the new or updated data for the field in the address record.

```
bool save_address_data(
    AddressHandle & address_handle,
    byte * data_ptr,
    int data_length,
    AddressFieldHandle field_handle = DB_INVALID_FIELD_HANDLE,
    AddressFieldType field_type = INVALID_FIELD_TYPE,
    bool UniqueField = true,
    bool AppendBinaryToText = false)
```

Parameters

`address_handle`	Handle for the address record to be saved. It should be set to `DB_NULL_HANDLE` to save a new address book record, or to the appropriate address handle to update an existing record.
`data_ptr`	Pointer to the data to be saved.
`data_length`	Length of the data to be saved.
`field_handle`	If set, the handle of the address field to be updated. To save a new field, set to `DB_INVALID_FIELD_HANDLE`; to update a field, set to the handle of that field.

`field_type`	Field type of the address record field to be saved. To save a new field, set to `INVALID_FIELD_TYPE`; to update a field, set to the type of that field.
`UniqueField`	If true, the field must be a unique field in this address record. If no fields of the given field type exist in this address record, a new field is created. Otherwise, the first field of this type is updated.
`AppendBinaryToText`	If true, the given data buffer points to binary (opaque) data that is appended to the text data for the given field.

Returns True if the operation is successful; false otherwise.

Description This function saves the new or updated data for the field within the address record. Note that either a valid field handle or a valid field type must be specified.

select_method (deprecated)

Selects the contact name and send method.

```
bool select_method(
    AddressHandle * address_handle,
    AddressFieldHandle * field_handle,
    char * search_string = NULL,
    const char * title = NULL,
    *AddressFieldType * field_type = NULL,
    char ** field_data = NULL,
    int * field_length = NULL,
    bool force_field_type = false)
```

Parameters

`address_handle`	Pointer to the data location in which the handle of the selected address record (contact) is to be placed.
`field_handle`	If set, pointer to a data location in which the handle of the selected field of the selected address record is to be placed.
`search_string`	If set, initializes the address list view to include all names that match the given search string. If NULL, all Address Book entries are shown in the initial address list view.
`title`	If set, the title to be displayed in the address list view
`field_type`	If set, pointer to a data location in which the type of the selected field of the selected address record will be placed. If `force_field_type` is true, it points to the type of field that must be selected.

`field_data`	If set, points to a data buffer pointer, where the location of the selected field data of the selected address record is to be placed. **Note:** The application should copy the given `field_data` length into its own data space, because the data location that is returned can be reused at any time by the Address Book application.
`field_length`	If set, pointer to a data location in which the length of the selected field data is to be placed.
`force_field_type`	If true, the chosen method must match the type specified in the `field_type`. If false, any method can be selected.

Returns True if an address and communication method are selected; false otherwise. If true is returned, then the following data locations are set, if requested by non-NULL pointers:

- `address_handle`
- `field_handle`
- `field_type`
- `field_data`
- `field_length`

Description This function is deprecated; instead, your code should use `select_method_using`.

select_method_using

Selects the contact name and send methods.

```
bool select_method_using(
    AddressHandle * address_handle,
    AddressFieldHandle * field_handle,
    char * search_string = NULL,
    const char * title = NULL,
    AddressFieldtype * field_type = NULL,
    char ** field_data = NULL,
    int * field_length = NULL,
    unsigned int send_using_method_list = UINT_MAX)
```

Parameters

`address_handle`	Pointer to the data location where the handle of the selected address record (contact) is to be placed.
`field_handle`	If set, pointer to a data location where the handle of the selected field of the selected address record is to be placed.
`search_string`	If set, initializes the address list view to include all names that match the given search string. If NULL, all Address Book entries are shown in the initial address list view.
`title`	If set, the title to be displayed in the address list view.
`field_type`	If set, pointer to a data location where the type of the selected field of the selected address record is to be placed.
`field_data`	If set, points to a data buffer pointer, where the location of the selected field data of the selected address record is to be placed. **Note:** The application should copy the given `field_data` length into its own data space, since the data location returned may be reused at any time by the Address Book application.
`field_length`	If set, pointer to a data location where the length of the selected field data is to be placed.
`send_using_method_list`	A bitmask of acceptable communications methods. Construct the bitmask using methods defined in `<addressFieldType.h>`. Those values are bit-shift values, not the actual bits to be set. For example, to specify that both the EMAIL and FAX methods are acceptable, use this code: `acceptable_methods= (1<<EMAIL) + (1<<FAX)` The method actually used comes from a list created by ANDing the bitmask of acceptable communications methods and the bitmask of methods supported by the network services.

Returns True if an address and communication method are selected; false otherwise. If true is returned, then if the appropriate parameters were non-NULL pointers, the following data locations are set:

- `address_handle`
- `field_handle`

- `field_type`
- `field_data`
- `field_length`

Description `select_method_using` requires the user to select the contact name and acceptable contact methods.

The user can select **`[Use Once]`** when selecting a communication method. In this case, the data locations store these special values:

- `* address_handle` is set to a special value, `ONESHOT_HANDLE`
- `field_handle` is set to `INVALID_FIELD_HANDLE`
- `field_type`, `field_data`, and `field_length` are as with any other communication method.

select_name

Displays a list of names in the address book, which enables the user to select a specific address.

```
bool select_name(
    AddressHandle * address_handle,
    char * search_string = NULL,
    const char * title = NULL)
```

Parameters

`address_handle`	Pointer to the data location in which the handle of the selected address record (contact) will be placed.
`search_string`	If set, initializes the address list view to include all names that match the given search string. If NULL, all Address Book entries are shown in the initial address list view.
`title`	If set, the title to be displayed in the address list view.

Returns True if an address is selected; false otherwise. If true, the data area pointed at by `address_handle` is set with the handle of the selected address record.

Description This function displays the list view of Address Book contacts, and prompts the user to select a name from the list. After the user selects an address, control is returned to the calling application, with the `address_handle` parameter set. The application can then call `get_name` to retrieve the name of the selected contact.

If `search_string` contains multiple words separated by spaces, the Address Book application tries to match all of the words. Thus, if you provide a contact's initials separated by a space, all Address Book entries with those initials appear to the user. If, for example, `search_string` is "M S", only the address entries with those initials appear. Similarly, if it is "Ma St", only names matching both prefixes appear.

Index of functions

Index