Address Book Programming Guide for Mac OS X

Data Management: Contact Data

2010-08-03

Contents

Tables and Listings

Introduction

Address Book is a technology that encompasses a centralized database for contact and group information, an application for viewing that information, and a programmatic interface for accessing that information in your own applications. The database contains information such as user names, street addresses, email addresses, phone numbers, and distribution lists. Applications that use the Address Book framework can share this contact information with other applications, including Mail and iChat, or extend it to include application-specific information.

The Address Book framework provides two APIs: one for Objective-C and one for C. Both are equally functional, but the majority of the code samples in this document are printed in Objective-C only. Where it is appropriate, this document addresses fundamental differences between the two. Developers using the C programming interface should refer to "Using the Address Book C API" (page 33) and *Address Book C Framework Reference for Mac OS X* for information about mapping the Objective-C code to C.

Who Should Read This Document?

This document is designed for anyone who wants to leverage the abilities of the Mac OS X Address Book technology in their application. You should read it to learn how to access a user's address book, add new properties to the address book database, and create action plug-ins for the Address Book application.

It is expected that you are already familiar with Xcode and the basics of Mac OS X application development.

Note: Developers who have used the Address Book technology on iOS should be aware that the programming interface for this technology is different on Mac OS X.

Organization of This Document

The document contains the following articles:

- "About the Address Book" (page 9) describes what's in the Address Book database and what you can do with it.
- "Managing Address Book Records " (page 11) describes how to add and remove people and groups, how to arrange people into groups, and how to find the record for the logged-in user.
- "Accessing Address Book Records" (page 15) describes how to access data in a person or group record.
- "Searching an Address Book" (page 21) describes how to perform searches on a user's address book.
- "Using Address Book Groups as Distribution Lists" (page 25) describes how to set up a group so you can use it as a mailing list, or other type of distribution list.

- "Adding Properties to Address Book Records" (page 27) describes how to customize an address book for your own applications by adding properties to it.
- "Creating and Using Address Book Action Plug-ins" (page 29) describes how to create action plug-ins which allow users to perform custom actions on address book data viewed within the Address Book application.
- "Importing and Exporting Address Book People and Groups" (page 31) describes how to import and export person records by using the vCard standard.
- "Using the Address Book C API" (page 33) contains special information for those using the Address Book C API.

See Also

- *Sync Services Programming Guide* discusses the data synchronization engine built-in to Mac OS X.
- *Identity Services Programming Guide* discusses a way to manage groups of users on a local system, including standard login accounts and sharing accounts.

About the Address Book

The Address Book framework uses a centralized database for contact and other personal information for people. Users only need to enter this information once, instead of entering it repeatedly whenever it is used. Applications that support the Address Book framework share this contact information with other applications, including Apple's Mail and iChat. Every user on the computer has one and only one address book. Every application shares the address book for the currently logged-in user.

How Address Book Handles Individuals and Groups

The Address Book framework supports two fundamental kinds of records: `ABPerson`, for individuals, and `ABGroup`, for groups. Both are subclasses of the same root class, `ABRecord`, and they can be used interchangeably in some places.

An `ABPerson` record contains properties such as the person's name, company, addresses, email addresses, phone numbers, instant messaging IDs, and a comments field.

An `ABGroup` object can contain any number of people and other groups; a person can be in any number of groups. For example, suppose you are a consultant who works with two companies, Acme Co. and Ajax Inc. You could set up an Acme employees group and an Ajax employees group, and make each company's employees members of their respective group. You could then set up a Professionals group that includes the Acme group, the Ajax group, all well as some additional people who aren't in either group.

In addition, group and person records have these characteristics:

- **Each group and person has a unique identifier.** It's set when the record is created, and guaranteed never to change even if a user changes the group's or person's name or other information. Use this identifier if your application needs to store a reference to a group or person. For more information, see the `ABRecord` method `uniqueId`.
- **The groups and people are stored in an extensible form.** As such, you can add custom properties to Address Book records that other applications will ignore, without worrying about data corruption or usability issues. For more information, see "Adding Properties to Address Book Records" (page 27).
- **Some of these properties can contain multiple values.** For example, a person can have any number of street addresses, phone numbers, and email addresses. For more information, see "Using Multivalue Lists" (page 16).

How Address Book Manages Individual Search Queries

The Address Book framework manages individual search queries using `ABSearchElement` objects, which can be created using class methods of `ABGroup` and `ABPerson`. This has an important implication—because the search objects are created using the these particular classes, a custom subclass of `ABRecord` will not contain the required methods to create such an object. For this reason, you are advised not to subclass `ABRecord`.

For more information about searching Address Book records, see "Searching an Address Book" (page 21).

Other Features

The Address Book framework:

- **Provides transparent record locking.** If two applications try to change the same property within a record at the same time, the application that saved its change last will succeed. The database will not be corrupted. If two applications change different properties of the same record, both changes are expected to succeed.
- **Does not provide any security above what's provided by Mac OS X.** Anyone who has read and write access to a user's home folder can also read and write that user's address book. For that reason, the Address Book may not be an appropriate place to store confidential information, such as credit card numbers.
- **Provides localized versions of the built-in property names and labels.** If you add properties or labels, you must provide your own way for localizing them.
- **Syncs its records using Sync Services.** The Address Book framework syncs the data stored in the default properties using Sync Services; applications should not try to sync this data. If your application depends on custom properties being synced, it must sync them and may use Sync Services to do so. See "Adding Properties to Address Book Records" (page 27).

Important: Attempting to sync parts of the Address Book database other than custom properties using Sync Services is unpredictable and may result in data loss.

Managing Address Book Records

You can manage the people and groups within a user's address book. This article explains how to obtain the user's address book, add and remove people and groups from that address book, manage groups, find the record that corresponds to the logged-in user, and save your changes.

Accessing the Address Book

There are two ways to get a copy of the address book. The preferred way is to use the `ABAddressBook` method `addressBook`. The address book object that it returns should only be used on the same thread that it was created on, and can be used with the `ABPerson` method `initWithAddressBook:`.

If you're just making one-off lookups and edits, the `ABAddressBook` method `sharedAddressBook` may be used. However, this method can cause a significant decrease in performance, especially in a tight loop.

For example, you can replace code such as the following:

```
for (id item in someDataStructure) {
    ABPerson* person = [[ABPerson alloc] init];
    // Populate the person from the item
}
[[ABAddressBook sharedAddressBook] save];
```

With code like the following, yielding a significant performance increase:

```
ABAddressBook* tempBook = [ABAddressBook addressBook];
for (id item in someDataStructure) {
    ABPerson* person = [[ABPerson alloc] initWithAddressBook:tempBook];
    // Populate the person from the item
}
[tempBook save];
```

Adding and Removing Person and Group Records

The `ABAddressBook` class provides methods for accessing, adding, and removing group and person records. For example, use the `groups` method to get an array of all the group records in the database, or the `people` method to get all the person records.

Adding a new person or group record takes the following steps:

- Get the address book. The preferred way to do this is with the `ABAddressBook` method `addressBook`.
- Create the person or group record . You must allocate and initialize the respective `ABPerson` or `ABGroup` object. The preferred initializer is `initWithAddressBook:`.

- Add the record to the Address Book using the `ABAddressBook` method `addRecord:`.

To remove a person or group, use the `ABAddressBook` method `removeRecord:`.

Managing Groups

The Address Book framework lets you add people and subgroups to groups, as well as find out all groups that a person or subgroup is in.

To add and remove people from a group, use the `addMember:` and `removeMember:` methods. A person record can only be added to a group after it has been saved to the address book. To get a list of all the groups a person is in, use the `parentGroups` methods.

You can also add groups to a group. For example, a user could have a group called Pet Lovers that contains the groups Dog Lovers and Cat Lovers. To add and remove groups from another group, use the `addSubgroup:` and `removeSubgroup:` methods. You cannot create a cycle. For example, if Dog Lovers is a subgroup of Pet Lovers, then Pet Lovers cannot be a subgroup of Dog Lovers, directly or indirectly. To get a list of all groups that another group is a subgroup of, use the `parentGroups` methods.

To get lists of what's in a group, use the `members` and `subgroups` methods.

Accessing the User's Record

The currently logged-in user can specify a record that contains information about himself or herself. That lets your application find the name, address, or phone number of the user, so you can use it when filling out forms, for example. To get the logged-in user's record, use the `ABAddressBook` method `me`. To set the logged-in user's record, use the `ABAddressBook` `setMe:` methods.

Saving Your Changes

When you modify the Address Book database, those changes are made in memory, and not to the database itself. Unless you save those changes, they will be lost.

To save your changes to the database, use the `ABAddressBook` method `save` or `saveAndReturnError:`. To test whether there are unsaved changes, use the `ABAddressBook` method `hasUnsavedChanges`.

Notification of Changes

The Address Book posts notifications if any application, including your own, makes changes to the database. Typically, you observe these notifications to update any dependent view or model objects in your application. The Address Book framework sends two notifications: `kABDatabaseChangedNotification` to indicate that the current process has made a change, and `kABDatabaseChangedExternallyNotification` to indicate that another process has made a change. Use NSNotificationCenter to register for the notifications you are interested in. Note that these notifications are not sent until after the sharedAddressBook method of the ABAddressBook class or the C function ABGetSharedAddressBook has been invoked.

If your application is using the shared address book object (returned by the sharedAddressBook method), the changes have already been merged in automatically and are available immediately when you receive the change notification. Non-shared address book objects (returned by theaddressBook method) are generally used only for short time, and do not process change notifications automatically.

An Example

This Objective-C example adds a person named John Doe to the current user's address book. Take note of how the code accesses the shared address book and how it allocates a new ABPerson object. Also note the properties used (in this case, just first name and last name), and the final save, which sends the changes to the user's address book:

```
ABAddressBook *addressBook;
ABPerson *newPerson;

addressBook = [ABAddressBook sharedAddressBook];

newPerson = [[[ABPerson alloc] init] autorelease];

[newPerson setValue:@"John"
        forProperty:kABFirstNameProperty];

[newPerson setValue:@"Doe"
        forProperty:kABLastNameProperty];

[addressBook addRecord:newPerson];
[addressBook save];
```

Accessing Address Book Records

After you have a record, you can retrieve the data within it. This chapter shows you how the data is organized and how to access it. It shows how to access properties from a records property list, how to handle properties that can have more than one value (such as addresses and phone numbers), how to get localized names for properties and labels, and how to associate a picture with a person.

Using Property Lists

Both groups and people store their data in property lists. This lets your application add properties to the Address Book records that other applications will ignore. See "Adding Properties to Address Book Records" (page 27).

- To get data from a record, such as a group's description or a person's first name, use the `valueForProperty:` method or the C function `ABRecordCopyValue`. For example, to get the first name for `aPerson`, use:

  ```
  [aPerson valueForProperty:kABFirstNameProperty];
  ```

- To set data, use the `setValue:forProperty:` method. For example, to set the name of `aGroup`, use:

  ```
  [aGroup setValue:@"Book Club" forProperty:kABGroupNameProperty];
  ```

- To find the names of the default properties, refer to Table 1.

Table 1 The documentation for property-list constants

Documentation	Class	Language
"Default Record Properties" in *Address Book Objective-C Constants Reference*	`ABRecord`	Objective-C
"Default Person Properties" in *Address Book Objective-C Constants Reference*	`ABPerson`	Objective-C
"Default Group Properties" in *Address Book Objective-C Constants Reference*	`ABGroup`	Objective-C
"Constants" in *ABPerson C Reference*	`ABPersonRef`	Procedural C
"Constants" in *ABGroup C Reference*	`ABGroupRef`	Procedural C

Using Multivalue Lists

Many properties can have multiple values. For example, a person can have several addresses, including work, home, summer home, and mailing addresses. These properties are stored as multivalue lists, of type `ABMultiValue`. Each item in a multivalue list has a numeric index, a unique identifier, a string label (such as Home or Work), and a value. Every value in the multivalue list must be of the same type. The label does not need to be unique; after all, someone could have more than one home or work address.

- To add an item to a multivalue list, use the `addValue:withLabel:` or `insertValue:withLabel:atIndex:` method.
- To retrieve an item, use the `valueAtIndex:` or `labelAtIndex:` method.
- To remove an entry in a multivalue list, use the `removeValueAndLabelAtIndex:` method.
- To replace values and labels, use the `replaceLabelAtIndex:withLabel:` or `replaceValueAtIndex:withValue:` methods.

You use the numeric index to access items in a multivalue list, but these indices may change as the user adds and removes values. If you want to save a reference to a specific value, use the unique identifier, which is guaranteed not to change. You can convert back and forth between indices and unique identifiers:

- To get the unique identifier for a value at a particular index, use the `identifierAtIndex:` method.
- To get the index for a identifier, use the `indexForIdentifier:` method.

Each multivalue list also has a primary value, which is the item the user most strongly associates with that person. For example, friends may have both home and work addresses, but the home address is their primary address. And coworkers may have both home and work phone numbers, but the work number is their primary number.

- To get the identifier for a multivalue list's primary value, use the `primaryIdentifier` method.
- To set the multivalue list's primary value, use the `setPrimaryIdentifier:` method.

The Picture Associated with a Person

A person may also have an associated picture or image. The image is not actually stored in the Address Book database (a property list)—it's stored in a separate image file. This means you need to use different methods to access the image data. You can set a person's image using the `setImageData:` method, or get an image using the `imageData` method. Use the `NSImage initWithData:` method to convert the `NSData` object returned by the `imageData` method to an `NSImage` object.

The Address Book framework locates images through a specific search hierarchy, in this order:

1. Check for an image set specifically by the user.
2. Check Directory Services for the local user's login picture.
3. Check for an image in `/Network/Library/Images/People/`*email*, where *email* is the user's primary email address.

4. Check for an image from the user's MobileMe account, first against the cache at `~/Library/Caches/com.apple.AddressBook/`*email* and then against the MobileMe servers.

Image files may be local or remote. Local images are any images in `.../Library/Images/People` and any images the user has set using the Address Book application. Remote images are images stored on the network. Remote images take time to download, so an asynchronous API for fetching remote images is provided.

Use the `beginLoadingImageDataForClient:` method if an image file is not local and you want to perform an asynchronous fetch. You pass a client object that implements the `ABImageClient` protocol as an argument to this method. The `beginLoadingImageDataForClient:` method returns an image tracking number. A `consumeImageData:forTag:` message is sent to your client object when the fetch is done. Implement this method to handle the new fetched image. Use the `cancelLoadingImageDataForTag:` class method if you want to cancel an asynchronous fetch.

Use the `beginLoadingImageDataForClient:` method if an image file is not local and you want to perform an asynchronous fetch:

1. Pass a client object that implements the `ABImageClient` protocol as an argument to this method.
2. The `beginLoadingImageDataForClient:` method returns an image tracking number.
3. A `consumeImageData:forTag:` message is sent to your client object when the fetch is done. Implement this method to handle the new fetched image.
4. Use the `cancelLoadingImageDataForTag:` class method if you want to cancel an asynchronous fetch.

Getting Localized Names for Properties and Labels

You can find the localized name for any of the default property names and labels that are listed in *Address Book Objective-C Constants Reference*. The functions `ABLocalizedPropertyOrLabel` and `ABCopyLocalizedPropertyOrLabel` returned a name that is localized for the user's selected language.

You must handle the localization of the names for the properties and labels you create; the Address Book framework does not provide any specific support for this.

An Example

Listing 1 is an Objective-C code sample that retrieves the country for the primary address of the logged-in user. If the country is a null string, it sets the country to USA.

Listing 1 Changing a person's address

```
ABPerson *aPerson = [[ABAddressBook sharedAddressBook] me];
ABMutableMultiValue *anAddressList =
    [[aPerson valueForProperty:kABAddressProperty] mutableCopy];
NSUInteger primaryIndex =
    [anAddressList indexForIdentifier:[anAddressList primaryIdentifier]];
NSMutableDictionary *anAddress =
    [[anAddressList valueAtIndex:primaryIndex] mutableCopy];
NSString *country =
    (NSString*) [anAddress objectForKey:kABAddressCountryKey];
if ([country isEqualToString:@""]) {
    [anAddress setObject:@"USA" forKey:kABAddressCountryKey];
    [anAddressList replaceValueAtIndex:primaryIndex withValue:anAddress];
    [aPerson setValue:anAddressList forProperty:kABAddressProperty];
    [[ABAddressBook sharedAddressBook] save];
}
```

Using the People Picker

The people picker is a view that provides access to the contents of a user's address book from any application. It offers a searchable, selectable list of people and groups that can be customized for your application.

To use a people picker in a Cocoa application, drag it from the library palette into your window. If you need to, you can create an instance of `ABPeoplePickerView` programmatically instead. There are two ways to set up the behavior of the people picker view—from Interface Builder and programmatically.

To set up the behavior from Interface Builder, open the Get Info panel. The attributes you can set include which columns are displayed by the view, whether or not multiple selections are allowed, whether or not group selections are allowed, and the autosave name for the view. These changes are reflected immediately within Interface Builder. Using the Test Interface feature of Interface Builder, a people picker view will display the address book of the logged-in user.

To set up the behavior programmatically, create an outlet of the type `ABPeoplePickerView` from your controller and connect it to the people picker view. Then use the appropriate instance methods to change attributes of the picker.

For Cocoa applications, the people picker also provides methods for using autosave data, so that it can retain the filter selections and column positions. Use the `ABPeoplePickerView` methods `autosaveName` and `autosaveName`.

Using the C API, the people picker is available only in a window form and cannot be used as a custom view. It must be created with `ABPickerCreate` and made visible with `ABPickerSetVisibility`, as follows:

```
ABPickerRef peoplePicker = ABPickerCreate();
ABPickerSetVisibility(peoplePicker, TRUE);
```

People Picker Example

The code from this example should be placed in the window controller for the people picker. Developers using the C API should refer to the *ABPicker Reference for C* to construct the analogue for their applications; most of the functions are named similarly. Instead of using notifications, you will need to register event handlers to handle changes to the window.

The following listing shows how you can set the attributes of a people picker programmatically. This code is typically part of the `awakeFromNib` method. All of these settings are also available in the attributes inspector, in Interface Builder.

```
    // Disallow multiple selections in the name list.
    [peoplePicker setAllowsMultipleSelection:NO];

    // Add the e-mail and telephone properties to the view.
    // By default, the people picker displays only the
    // Name column.
    [peoplePicker addProperty:kABEmailProperty];
    [peoplePicker addProperty:kABPhoneProperty];
```

The following listing shows how to register for a notification when the user selects a person record in the people picker. This code is typically part of the `awakeFromNib` method.

```
    NSNotificationCenter* center;
    center = [NSNotificationCenter defaultCenter];

    // Set up a responder for one of the four available notifications,
    // in this case to tell us when the selection in the people picker
    // has changed.
    [center addObserver:self
            selector:@selector(recordChanged:)
            name:ABPeoplePickerNameSelectionDidChangeNotification
            object:peoplePicker];
```

The following listing shows an example of how to respond to the user selecting a person record in the people picker:

```
- (void)recordChanged:(NSNotification*)notification {
    NSArray *array;
    NSImage *personImage;
    NSString *personFirstName;
    NSString *personLastName;

    array = [peoplePicker selectedRecords];
    NSAssert([array count] == 1,
             @"Picker returned multiple selected records");
    ABPerson *person = [array objectAtIndex:0];

    personImage = [[NSImage alloc] initWithData:[person imageData]];
    personFirstName = [person valueForProperty:kABFirstNameProperty],
    personLastName = [person valueForProperty:kABLastNameProperty];

    /* ...do something with the image and name... */

    [personImage release];
```

Searching an Address Book

You can quickly search a user's address book, using arbitrarily complex criteria. For example, you can search for all people named Smith, or for all people who work at Acme and live in San Francisco, or for all people who work at Ajax and live in Seattle.

To perform the search, you encapsulate the criteria in a search element to pass it to the Address Book framework. The framework performs the search on your behalf, and returns the results. Letting the framework handle the search can yield significant performance benefits compared to performing the search inside your application, because the framework is aware of the low-level layout of the underlying database, and it can optimize disk access accordingly.

Note: You can only seach the user's local Address Book database, not remote directories such as CardDAV or Exchange.

Creating a Search Element for a Single Property

To create a search element for a person, use the `ABPerson` class method `searchElementForProperty:label:key:value:comparison:`. To create a search element for a group, use the `ABGroup` class method `searchElementForProperty:label:key:value:comparison:`.

If you want to search for people or groups that have a particular property set, regardless of the value it is set to, pass `nil` as the value and `kABNotEqual` as the comparison. To search for people or groups that do not have a property set, pass `nil` as the value and `kABEqual` as the comparison.

Creating a Search Element for Multiple Properties

To combine search elements, use the `ABSearchElement` class method `searchElementForConjunction:children:`. This method takes two arguments:

- *`conjunctionOperator`* describes how to combine the search elements. It can be `kABSearchAnd` or `kABSearchOr`.
- *`children`* is an NSArray of search elements. The search elements can be a simple elements that specifies only one property, or complex elements that specifies several. This lets you create arbitrarily complex search elements. You cannot combine search elements for groups with search elements for people.

Finding Records That Match a Search Element

To search the address book for records that match a search element, use the `ABAddressBook` method `recordsMatchingSearchElement:`, which returns an `NSArray` of records. Use the `ABSearchElement` method `matchesRecord:` to test whether a specific record matches a query.

Search Examples

Listing 1 shows the code to find everyone whose last name is Smith.

Listing 1 A simple search

```
ABAddressBook *AB = [ABAddressBook sharedAddressBook];
ABSearchElement *nameIsSmith =
    [ABPerson searchElementForProperty:kABLastNameProperty
                                 label:nil
                                   key:nil
                                 value:@"Smith"
                            comparison:kABEqualCaseInsensitive];
NSArray *peopleFound =
    [AB recordsMatchingSearchElement:nameIsSmith];
```

Listing 2 shows the code to find everyone who lives in San Francisco and works for Acme, or who lives in Seattle and works for Ajax. Note that the addresses are searched using the `kABHomeLabel` label—we only want to know if they live in the city we are searching, not if they work in the same city.

Listing 2 A complex search

```
ABAddressBook *AB = [ABAddressBook sharedAddressBook];
ABSearchElement *inSF =
    [ABPerson searchElementForProperty:kABAddressProperty
                                 label:kABHomeLabel
                                   key:kABAddressCityKey
                                 value:@"San Francisco"
                            comparison:kABEqualCaseInsensitive];
ABSearchElement *atAcme =
    [ABPerson searchElementForProperty:kABOrganizationProperty
                                 label:nil
                                   key:nil
                                 value:@"Acme"
                            comparison:kABContainsSubStringCaseInsensitive];
ABSearchElement *inSeattle =
    [ABPerson searchElementForProperty:kABAddressProperty
                                 label:kABHomeLabel
                                   key:kABAddressCityKey
                                 value:@"Seattle"
                            comparison:kABEqualCaseInsensitive];
ABSearchElement *atAjax =
    [ABPerson searchElementForProperty:kABOrganizationProperty
                                 label:nil
                                   key:nil
                                 value:@"Ajax"
                            comparison:kABContainsSubStringCaseInsensitive];
ABSearchElement *inSFAndAtAcme =
    [ABSearchElement searchElementForConjunction:kABSearchAnd
                                        children:[NSArray arrayWithObjects:
                                            inSF, atAcme, nil]];
ABSearchElement *inSeattleAndAtAjax =
    [ABSearchElement searchElementForConjunction:kABSearchAnd
                                        children:[NSArray arrayWithObjects:
                                            inSeattle, atAjax, nil]];
ABSearchElement *inSFAndAtAcmeOrInSeattleAndAtAjax =
    [ABSearchElement searchElementForConjunction:kABSearchOr
                                        children:[NSArray arrayWithObjects:
                                            inSFAndAtAcme, inSeattleAndAtAjax,
 nil]];
NSArray *peopleFound =
    [AB recordsMatchingSearchElement:inSFAndAtAcmeOrInSeattleAndAtAjax];
```

Using Address Book Groups as Distribution Lists

An Address Book group can be used as a distribution list. For example, suppose you lead a book discussion club on the weekends. You can use a group to keep a list of all the people in the group. For multivalue properties such as telephone number and email address, the distribution list lets the you indicate which value should be used when sending a message to this group.

Generally, users will want to use the value marked as primary as the distribution identifier, but in some cases they will want to make an exception. For example, for coworkers, their primary email addresses are probably their work address. But messages about your weekend book club should be sent to their home addresses. This is accomplished by setting distribution identifier for the Book Club group to their home addresses.

To choose the value of a multivalue property that a group should use, use the `ABGroup` method `setDistributionIdentifier:forProperty:person:`. Every group can use a different value for each person. Users can also edit this from the Address Book application, by selecting Edit Distribution List from the Edit menu.

To get a group's chosen value for a multivalue property, use the `ABGroup` method `distributionIdentifierForProperty:person:`. If a distribution identifier is not set, this method returns the multivalue's primary identifier. If either the property or the person is `nil`, the method returns `nil`. the method also returns `nil` if the property is not a multivalue list property, or if the person is not a member of the group. Use the `ABMultiValue` method `valueForIdentifier:` to get the value from with the distribution identifier.

Adding Properties to Address Book Records

You can add your own properties to the people and groups in the address book. For example, if you're creating a small application to manage a dog club, you could add properties to each person that specify the name and breed of that person's dog. Or if you're creating an application to manage business contacts, you could add a property that lists all the meetings and phone calls a user has had with that person. These properties are stored in the Address Book database. Applications that don't know about the new properties aren't affected by them and don't modify them.

Note: Data stored in custom properties are not synced across MobileMe. If your application depends on this data being synced on multiple computers, it needs to sync the data itself. See *Sync Services Programming Guide* for more details on syncing.

Do not sync any of the data stored in the default properties. The Address Book framework already syncs this data, and attempting to sync it from your application may lead to data loss.

When deciding whether to add a property to the Address Book record, keep these issues in mind:

- Avoid properties for confidential information, such as credit card numbers. The Address Book framework does not provide any security above what's provided by Mac OS X. Anyone who has read and write access to a user's home folder can also read and write that user's address book.
- Avoid properties that are not useful for everyone in the address book database. If you want to store information for just the logged-in user, for Cocoa applications refer to *NSUserDefaults Class Reference*, and for C-based applications refer to *Preferences Utilities Reference*.
- Use a multivalue list if you think a person may have more than one of that property. Your new multivalue list has the same capabilities as the other multivalue lists in the address book. The user can choose a primary value in the list and can create distribution lists for it.

To add properties to every person or group, use the `ABPerson` or `ABGroup` class method `addPropertiesAndTypes:`. These procedures take a dictionary, in which the keys are the names of the new properties and the values are their types. Note that the property names must be unique. You may want to use reverse-DNS style names for your properties, to make sure no one else uses the same name; for example, `org.dogclub.dogname` or `com.mycompany.buildingNumber`. The type can be one of the types or a multivalue list of one of the types listed in "Property Types" in *Address Book Objective-C Constants Reference*.

The following code listing adds a custom property, and then removes it:

```
NSNumber* stringProperty = [NSNumber numberWithInteger:kABStringProperty];
NSString* testProperty = @"com.apple.devpubs.testProperty";
NSDictionary* dict = [NSDictionary dictionaryWithObject:stringProperty
                                                 forKey:testProperty];

NSInteger result = [ABPerson addPropertiesAndTypes:dict];
NSLog(@"Added %d properties.", result);

result = [ABPerson removeProperties:[NSArray arrayWithObject:testProperty]];
NSLog(@"Removed %d properties.", result);
```

Creating and Using Address Book Action Plug-ins

A unique aspect of the Address Book application is its ability to act on data contained within a person's card. You can install your own custom plug-ins to add additional actions to a given record. An example of an existing action is the Large Type action, which works on any phone number entry. When selected from its contextual menu, it displays the number in large type across the screen.

Each action plug-in can implement only one action. Actions can only apply to items with labels. An action can display a simple window in the Address Book application. If your action actions needs to do anything else, it should launch your own application to perform the action.

The `ABActionDelegate` protocol, which must be followed for the Address Book application to recognize the plug-in, is summarized in Table 1. See *ABActionDelegate Protocol Reference* for full details. C-based actions must implement a function named `ABActionRegisterCallbacks`, as described in *Address Book Actions Reference*.

Table 1 Action methods for an Address Book action plug-in

Method	Purpose
`actionProperty`	Returns the `NSString` constant identifying the property that the action applies to.
`titleForPerson: identifier:`	Returns the title of the menu item for the action. This method should not return `nil`.
`performActionForPerson: identifier:`	Performs the appropriate action for the plug-in. Each plug-in can only have one action.
`shouldEnableActionForPerson: identifier:`	Returns `YES` if the action is applicable and `NO` otherwise. This allows your plug-in to enable and disable its menu item. (Optional.)

To create a plug-in, use the Address Book action plug-in template from the Xcode New Project window. The template creates an action plug-in designed to create a contextual menu item on any phone number. When the menu item is selected, the sample plug-in uses Mac OS X's speech synthesis framework to speak the phone number. Replace this sample code with the code you need for your new plug-in. After you build your project, place the completed bundle in `.../Library/Address Book Plug-Ins`.

After an action plug-in has been loaded, its menu item is displayed in the contextual menu with the title returned from the `titleForPerson:identifier:` method; this method should not return `nil`. The plug-in can enable and disable this menu item using the `shouldEnableActionForPerson:identifier:` method.

Importing and Exporting Person and Group Records

You can import and export person records using the vCard format. To create a vCard representation of a person, use the `ABPerson` method `vCardRepresentation`. This method creates an `NSData` structure that you can use in your program or save to a file. To enable drag and drop for this data, use a file promise as described in "Dragging Files".

To create a person record from a vCard representation, use the `ABPerson` method `initWithVCardRepresentation:`.

Using the Address Book C API

For the most part, the Objective-C API has close method and syntax parity with the C API. This makes it easy to determine, for example, which function corresponds to a given Objective-C method.

There are a couple of primary differences that developers need to be aware of when using the Address Book C API:

- The people picker comes only in window form and does not have a C API for setting an accessory view. In addition, changes in selection and displayed properties are sent via Carbon Events.
- When creating a C-based action plug-in, your bundle must implement a function called `ABActionRegisterCallbacks`, which will return an `ABActionCallbacks` structure. The structure needs to be formed according to this type definition:

```
typedef struct {
    // The version of this struct is 0
    CFIndex                     version;

    // A pointer to a function that returns the AddressBook
    // property this action applies to.
    ABActionPropertyCallback     property;

    // A pointer to a function that returns the AddressBook
    // property this action applies to. Only items with labels
    // may have actions at this time.
    ABActionTitleCallback        title;

    // A pointer to a function which returns YES if the action
    // should be enabled for the passed ABPersonRef and item
    // identifier. The item identifier will be NULL for single value
    // properties. This field may be NULL. Actions with NULL enabled
    // callbacks will always be enabled.
    ABActionEnabledCallback      enabled;

    // A pointer to a function which will be called when the user
    // selects this action. It's passed an ABPersonRef and item
    // identifier. The item identifier will be NULL for single
    // value properties.
    ABActionSelectedCallback     selected;

} ABActionCallbacks
```

To access the user’s shared address book using the C programming interface, you need to use the value returned by `ABGetSharedAddressBook`.

```
ABAddressBookRef addressBook = ABGetSharedAddressBook();
```

Compare this with the corresponding code using the Objective-C programming interface, noting the mapping between corresponding method and function names.

```
ABAddressBook *addressBook = [ABAddressBook sharedAddressBook];
```

This example from "Searching an Address Book" (page 21), Listing 1, searches for anyone named Smith in the current user's address book and returns an array of results.

Listing 1 A simple search, in Objective-C

```
ABAddressBook *AB = [ABAddressBook sharedAddressBook];

ABSearchElement *nameIsSmith =
    [ABPerson searchElementForProperty:kABLastNameProperty
                                 label:nil
                                   key:nil
                                 value:@"Smith"
                            comparison:kABEqualCaseInsensitive];

NSArray *peopleFound =
    [AB recordsMatchingSearchElement:nameIsSmith];
```

Listing 2 performs the same search using the C API. Again, note the mapping between corresponding method and function names.

Listing 2 A simple search, in C

```
ABAddressBookRef AB = ABGetSharedAddressBook();

ABSearchElementRef nameIsSmith =
    ABPersonCreateSearchElement(kABLastNameProperty,
                    NULL,
                    NULL,
                    CFSTR("Smith"),
                    kABEqualCaseInsensitive);

CFArrayRef peopleFound =
    ABCopyArrayOfMatchingRecords(AB, nameIsSmith);
```

For more details about using the C API for the Address Book framework, refer to *Address Book C Framework Reference for Mac OS X*.

Document Revision History

This table describes the changes to *Address Book Programming Guide for Mac OS X*.

Date	Notes
2010-08-03	Minor editorial changes throughout.
2010-02-24	Updated the sections Using the People Picker, Searching an Address Book, Using Address Book Groups as Distribution Lists, and Creating and Using Address Book Action Plug-in. Added sample code to Adding Properties to Address Book Records. Formatting and editorial changes throughout.
2009-08-07	Minor changes throughout.
2009-05-28	Made minor updates to searching and drag-and-drop exporting.
2006-04-04	Made minor editorial corrections throughout.
2005-04-29	Made minor sample code changes. Added important note about syncing with Sync Services to the introduction.
2004-04-21	Added major updates. New sections include "Creating and Using Address Book Action Plug-ins" (page 29), "Using the People Picker" (page 18), and "Using the Address Book C API" (page 33) for Carbon developers. Other sections have new sample code and more detailed content.
2003-10-30	Made minor correction in "Search Examples" (page 22).
2003-08-21	Added revision history, which records changes to the content of *Address Book*.