Inside Mac OS X: Address Book

The Address Book is a centralized database for contact and other personal information for people. Applications that support the Address Book framework share this contact information with other applications, include Apple's Mail and iChat. Both Carbon and Cocoa applications can access it.

Discussion

This concept describes gives an overview of the Address Book framework:

- "About the Address Book" (page 3) describes what's in the Address Book database and what you can do with it.

These tasks describe what you can do with the Address Book database:

- "Managing Address Book Records" (page 5) describes how to add and remove people and groups, how to arrange people into groups, and how to find the record for logged in user.
- "Accessing What's in Address Book Records" (page 9) describes how to access data in a person or group record.
- "Searching the Address Book" (page 13) describes how to perform searches on the Address Book database.
- "Using Address Book Groups As Distribution Lists" (page 19) 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 21) describes how to customize the Address Book database for your own applications by adding properties to it.
- "Importing and Exporting Address Book People" (page 23) describes how to import and export person records by using the vCard standard.

About the Address Book

The Address Book is a centralized database for contact and other personal information for people. Users need to enter personal information about themselves and their friends only once, instead of entering it repeatedly whenever the information is used. Applications that support the Address Book framework share this contact information with other applications, include Apple's Mail and iChat. Both Carbon and Cocoa applications can access it.

This database contains people's names, street addresses, email addresses, phone numbers, home pages, and more. Your applications can use this data as it is, or extend it to include information specific to your applications. Every user on the computer has one and only one address book. Every application shares the address book for the currently logged-in user.

The API also provides the concept of a "Me" record. This records contains contact information for the currently logged-in user. Applications such as web browsers can use it to fill in forms automatically.

The Address Book contains two basic sorts of records: ABPerson and ABGroup. Both are subclasses of the same root class ABRecord, and they can be used interchangeably in some places.

A person's record contains such properties as the person's name, company, addresses, email addresses, phone numbers, instant messaging IDs, and a comments field. Some of these properties can contain multiple values. For example, a person can have any number of street addresses, phone numbers, and email addresses. A user can also specify that one of those multiple values is the primary value. For example, a user can specify that email t should be sent to one person's work address and another person's home address, unless otherwise specified.

A group can contain any number of people and other groups For example, you could have an Acme employees group, an Ajax employees group, and a

Professional group that includes the Acme group, the Ajax group, and some people that aren't in any other group. A person can be in any number of groups. A group also contains a name.

The groups and people are stored in an extensible. That means you can add properties to the Address Book records that other applications will ignore.

Each group and person has a unique identifier that's set when the record is created. It's 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.

The Address Book framework provides transparent record locking. If two applications try to change the same record simultaneously, the application that tried to change it last will succeed. The database will not be corrupted.

The Address Book 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.

The Address Book API 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.

Managing Address Book Records

You can manage the people and groups within a user's Address Book. This document explains how get 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

To get the address book for the currently logged-in user, use the ABAddressBook method `sharedAddressBook` or the function `ABGetSharedAddressBook`. If you call these procedures more than once or try to create a new address book, you get a pointer to the shared address book.

Adding and Removing People and Groups

Adding a new person or group takes two steps: creating it and then adding it to the Address Book.

First, create the person or group. If you're using Objective-C, allocate and initialize the person or group in the usual way. If you're using C, use the function `ABPersonCreate` or `ABGroupCreate`.

Second, add the person or group to the Address Book. If you're using Objective-C, use the ABAddressBook method `addRecord:`. If you're using C, use the function `ABAddRecord`.

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

Managing Groups

The Address Book 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 methods `addMember:` and `removeMember:` or the functions `ABGroupAddMember` and `ABGroupRemoveMember`. To get a list of all the groups a person is in, use the method `parentGroups` or the function `ABPersonCopyParentGroups`.

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 methods `addSubgroup:` and `removeSubgroup:` or the functions `ABGroupAddGroup` and `ABGroupRemoveGroup`. The Address Book will not let you create a circular dependency. For example, if "Dog Lovers" is a subgroup of "Pet Lovers," then "Pet Lovers" cannot be a subgroup of "Dog Lovers." To get a list of all groups that another group is a subgroup of, use the method `parentGroups` or the function `ABGroupCopyParentGroup`.

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

Accessing the User's Record

The 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 logged-in 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` or the function `ABGetMe`. To set the logged-in user's record, use the ABAddressBook method `setMe:` or the function `ABSetMe`.

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 address book, use the ABAddressBook method `save` or the function `ABSave`. To test whether there are unsaved changes to the address book, use the ABAddressBook method `hasUnsavedChanges` or the function `ABHasUnsavedChanges`.

Accessing What's in Address Book Records

Once you have a record, you can retrieve the data within it. This document 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 21).

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

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

or

```
ABRecordCopyValue(aPerson, kABFirstNameProperty);
```

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

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

or

```
ABRecordSetValue(aGroup, CFSTR("Book Club"), kABGroupNameProperty);
```

To find out the names of the properties, look at these header files in the Address Book framework: `ABGlobals.h` for Cocoa and `ABGlobalsC.h` for Carbon.

Using Multiple-Value 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 multi-value lists, of type ABMultiValue or ABMultiValueRef. Each item in a multi-value list has a numeric index, a unique identifier, a string label (such as "Home" or "Work"), and a value. Every value in the multi-value 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. You access the items with the numeric index. To add an item to a multi-value list, use the method `addValue:withLabel:` or the function `ABMultiValueAdd`. To retrieve an item, use the methods `valueAtIndex:` and `labelAtIndex:` or the functions `ABMultiValueCopyValueAtIndex` and `ABMultiValueCopyLabelAtIndex`.

If you want to save a reference to a specific value, use the unique identifier. The numeric index may change as the user adds and removes values, but the identifier is guaranteed never to change. To get the unique identifier for a value at a particular index, use the method `identifierAtIndex:` or the function `ABMultiValueCopyIdentifierAtIndex`. To get the index for a identifier, use the method `indexForIdentifier:` or the function `ABMultiValueIndexForIdentifier`.

Each multi-value 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 multi-value list's primary value, use the method `primaryIdentifier` or the function `ABMultiValueCopyPrimaryIdentifier`. To set the multi-value list's primate value, use the method `setPrimaryIdentifier:` the function `ABMultiValueSetPrimaryIdentifier`.

Associating a Picture With a Person

You can associate a picture that identifies a person in your Address Book database. Because these pictures are not stored in the same way as the other Address Book, you need to use different methods to access them.

To associate a TIFF to a person, use the method `setImageData:` or the function `ABPersonSetImageData`. To get the TIFF data for person's image, use the method `imageData` or the function `ABPersonCopyImageData`. Note that both NSImage and QuickTime have functions for converting the data into a TIFF file.

Getting Localized Names for Properties and Labels

You can find the localized name for any of the property names and labels that are in the header files `ABGlobals.h` and `ABGlobalsC.h`. The function `ABCopyLocalizedPropertyOrLabel` returned a name that's localized for the user's selected language.

If you want to localize names for the properties and labels that you create, you must handle it yourself. The Address Book framework does not have support for this.

An Example

Listing 1 (page 12) 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, in Objective-C

```
ABPerson *aPerson = [[ABAddressBook sharedAddressBook] me];
ABMutableMultiValue *anAddressList =
    [[aPerson valueForProperty:kABAddressProperty] mutableCopy];
int 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];
}
```

Listing 2 (page 12) does the same thing in C.

Listing 2 Changing a Person's Address, in C

```
ABPersonRef me = ABGetMe(ABGetSharedAddressBook());
ABMutableMultiValueRef anAddressList =
   ABMultiValueCreateMutableCopy(ABRecordCopyValue(me, kABAddressProperty));
int primaryIndex = ABMultiValueIndexForIdentifier(anAddressList,
    ABMultiValueCopyPrimaryIdentifier(anAddressList));
CFMutableDictionaryRef anAddress =
    CFDictionaryCreateMutableCopy(NULL, 0,
        ABMultiValueCopyValueAtIndex(anAddressList, primaryIndex));
CFStringRef country =
    (CFStringRef) CFDictionaryGetValue(anAddress, kABAddressCountryKey);
if (CFStringCompare(country, CFSTR(""), NULL) == kCFCompareEqualTo) {
    CFDictionarySetValue(anAddress, kABAddressCountryKey, CFSTR("USA"));
    ABMultiValueReplaceValue(anAddressList, anAddress,
        primaryIndex);
    ABRecordSetValue(me, kABAddressProperty, anAddressList);
}
```

Searching the Address Book

You can quickly search a user's address book, using arbitrarily complex criterion. The address book is fully indexed, automatically. And the Address Book framework's searching features let you easily create complex searches. For example, you can search for all people named "Smith," or for all people who work at Apple and live in California or who work at Microsoft and live in Washington.

Here's how you create a search query. For each property you want to search on, create a search element. If you want to search on multiple properties, combine the search elements together with boolean operators to create a complex search element. Then search the Address Book with that simple or complex search element.

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:` or the function `ABPersonCreateSearchElement`. To create a search element for a group, use the ABGroup class method `searchElementForProperty:label:key:value:comparison:` or the function `ABGroupCreateSearchElement`. These procedures take the following arguments:

- *property* is the name of the property to search on, such as `kABAddressProperty` or `kABLastNameProperty`. It cannot be `nil`. For a full list of the properties, see `ABGlobals.h` or `ABGlobalsC.h`.
- *label* is the label name for a multi-value list, such as `kABAdressHomeLabel`, `kABPhoneWorkLabel`, or a user-specified label, such as `"Summer Home"`. If the

specified property does not have multiple values, pass `nil`. If the specified property does have multiple values, pass `nil` to search all the values.

- *key* is the key name for a dictionary, such as `kABAddressCityKey` or `kABAddressStreetKey`. If the specified property is not a dictionary, pass `nil`. If the specified property is a dictionary, pass `nil` to search all keys.
- *value* is what you're searching for. It cannot be `nil`.
- *comparison* specifies the type of comparison to perform. You can choose
  - To search for elements that are equal or not equal to the value, use `kABEqual`, `kABNotEqual`, or `kABEqualCaseInsensitive`
  - To search for elements that are less than or greater than the value, use `kABLessThan`, `kABLessThanOrEqual`, `kABGreaterThan`, or `kABGreaterThanOrEqual`.
  - To search for elements `kABContainsSubString`, `kABContainsSubStringCaseInsensitive`, `kABPrefixMatch`, or `kABPrefixMatchCaseInsensitive`.

Creating a Search Element for Multiple Properties

To combine search elements, use the ABSearchElement class method `searchElementForConjunction:children:` or the function `ABSearchElementCreateWithConjunction`. These procedures take two arguments:

- *conjunctionOperator* describes how to combine the search elements. It can be `kABSearchAnd` or `kABSearchOr`.
- *children* is a NSArray or CFArray 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:` or the function `ABCopyArrayOfMatchingRecords`. These procedures return an NSArray or CFArray of records.

Two Examples

Listing 3 (page 15) a simple search in Objective-C. This code finds all the people whose last name is "Smith."

Listing 3 Simple Search, in Objective-C

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

Listing 4 (page 15) shows the same simple search in C.

Listing 4 Simple Search, in C

```
ABSearchElementRef nameIsSmith =
    ABPersonCreateSearchElement(kABLastNameProperty, NULL,
                                NULL, CFSTR("Smith"),
                                kABEqualCaseInsensitive);
CFArrayRef peopleFound =
    ABCreateArrayOfMatchingRecords(AB, nameIsSmith);
```

Listing 5 (page 16) is a complex search in Objective-C. It searches for anyone who lives in San Francisco and works for Apple, or for anyone who lives in Seattle and lives in Seattle.

Listing 5 Complex Search, in Objective-C

```
ABAddressBook *AB = [ABAddressBook sharedAddressBook];
ABSearchElement *inSF =
    [ABPerson searchElementForProperty:kABAddressProperty
                                 label:kABHomeLabel
                                   key:kABAddressCityKey
                                 value:@"San Francisco"
                            comparison:kABEqualCaseInsensitive];
ABSearchElement *atApple =
    [ABPerson searchElementForProperty:kABOrganizationProperty
                                 label:nil
                                   key:nil
                                 value:@"Apple"
                            comparison:kABContainsSubStringCaseInsensitive];
ABSearchElement *inSeattle =
    [ABPerson searchElementForProperty:kABAddressProperty
                                 label:kABHomeLabel
                                   key:kABAddressCityKey
                                 value:@"Seattle"
                            comparison:kABEqualCaseInsensitive];
ABSearchElement *atMS =
    [ABPerson searchElementForProperty:kABOrganizationProperty
                                 label:nil
                                   key:nil
                                 value:@"Microsoft"
                            comparison:kABContainsSubStringCaseInsensitive];
ABSearchElement *inSFAndAtApple =
    [ABSearchElement searchElementForConjunction:kABSearchAnd
                                        children:[NSArray arrayWithObjects:
```

```
                                       inSF, atApple, nil]];
ABSearchElement *inSeattleAndAtMS =
    [ABSearchElement searchElementForConjunction:kABSearchAnd
                                    children:[NSArray arrayWithObjects:
                                        inSeattle, atMS, nil]];
ABSearchElement *inSFAndAtAppleOrInSeattleAndAtMS =
    [ABSearchElement searchElementForConjunction:kABSearchOr
                                    children:[NSArray arrayWithObjects:
                                inSFAndAtApple, inSeattleAndAtMS, nil]];
NSArray *peopleFound =
    [AB recordsMatchingSearchElement:inSFAndAtAppleOrInSeattleAndAtMS];
```

Listing 6 (page 17) is a complex search in C.

Listing 6 Complex Search, in C

```
ABAddressBookRef AB = ABGetSharedAddressBook();
ABSearchElementRef inSF =
    ABPersonCreateSearchElement(kABLastNameProperty, kABHomeLabel,
                            kABAddressCityKey, CFSTR("San Francisco"),
                            kABEqualCaseInsensitive);
ABSearchElementRef atApple =
    ABPersonCreateSearchElement(kABOrganizationProperty, NULL,
                            NULL, CFSTR("Apple"),
                            kABContainsSubStringCaseInsensitive);
ABSearchElementRef inSeattle =
    ABPersonCreateSearchElement(kABLastNameProperty, kABHomeLabel,
                            kABAddressCityKey,  CFSTR("Seattle"),
                            kABEqualCaseInsensitive);
ABSearchElementRef atMS =
    ABPersonCreateSearchElement(kABOrganizationProperty,NULL,
                            NULL, CFSTR("Microsoft"),
                            kABContainsSubStringCaseInsensitive);
ABSearchElementRef array1[] = {inSF, atApple};
ABSearchElementRef array2[] = {inSeattle, atMS};
ABSearchElementRef inSFAndAtApple =
    ABSearchElementCreateWithConjunction(kABSearchAnd,
        CFArrayCreate(NULL, (void*)array1, 2, NULL));
ABSearchElementRef inSeattleAndAtMS =
    ABSearchElementCreateWithConjunction(kABSearchAnd,
        CFArrayCreate(NULL, (void*)array2, 2, NULL));
ABSearchElementRef array3[] = {inSFAndAtApple, inSeattleAndAtMS};
ABSearchElementRef inSFAndAtAppleOrInSeattleAndAtMS =
    ABSearchElementCreateWithConjunction(kABSearchOr,
        CFArrayCreate(NULL, (void*)array3, 2, NULL));
CFArrayRef peopleFound = ABCopyArrayOfMatchingRecords(AB,
    inSFAndAtAppleOrInSeattleAndAtMS);
```

Using Address Book Groups As Distribution Lists

An Address Book group can be used as a distribution list. For example, a user can have a Christmas Card group of all the people he or she mails Christmas cards to. Or a user can have a Book Club group of all the people he or she emails book club announcements to. The Address Book framework provides you with a special feature that helps you maintain distribution lists.

If a property has multiple values, like a street address or email address, it lets the user choose one as the value that this group uses. Generally, the user will want to use the value he or she has already marked as primary. But in some cases the user might want to make an exception. For example, for co-workers, the user would want their work email addresses to be their primary email addresses. But when the user notifies some of them about a book club that happens on the weekends, he or she would want to send email to their home addresses. Each group can use a different value for each person.

To choose the value in a multi-value list that a group uses, use the ABGroup method `setDistributionIdentifier:forProperty:person:` or the function `ABGroupSetDistributionIdentifier`. To get a group's chosen value for a multi-value list, use the ABGroup method `distributionIdentifierForProperty:person:` or the function `ABGroupCopyDistributionIdentifier`. These procedures return the identifier for the value chosen for this group or the identifier for the primary value, if no value is chosen for the group.

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 handle 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 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.

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

- Avoid properties for confidential information, such as credit card numbers. The Address Book 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. If you want to store information for just the logged-in user, consider using the NSUserDefaults or CFPreference APIs.
- Use a multi-value list if you think a person may have more than one of that property. Your new multi-value list has the same capabilities as the other multi-value 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:` or the function `ABAddPropertiesAndTypes`. These procedures take a NSDictionary or CFDictionary, 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 Java-style package names for your properties, to make sure no one else uses the same name; for example, `"org.dogclub.dogname"` or `"com.mycompany.meetinglist"`. The type can be one of five types or a multi-value list of one of those types. Here are the types:

- `kABStringProperty` or `kABMultiStringProperty`
- `kABIntegerProperty` or `kABMultiIntegerProperty`
- `kABRealProperty` or `kABMultiRealProperty` (a floating-point number)
- `kABDateProperty` or `kABMultiDateProperty` (an NSDate or CFDate)
- `kABArrayProperty` or `kABMultiArrayProperty` (an NSArray or CFArray)
- `kABDictionaryProperty` or `kABMultiDictionaryProperty` (an NSDictionary or CFDictionary)
- `kABDataProperty` or `kABMultiDataProperty` (an NSData or CFData)

Importing and Exporting Address Book People

You can import and export people in the Address Book using the vCard format. To create a vCard representation of a person, use the method `vCardRepresentation` or the function `ABPersonCopyVCardRepresentation`. These procedures create an NSData or CFData structure that you can use in your program or save to a file.

To create a person from a vCard representation, use the method `initWithVCardRepresentation:` or the function `ABPersonCreateWithVCardRepresentation`.