Sync Services Reference

2006-02-07

Contents

Introduction Sync Services Reference 5

Part I Classes 7

Chapter 1 ISyncChange 9

Class Description 9
Methods by Task 10
Class Methods 10
Instance Methods 11
Constants 12

Chapter 2 ISyncClient 15

Class Description 15
Methods by Task 16
Instance Methods 18
Constants 28

Chapter 3 ISyncFilter 31

Class Description 31
Methods by Task 31
Class Methods 32

Chapter 4 ISyncManager 33

Class Description 33
Methods by Task 34
Class Methods 34
Instance Methods 35
Constants 38
Notifications 38

Chapter 5 ISyncRecordSnapshot 39

Class Description 39
Methods by Task 40

Instance Methods 40

Chapter 6 ISyncSession 43

Class Description 43
Methods by Task 44
Class Methods 47
Instance Methods 49
Constants 59

Part II Protocols 61

Chapter 7 ISyncFiltering 63

Protocol Description 63
Methods by Task 64
Instance Methods 64

Part III Types and Constants 67

Chapter 8 Constants 69

Enumerations 69
Global Variables 70

Index 73

Sync Services Reference

Framework	/System/Library/Frameworks/SyncServices.framework
Header file directories	/System/Library/Frameworks/SyncServices.framework/Headers

This document was previously titled *Sync Services Reference for Objective-C*.

The Sync Services framework provides all the classes and protocols you need to sync your application's data with other applications and devices on the same computer or—using a .Mac account—with applications and devices on other computers. Developers use this framework to manage their sync sessions and communicate with the sync engine to push and pull changes.

P A R T   I

Classes

PART I
Classes

ISyncChange

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Declared in:	SyncServices/ISyncChange.h
Availability:	Available in Mac OS X v10.4 and later.
Companion guide:	Sync Services Programming Guide

Class Description

An ISyncChange object encapsulates a set of changes to a single record such as adding, deleting, and modifying a record. You use ISyncChange objects to push your changes to the sync engine. Similarly, you pull ISyncChange objects from the sync engine when applying sync engine changes.

You use the `changeWithType:recordIdentifier:changes:` (page 10) method to create an ISyncChange object specifying the type of change (an add, delete, or modify), the identifier for the record that changed, and the new property values if any. Use the `pushChange:` (page 56) ISyncSession method to push the change to the sync engine. You can also use the `deleteRecordWithIdentifier:` (page 54) ISyncSession method to delete a record without needing to create an ISyncChange object.

Use the `changeEnumeratorForEntityNames:` (page 49) ISyncSession method to pull changes from the sync engine. Use the returned object enumerator to process each ISyncChange object. Use the `type` (page 12) method to each ISyncChange object to get the type of change (add, modify or delete), the `recordIdentifier` (page 12) method to get the record identifier, and the `changes` (page 11) method get descriptions of the change.

Methods by Task

Creating and initializing instances

`+ changeWithType:recordIdentifier:changes:` (page 10)
Creates an ISyncChange object.

`- initWithChangeType:recordIdentifier:changes:` (page 11)
Initializes an ISyncChange object.

Getting attributes

`- type` (page 12)
Returns the type of change the receiver represents.

`- recordIdentifier` (page 12)
Returns the unique record identifier for the record that changed.

`- record` (page 11)
Returns a dictionary representation of the record that changed.

`- changes` (page 11)
Returns an array of changes made to the record.

Class Methods

changeWithType:recordIdentifier:changes:

Creates an ISyncChange object.

```
+ (id)changeWithType:(ISyncChangeType)type recordIdentifier:(NSString
    *)recordIdentifier changes:(NSArray *)changes
```

Discussion
Creates and returns an ISyncChange object of the type specified by `type`, for the record identified by `recordIdentifier`, and with the changes specified by `changes`. The `type` argument should be one of the `ISyncChangeTypeAdd` (page 13), `ISyncChangeTypeModify` (page 13) or `ISyncChangeTypeDelete` (page 13) constants. The `changes` array encapsulates multiple property changes to a single record—it is expected to contain dictionaries that use the keys described in "Constants" (page 12). Each dictionary in the array encapsulates the change to a single property and specifies the property name, action, and new value if applicable. Use this method to create changes for pushing to the sync engine. Use the `pushChange:` (page 56) ISyncSession method to push the change to the sync engine.

Availability
Available in Mac OS X v10.4 and later.

Instance Methods

changes

Returns an array of changes made to the record.

```
- (NSArray *)changes
```

Discussion
Returns an array of the changes made to the record returned by `record` (page 11) with the identifier returned by `recordIdentifier` (page 12). The changes array encapsulates multiple property changes to a single record. The returned array contains dictionaries with keys specifying the type of change to a property and its new value. See "Constants" (page 12) for a description of the keys used in these dictionaries. See `ISyncChangePropertyValueKey` (page 13) for a description of the value of relationship properties. Returns `nil` if the change type is `ISyncChangeTypeDelete`.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- type` (page 12)

initWithChangeType:recordIdentifier:changes:

Initializes an ISyncChange object.

```
- (id)initWithChangeType:(ISyncChangeType)type recordIdentifier:(NSString
    *)recordIdentifier changes:(NSArray *)changes
```

Discussion
Initializes an ISyncChange object of the type specified by *type*, for the record identified by *recordIdentifier*, and with the changes specified by *changes*. The type argument should be one of the `ISyncChangeTypeAdd` (page 13), `ISyncChangeTypeModify` (page 13) or `ISyncChangeTypeDelete` (page 13) constants. The *changes* array encapsulates multiple property changes to a single record—it is expected to contain dictionaries that use the keys described in "Constants" (page 12). Each dictionary encapsulates the change to a single property and specifies the property name, action, and new value if applicable. Use this method to create changes for pushing to the sync engine. Use the `pushChange:` (page 56) ISyncSession method to push the change to the sync engine. This is the designated initializer for this class.

Availability
Available in Mac OS X v10.4 and later.

record

Returns a dictionary representation of the record that changed.

```
- (NSDictionary *)record
```

Discussion
The dictionary contains a key-value pair for each property unless the value of a property is unspecified. Only changes created by the sync engine have an associated record. Returns `nil` if the client created the receiver to push changes, or this is a delete change.

When pulling changes, this method returns a copy of the record as it appears in the truth database. Only those properties supported by the client are included in this record. Use the `changes` (page 11) method to get the changes made to this record, and use the `recordIdentifier` (page 12) method to get the record's unique identifier.

Availability
Available in Mac OS X v10.4 and later.

recordIdentifier

Returns the unique record identifier for the record that changed.

```
- (NSString *)recordIdentifier
```

Discussion
Returns the unique identifier for the record returned by `record` (page 11), or the record identifier you specified when creating the receiver. Use the `changes` (page 11) method to get the changes that were made to this record.

Availability
Available in Mac OS X v10.4 and later.

type

Returns the type of change the receiver represents.

```
- (ISyncChangeType)type
```

Discussion
Returns whether or not this change was an add, delete or modify. See "Constants" (page 12) for a description of the possible return values.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- changes` (page 11)

Constants

The following constants are used to determine the type of change. Use one of these values to set the *`type`* argument when creating an ISyncChange object using either the `changeWithType:recordIdentifier:changes:` (page 10) class method or the `initWithChangeType:recordIdentifier:changes:` (page 11) instance method. The `type` (page 12) method also returns one of these values.

Constant	Description
`ISyncChangeTypeAdd`	Indicates a record was added. Available in Mac OS X v10.4 and later.
`ISyncChangeTypeModify`	Indicates a record was modified. Available in Mac OS X v10.4 and later.
`ISyncChangeTypeDelete`	Indicates a record was deleted. Available in Mac OS X v10.4 and later.

The following constants are used as keys for individual property changes encapsulated in an ISyncChange object. When pushing changes, use these keys to set key-value pairs for dictionaries you add to the *`changes`* argument passed to either the `changeWithType:recordIdentifier:changes:` (page 10) class method or the `initWithChangeType:recordIdentifier:changes:` (page 11) instance method. When pulling changes, you also use these keys to get the attributes of each change. Use the object enumerator returned by the `changes` (page 11) method to iterate through the changes array.

Constant	Description
`ISyncChangeProperty-ActionKey`	Specifies whether or not the property is being set or deleted. The value for this key should be either `ISyncChangePropertySet` or `ISyncChangePropertyClear` described below. Available in Mac OS X v10.4 and later.
`ISyncChangeProperty-NameKey`	Key for the name of the property. Available in Mac OS X v10.4 and later.
`ISyncChangeProperty-ValueKey`	Key for the new value of the property. Not used if the action is `ISyncChangePropertyClear`. However, the absence of this key does not imply the property is being deleted. This key-value pair may be omitted if the value is unspecified. You can also set the value to `nil`. If the property is a relationship, then the value is an array of record identifiers belonging to the destination objects. If the relationship is to-one, this array contains a single record identifier. Available in Mac OS X v10.4 and later.

The following constants are possible values for the `ISyncChangePropertyActionKey` (page 13) key used to describe the type of change to single property.

Constant	Description
`ISyncChangePropertySet`	Indicates the property was modified. Available in Mac OS X v10.4 and later.
`ISyncChangePropertyClear`	Indicates the property was deleted. Available in Mac OS X v10.4 and later.

ISyncClient

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Declared in:	SyncServices/ISyncClient.h
Availability:	Available in Mac OS X v10.4 and later.
Companion guide:	Sync Services Programming Guide

Class Description

ISyncClient represents an application, tool, or device that syncs records—for example, Address Book, .Mac, or a mobile phone.

An ISyncClient object encapsulates information that assists the sync engine in identifying your client, determining its capabilities, and maintaining its state. For example, you use an ISyncClient object to get the list of entities that a client supports, find out when an entity was last synced, and setup filters. ISyncClient also provides some methods for controlling the sync mode.

You create an ISyncClient object by registering a unique client identifier with the shared ISyncManager object. Send either the `registerClientWithIdentifier:descriptionFilePath:` (page 35) or `clientWithIdentifier:` (page 35) message to the shared ISyncManager object. You obtain the shared instance by sending `sharedManager` (page 34) to ISyncManager class. You unregister a client, remove all information the sync engine knows about that client, using the `unregisterClient:` (page 37) ISyncManager method. See *Sync Services Programming Guide* for more information on registering and unregistering clients. You should never subclass or instantiate ISyncClient directly.

When you create a client using the `registerClientWithIdentifier:descriptionFilePath:` (page 35) ISyncManager method, you specify the client's capabilities using a client description file. Some of the ISyncClient methods are simply accessors that you can use to get or set the properties of this client description. For example, you use the client description to specify the entities and properties that a client supports, and you use the `supportedEntityNames` (page 27) method to get those supported entities. You can also use the `canPushChangesForEntityName:` (page 19) and `canPullChangesForEntityName:` (page 18) methods to find out which entities your client can push and pull. See *Sync Services Programming Guide* to learn more about the properties of a client description file.

Typically, the user requests that an application or device be resets (so that all the records on the client are replaced by the records in the truth database). The preference panel or configuration tool that receives this user request sends a `setShouldReplaceClientRecords:forEntityNames:` (page 24) message to the ISyncClient so that the next time the client syncs the truth is pulled. This is called a **pull the truth** sync mode and must be requested before the sync session enters the negotiation state.

Clients can optionally sync simultaneously. Use the `setShouldSynchronize:withClientsOfType:` (page 24) method to specify the type of client your client is interested in syncing simultaneously with. If you want to participate in a sync when your application isn't running, use the `setSyncAlertToolPath:` (page 26) method to specify that an alert tool be launched. Otherwise, use the `setSyncAlertHandler:selector:` (page 25) method to specify that a target and action be invoked when another client of the specified type syncs. If both a sync tool and sync target-action are registered, only the sync target-action is invoked.

If your application uses only a subset of the entities, attributes, and relationships defined in a schema, then you can restrict pulled records to that subset using custom filters. You set filters using the `setFilters:` (page 23) method. Each filter is expected to conform to the ISyncFiltering protocol and are used to reject or accept records from the sync engine before they are pulled. Use the `filters` (page 20) method to get the filters currently used by a client. See *Sync Services Programming Guide* for more information on using filters.

Methods by Task

Getting and setting attributes

– `clientIdentifier` (page 19)
  Returns the client's identifier specified when registering the client.

– `clientType` (page 19)
  Returns the receiver's client type.

– `displayName` (page 19)
  Returns the receiver's display name specified in the client description file when registering the client or by sending `setDisplayName:` (page 22) to the receiver.

– `imagePath` (page 20)
  Returns the absolute path to the image representation of the client.

– `setDisplayName:` (page 22)
  Sets the display name for the receiver to *`displayName`*.

– `setImagePath:` (page 23)
  Sets the receiver's absolute image path to *`path`*.

– `objectForKey:` (page 22)
  Returns the object for *`key`* that was specified using the `setObject:forKey:` (page 23) method.

`- setObject:forKey:` (page 23)
Associates arbitrary information specified by a key-value pair to the receiver.

Specifying supported entities

`- canPushChangesForEntityName:` (page 19)
Returns `YES` if the client supports pushing changes to entity records specified by *`entityName`*, `NO` otherwise.

`- canPullChangesForEntityName:` (page 18)
Returns `YES` if the client supports pulling changes to entity records specified by *`entityName`*, `NO` otherwise.

`- supportedEntityNames` (page 27)
Returns an array of NSString objects containing the names of the entities the client supports.

Getting sync status

`- lastSyncDateForEntityName:` (page 21)
Returns the start date of the last time an entity, specified by *`entityName`*, was synced.

`- lastSyncStatusForEntityName:` (page 21)
Returns the status of the last time an entity, specified by *`entityName`* was synced.

Enabling entities

`- enabledEntityNames` (page 20)
Returns an array of NSString objects containing the names of the entities that are enabled.

`- isEnabledForEntityName:` (page 21)
Returns `YES` if the entity specified by *`entityName`* is enabled, `NO` otherwise.

`- setEnabled:forEntityNames:` (page 22)
If *`flag`* is `YES`, enables the entities specified by *`entityNames`*, otherwise disables them.

Replacing records

`- setShouldReplaceClientRecords:forEntityNames:` (page 24)
Sets whether or not a client should pull the truth—replace all its records for the specified entities on the next sync.

`- shouldReplaceClientRecordsForEntityName:` (page 27)
Returns `YES` if the client should replace all records for the entity specified by entityName during the next sync, `NO` otherwise.

Filtering

`- filters` (page 20)
: Returns an array of filters that define a subset of the records the client syncs.

`- setFilters:` (page 23)
: Sets the receiver's filters used to control the records pulled from the sync engine to *`filters`*, an array of objects conforming to the ISyncFiltering protocol.

Alerting clients

`- shouldSynchronizeWithClientsOfType:` (page 27)
: Returns `YES` if the client is registered to receive alerts when clients of *`clientType`* sync, `NO` otherwise.

`- setShouldSynchronize:withClientsOfType:` (page 24)
: Adds the receiver as an observer of alerts when clients of the specified type sync.

`- setSyncAlertToolPath:` (page 26)
: Specifies the absolute path to a tool that is launched when an observed client creates a session and begins syncing.

`- setSyncAlertHandler:selector:` (page 25)
: Sets the target and action to be invoked when an observed client creates a session and begins syncing.

`- syncAlertToolPath` (page 27)
: Returns the path to the tool that is launched when an observed client begins syncing.

Instance Methods

canPullChangesForEntityName:

Returns `YES` if the client supports pulling changes to entity records specified by *`entityName`*, `NO` otherwise.

```
- (BOOL)canPullChangesForEntityName:(NSString *)entityName
```

Discussion

Use this method to determine if a client is capable of pulling entity records. For example, an iPod or phone client might pull but never push changes to contacts and calendars. This property is set when registering the client.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- canPushChangesForEntityName:` (page 19) (ISyncManager)
`- registerClientWithIdentifier:descriptionFilePath:` (page 35)

canPushChangesForEntityName:

Returns `YES` if the client supports pushing changes to entity records specified by *`entityName`*, `NO` otherwise.

```
- (BOOL)canPushChangesForEntityName:(NSString *)entityName
```

Discussion
Use this method to determine if a client is capable of pushing entity records. For example, an iPod or phone client might pull but never push changes to contacts and calendars. This property is set when registering the client.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `- canPullChangesForEntityName:` (page 18) (ISyncManager)
- `- registerClientWithIdentifier:descriptionFilePath:` (page 35)

clientIdentifier

Returns the client's identifier specified when registering the client.

```
- (NSString *)clientIdentifier
```

Discussion
You set the client identifier when registering the client using the `registerClientWithIdentifier:descriptionFilePath:` (page 35) ISyncManager method. The client identifier is expected to be unique across all clients and is typically a DNS-style name.

Availability
Available in Mac OS X v10.4 and later.

clientType

Returns the receiver's client type.

```
- (NSString *)clientType
```

Discussion
The returned string is expected to be one of the ISyncClientType constants (see "Constants" (page 28)). The client type is used to match clients that want to sync simultaneously. You specify the client type in the client description file when registering the client using the `registerClientWithIdentifier:descriptionFilePath:` (page 35) ISyncManager method.

Availability
Available in Mac OS X v10.4 and later.

displayName

Returns the receiver's display name specified in the client description file when registering the client or by sending `setDisplayName:` (page 22) to the receiver.

```
- (NSString *)displayName
```

Availability
Available in Mac OS X v10.4 and later.

See Also
`- registerClientWithIdentifier:descriptionFilePath:` (page 35) (ISyncManager)

enabledEntityNames

Returns an array of NSString objects containing the names of the entities that are enabled.

```
- (NSArray *)enabledEntityNames
```

Discussion
The enabled entities may be a subset of the supported entities. Use `setEnabled:forEntityNames:` (page 22) to enable or disable an entity. You should pass the returned array as the *`entityNames`* argument to one of the `beginSessionWithClient...` ISyncSession class methods when creating a session.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- isEnabledForEntityName:` (page 21)

filters

Returns an array of filters that define a subset of the records the client syncs.

```
- (NSArray *)filters
```

Discussion
Objects in the returned array are expected to conform to the ISyncFiltering protocol

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setFilters:` (page 23)

imagePath

Returns the absolute path to the image representation of the client.

```
- (NSString *)imagePath
```

Discussion
You can specify an image path in the client description file when registering a client or by sending `setImagePath:` (page 23) to the receiver.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `registerClientWithIdentifier:descriptionFilePath:` (page 35) (ISyncManager)

isEnabledForEntityName:

Returns `YES` if the entity specified by *`entityName`* is enabled, `NO` otherwise.

```
- (BOOL)isEnabledForEntityName:(NSString *)entityName
```

Discussion
If this method returns `NO`, the sync engine does not allow the client to sync records of type *`entityName`*.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `enabledEntityNames` (page 20)
- `setEnabled:forEntityNames:` (page 22)

lastSyncDateForEntityName:

Returns the start date of the last time an entity, specified by *`entityName`*, was synced.

```
- (NSDate *)lastSyncDateForEntityName:(NSString *)entityName
```

Discussion
Returns a start date of the last sync even if the last sync failed. Returns the start date of the previous sync if the client is currently syncing the entity. Returns `nil` if the client never synced the specified entity or the entity is not supported.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `lastSyncStatusForEntityName:` (page 21)

lastSyncStatusForEntityName:

Returns the status of the last time an entity, specified by *`entityName`* was synced.

```
- (ISyncStatus)lastSyncStatusForEntityName:(NSString *)entityName
```

Discussion
For example, the last sync may have succeeded, may have failed, may be in progress, or may have been canceled (see "Constants" (page 28) for other possible return values). Returns `ISyncStatusNever` (page 29) if the client never synced the specified entity, or the entity is not supported.

The sync engine maintains the last sync information for as long as the client supports *`entityName`*. When a client stops supporting *`entityName`*, the last sync information for that entity is removed. If the client starts supporting *`entityName`* again, this method behaves as if the client never synced the entity.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- lastSyncDateForEntityName:` (page 21)

objectForKey:

Returns the object for `key` that was specified using the `setObject:forKey:` (page 23) method.

```
- (id)objectForKey:(NSString *)key
```

Availability
Available in Mac OS X v10.4 and later.

setDisplayName:

Sets the display name for the receiver to `displayName`.

```
- (void)setDisplayName:(NSString *)displayName
```

Discussion
The display name may be used by GUI applications to graphically identify the client to users. You can also specify a display name when registering the client using the client description file.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- registerClientWithIdentifier:descriptionFilePath:` (page 35) (ISyncManager)
`- displayName` (page 19)

setEnabled:forEntityNames:

If `flag` is `YES`, enables the entities specified by `entityNames`, otherwise disables them.

```
- (void)setEnabled:(BOOL)flag forEntityNames:(NSArray *)entityNames
```

Discussion
The `entityNames` array of NSString objects is expected to contain names of supported entities, otherwise an exception is raised.

The first time a client syncs, a panel appears asking the user if it’s OK to sync entities belonging to a data class (a panel may appear for each data class). If the user declines then the entities are disabled, otherwise they are enabled. If you want to allow the user to enable entities, invoke this method by passing `YES` as the `flag` argument and all the entity names in the data class as the `entityNames` argument. Then the next time the client syncs, a panel appears again asking the user if it’s OK to sync the data class.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `enabledEntityNames` (page 20)
- `isEnabledForEntityName:` (page 21)

setFilters:

Sets the receiver's filters used to control the records pulled from the sync engine to *`filters`*, an array of objects conforming to the ISyncFiltering protocol.

```
- (void)setFilters:(NSArray *)filters
```

Discussion
You use filters to define a subset of the records that this client syncs.

When pulling changes, the sync engine passes each record to each filter before giving changes to that record to the client. If any one of the filters rejects the record, it is not be given to the client. See ISyncFilter for some default filters.

This method recomputes the records that need to be sent to the client during the next sync operation which can be expensive. Consequently, do not invoke this method frequently.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `filters` (page 20)

setImagePath:

Sets the receiver's absolute image path to *`path`*.

```
- (void)setImagePath:(NSString *)path
```

Discussion
The image may be used by GUI applications to represent the client. You can also specify an image path when registering the client using the client description file.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `imagePath` (page 20) (ISyncManager)
- `registerClientWithIdentifier:descriptionFilePath:` (page 35)

setObject:forKey:

Associates arbitrary information specified by a key-value pair to the receiver.

```
- (void)setObject:(id <NSCoding>)value forKey:(NSString *)key
```

Discussion

This method retains *value* and copy *key*. Pass `nil` for *value* to release a previously retained value. Use `objectForKey:` (page 22)to retrieve the value for a given key. The *value* is released when the client is unregistered.

This method is provided as a convenience for developers who have additional data they want to store with an object that is not defined in the schema. For example, use this method to store client-specific configuration information if multiple clients are associated with the same user defaults domain or if you want to store a sync anchor.

A sync anchor is an identifier exchanged between a client and a device, or between two clients running on different computers. Typically, the client that initiates a sync is passed a sync anchor to the device or another client at the end of a successful sync. The next time the client syncs, the recipient of the sync anchor passes the anchor back to the original client to verify that it is in a known state.

Availability

Available in Mac OS X v10.4 and later.

setShouldReplaceClientRecords:forEntityNames:

Sets whether or not a client should pull the truth—replace all its records for the specified entities on the next sync.

```
- (void)setShouldReplaceClientRecords:(BOOL)flag forEntityNames:(NSArray
    *)entityNames
```

Discussion

If *flag* is `YES`, the client should replace all its local records with the records pulled from the sync engine.

After invoking this method, sending `shouldReplaceClientRecordsForEntityName:` (page 27) to any new sessions created for this client returns `YES`, and sending `shouldPushChangesForEntityName:` (page 58) or `shouldPushAllRecordsForEntityName:` (page 58) returns `NO`.

This request takes effect on the next session created after invoking this method and remains in effect until the client successfully passes through the pull phase of that session. The sync engine needs to know whether a client is going to pull the truth before entering the negotiation phase. This is necessary to detect conflicting push the truth and pull the truth requests.

A client should not remove its local records until after the records are successfully pulled from the sync engine. The local records can be safely removed after `shouldReplaceClientRecordsForEntityName:` (page 27) returns `YES`.

This method is typically used by a configuration tool that allows the user to revert to the state of the truth.

Availability

Available in Mac OS X v10.4 and later.

setShouldSynchronize:withClientsOfType:

Adds the receiver as an observer of alerts when clients of the specified type sync.

```
- (void)setShouldSynchronize:(BOOL)flag withClientsOfType:(NSString *)clientType
```

Discussion

If *flag* is `YES` the receiver is added; otherwise the receiver is removed as an observer for alerts of the specified type. Alternatively, you can specify this information when registering the client using the client description file. You can invoke this method multiple times to register additional client types.

Typically, you use this method to setup a dependency between two clients. For example, Address Book might observe all types of clients, and is given an opportunity to join any syncs which synchronize entities defined in the contacts schema. The .Mac client might observe only device clients, so it can join a Palm or phone sync session. The client is notified only if it has entities in common with the client that initiated the sync.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- shouldSynchronizeWithClientsOfType:` (page 27) (ISyncManager)
`- setSyncAlertHandler:selector:` (page 25)
`- setSyncAlertToolPath:` (page 26)
`- registerClientWithIdentifier:descriptionFilePath:` (page 35)

setSyncAlertHandler:selector:

Sets the target and action to be invoked when an observed client creates a session and begins syncing.

```
- (void)setSyncAlertHandler:(id)handler selector:(SEL)selector
```

Discussion

When *selector* is sent to *handler*, your client has the opportunity to join the sync session.

The *selector* method is expected to take the receiver (an ISyncClient object) as the first argument and an array of entity names (an NSArray object) as the second argument. The method signature for *selector* should look like:

```
- (void)client:(ISyncClient *)client willSyncEntityNames:(NSArray  *)entityNames
```

If *selector* returns without creating a session, the sync engine assumes the client will not join the session. If this client already has another handler registered—for example, from another client process—this method raises an exception. An observer is automatically removed when the client terminates.

When you create a session using the `beginSessionWithClient:entityNames:beforeDate:` (page 47) ISyncSession class method, you specify how long you are willing to wait for the sync session. This is the length of time you are willing to wait for all the other clients to join the session. If a client takes too long to join a session, the sync engine may proceed without it.

Use this method instead of `setSyncAlertToolPath:` (page 26) if you want to notify a running application only. Use `setShouldSynchronize:withClientsOfType:` (page 24) to specify the types of clients the receiver wishes to observe. If both a tool and an observer are registered, only the observer is notified.

Note: If your client is multithreaded, the thread that registers the alert handler has to exist, otherwise the client does not receive the alert.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- shouldSynchronizeWithClientsOfType:` (page 27)
`- syncAlertToolPath` (page 27)

setSyncAlertToolPath:

Specifies the absolute path to a tool that is launched when an observed client creates a session and begins syncing.

```
- (void)setSyncAlertToolPath:(NSString *)path
```

Discussion
The sync engine retains this path until the client is unregistered or you explicitly change the path using this method. Pass `nil` if you want to disable the sync alert tool.

When the tool is launched it passes the following command-line arguments:

```
    --sync <clientIdentifier> --entitynames <entityNames>
```

The *`clientIdentifier`* argument is the identifier of the observed client. The *`entityNames`* argument is a single string containing the entity names deliminated by commas that is synced. You can send `componentsSeparatedByString:` to the string with `@","` as the argument to convert it to an array of entity names. The order of the key-value pairs, where `--sync` and `--entitynames` are keys, is arbitrary. If the tool terminates without creating a sync session, the sync engine assumes the client will not join the session.

When you create a session using the `beginSessionWithClient:entityNames:beforeDate:` (page 47) ISyncSession class method, you specify how long you are willing to wait for the sync session. This is the time you are willing to wait for all the other clients to join the session. If a client takes too long to join a session, the sync engine may proceed without it.

Use this method instead of `setSyncAlertHandler:selector:` (page 25) if you want to notify an application or tool that may not be running. Use `setShouldSynchronize:withClientsOfType:` (page 24) to specify the types of clients the receiver wishes to observe. If both a tool and a handler are registered, only the handler is notified.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- shouldSynchronizeWithClientsOfType:` (page 27)
`- syncAlertToolPath` (page 27)

shouldReplaceClientRecordsForEntityName:

Returns `YES` if the client should replace all records for the entity specified by entityName during the next sync, `NO` otherwise.

```
- (BOOL)shouldReplaceClientRecordsForEntityName:(NSString *)entityName
```

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setShouldReplaceClientRecords:forEntityNames:` (page 24)

shouldSynchronizeWithClientsOfType:

Returns `YES` if the client is registered to receive alerts when clients of *`clientType`* sync, `NO` otherwise.

```
- (BOOL)shouldSynchronizeWithClientsOfType:(NSString *)clientType
```

Availability
Available in Mac OS X v10.4 and later.

See Also
`- setShouldSynchronize:withClientsOfType:` (page 24)
`- setSyncAlertHandler:selector:` (page 25)
`- setSyncAlertToolPath:` (page 26)
`- syncAlertToolPath` (page 27)

supportedEntityNames

Returns an array of NSString objects containing the names of the entities the client supports.

```
- (NSArray *)supportedEntityNames
```

Discussion
This property is set when registering the client.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- registerClientWithIdentifier:descriptionFilePath:` (page 35) (ISyncManager)

syncAlertToolPath

Returns the path to the tool that is launched when an observed client begins syncing.

```
- (NSString *)syncAlertToolPath
```

Availability
Available in Mac OS X v10.4 and later.

See Also

- `setShouldSynchronize:withClientsOfType:` (page 24)
- `setSyncAlertHandler:selector:` (page 25)
- `setSyncAlertToolPath:` (page 26)
- `shouldSynchronizeWithClientsOfType:` (page 27)

Constants

Use the following constants to specify the type of client you might want to sync simultaneously with using the `setShouldSynchronize:withClientsOfType:` (page 24) method. The `clientType` (page 19) method also returns one of these constants.

Constant	Description
`ISyncClientTypeApplication`	Indicates the client is an application, such as Mail or iCal. Available in Mac OS X v10.4 and later.
`ISyncClientTypeDevice`	Indicates the client is used to sync a device such as a phone or an iPod. Available in Mac OS X v10.4 and later.
`ISyncClientTypeServer`	Indicates the client is used to sync a remote server such as .Mac. Available in Mac OS X v10.4 and later.
`ISyncClientTypePeer`	Indicates the client is a peer, such as another computer. Available in Mac OS X v10.4 and later.

The following constants are returned by the `lastSyncStatusForEntityName:` (page 21) method to indicate the state of the last sync session.

Constant	Description
`ISyncStatusRunning`	Indicates the client is syncing. Available in Mac OS X v10.4 and later.
`ISyncStatusSuccess`	Indicates the last sync was successful. Available in Mac OS X v10.4 and later.
`ISyncStatusWarnings`	Indicates the last sync resulted in warnings. Available in Mac OS X v10.4 and later.
`ISyncStatusErrors`	Indicates the last sync resulted in errors. Available in Mac OS X v10.4 and later.
`ISyncStatusCancelled`	Indicates the last sync was canceled. Available in Mac OS X v10.4 and later.
`ISyncStatusFailed`	Indicates the last sync failed to complete (for example, the client crashed). Available in Mac OS X v10.4 and later.

Constant	Description
`ISyncStatusNever`	Indicates the client has never synced. Available in Mac OS X v10.4 and later.

ISyncFilter

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Declared in:	SyncServices/ISyncFilter.h
Availability:	Available in Mac OS X v10.4 and later.
Companion guide:	Sync Services Programming Guide

Class Description

ISyncFilter provides a set of standard filters that can be used by any client, and some utility methods for creating filters. You should never instantiate or subclass ISyncFilter directly.

If your application uses only a subset of the entities, attributes, and relationships defined in a schema, then you can restrict pulled records to that subset using filters. A filter is any object that conforms to the ISyncFiltering protocol. You set filters using the `setFilters:` (page 23) ISyncClient method.

Use the methods in this class to create new filters by applying logical `AND` and `OR` binary operators on a collection of filters. Use the `filterMatchingAllFilters:` (page 32) to apply an `AND` operator and `filterMatchingAtLeastOneFilter:` (page 32) to apply an `OR` operator.

See *Sync Services Programming Guide* for more information on using filters.

Methods by Task

Creating and initializing instances

`+ filterMatchingAllFilters:` (page 32)
Returns a filter that is the logical `AND` of the filters specified in the *`filters`* array.

`+ filterMatchingAtLeastOneFilter:` (page 32)
Returns a filter that is the logical `OR` of the filters specified in the *`filters`* array.

Class Methods

filterMatchingAllFilters:

Returns a filter that is the logical `AND` of the filters specified in the *`filters`* array.

```
+ (id <ISyncFiltering>)filterMatchingAllFilters:(NSArray *)filters
```

Discussion
All the filters are expected to conform to the ISyncFiltering protocol.

Availability
Available in Mac OS X v10.4 and later.

filterMatchingAtLeastOneFilter:

Returns a filter that is the logical `OR` of the filters specified in the *`filters`* array.

```
+ (id <ISyncFiltering>)filterMatchingAtLeastOneFilter:(NSArray *)filters
```

Discussion
All the filters are expected to conform to the ISyncFiltering protocol.

Availability
Available in Mac OS X v10.4 and later.

ISyncManager

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Declared in:	SyncServices/ISyncManager.h
Availability:	Available in Mac OS X v10.4 and later.
Companion guide:	Sync Services Programming Guide

Class Description

You use an ISyncManager object to communicate directly with the sync engine to perform administrative operations. A client must register itself with an ISyncManager object before it can sync its data. If a client is not using an existing schema, it must register the schema before it registers itself. You also use an ISyncManager to look up an existing client and unregister a client.

There's only one ISyncManager instance per client process you obtain using the `sharedManager` (page 34) class method. You should never instantiate or subclass ISyncManager directly.

Sync Services provides three canonical schemas: `Bookmarks.syncschema`, `Contacts.syncschema`, and `Calendars.syncschema`. If you want to extend one of these existing schemas or define your own schema, then you need to register that schema with the shared ISyncManager object. You use the `registerSchemaWithBundlePath:` (page 36) method to register a schema with the sync engine or update an existing schema. Occasionally, you might use `unregisterSchemaWithName:` (page 37) to remove a schema and all associated records. Removing a schema impacts every client that uses that schema. Typically, you just register a schema once and reregister it when it changes.

You also use the shared ISyncManager object to create and register a sync client—that is, an instance of ISyncClient—with a unique identifier that you specify. Use `registerClientWithIdentifier:descriptionFilePath:` (page 35) to register your client—this method returns either a new client object or an existing client. You also use this method to describe the capabilities of the client—for example, describe what entities and properties the client supports. You use the `unregisterClient:` (page 37) method to unregister a client. See *Sync Services Programming Guide* for more information on registering and unregistering clients.

Methods by Task

Getting the default manager

`+ sharedManager` (page 34)
Returns a shared ISyncManager object.

Getting a manager's state

`- isEnabled` (page 35)
Returns `NO` if the sync engine is disabled, `YES` otherwise.

Registering schemas

`- registerSchemaWithBundlePath:` (page 36)
Registers a schema property list located in a bundle at *`bundlePath`*.

`- unregisterSchemaWithName:` (page 37)
Unregisters a schema uniquely identified by *`schemaName`*, and removes all associated records.

Registering clients

`- clientWithIdentifier:` (page 35)
Returns the sync client identified by *`clientIdentifier`*, or `nil` if not found.

`- registerClientWithIdentifier:descriptionFilePath:` (page 35)
Returns an existing or new sync client uniquely identified by *`clientIdentifier`*.

`- unregisterClient:` (page 37)
Unregisters a sync client represented by *`client`*.

Getting snapshots

`- snapshotOfRecordsInTruthWithEntityNames:usingIdentifiersForClient:` (page 36)
Returns an immutable snapshot of the records for *`entityNames`* from the truth database.

Class Methods

sharedManager

Returns a shared ISyncManager object.

```
+ (ISyncManager *)sharedManager
```

Availability
Available in Mac OS X v10.4 and later.

Instance Methods

clientWithIdentifier:

Returns the sync client identified by `clientIdentifier`, or `nil` if not found.

```
- (ISyncClient *)clientWithIdentifier:(NSString *)clientIdentifier
```

Availability
Available in Mac OS X v10.4 and later.

See Also
`- registerClientWithIdentifier:descriptionFilePath:` (page 35)
`- unregisterClient:` (page 37)

isEnabled

Returns `NO` if the sync engine is disabled, `YES` otherwise.

```
- (BOOL)isEnabled
```

Discussion
You should not begin a sync session when this method returns `NO`. However, you can register for the `ISyncAvailabilityChangedNotification` (page 38) notification, which is sent when the sync engine state changes.

Availability
Available in Mac OS X v10.4 and later.

registerClientWithIdentifier:descriptionFilePath:

Returns an existing or new sync client uniquely identified by `clientIdentifier`.

```
- (ISyncClient *)registerClientWithIdentifier:(NSString *)clientIdentifier
    descriptionFilePath:(NSString *)descriptionFilePath
```

Discussion
There are no restrictions on the content or length of `clientIdentifier`, but it must be unique across all clients. Typically, it's a DNS-style name such as `com.apple.iCal`.

The client description file located at `descriptionFilePath` is a property list that specifies client information that the sync engine needs to know to sync its records. For example, the client description file a list of the client supported entities and properties. See *Sync Services Programming Guide* for a complete description of the client description file.

If the client already exists, then invoking this method updates the client description. If the set of supported entities and properties changes, the sync engine may force the client to slow sync the next time it syncs. This can be expensive, so only reregister a client if necessary.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `clientWithIdentifier:` (page 35)
- `unregisterClient:` (page 37)
- `canPullChangesForEntityName:` (page 18) (ISyncClient)
- `canPushChangesForEntityName:` (page 19) (ISyncClient)
- `displayName` (page 19) (ISyncClient)
- `imagePath` (page 20) (ISyncClient)

registerSchemaWithBundlePath:

Registers a schema property list located in a bundle at *bundlePath*.

```
- (BOOL)registerSchemaWithBundlePath:(NSString *)bundlePath
```

Discussion
The schema can define new entities and properties, and extend existing entities. The schema bundle may contain other files, such as images and localization files. See *Sync Services Programming Guide* for more details on the schema format and contents of the schema bundle.

If a schema of the same name exists, invoking this method updates that schema. Consequently, records and properties of records may be removed if an entity or property is removed from the schema. This action may cause clients that use this schema to slow sync the next time they sync. This process can be expensive, so reregister a schema only if necessary.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `unregisterSchemaWithName:` (page 37)

snapshotOfRecordsInTruthWithEntityNames:usingIdentifiersForClient:

Returns an immutable snapshot of the records for *entityNames* from the truth database.

```
- (ISyncRecordSnapshot *)snapshotOfRecordsInTruthWithEntityNames:(NSArray
    *)entityNames usingIdentifiersForClient:(ISyncClient *)client
```

Discussion
The truth database stores a copy of all the synced records and contains the amalgamation of all entities and properties from all clients. The snapshot is made of the records for entities specified by the *entityNames* argument, an array of NSString objects containing the names of entities. You access the records by sending messages to the returned ISyncRecordSnapshot object.

Each client has its own name space for record identifiers. The *client* argument specifies the name space you want to use. If *client* is `nil` or invalid, the record identifiers from the sync engine's global name space are used.

The snapshot is an immutable copy of the records taken at the time returned object is created. If the truth database is subsequently modified, the changes are not be reflected in the snapshot. You should create a new snapshot if you want up-to-date records.

Do not use this method if you are syncing and want a snapshot that is consistent with the sync session. Another client may be pushing changes that you have not pulled yet. Instead, you can use the ISyncSession `snapshotOfRecordsInTruth` (page 59) method to get the state of a session.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- clientWithIdentifier:` (page 35)

unregisterClient:

Unregisters a sync client represented by *client*.

```
- (void)unregisterClient:(ISyncClient *)client
```

Discussion
Does nothing if *client* is not registered.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- clientWithIdentifier:` (page 35)
`- registerClientWithIdentifier:descriptionFilePath:` (page 35)

unregisterSchemaWithName:

Unregisters a schema uniquely identified by *schemaName*, and removes all associated records.

```
- (void)unregisterSchemaWithName:(NSString *)schemaName
```

Discussion
This action causes clients that use this schema to slow sync the next time they sync. This can be expensive and results in the loss of data, so only unregister a schema if necessary.This method does nothing if the schema is not registered.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- registerSchemaWithBundlePath:` (page 36)

Constants

The following constant is thrown whenever the connection with the sync engine is lost.

Constant	Description
`ISyncServer-UnavailableException`	A string aggregating the name, reason, and user info from the originating exception. Thrown by any ISyncManager method when communication to the server is lost. Available in Mac OS X v10.4 and later.

Notifications

ISyncAvailabilityChangedNotification

Posted by the distributed notification center when syncing is enabled or disabled. The notification object is an NSString equal to "`YES`" if enabled and "`NO`" if disabled. The receiver should still invoke `isEnabled` (page 35) before beginning a sync session. This notification does not contain a `userInfo` dictionary.

Availability
Available in Mac OS X v10.4 and later.

ISyncRecordSnapshot

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Declared in:	SyncServices/ISyncRecordSnapshot.h
Availability:	Available in Mac OS X v10.4 and later.
Companion guide:	Sync Services Programming Guide

Class Description

The ISyncRecordSnapshot class provides a client with access to an immutable copy of the records in the truth database. The truth database is an aggregate of all records of all supported entities and properties of all registered clients. ISyncRecordSnapshot provides a variety of methods for querying the records in the snapshot.

There is no mutable variant of ISyncRecordSnapshot for changing records—only the sync engine updates the truth by clients pushing records during a sync session. If the sync engine modifies records after a snapshot is created, the snapshot will not contain those changes. You must create a new snapshot to get the latest records.

Use the `snapshotOfRecordsInTruthWithEntityNames:usingIdentifiersForClient:` (page 36) ISyncManager method to create a snapshot of the truth database. Use the `recordsWithIdentifiers:` (page 40) method to get a dictionary representation of all the specified records where the keys in the returned dictionary are the record identifiers. For example, a configuration tool might create a snapshot to query records in the truth database and provide various filtering.

However, if you are syncing and want a snapshot that is consistent with the sync session (for example, another client is pushing changes that you have not pulled yet), then you need to use the ISyncSession `snapshotOfRecordsInTruth` (page 59) method to get a snapshot.

Other methods in this class are used to query the snapshot. Use the `recordsWithIdentifiers:` (page 40) method to get specific records (it is more efficient to request records in batches, not all at once). Use the `recordsWithMatchingAttributes:` (page 41) method to get records that match specific attribute values, and the `sourceIdentifiersForRelationshipName:withTargetIdentifier:` (page 41) or `targetIdentifiersForRelationshipName:withSourceIdentifier:` (page 41) method to get records that match relationship values. Because all of these query methods can be computationally intensive, use them with care.

Methods by Task

Getting records

- `recordsWithIdentifiers:` (page 40)
  Returns a dictionary containing the records.

Getting records in a relationship

- `targetIdentifiersForRelationshipName:withSourceIdentifier:` (page 41)
  Returns an array of record identifiers belonging to the target objects of a relationship.
- `sourceIdentifiersForRelationshipName:withTargetIdentifier:` (page 41)
  Returns an array of the record identifiers belonging to the source objects of a relationship.

Searching for records

- `recordsWithMatchingAttributes:` (page 41)
  Returns a dictionary containing all the records that match a query.

Instance Methods

recordsWithIdentifiers:

Returns a dictionary containing the records.

```
- (NSDictionary *)recordsWithIdentifiers:(NSArray *)recordIdentifiers
```

Discussion
Returns a dictionary containing the records specified by *`recordIdentifiers`*, an array of record identifiers. The dictionary keys are the record identifiers, and the values are the record dictionaries. The returned dictionary does not contain key-value pairs for deleted records or invalid record identifiers.

Availability
Available in Mac OS X v10.4 and later.

recordsWithMatchingAttributes:

Returns a dictionary containing all the records that match a query.

```
- (NSDictionary *)recordsWithMatchingAttributes:(NSDictionary *)attributes
```

Discussion

The `attributes` argument specifies the query and supports key-value pairs for record properties. For example, a key might be `firstName` and a value might be “`Jane`” in the `attributes` dictionary. Optionally, you can use the `ISyncRecordEntityNameKey` key to limit the search to a particular entity. The keys in the returned dictionary are the record identifiers, and the values are the record dictionaries that match the query.

Availability

Available in Mac OS X v10.4 and later.

sourceIdentifiersForRelationshipName:withTargetIdentifier:

Returns an array of the record identifiers belonging to the source objects of a relationship.

```
- (NSArray *)sourceIdentifiersForRelationshipName:(NSString *)relationshipName
    withTargetIdentifier:(NSString *)targetIdentifier
```

Discussion

Returns an array of the record identifiers of the source objects belonging to the relationship, specified by `relationshipName`, whose target object identifier is `targetIdentifier`. You can use this method to get the inverse of a to-many or to-one relationship when that relationship is not defined in the schema.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- targetIdentifiersForRelationshipName:withSourceIdentifier:` (page 41)

targetIdentifiersForRelationshipName:withSourceIdentifier:

Returns an array of record identifiers belonging to the target objects of a relationship.

```
- (NSArray *)targetIdentifiersForRelationshipName:(NSString *)relationshipName
    withSourceIdentifier:(NSString *)sourceIdentifier
```

Discussion

Returns an array of the record identifiers of the target objects belonging to the relationship, specified by `relationshipName`, whose source object identifier is `sourceIdentifier`. If `relationshipName` is a to-one relationship, then the returned array contains no more than one object, otherwise it may contain multiple objects. If the relationship is ordered, the order of the record identifiers matches the order of the records. A record identifier may appear in the array multiple times. Use this method if you want to resolve the destination objects of a relationship.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- sourceIdentifiersForRelationshipName:withTargetIdentifier:` (page 41)

ISyncSession

Inherits from:	NSObject
Conforms to:	NSObject (NSObject)
Declared in:	SyncServices/ISyncSession.h
Availability:	Available in Mac OS X v10.4 and later.
Companion guide:	Sync Services Programming Guide

Class Description

An ISyncSession object is used to manage a single sync operation. It coordinates communication between a client, the sync engine, and any other clients that sync simultaneously. You create an ISyncSession object when you want to sync—you use it to sync your records and then throw it away.

A sync session is modeled as a finite state machine whose four main states are negotiation, pushing, mingling, and pulling. During the negotiation state, the client requests a sync mode, such as **slow sync**, **fast sync**, or **pull the truth**. The client then pushes changes to the sync engine. When all changes from all participating clients are pushed, the sync engine enters the mingling state. During mingling, it processes all the pushed records and computes the changes that are pulled by each client. After mingling, the client pulls changes from the sync engine.

The client can finish or cancel a session at any time. Read *Sync Services Programming Guide* for more details about each state in the finite state machine, transactions within each state, and the consequences of invoking ISyncSession methods. Refer to *Sync Services Programming Guide* for definition of other sync terms.

Use the `beginSessionWithClient:entityNames:beforeDate:` (page 47) method to create an ISyncSession object for the specified client and entities. Alternatively, use the `beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 47) method if you don't want to block when creating a session. Creating a session might block if the sync engine is waiting for other clients to join the sync.

During the negotiation state, use the `clientWantsToPushAllRecordsForEntityNames:` (page 53) or the `clientDidResetEntityNames:` (page 51) method to request a different sync mode than the default mode.

During the pushing state, use the `pushChange:` (page 56) method to push just the changes for a particular record. Otherwise, use the `pushChangesFromRecord:withIdentifier:` (page 57) method to push the entire record and let the sync engine figure out which properties changed. Use the `deleteRecordWithIdentifier:` (page 54) to push a delete change.

Use the `prepareToPullChanges`... methods to begin the mingling state and transition to the pulling state. During the mingling state, the sync engine merges all the changes between multiple clients and computes what changes need to be pulled by clients. Hence, these methods prepare the sync session for the pulling state.

During the pulling state, use the `changeEnumeratorForEntityNames:` (page 49) method to pull all the changes for the specified entities. Use the returned object enumerator to iterate through and apply the changes to your data. Use the `clientAcceptedChangesForRecordWithIdentifier:formattedRecord:newRecordIdentifier:` (page 50) method to accept changes and the `clientRefusedChangesForRecordWithIdentifier:` (page 53) method to reject changes. Use the `clientCommittedAcceptedChanges` (page 51) method to commit the accepted changes and close a transaction within the pulling state.

Use `finishSyncing` (page 54) to terminate a sync session by closing an open transaction. Or use `cancelSyncing` (page 49) to cancel a sync session which rolls back the state of the client to the previously closed transaction. All changes that were applied in an open transaction need to be reapplied on the next sync. The `finishSyncing` (page 54) and `cancelSyncing` (page 49) methods can be invoked at any time. However, the ISyncSession object should be released after invoking these methods, because the object cannot be used in a subsequent sync.

Methods by Task

Creating a sync session

`+ beginSessionWithClient:entityNames:beforeDate:` (page 47)
Creates and returns a new sync session for the specified client.

`+ beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 47)
Creates a new sync session for the specified client.

`+ cancelPreviousBeginSessionWithClient:` (page 48)
Cancels a previous request to create a session using `beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 47) for *`client`*.

Negotiating a sync mode

`- clientDidResetEntityNames:` (page 51)
Tells the sync engine to perform a refresh sync of all the records for the specified entities.

- `clientWantsToPushAllRecordsForEntityNames:` (page 53)
  Forces a slow sync of all the records for the specified entities.

- `shouldPushChangesForEntityName:` (page 58)
  Returns `YES` if the client should push changes to records for *`entityName`* since the last sync, `NO` otherwise.

- `shouldPushAllRecordsForEntityName:` (page 58)
  Returns `YES` if the client should push all the records for *`entityName`* to the sync engine, `NO` otherwise.

- `shouldPullChangesForEntityName:` (page 57)
  Returns `YES` if the client should pull changes to records for *`entityName`*, `NO` otherwise.

- `shouldReplaceAllRecordsOnClientForEntityName:` (page 59)
  Returns `YES` if the client should delete all the records for the entity, specified by *`entityName`*, and replace them with records pulled from the sync engine, `NO` otherwise.

Pushing changes

- `pushChange:` (page 56)
  Pushes changes made to a single record, specified by *`change`* , to the sync engine.

- `pushChangesFromRecord:withIdentifier:` (page 57)
  Compares *`record`* to the client's previous known state of the record, identified by *`recordIdentifier`*, and pushes the changes to the sync engine.

- `deleteRecordWithIdentifier:` (page 54)
  Creates a delete change for the record specified by *`recordIdentifier`* and pushes the change to the sync engine.

Mingling

- `prepareToPullChangesForEntityNames:beforeDate:` (page 55)
  Returns `NO` if *`date`* expires before the sync engine is ready for the client to begin pulling changes, `YES` otherwise.

- `prepareToPullChangesInBackgroundForEntityNames:target:selector:` (page 55)
  Moves the receiver to the mingling state and sends *`selector`* to *`target`* when the sync engine is ready for the client to begin pulling changes to the specified entities.

Pulling changes

- `changeEnumeratorForEntityNames:` (page 49)
  Returns the object enumerator for the ISyncChange objects which contain all the changes the client should apply to its local data.

- `clientAcceptedChangesForRecordWithIdentifier:formattedRecord:newRecordIdentifier:` (page 50)
  Informs the sync engine that the client has accepted the changes to the record identified by *recordIdentifier* during the pulling state.

- `clientRefusedChangesForRecordWithIdentifier:` (page 53)
  Informs the sync engine during the pulling state that the client has refused to apply the changes for the record specified by *recordIdentifier*.

- `clientCommittedAcceptedChanges` (page 51)
  Informs the sync engine that all accepted and rejected changes in the current transaction during the pulling state should be committed.

- `clientChangedRecordIdentifiers:` (page 51)
  Changes the record identifiers of the given records.

Pulling and pulling changes

- `clientLostRecordWithIdentifier:shouldReplaceOnNextSync:` (page 52)
  Tells the sync engine that a record identified by *recordIdentifier*, no longer exists on the client, and indicates whether or not it should be replaced.

Finishing syncing

- `finishSyncing` (page 54)
  Tells the sync engine that the client is done syncing. Invoking this method closes any open transactions in the pushing or pulling states.

Cancelling syncing

- `isCancelled` (page 55)
  Returns `YES` if the receiver was canceled, `NO` otherwise.

- `cancelSyncing` (page 49)
  Cancels the current session.

Getting and setting client information

- `clientInfoForRecordWithIdentifier:` (page 52)
  Returns a client-specific, nonsynchronized object that stores additional information about a record specified by *recordIdentifier*.

- `setClientInfo:forRecordWithIdentifier:` (page 57)
  Associates a client-specific, non synchronized object, *clientInfo*, to a record specified by *recordIdentifier* .

Getting snapshots

- `snapshotOfRecordsInTruth` (page 59)
  Returns an immutable snapshot of the records in the truth database.

Class Methods

beginSessionInBackgroundWithClient:entityNames:target:selector:

Creates a new sync session for the specified client.

```
+ (void)beginSessionInBackgroundWithClient:(ISyncClient *)client entityNames:(NSArray
     *)entityNames target:(id)target selector:(SEL)selector
```

Discussion

Returns immediately and sends *selector* to *target* when a session was successfully or unsuccessfully created. The *selector* is passed two arguments: the first argument is the ISyncClient object and the second argument is the new ISyncSession object. If the sync engine is disabled and this method failed to create a session, *selector* is sent to *target* with `nil` as the ISyncSession object.

The *entityNames* argument specifies which entities the client wants to sync during this session. They can be a subset of the client's supported entities and may include entities which have been disabled. However, the sync engine does not allow the client to push changes to disabled entities nor does it provide changes to disabled entities while pulling changes. Typically, you use the array returned by sending `enabledEntityNames` (page 20) to your ISyncClient object as the *entityNames* argument to this method.

Creating a session for *client* may trigger notifications to other clients observing syncs of this client type. This method differs from `beginSessionWithClient:entityNames:beforeDate:` (page 47) by not blocking and sending *selector* to *target* when all dependent clients have joined the sync session. (Send `setShouldSynchronize:withClientsOfType:` (page 24) to an ISyncClient object to setup these dependencies.) This method requires the client have a run loop running in the default mode.

Note: ISyncSession is not thread-safe. You can pass an ISyncSession object between threads but you should not use it concurrently. Asynchronous callbacks from ISyncSession are delivered to any client thread that used any of the SyncServices API. The client is responsible for directing the callback to an appropriate thread.

Availability

Available in Mac OS X v10.4 and later.

See Also

+ `cancelPreviousBeginSessionWithClient:` (page 48)
- `isEnabled` (page 35) (ISyncManager)

beginSessionWithClient:entityNames:beforeDate:

Creates and returns a new sync session for the specified client.

```
+ (ISyncSession *)beginSessionWithClient:(ISyncClient *)client entityNames:(NSArray
    *)entityNames beforeDate:(NSDate *)date
```

Discussion

This method returns when all dependent clients have had the opportunity to join the sync session or `date` has expired, which ever occurs first. This method might block if another client is syncing an entity specified in `entityNames`—consequently, invoking this method may block. Returns `nil` if the sync engine is disabled.

The `entityNames` argument specifies which entities the client wants to sync during this session. They can be a subset of the client's supported entities, and may include entities which have been disabled. However, the sync engine does not allow the client to push changes to disabled entities nor does it provide changes to disabled entities while pulling changes. Typically, you use the array returned by sending `enabledEntityNames` (page 20) to your ISyncClient object as the `entityNames` argument to this method.

Creating a session for `client` may trigger notifications to other clients observing syncs of this client type. The `date` argument specifies how long this client is willing to wait for other clients to join this session. If `date` is in the distant future, this method blocks until all dependent clients have joined the session or `date` has expired. If `date` is the current date or a past date, this method returns `nil` if the session can not be created immediately.

Choose a future date carefully before invoking this method. If `date` is too small, dependent clients may be excluded from joining the sync. Typically, a client specifies the longest delay possible. However, if you sync before terminating an application, you might specify a zero delay.

Send `setShouldSynchronize:withClientsOfType:` (page 24) to an ISyncClient object to setup these dependencies. Use the `beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 47) method if you don't want to block when creating a session.

Availability

Available in Mac OS X v10.4 and later.

See Also

`+ cancelPreviousBeginSessionWithClient:` (page 48)

`- isEnabled` (page 35) (ISyncManager)

cancelPreviousBeginSessionWithClient:

Cancels a previous request to create a session using `beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 47) for `client`.

```
+ (void)cancelPreviousBeginSessionWithClient:(ISyncClient *)client
```

Discussion

Use the `beginSessionWithClient:entityNames:beforeDate:` (page 47) method if you prefer to block when creating a session.

Availability

Available in Mac OS X v10.4 and later.

See Also

`+ beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 47)

`+ beginSessionWithClient:entityNames:beforeDate:` (page 47)

Instance Methods

cancelSyncing

Cancels the current session.

```
- (void)cancelSyncing
```

Discussion
May close an open pull or push transaction, and rolls back the state of the client to the previous transaction.

In the case of a pull transaction, the sync engine assumes the client is able to reapply the same changes on the next sync. In the case of a push transaction, the changes are reapplied to the client on the next sync. If the client cannot push or pull the same changes, it must force a slow sync by sending `clientWantsToPushAllRecordsForEntityNames:` (page 53) to the session before the next sync.

Use this method at any time but the receiver should be released soon afterward because you cannot continue using a canceled session.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- isCancelled` (page 55)
`- finishSyncing` (page 54)

changeEnumeratorForEntityNames:

Returns the object enumerator for the ISyncChange objects which contain all the changes the client should apply to its local data.

```
- (NSEnumerator *)changeEnumeratorForEntityNames:(NSArray *)entityNames
```

Discussion
Use the returned object to iterate through the record changes during the pulling state. The *`entityNames`* argument can contain a subset of the supported entities.

The sync engine applies client filters to the returned changes so that the client does not receive changes that were rejected. See *Sync Services Programming Guide* for more information on setting filters.

When the client applies a change described by an ISyncChange object, it must either accept or reject the change. You accept a change by sending `clientAcceptedChangesForRecordWithIdentifier:formattedRecord:newRecordIdentifier:` (page 50) to the sync session, and reject a change by sending `clientRefusedChangesForRecordWithIdentifier:` (page 53) to the sync session. After processing all changes (saving them on the device or locally) you send `clientCommittedAcceptedChanges` (page 51) to the sync session to commit those changes. Any uncommitted accepted changes or rejected changes are sent again to the client during the next sync.

Use this method during the pulling state only, otherwise an exception is raised.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- prepareToPullChangesForEntityNames:beforeDate:` (page 55)

clientAcceptedChangesForRecordWithIdentifier:formattedRecord:newRecordIdentifier:

Informs the sync engine that the client has accepted the changes to the record identified by *`recordIdentifier`* during the pulling state.

```
- (void)clientAcceptedChangesForRecordWithIdentifier:(NSString *)recordIdentifier
    formattedRecord:(NSDictionary *)formattedRecord newRecordIdentifier:(NSString
    *)newRecordIdentifier
```

Discussion
If your client or device can not store properties using the defined schema, then you can use the *`formattedRecord`* to specify an alternate format. By specifying an alternate format, you assist the sync engine in figuring out which records and properties are equal during the mingling state, so that the sync engine doesn't generate false changes for records that were simply reformatted. The *`formattedRecord`* dictionary should contain the entire record—key-value pairs for the formatted and unformatted properties including the `ISyncRecordEntityNameKey` (page 60) key identifying the record's entity name. Otherwise, the *`formattedRecord`* argument should be `nil`.

For example, if you are syncing a device and the device truncates first and last names to 20 characters long, then you should specify a *`formattedRecord`* record for truncated values when invoking this method. The *`formattedRecord`* dictionary should contain key-value pairs for only the properties that have different formats (were truncated by the device). See *Sync Services Programming Guide* for more details on formatting records.

The sync engine creates a new identifier for added records using CFUUIDRef. However, if your client generates its own unique record identifiers, then you can use the *`newRecordIdentifier`* argument to request that your identifier be used in future communications with the sync engine.

You can use this method to batch up changes by invoking it repeatedly. You can also use the `clientRefusedChangesForRecordWithIdentifier:` (page 53) method to reject changes. However, after a sequence of accepting and rejecting changes, you need to invoke `clientCommittedAcceptedChanges` (page 51) to end the transaction and actually commit them.

Use this method during the pulling state only, otherwise an exception is raised.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- clientChangedRecordIdentifiers:` (page 51)
`- clientLostRecordWithIdentifier:shouldReplaceOnNextSync:` (page 52)

clientChangedRecordIdentifiers:

Changes the record identifiers of the given records.

```
- (void)clientChangedRecordIdentifiers:(NSDictionary *)oldToNew
```

Discussion

The `oldToNew` dictionary should contain key-value pairs where the keys are the old record identifiers, currently used by the sync engine, and the values are the new record identifiers. Use this method if your application generates its own unique record identifier. Alternatively, you can change individual identifiers when adding a record during the pulling state by sending `clientAcceptedChangesForRecordWithIdentifier:formattedRecord:newRecordIdentifier:` (page 50) to the session. This method can be invoked when pushing or pulling records.

Availability

Available in Mac OS X v10.4 and later.

clientCommittedAcceptedChanges

Informs the sync engine that all accepted and rejected changes in the current transaction during the pulling state should be committed.

```
- (void)clientCommittedAcceptedChanges
```

Discussion

Invoke this method after you save the accepted changes locally or on the device you are syncing.

You accept a change by sending `clientAcceptedChangesForRecordWithIdentifier:formattedRecord:newRecordIdentifier:` (page 50) to the sync session, and reject a change by sending `clientRefusedChangesForRecordWithIdentifier:` (page 53) to the sync session.

Invoking this method ends an open pull transaction that began by sending either `prepareToPullChangesForEntityNames:beforeDate:` (page 55)or `prepareToPullChangesInBackgroundForEntityNames:target:selector:` (page 55) to the receiver. Once a transaction ends the sync engine commits the changes to the truth database, and opens a new pull transaction.

Use this method during the pulling state only, otherwise an exception is raised.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- cancelSyncing` (page 49)
`- finishSyncing` (page 54)

clientDidResetEntityNames:

Tells the sync engine to perform a refresh sync of all the records for the specified entities.

```
- (void)clientDidResetEntityNames:(NSArray *)entityNames
```

Discussion

The *entityNames* array can contain a subset of the supported entity names. Use this method if a user hard-resets a device by removing all its records, or if the local client data file is accidently deleted.

After invoking this method, the client is expected to push all the records for the specified entities similar to a slow sync. However, during a refresh sync, the sync engine resets the client's sync state (as if it never synced before) and consequently, does not generate any delete changes when the client pulls records. Although the client may pull delete changes if the sync engine detects duplicate records.

Use this method during the negotiation state only, otherwise an exception is raised.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- shouldPullChangesForEntityName:` (page 57)

`- shouldReplaceAllRecordsOnClientForEntityName:` (page 59)

clientInfoForRecordWithIdentifier:

Returns a client-specific, nonsynchronized object that stores additional information about a record specified by *recordIdentifier*.

```
- (id)clientInfoForRecordWithIdentifier:(NSString *)recordIdentifier
```

Discussion

Returns `nil` if the record has no client information or doesn't exist.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- setClientInfo:forRecordWithIdentifier:` (page 57)

clientLostRecordWithIdentifier:shouldReplaceOnNextSync:

Tells the sync engine that a record identified by *recordIdentifier*, no longer exists on the client, and indicates whether or not it should be replaced.

```
- (void)clientLostRecordWithIdentifier:(NSString *)recordIdentifier
    shouldReplaceOnNextSync:(BOOL)flag
```

Discussion

If this method is invoked during the pushing state and *flag* is `YES`, then the record is added during the subsequent pulling state in the same sync session. However, if this method is invoked during the pulling state (pushing has already taken place), it is added the next time the client syncs.

If *flag* is `NO`, the sync engine treats the record as if it had been filtered and does not send any more changes for the record. If invoked during the pulling state, this is equivalent to refusing a record using the `clientRefusedChangesForRecordWithIdentifier:` (page 53) method.

Use this method if you inadvertently deleted a record when pushing or pulling changes. For example, use this method if the contacts on a phone device are full and the user must select a record to delete in order to add a record. The user's only choice is to delete an existing record on the phone but they don't want the record deleted from the Address Book on their computer.

You can use this method during the pulling and pushing state.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- deleteRecordWithIdentifier:` (page 54)
`- clientAcceptedChangesForRecordWithIdentifier:formattedRecord:newRecordIdentifier:` (page 50)

clientRefusedChangesForRecordWithIdentifier:

Informs the sync engine during the pulling state that the client has refused to apply the changes for the record specified by *`recordIdentifier`*.

```
- (void)clientRefusedChangesForRecordWithIdentifier:(NSString *)recordIdentifier
```

Discussion
This method applies only to add and modify changes, not deletes. After invoking this method, the sync engine does not send the same change during any subsequent syncs unless the record is modified. Refusing a record does not change the local identifier mapping for the client. Invoking this method does not affect other clients participating in the same sync session.

Note that if a client refresh syncs, the entire client store is wiped out, so any previously refused records are reapplied to the client. Use filtering if you want to permanently ignore some records. Do not use this method if you want to refuse deletes. Instead push the records that you want to keep during the next sync session.

Use this method during the pulling state only, otherwise an exception is raised.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- clientAcceptedChangesForRecordWithIdentifier:formattedRecord:newRecordIdentifier:` (page 50)
`- clientCommittedAcceptedChanges` (page 51)
`- clientLostRecordWithIdentifier:shouldReplaceOnNextSync:` (page 52)

clientWantsToPushAllRecordsForEntityNames:

Forces a slow sync of all the records for the specified entities.

```
- (void)clientWantsToPushAllRecordsForEntityNames:(NSArray *)entityNames
```

Discussion
The `entityNames` array can contain a subset of the supported entity names. A **slow sync** is when the client pushes all of its records to the sync engine, and the sync engine determines, by comparing records, what changes need to be applied and pushed to the client. By default, the sync engine assumes the client is **fast syncing** and expects the client to push only the changes to records since the last sync (added, modified, and deleted records). Use this method if you want to change this default behavior by forcing a slow sync. For example, if the client can't determine what records changed.

Use this method during the negotiation state only, otherwise an exception is raised.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- shouldPushAllRecordsForEntityName:` (page 58)
`- shouldPushChangesForEntityName:` (page 58)

deleteRecordWithIdentifier:

Creates a delete change for the record specified by *`recordIdentifier`* and pushes the change to the sync engine.

```
- (BOOL)deleteRecordWithIdentifier:(NSString *)recordIdentifier
```

Discussion
Use this method during the negotiation state or pushing state only, otherwise an exception is raised. Invoking this method in the negotiation state transitions the receiver to the pushing state.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- pushChange:` (page 56)
`- pushChangesFromRecord:withIdentifier:` (page 57)
`- clientLostRecordWithIdentifier:shouldReplaceOnNextSync:` (page 52)
`- shouldPushAllRecordsForEntityName:` (page 58)

finishSyncing

Tells the sync engine that the client is done syncing. Invoking this method closes any open transactions in the pushing or pulling states.

```
- (void)finishSyncing
```

Discussion
You must invoke this method to cleanly terminate the session.

Use this method at any time but the receiver should be released soon afterwards since you can not continue using a finished session.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `cancelSyncing` (page 49)
- `isCancelled` (page 55)

isCancelled

Returns `YES` if the receiver was canceled, `NO` otherwise.

```
- (BOOL)isCancelled
```

Discussion
You can not continue using a canceled session.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `cancelSyncing` (page 49)
- `finishSyncing` (page 54)

prepareToPullChangesForEntityNames:beforeDate:

Returns `NO` if *`date`* expires before the sync engine is ready for the client to begin pulling changes, `YES` otherwise.

```
- (BOOL)prepareToPullChangesForEntityNames:(NSArray *)entityNames beforeDate:(NSDate
    *)date
```

Discussion
Moves the receiver to the mingling state and returns when the sync engine is ready for the client to begin pulling changes to the specified entities. Invoke this method after you have finished pushing changes to the sync engine and want to begin pulling changes. The `entityNames` array can contain a subset of the supported entity names. If this method returns `NO`, you can invoke it multiple times until it returns `YES` or the sync session is canceled or finished.

Use this method during the pushing state to transition to the mingling state, otherwise an exception is raised. The mingling state ends when this method returns `YES`.

Availability
Available in Mac OS X v10.4 and later.

See Also
- `prepareToPullChangesInBackgroundForEntityNames:target:selector:` (page 55)

prepareToPullChangesInBackgroundForEntityNames:target:selector:

Moves the receiver to the mingling state and sends *`selector`* to *`target`* when the sync engine is ready for the client to begin pulling changes to the specified entities.

```
- (void)prepareToPullChangesInBackgroundForEntityNames:(NSArray *)entityNames
   target:(id)target selector:(SEL)selector
```

Discussion

The `selector` method is passed two arguments: the first argument is the ISyncClient object and the second argument is the ISyncSession object. The `entityNames` array can contain a subset of the supported entity names. Use the `cancelSyncing` (page 49) method to cancel this method and the entire sync session. If the session is canceled, `selector` is sent to `target` passing `nil` as the ISyncSession object.

Use this method during the pushing state to transition to the mingling state, otherwise an exception is raised. The mingling state ends when `selector` is sent to `target`.

Note: ISyncSession is not thread-safe. You can pass an ISyncSession object between threads but you should not use it concurrently. Asynchronous callbacks from ISyncSession are delivered to any client thread that used any of the SyncServices API. The client is responsible for directing the callback to an appropriate thread.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- prepareToPullChangesForEntityNames:beforeDate:` (page 55)

pushChange:

Pushes changes made to a single record, specified by `change`, to the sync engine.

```
- (void)pushChange:(ISyncChange *)change
```

Discussion

A client can push only one ISyncChange object per record. The `change` object encapsulates an add, modify, or delete record change. If the change is an add or modify, the `change` object can contain changes to multiple properties including deleting properties. The change is not actually pushed to the sync engine until the sync session leaves the pushing state.

When slow syncing, a client should push add changes only. When fast syncing, a client should push only the delta changes since the last time the client synced. These changes may include new records, modified records, and deleted records.

You can also delete records without creating an ISyncChange object using the `deleteRecordWithIdentifier:` (page 54) method. Use the `pushChangesFromRecord:withIdentifier:` (page 57) method if your client knows a record changed but doesn't keep track of individual property changes.

Use this method during the negotiation or pushing state only, otherwise an exception is raised. Invoking this method during the negotiation state transitions to the pushing state.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- shouldPushAllRecordsForEntityName:` (page 58)
`- shouldPushChangesForEntityName:` (page 58)

pushChangesFromRecord:withIdentifier:

Compares *record* to the client's previous known state of the record, identified by *recordIdentifier*, and pushes the changes to the sync engine.

```
- (void)pushChangesFromRecord:(NSDictionary *)record withIdentifier:(NSString
    *)recordIdentifier
```

Discussion
Use this method if you know a record changed but don't know what properties changed. Otherwise, use the `pushChange:` (page 56) method to specify the exact property changes made to this record.

Use this method during the negotiation or pushing state only, otherwise an exception is raised. Invoking this method during the negotiation state transitions to the pushing state.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- shouldPushAllRecordsForEntityName:` (page 58)
`- shouldPushChangesForEntityName:` (page 58)

setClientInfo:forRecordWithIdentifier:

Associates a client-specific, non synchronized object, *clientInfo*, to a record specified by *recordIdentifier*.

```
- (void)setClientInfo:(id <NSCoding>)clientInfo forRecordWithIdentifier:(NSString
    *)recordId
```

Discussion
The *clientInfo* argument can be any object that conforms to the NSCoding protocol, and are deleted when the record is deleted. Pass `nil` for *clientInfo* to remove a previously set client information object. Use this method to store additional information with a record.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- clientInfoForRecordWithIdentifier:` (page 52)

shouldPullChangesForEntityName:

Returns `YES` if the client should pull changes to records for *entityName*, `NO` otherwise.

```
- (BOOL)shouldPullChangesForEntityName:(NSString *)entityName
```

Discussion
However, a return value of `YES`, doesn't imply the sync engine has changes for the entity. The sync engine doesn't know if there are any changes until after the mingling state. Use this method to determine if the client should try to pull changes for *entityName*. You can invoke this method at any time.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- shouldReplaceAllRecordsOnClientForEntityName:` (page 59)
`- clientDidResetEntityNames:` (page 51)

shouldPushAllRecordsForEntityName:

Returns `YES` if the client should push all the records for *`entityName`* to the sync engine, `NO` otherwise.

```
- (BOOL)shouldPushAllRecordsForEntityName:(NSString *)entityName
```

Discussion
For example, returns `YES` if you previously sent `clientWantsToPushAllRecordsForEntityNames:` (page 53) or `clientDidResetEntityNames:` (page 51)to the session to force a slow sync. This method also returns `YES` if the sync engine decides to slow sync *`entityName`*. If this method returns `NO`, the client should only push changes made since the last sync. You can invoke this method at any time.

Warning: If this method returns `YES` and the client does not push a record that the client was known to have on the last sync, the sync engine assumes the record was deleted and deletes it from the truth.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- shouldPushChangesForEntityName:` (page 58)
`- pushChange:` (page 56)
`- pushChangesFromRecord:withIdentifier:` (page 57)
`- deleteRecordWithIdentifier:` (page 54)

shouldPushChangesForEntityName:

Returns `YES` if the client should push changes to records for *`entityName`* since the last sync, `NO` otherwise.

```
- (BOOL)shouldPushChangesForEntityName:(NSString *)entityName
```

Discussion
For example, this method returns `NO` if you previously sent `setShouldReplaceClientRecords:forEntityNames:` (page 24) to the client for this session passing `YES` as the *`flag`* argument. If this method returns `NO`. the sync engine does not accept any changes from the client for this entity. You can invoke this method at any time.

Availability
Available in Mac OS X v10.4 and later.

See Also
`- shouldPushAllRecordsForEntityName:` (page 58)

`- clientWantsToPushAllRecordsForEntityNames:` (page 53)

shouldReplaceAllRecordsOnClientForEntityName:

Returns `YES` if the client should delete all the records for the entity, specified by *`entityName`*, and replace them with records pulled from the sync engine, `NO` otherwise.

```
- (BOOL)shouldReplaceAllRecordsOnClientForEntityName:(NSString *)entityName
```

Discussion

Send `setShouldReplaceClientRecords:forEntityNames:` (page 24) to the client for this session to request a different behavior.

You can invoke this method at any time. However, you should not delete the local client data until the session enters the pulling state (after `prepareToPullChangesForEntityNames:beforeDate:` (page 55) returns). Otherwise, if the session is canceled prematurely, the client may be left in an non synchronized state with no data.

Availability

Available in Mac OS X v10.4 and later.

See Also

`- shouldPullChangesForEntityName:` (page 57)
`- clientDidResetEntityNames:` (page 51)

snapshotOfRecordsInTruth

Returns an immutable snapshot of the records in the truth database.

```
- (ISyncRecordSnapshot *)snapshotOfRecordsInTruth
```

Discussion

Use this method if you are syncing and want a snapshot that is consistent with the sync session. Otherwise, you can create a snapshot at any time using the `snapshotOfRecordsInTruthWithEntityNames:usingIdentifiersForClient:` (page 36) ISyncRecordSnapshot method. Snapshots are useful if you want to compare records or perform queries.

Availability

Available in Mac OS X v10.4 and later.

Constants

This constant is a key used by a record pushed to the sync engine.

Constant	Description
`ISyncRecordEntityNameKey`	Each record pushed to the sync engine must have a value for this key that specifies the record's entity name. Available in Mac OS X v10.4 and later.

These are the exceptions that might be thrown by ISyncSession methods during a sync.

Constant	Description
`ISyncSession-CancelledException`	Thrown by any method if invoked after the session was canceled. Available in Mac OS X v10.4 and later.
`ISyncSession-UnavailableException`	Thrown if a session cannot be created, for example, if a client is already syncing. Available in Mac OS X v10.4 and later.
`ISyncInvalidEntityException`	Thrown if a client tries creating a session with an entity that does not exist. Available in Mac OS X v10.4 and later.
`ISyncUnsupported-EntityException`	Thrown if a client tries creating a session with an entity that exists but the client does not support. Available in Mac OS X v10.4 and later.
`ISyncInvalidRecordException`	Thrown if a client pushes a malformed record. Available in Mac OS X v10.4 and later.

These keys are used in the *`userInfo`* dictionary when a `ISyncInvalidRecordException` (page 60) exception is raised.

Constant	Description
`ISyncInvalidRecord-IdentifiersKey`	An array of the record identifiers that raised the exception. Available in Mac OS X v10.4 and later.
`ISyncInvalidRecord-ReasonsKey`	A dictionary where keys are the invalid record identifies and the values are the reasons for the exception. Available in Mac OS X v10.4 and later.
`ISyncInvalidRecordsKey`	A dictionary where the keys are the invalid record identifiers and the values are the property keys that raised the exception. Available in Mac OS X v10.4 and later.

P A R T I I

Protocols

ISyncFiltering

Conforms to:	NSCoding
Declared in:	SyncServices/ISyncFilter.h
Availability:	Available in Mac OS X v10.4 and later.
Companion guide:	Sync Services Programming Guide

Protocol Description

ISyncFiltering is a protocol implemented by objects that filter records for a client. A client can filter the records it pulls from the sync engine using these objects. Before a record is pulled by a client, the sync engine passes it to each filter associated with the client. A filter can then accept or reject the record. If a filter rejects a record, it is not passed to the client.

For example, a user might want to sync only contacts with phone numbers to their mobile phone. The filter for the phone client would examine each contact and reject it if the contact has no phone number.

Use the `setFilters:` (page 23) ISyncClient method to set the filters for a client. The sync engine archives the filters so that they persist after the client process terminates. Because the filters persist, they must conform to the NSCoding protocol. In addition, any process that loads the filters using the `filters` (page 20) ISyncClient method must have the classes for those filters. When the set of filters changes, all records for a client must be refiltered to determine if they need to be pulled during the next sync. This is a potentially computationally expensive operation, so only change filters when necessary.

Note

You can use the ISyncFilter class methods to combine multiple filters into a single filter using logical `AND` or `OR` binary operators on a set of filters.

Methods by Task

Testing for equality

`- isEqual:` (page 64)
Returns `YES` if the receiver and *`anotherFilter`* are equal, `NO` otherwise.

Getting supported entities

`- supportedEntityNames` (page 65)
Returns an array of entity names that this filter supports.

Filtering records

`- shouldApplyRecord:withRecordIdentifier:` (page 64)
Returns `YES` if the client should pull *`record`* uniquely identified by *`recordIdentifier`*, `NO` otherwise.

Instance Methods

isEqual:

Returns `YES` if the receiver and *`anotherFilter`* are equal, `NO` otherwise.

```
- (BOOL)isEqual:(id)anotherFilter
```

Discussion
When setting a filter using the `setFilters:` (page 23) ISyncClient method, the sync engine uses this method to compare the new filter with the previous filter (if there is one). If the filters are not equal, the sync engine recomputes all records that should be pushed to a client.

When setting filters, the sync engine compares the new filters with the old filters using `isEqual:`. If a filter has changed, the sync engine must refilter all the client's records—an expensive operation. Therefore, it's important that this implementation returns `NO` only if two filters differ in such a way that the records need to be refiltered.

Availability
Available in Mac OS X v10.4 and later.

shouldApplyRecord:withRecordIdentifier:

Returns `YES` if the client should pull *`record`* uniquely identified by *`recordIdentifier`*, `NO` otherwise.

```
- (BOOL)shouldApplyRecord:(NSDictionary *)record withRecordIdentifier:(NSString
    *)recordIdentifier
```

Discussion

This is the method that implements the actual filtering logic.

Availability

Available in Mac OS X v10.4 and later.

supportedEntityNames

Returns an array of entity names that this filter supports.

```
- (NSArray *)supportedEntityNames
```

Discussion

This filter is used only to filter records of the supported entities. An exception is raised if this method returns an empty array or `nil`.

Availability

Available in Mac OS X v10.4 and later.

PART III

Types and Constants

Constants

This chapter describes the types and constants found in the Sync Services:

Enumerations

ISyncChange—Change Types

```
typedef int ISyncChangeType;
enum __ISyncChangeType {
 ISyncChangeTypeAdd=1,
 ISyncChangeTypeModify,
 ISyncChangeTypeDelete
};
```

Discussion
These types are described in "ISyncChange" (page 9).

ISyncClient—Client Status

```
typedef int ISyncStatus;
enum __ISyncStatus {
 ISyncStatusRunning=1,
 ISyncStatusSuccess,
 ISyncStatusWarnings,
 ISyncStatusErrors,
 ISyncStatusCancelled,
 ISyncStatusFailed,
 ISyncStatusNever
};
```

Discussion
These types are described in "ISyncClient" (page 15).

Global Variables

ISyncChange—Property Keys

```
extern NSString * const ISyncChangePropertyActionKey;
extern NSString * const ISyncChangePropertySet;
extern NSString * const ISyncChangePropertyClear;
extern NSString * const ISyncChangePropertyNameKey;
extern NSString * const ISyncChangePropertyValueKey;
```

Discussion
These constants are defined in "ISyncChange" (page 9).

ISyncClient—Client Types

```
extern NSString * const ISyncClientTypeApplication;
extern NSString * const ISyncClientTypeDevice;
extern NSString * const ISyncClientTypeServer;
extern NSString * const ISyncClientTypePeer;
```

Discussion
These constants are defined in "ISyncClient" (page 15).

ISyncManager—Exceptions

```
extern NSString * const ISyncServerUnavailableException
```

Discussion
This constant is defined in "ISyncManager" (page 33).

ISyncSession—Exceptions

```
extern NSString * const ISyncSessionCancelledException;
extern NSString * const ISyncSessionUnavailableException;
extern NSString * const ISyncInvalidRecordException;
extern NSString * const ISyncInvalidRecordIdentifiersKey;
extern NSString * const ISyncInvalidRecordReasonsKey;
extern NSString * const ISyncInvalidRecordsKey;
extern NSString * const ISyncInvalidEntityException;
extern NSString * const ISyncUnsupportedEntityException;
```

Discussion
These constants are defined in "ISyncSession" (page 43).

ISyncSession—Keys

```
extern NSString * const ISyncRecordEntityNameKey;
```

Discussion

These constants are defined in "ISyncSession" (page 43).

Index

B

beginSessionInBackgroundWithClient:entityNames: target:selector: class method 47
beginSessionWithClient:entityNames:beforeDate: class method 47

C

cancelPreviousBeginSessionWithClient: class method 48
cancelSyncing instance method 49
canPullChangesForEntityName: instance method 18
canPushChangesForEntityName: instance method 19
changeEnumeratorForEntityNames: instance method 49
changes instance method 11
changeWithType:recordIdentifier:changes: class method 10
clientAcceptedChangesForRecordWithIdentifier: formattedRecord:newRecordIdentifier: instance method 50
clientChangedRecordIdentifiers: instance method 51
clientCommittedAcceptedChanges instance method 51
clientDidResetEntityNames: instance method 51
clientIdentifier instance method 19
clientInfoForRecordWithIdentifier: instance method 52
clientLostRecordWithIdentifier: shouldReplaceOnNextSync: instance method 52
clientRefusedChangesForRecordWithIdentifier: instance method 53
clientType instance method 19
clientWantsToPushAllRecordsForEntityNames: instance method 53
clientWithIdentifier: instance method 35

D

deleteRecordWithIdentifier: instance method 54
displayName instance method 19

E

enabledEntityNames instance method 20

F

filterMatchingAllFilters: class method 32
filterMatchingAtLeastOneFilter: class method 32
filters instance method 20
finishSyncing instance method 54

I

imagePath instance method 20
initWithChangeType:recordIdentifier:changes: instance method 11
isCancelled instance method 55
isEnabled instance method 35
isEnabledForEntityName: instance method 21
isEqual: protocol instance method 64
ISyncAvailabilityChangedNotification notification 38
ISyncChangePropertyActionKey constant 13
ISyncChangePropertyClear constant 13
ISyncChangePropertyNameKey constant 13
ISyncChangePropertySet constant 13
ISyncChangePropertyValueKey constant 13
ISyncChangeTypeAdd constant 13
ISyncChangeTypeDelete constant 13
ISyncChangeTypeModify constant 13
ISyncChange—Change Types 69

ISyncChange—Property Keys 70
ISyncClientTypeApplication constant 28
ISyncClientTypeDevice constant 28
ISyncClientTypePeer constant 28
ISyncClientTypeServer constant 28
ISyncClient—Client Status 69
ISyncClient—Client Types 70
ISyncInvalidEntityException constant 60
ISyncInvalidRecordException constant 60
ISyncInvalidRecordIdentifiersKey constant 60
ISyncInvalidRecordReasonsKey constant 60
ISyncInvalidRecordsKey constant 60
ISyncManager—Exceptions 70
ISyncRecordEntityNameKey constant 60
ISyncServerUnavailableException constant 38
ISyncSessionCancelledException constant 60
ISyncSessionUnavailableException constant 60
ISyncSession—Exceptions 70
ISyncSession—Keys 70
ISyncStatusCancelled constant 28
ISyncStatusErrors constant 28
ISyncStatusFailed constant 28
ISyncStatusNever constant 29
ISyncStatusRunning constant 28
ISyncStatusSuccess constant 28
ISyncStatusWarnings constant 28
ISyncUnsupportedEntityException constant 60

L

lastSyncDateForEntityName: instance method 21
lastSyncStatusForEntityName: instance method 21

O

objectForKey: instance method 22

P

prepareToPullChangesForEntityNames:beforeDate: instance method 55
prepareToPullChangesInBackgroundForEntityNames: target:selector: instance method 55
pushChange: instance method 56
pushChangesFromRecord:withIdentifier: instance method 57

R

record instance method 11
recordIdentifier instance method 12
recordsWithIdentifiers: instance method 40
recordsWithMatchingAttributes: instance method 41
registerClientWithIdentifier:descriptionFilePath: instance method 35
registerSchemaWithBundlePath: instance method 36

S

setClientInfo:forRecordWithIdentifier: instance method 57
setDisplayName: instance method 22
setEnabled:forEntityNames: instance method 22
setFilters: instance method 23
setImagePath: instance method 23
setObject:forKey: instance method 23
setShouldReplaceClientRecords:forEntityNames: instance method 24
setShouldSynchronize:withClientsOfType: instance method 24
setSyncAlertHandler:selector: instance method 25
setSyncAlertToolPath: instance method 26
sharedManager class method 34
shouldApplyRecord:withRecordIdentifier: protocol instance method 64
shouldPullChangesForEntityName: instance method 57
shouldPushAllRecordsForEntityName: instance method 58
shouldPushChangesForEntityName: instance method 58
shouldReplaceAllRecordsOnClientForEntityName: instance method 59
shouldReplaceClientRecordsForEntityName: instance method 27
shouldSynchronizeWithClientsOfType: instance method 27
snapshotOfRecordsInTruth instance method 59
snapshotOfRecordsInTruthWithEntityNames: usingIdentifiersForClient: instance method 36
sourceIdentifiersForRelationshipName: withTargetIdentifier: instance method 41
supportedEntityNames instance method 27
supportedEntityNames protocol instance method 65
syncAlertToolPath instance method 27

T

`targetIdentifiersForRelationshipName:withSourceIdentifier:` instance method 41
`type` instance method 12

U

`unregisterClient:` instance method 37
`unregisterSchemaWithName:` instance method 37