ISyncSession Class Reference

Cocoa > Syncing

2008-11-19

Contents

ISyncSession Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/SyncServices.framework
Availability	Available in Mac OS X v10.4 and later.
Companion guide	Sync Services Programming Guide
Declared in	ISyncSession.h
Related sample code	People SeeMyFriends StickiesExample

Overview

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.

You should use sync anchors so your application is more resilient by avoiding serious errors that can occur during a sync session—such as a communication failure between a client and its data store that corrupts data. Use sync anchors unless you implement your own mechanism for tracking whether records are successfully pushed and pulled. A sync anchor is an object that is unique per client and per entity that is saved periodically throughout a sync session. The sync engine compares the client's locally stored sync anchors with its copies to determine the next sync mode. For example, if there is a discrepancy in sync anchors and the client is an application, an alert panel appears asking the user to select an appropriate sync mode. If the client is not an application, it slow syncs.

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 definitions of other sync terms.

Use the `beginSessionWithClient:entityNames:beforeDate:lastAnchors:` (page 13) method to create an `ISyncSession` object for the specified client and entities. Alternatively, use the `beginSessionInBackgroundWithClient:entityNames:target:selector:lastAnchors:` (page 10) 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 22) or the `clientDidResetEntityNames:` (page 19) method to request a sync mode different from the default mode.

During the pushing state, use the `pushChange:` (page 25) method to push just the changes for a particular record. Otherwise, use the `pushChangesFromRecord:withIdentifier:` (page 26) method to push the entire record and let the sync engine figure out which properties changed. Use the `deleteRecordWithIdentifier:` (page 22) method to push a delete change. After successfully pushing your entities, use the `clientFinishedPushingChangesWithNextAnchors:` (page 20) method to change the sync anchors stored with the sync engine.

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 15) 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 16) method to accept changes and the `clientRefusedChangesForRecordWithIdentifier:` (page 21) method to reject changes. Use the `clientCommittedAcceptedChangesWithNextAnchors:` (page 18) method to commit the accepted changes and close a transaction within the pulling state.

Use `finishSyncing` (page 23) to terminate a sync session by closing an open transaction. Or use `cancelSyncing` (page 15) to cancel a sync session that 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 23) and `cancelSyncing` (page 15) 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.

If you are using Mac OS X v10.4 or earlier, use the `beginSessionWithClient:entityNames:beforeDate:` (page 12) method to create an `ISyncSession` object for the specified client and entities. Alternatively, use the `beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 9) method if you don't want to block when creating a session. Similarly, use the non-anchor `clientCommittedAcceptedChanges` (page 18) method to commit the accepted changes and close a transaction within the pulling state.

Tasks

Creating a Sync Session

`+ beginSessionWithClient:entityNames:beforeDate:lastAnchors:` (page 13)
: Creates and returns a new sync session for the specified client using sync anchors.

`+ beginSessionInBackgroundWithClient:entityNames:target:selector:lastAnchors:` (page 10)
: Creates a new sync session for the specified client asynchronously using sync anchors.

`+ beginSessionWithClient:entityNames:beforeDate:` (page 12)
: Creates and returns a new sync session for the specified client.

`+ beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 9)
: Creates a new sync session for the specified client asynchronously.

`+ cancelPreviousBeginSessionWithClient:` (page 14)
: Cancels a previous request to create a session using `beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 9) for *client*.

Negotiating a Sync Mode

`- clientDidResetEntityNames:` (page 19)
: Tells the sync engine to perform a refresh sync of all the records for the specified entities.

`- clientWantsToPushAllRecordsForEntityNames:` (page 22)
: Forces a slow sync of all the records for the specified entities.

`- shouldPushChangesForEntityName:` (page 28)
: Returns `YES` if the client should push changes to records for *entityName* since the last sync, `NO` otherwise.

`- shouldPushAllRecordsForEntityName:` (page 27)
: Returns `YES` if the client should push all the records for *entityName* to the sync engine; otherwise, `NO`.

`- shouldPullChangesForEntityName:` (page 27)
: Returns `YES` if the client should pull changes to records for *entityName*, `NO` otherwise.

`- shouldReplaceAllRecordsOnClientForEntityName:` (page 28)
: 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 25)
: Pushes changes made to a single record, specified by *change*, to the sync engine.

`- pushChangesFromRecord:withIdentifier:` (page 26)
: 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 22)
: Creates a delete change for the record specified by *`recordIdentifier`* and pushes the change to the sync engine.

`- clientFinishedPushingChangesWithNextAnchors:` (page 20)
: Sets the sync anchors for each entity whose records were successfully pushed by the client.

Mingling

`- prepareToPullChangesForEntityNames:beforeDate:` (page 24)
: 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.

`- prepareToPullChangesInBackgroundForEntityNames:target:selector:` (page 25)
: Moves the receiver to the mingling state and sends a message to a specified target when the sync engine is ready for the client to begin pulling changes to the specified entities.

Pulling Changes

`- changeEnumeratorForEntityNames:` (page 15)
: 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 16)
: Informs the sync engine that the client has accepted the changes to the record identified by *`recordIdentifier`* during the pulling state.

`- clientRefusedChangesForRecordWithIdentifier:` (page 21)
: 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 18)
: Informs the sync engine that all accepted and rejected changes in the current transaction during the pulling state should be committed.

`- clientCommittedAcceptedChangesWithNextAnchors:` (page 18)
: Sets the sync anchors for each entity whose records were successfully updated during the pulling phase.

`- clientChangedRecordIdentifiers:` (page 17)
: Changes the record identifiers of the given records.

Pushing and Pulling Changes

`- clientLostRecordWithIdentifier:shouldReplaceOnNextSync:` (page 21)
: 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 23)
: Tells the sync engine that the client is done syncing. Invoking this method closes any open transactions in the pushing or pulling states.

Canceling Syncing

`- isCancelled` (page 23)
: Returns `YES` if the receiver was canceled, `NO` otherwise.

`- cancelSyncing` (page 15)
: Cancels the current session.

Getting and Setting Client Information

`- clientInfoForRecordWithIdentifier:` (page 20)
: Returns a client-specific, nonprepsynchronized object that stores additional information about a record specified by *`recordIdentifier`*.

`- setClientInfo:forRecordWithIdentifier:` (page 27)
: Associates a client-specific, non synchronized object, *`clientInfo`*, to a record specified by *`recordIdentifier`*.

Getting Snapshots

`- snapshotOfRecordsInTruth` (page 29)
: 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 asynchronously.

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

Parameters

`client`
: The client that is syncing.

entityNames

An array of entity names that the client wants to sync.

The *entityNames* parameter can be a subset of the client's supported entities and may include entities that 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` to your `ISyncClient` object as the *entityNames* parameter to this method.

target

The recipient of *selector*.

selector

The message to send to *target*.

The *selector* message is passed two parameters: The first parameter is the `ISyncClient` object and the second parameter is the new `ISyncSession` object. If the sync engine is disabled or another client already created a session for this client, then this method fails to create a session and *selector* is sent to *target* with `nil` as the `ISyncSession` object.

Discussion

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 12) by not blocking—returning immediately—and sending *selector* to *target* when all dependent clients have joined the sync session. (Send `setShouldSynchronize:withClientsOfType:` 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 Sync Services methods. 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 14)

`– isEnabled` (ISyncManager)

Related Sample Code

People

SeeMyFriends

Declared In

`ISyncSession.h`

beginSessionInBackgroundWithClient:entityNames:target:selector:lastAnchors:

Creates a new sync session for the specified client asynchronously using sync anchors.

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

Parameters

client

The client that is syncing.

entityNames

An array of entity names that the client wants to sync.

The *entityNames* parameter can be a subset of the client's supported entities and may include entities that 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` to your `ISyncClient` object as the *entityNames* parameter to this method.

target

The recipient of *selector*.

selector

The message to send to *target*.

The *selector* message is passed two parameters: the first parameter is the `ISyncClient` object and the second parameter is the new `ISyncSession` object. If the sync engine is disabled or another client already created a session for this client, then this method fails to create a session and *selector* is sent to *target* with `nil` as the `ISyncSession` object.

anchors

The sync anchors used in the last sync which are compared to those saved by the sync engine to determine the sync mode.

The keys are the entity names and the values are the sync anchors. Sync anchors are globally unique `NSString` objects, typically containing a UUID or date. If this parameter is `nil`, the client refresh syncs.

As a convenience, you may use the same sync anchor for all entities of a data class by specifying a sync anchor for only one entity in that data class. If you provide sync anchors for two or more entities per data class, then you need to specify sync anchors for all entities in that data class. A missing sync anchor for an entity causes that entity to refresh sync.

Discussion

This method is the nonblocking variation of the `beginSessionWithClient:entityNames:beforeDate:lastAnchors:` (page 13) method similar to the nonblocking `beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 9) method that doesn't use sync anchors. Read the `beginSessionWithClient:entityNames:beforeDate:lastAnchors:` (page 13) description for more details.

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 Sync Services methods. The client is responsible for directing the callback to an appropriate thread.

Availability

Available in Mac OS X v10.5 and later.

See Also

`+ beginSessionWithClient:entityNames:beforeDate:lastAnchors:` (page 13)

`- clientFinishedPushingChangesWithNextAnchors:` (page 20)

`- clientCommittedAcceptedChangesWithNextAnchors:` (page 18)

Declared In
`ISyncSession.h`

beginSessionWithClient:entityNames:beforeDate:

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

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

Parameters

`client`

The client that is syncing.

`entityNames`

An array of entity names that the client wants to sync.

The *`entityNames`* parameter can be a subset of the client's supported entities and may include entities that 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` to your `ISyncClient` object as the *`entityNames`* parameter to this method.

`date`

How long the 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 cannot 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 by passing `[NSDate date]`.

Return Value
Returns the new sync session or `nil` if the session cannot be created immediately.

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`*. Returns `nil` if the sync engine is disabled.

Discussion
Creating a session for *`client`* may trigger notifications to other clients observing syncs of this client type. Other clients then have the opportunity to join this sync session; therefore, this method may block. Send `setShouldSynchronize:withClientsOfType:` to an `ISyncClient` object to setup these dependencies. Use the `beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 9) 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 14)
`- isEnabled` (ISyncManager)

Related Sample Code
StickiesExample

Declared In
`ISyncSession.h`

beginSessionWithClient:entityNames:beforeDate:lastAnchors:

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

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

Parameters

`client`

The client that is syncing.

`entityNames`

An array of entity names that the client wants to sync.

The *`entityNames`* parameter can be a subset of the client's supported entities and may include entities that 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` to your `ISyncClient` object as the *`entityNames`* parameter to this method.

`date`

How long the 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 cannot 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 by passing `[NSDate date]`.

`anchors`

Specifies the sync anchors used in the last sync which are compared to those saved by the sync engine to determine the sync mode.

The keys are the entity names and the values are the sync anchors. Sync anchors are globally unique `NSString` objects typically, containing a UUID or date. If this parameter is `nil`, the client refresh syncs.

As a convenience, you may use the same sync anchor for all entities of a data class by specifying a sync anchor for only one entity in that data class. If you provide sync anchors for two or more entities per data class, then you need to specify sync anchors for all entities in that data class. A missing sync anchor for an entity causes that entity to refresh sync.

Return Value
Returns the new sync session or `nil` if the session cannot be created immediately.

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

Discussion

Use this method instead of `beginSessionWithClient:entityNames:beforeDate:` (page 12) if you are using sync anchors to improve the selection of a sync mode—that is, fast sync when possible and avoid unnecessary slow syncs. Use the `beginSessionInBackgroundWithClient:entityNames:target:selector:lastAnchors:` (page 10) method if you don't want to block when creating a session.

If the client is syncing an entity for the first time, the sync anchor for the entity should be `[NSNull null]` which causes a refresh sync. A null value for a sync anchor can also indicate that local records for the corresponding entity were removed or lost, and consequently, the client needs to refresh sync.

Otherwise, the sync anchors parameter should be the sync anchors you saved locally and passed to either the `clientFinishedPushingChangesWithNextAnchors:` (page 20) or `clientCommittedAcceptedChangesWithNextAnchors:` (page 18) methods during the last sync session.

The sync anchors you provide in this method are compared to the sync anchors from the last sync session to determine the sync mode. If a sync anchor doesn't match a sync anchor from the previous sync session and the client is an application, an alert panel appears asking the user to select an appropriate sync mode. If the user selects slow sync or the client is not an application, then the `shouldPushAllRecordsForEntityName:` (page 27) method returns `YES` for all entities in a data class corresponding to that sync anchor.

If you use this method you must also use the `clientFinishedPushingChangesWithNextAnchors:` (page 20) and `clientCommittedAcceptedChangesWithNextAnchors:` (page 18) methods to set sync anchors at the end of the pushing and pulling phases of the sync session. Otherwise, the sync engine assumes an error occurred and the client refresh syncs.

Availability

Available in Mac OS X v10.5 and later.

See Also

`+ beginSessionInBackgroundWithClient:entityNames:target:selector:lastAnchors:` (page 10)

`- clientFinishedPushingChangesWithNextAnchors:` (page 20)

`- clientCommittedAcceptedChangesWithNextAnchors:` (page 18)

Declared In

`ISyncSession.h`

cancelPreviousBeginSessionWithClient:

Cancels a previous request to create a session using `beginSessionInBackgroundWithClient:entityNames:target:selector:` (page 9) for *`client`*.

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

Discussion

Use the `beginSessionWithClient:entityNames:beforeDate:` (page 12) 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 9)
`+ beginSessionWithClient:entityNames:beforeDate:` (page 12)

Declared In
`ISyncSession.h`

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 22) 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 23)
`- finishSyncing` (page 23)

Related Sample Code
People

Declared In
`ISyncSession.h`

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`* parameter 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 16) to the sync session, and reject a change by sending `clientRefusedChangesForRecordWithIdentifier:` (page 21) to the sync session. After processing all changes (saving them on the device or locally) you send `clientCommittedAcceptedChanges` (page 18) 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 24)

Related Sample Code
People
SeeMyFriends

Declared In
`ISyncSession.h`

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 cannot store properties using the defined schema, then you can use the *`formattedRecord`* parameter 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 that the client stores and pushes. This may be a subset of the entity properties if a client doesn't use the omitted properties. This dictionary should include the `ISyncRecordEntityNameKey` (page 29) key identifying the record's entity name. Otherwise, the *`formattedRecord`* parameter 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 containing the truncated values when invoking this method. 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`* parameter to request that your identifier be used in future communications with the sync engine. Although conflicts can occur when changing record identifiers that are targets of relationships pulled in the same sync session. Read Syncing Relationships in *Sync Services Programming Guide* for more information on resolving pulled relationships.

You can use this method to batch up changes by invoking it repeatedly. You can also use the `clientRefusedChangesForRecordWithIdentifier:` (page 21) method to reject changes. However, after a sequence of accepting and rejecting changes, you need to invoke `clientCommittedAcceptedChanges` (page 18) 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 17)
- `clientLostRecordWithIdentifier:shouldReplaceOnNextSync:` (page 21)

Related Sample Code
People
SeeMyFriends

Declared In
`ISyncSession.h`

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 16) to the session. This method can be invoked when pushing or pulling records.

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

Related Sample Code
StickiesExample

Declared In
`ISyncSession.h`

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 16) to the sync session, and reject a change by sending `clientRefusedChangesForRecordWithIdentifier:` (page 21) to the sync session.

Invoking this method ends an open pull transaction that began by sending either `prepareToPullChangesForEntityNames:beforeDate:` (page 24)or `prepareToPullChangesInBackgroundForEntityNames:target:selector:` (page 25) 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 15)
`– finishSyncing` (page 23)

Related Sample Code
People
SeeMyFriends

Declared In
`ISyncSession.h`

clientCommittedAcceptedChangesWithNextAnchors:

Sets the sync anchors for each entity whose records were successfully updated during the pulling phase.

```
- (void)clientCommittedAcceptedChangesWithNextAnchors:(NSDictionary *)anchors
```

Parameters

`anchors`
: The sync anchors for the entities that were successfully pulled from the sync engine.

  The keys are the entity names and the values are the sync anchors. Sync anchors are globally unique `NSString` objects, typically containing a UUID or date. If this parameter is `nil`, the client refresh syncs.

  As a convenience, you may use the same sync anchor for all entities of a data class by specifying a sync anchor for only one entity in that data class. If you provide sync anchors for two or more entities per data class, then you need to specify sync anchors for all entities in that data class. A missing sync anchor for an entity causes that entity to refresh sync.

Discussion
If you are using sync anchors, create a new sync anchor per entity or data class that is successfully pulled from the sync engine, store them locally, and invoke this method to register the new sync anchors with the sync engine. Then pass these new sync anchors to the sync engine at the beginning of the next sync by invoking either the `beginSessionWithClient:entityNames:beforeDate:lastAnchors:` (page 13) or `beginSessionInBackgroundWithClient:entityNames:target:selector:lastAnchors:` (page 10) methods. An exception is raised if this method is invoked on a session that was not created using one of these `...lastAnchors:` methods.

Use the `clientFinishedPushingChangesWithNextAnchors:` (page 20) method to change the sync anchors for entities that are successfully pushed.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ beginSessionWithClient:entityNames:beforeDate:lastAnchors:` (page 13)
`+ beginSessionInBackgroundWithClient:entityNames:target:selector:lastAnchors:` (page 10)
`- clientFinishedPushingChangesWithNextAnchors:` (page 20)

Declared In
`ISyncSession.h`

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 27)
`- shouldReplaceAllRecordsOnClientForEntityName:` (page 28)

Related Sample Code
People

Declared In
`ISyncSession.h`

clientFinishedPushingChangesWithNextAnchors:

Sets the sync anchors for each entity whose records were successfully pushed by the client.

```
- (void)clientFinishedPushingChangesWithNextAnchors:(NSDictionary *)anchors
```

Parameters

anchors

The sync anchors for the entities that were successfully pushed to the sync engine.

The keys are the entity names and the values are the sync anchors. Sync anchors are globally unique `NSString` objects, typically containing a UUID or date. If this parameter is `nil`, the client refresh syncs.

As a convenience, you may use the same sync anchor for all entities of a data class by specifying a sync anchor for only one entity in that data class. If you provide sync anchors for two or more entities per data class, then you need to specify sync anchors for all entities in that data class. A missing sync anchor for an entity causes that entity to refresh sync.

Discussion

If you are using sync anchors, create a new sync anchor per entity or data class that is successfully pushed to the sync engine, store them locally, and invoke this method to register the new sync anchors with the sync engine. Then pass these new sync anchors to the sync engine at the beginning of the next sync by invoking either the `beginSessionWithClient:entityNames:beforeDate:lastAnchors:` (page 13) or `beginSessionInBackgroundWithClient:entityNames:target:selector:lastAnchors:` (page 10) methods. An exception is raised if this method is invoked on a session that was not created using one of these methods.

Use the `clientCommittedAcceptedChangesWithNextAnchors:` (page 18) method to change the sync anchors for entities that are successfully pulled.

Availability

Available in Mac OS X v10.5 and later.

See Also

`+ beginSessionWithClient:entityNames:beforeDate:lastAnchors:` (page 13)

`+ beginSessionInBackgroundWithClient:entityNames:target:selector:lastAnchors:` (page 10)

`- clientCommittedAcceptedChangesWithNextAnchors:` (page 18)

Declared In

`ISyncSession.h`

clientInfoForRecordWithIdentifier:

Returns a client-specific, nonprepsynchronized 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 27)

Declared In
`ISyncSession.h`

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 21) 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 22)
- `clientAcceptedChangesForRecordWithIdentifier:formattedRecord:newRecordIdentifier:` (page 16)

Declared In
`ISyncSession.h`

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 16)
`- clientCommittedAcceptedChanges` (page 18)
`- clientLostRecordWithIdentifier:shouldReplaceOnNextSync:` (page 21)

Declared In
`ISyncSession.h`

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 27)
`- shouldPushChangesForEntityName:` (page 28)

Related Sample Code
People

Declared In
`ISyncSession.h`

deleteRecordWithIdentifier:

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

```
- (void)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 25)
`- pushChangesFromRecord:withIdentifier:` (page 26)
`- clientLostRecordWithIdentifier:shouldReplaceOnNextSync:` (page 21)
`- shouldPushAllRecordsForEntityName:` (page 27)

Related Sample Code
People

Declared In
`ISyncSession.h`

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 cannot continue using a finished session.

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

See Also
`- cancelSyncing` (page 15)
`- isCancelled` (page 23)

Related Sample Code
People
SeeMyFriends
StickiesExample

Declared In
`ISyncSession.h`

isCancelled

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

```
- (BOOL)isCancelled
```

Discussion
You cannot continue using a canceled session.

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

See Also
`- cancelSyncing` (page 15)
`- finishSyncing` (page 23)

Declared In
`ISyncSession.h`

prepareToPullChangesForEntityNames:beforeDate:

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.

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

Parameters

entityNames
The entity names to use in the pulling phase. Can be a subset of the supported entity names.

date
The date/time that the client is willing to wait for the mingling process to complete.

Return Value
Returns `NO` if *date* expires before the sync engine is ready for the client to begin pulling changes, `YES` otherwise. If this method returns `NO`, the pushed changes and sync anchors are saved but not applied until the next time the client syncs. If this method returns `NO`, you can invoke it multiple times until it returns `YES` or the sync session is canceled or finished. If this method returns `YES`, the mingling state ends.

Discussion
Invoke this method after you have finished pushing changes to the sync engine and want to begin pulling changes.

This method raises an exception if validation errors occur when saving pushed changes or the sync session is canceled. Similar to this method returning `NO`, when an `ISyncSessionCancelledException` exception is raised, the pushed changes and anchors are saved but not applied until the next sync. If a validation exception is raised, the pushed changes and anchors are not saved. If you pass a zero delay, `[Date date]`, as the *date* parameter, you can use this method to validate your pushed changes.

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

See Also
`- prepareToPullChangesInBackgroundForEntityNames:target:selector:` (page 25)

Related Sample Code
SeeMyFriends

Declared In

`ISyncSession.h`

prepareToPullChangesInBackgroundForEntityNames:target:selector:

Moves the receiver to the mingling state and sends a message to a specified 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
```

Parameters

`entityNames`

The entity names to use in the pulling phase. Can be a subset of the supported entity names.

`target`

The object to send *`selector`* to when the mingling process is complete.

`selector`

The message to send to *`target`* when the mingling process is complete. The *`selector`* method is passed two parameters: The first parameter is the `ISyncClient` object and the second parameter is the `ISyncSession` object.

Discussion

Use this method during the pushing state to transition to the mingling state. The mingling state ends when *`selector`* is sent to *`target`*. Use the `cancelSyncing` (page 15) 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. This method may raise and exception before returning if a validation error occurs.

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 Sync Services methods. 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 24)

Declared In

`ISyncSession.h`

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 22) method. Use the `pushChangesFromRecord:withIdentifier:` (page 26) 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 27)
`– shouldPushChangesForEntityName:` (page 28)

Declared In
`ISyncSession.h`

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 25) 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 27)
`– shouldPushChangesForEntityName:` (page 28)

Related Sample Code
People

Declared In
`ISyncSession.h`

setClientInfo:forRecordWithIdentifier:

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

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

Discussion
The *`clientInfo`* parameter 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 20)

Declared In
`ISyncSession.h`

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 28)
`- clientDidResetEntityNames:` (page 19)

Related Sample Code
People

Declared In
`ISyncSession.h`

shouldPushAllRecordsForEntityName:

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

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

Discussion

For example, returns `YES` if you previously sent `clientWantsToPushAllRecordsForEntityNames:` (page 22) or `clientDidResetEntityNames:` (page 19)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 28)
- `pushChange:` (page 25)
- `pushChangesFromRecord:withIdentifier:` (page 26)
- `deleteRecordWithIdentifier:` (page 22)

Declared In

`ISyncSession.h`

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:` to the client for this session passing `YES` as the *`flag`* parameter. 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 27)
- `clientWantsToPushAllRecordsForEntityNames:` (page 22)

Declared In

`ISyncSession.h`

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:` 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 24) 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 27)
`- clientDidResetEntityNames:` (page 19)

Declared In
`ISyncSession.h`

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:` ISyncManager method. Snapshots are useful if you want to compare records or perform queries.

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

Declared In
`ISyncSession.h`

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.

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

Constant	Description
`ISyncSessionCancelledException`	Thrown by any method if invoked after the session was canceled.
`ISyncSessionUnavailableException`	Thrown if a session cannot be created, for example, if a client is already syncing.
`ISyncInvalidEntityException`	Thrown if a client tries creating a session with an entity that does not exist.
`ISyncUnsupportedEntityException`	Thrown if a client tries creating a session with an entity that exists but the client does not support.
`ISyncInvalidRecordException`	Thrown if a client pushes a malformed record.

These keys are used in the *`userInfo`* dictionary when a `ISyncInvalidRecordException` (page 30) exception is raised.

Constant	Description
`ISyncInvalidRecord-IdentifiersKey`	An array of the record identifiers that raised the exception.
`ISyncInvalidRecordReasonsKey`	A dictionary where keys are the invalid record identifies and the values are the reasons for the exception.
`ISyncInvalidRecordsKey`	A dictionary where the keys are the invalid record identifiers and the values are the property keys that raised the exception.

Document Revision History

This table describes the changes to *ISyncSession Class Reference*.

Date	Notes
2008-11-19	Corrected typographical errors.
2007-07-11	Updated for Mac OS X v10.5. First publication of this content as a separate document. Fixed return type for the deleteRecordWithIdentifier: method. Added new creation and anchor methods.

Index