Game Kit Framework Reference

Developer

Contents

Tables

Introduction

Framework	/System/Library/Frameworks/GameKit.framework
Header file directories	/System/Library/Frameworks/GameKit.framework/Headers
Companion guide	Game Center Programming Guide
Declared in	GKAchievement.h GKAchievementDescription.h GKAchievementViewController.h GKChallenge.h GKChallengeEventHandler.h GKError.h GKFriendRequestComposeViewController.h GKGameCenterViewController.h GKLeaderboard.h GKLeaderboardViewController.h GKLocalPlayer.h GKMatch.h GKMatchmaker.h GKMatchmakerViewController.h GKNotificationBanner.h GKPeerPickerController.h GKPlayer.h GKPublicConstants.h GKPublicProtocols.h GKScore.h GKSession.h GKSessionError.h GKTurnBasedMatch.h GKTurnBasedMatchmakerViewController.h GKVoiceChat.h GKVoiceChatService.h

Game Kit offers features that you can use to create great social games.

Game Kit Features

Game Kit provides three separate pieces of functionality:

- **Game Center** offers a centralized game service that connects players to each other. Game Center implements many different features:
  - **Friends** allow players to create anonymous online personas. Users connect to Game Center and interact with other players through an *alias*. Players can set status messages as well as mark other players as friends.
  - **Multiplayer** allows your game to create network matches that connect players through Game Center. Players can invite their friends or be connected to anonymous players. Most importantly, players can receive invitations to join a match even when your game is not running. Your game is running on each device and the instances of your game exchange match and voice data with each other.
  - **Turn-Based Gaming** provides store-and-forward network match infrastructure where the match is played out over a series of discrete turns. This kind of match can be played without requiring all of the players to be connected to Game Center simultaneously.
  - **Leaderboards** allow your game to store and fetch player scores from Game Center.
  - **Achievements** provide the ability to track a player's accomplishments in your game.
  - **Challenges** allow a player to challenge other players to complete an achievement or to beat a leaderboard score.

  Game Center is available on iOS and OS X.
- **Peer-to-peer connectivity** allows your game to create an ad hoc Bluetooth or wireless network between multiple iPhones in the same local area. Although designed with games in mind, this network is useful for any type of data exchange among users of your app. For example, an app could use peer-to-peer connectivity to share electronic business cards or other data. This functionality is only available on iOS. You can also get the same functionality using Game Center.
- **In-game voice chat** allows your game to provide voice communication between two iPhones. In-game voice relies on your game's network connection to another user to create its own network connection to transmit voice data. This functionality is only available on iOS. You can also get the same functionality using Game Center.

Classes

GKAchievement Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKAchievement.h
Companion guide	Game Center Programming Guide
Related sample code	GKAchievements GKTapper

Overview

Your game uses a `GKAchievement` object to communicate with Game Center about the local player's progress towards completing an achievement.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication see *Game Center Programming Guide*.

To request the local player's progress on achievements, call the `loadAchievementsWithCompletionHandler:` (page 18) class method.

To report the local player's progress on an achievement, your game updates the `percentComplete` (page 17) property of an achievement object and calls the object's `reportAchievementWithCompletionHandler:` (page 22) method. To report progress on multiple achievements at once, use the `reportAchievements:withCompletionHandler:` (page 19) class method.

By default, when an achievement is completed, Game Kit displays a notification banner to the player. If your game wants to display its own achievement user interface, you can prevent the banner from being displayed by setting the `showsCompletionBanner` (page 17) property of the achievement object to `NO` before reporting the player's progress.

To clear all progress the local player has made on achievements, call the `resetAchievementsWithCompletionHandler:` (page 20) class method.

Tasks

Retrieving Achievement Progress from Game Center

`+ loadAchievementsWithCompletionHandler:` (page 18)
Loads previously submitted achievement progress for the local player from Game Center.

Initializing an Achievement Object

`– initWithIdentifier:` (page 21)
Initializes a new achievement object.

Configuring an Achievement

`identifier` (page 16) *property*
A string used to uniquely identify the specific achievement the object refers to.

`percentComplete` (page 17) *property*
A percentage value that states how far the player has progressed on this achievement.

Reading the State of an Achievement

`completed` (page 15) *property*
A Boolean value that states whether the player has completed the achievement. (read-only)

`lastReportedDate` (page 17) *property*
The last time that progress on the achievement was successfully reported to Game Center. (read-only)

`hidden` (page 16) *property*

A Boolean value that states whether this achievement is normally kept secret from the player. (read-only) (Deprecated. Use the `hidden` property on the `GKAchievementDescription` class instead.)

Reporting Progress on an Achievement

`+ reportAchievements:withCompletionHandler:` (page 19)

Reports progress on an array of achievements.

`– reportAchievementWithCompletionHandler:` (page 22)

Reports the player's progress to Game Center.

Displaying a Notification Banner For an Achievement

`showsCompletionBanner` (page 17) *property*

A Boolean value that states whether a banner is displayed when the achievement is completed.

Resetting the Player's Progress on Achievements

`+ resetAchievementsWithCompletionHandler:` (page 20)

Resets all achievement progress for the local player.

Issuing Achievement Challenges

`– issueChallengeToPlayers:message:` (page 21)

Issue an achievement challenge to a list of players.

`– selectChallengeablePlayerIDs:withCompletionHandler:` (page 23)

Finds the subset of players that can earn an achievement.

Properties

completed

A Boolean value that states whether the player has completed the achievement. (read-only)

```
@property(nonatomic, readonly, getter=isCompleted) BOOL completed
```

Discussion
The value of this property is `YES` if the `percentComplete` property is equal to `100.0`; otherwise, it is `NO`.

Availability
Available in iOS 4.1 and later.

Related Sample Code
GKAchievements

Declared in
`GKAchievement.h`

hidden

A Boolean value that states whether this achievement is normally kept secret from the player. (read-only) (Deprecated in iOS 6.0. Use the `hidden` property on the `GKAchievementDescription` class instead.)

```
@property(nonatomic, assign, getter=isHidden, readonly) BOOL hidden
```

Availability
Available in iOS 4.1 and later.

Deprecated in iOS 6.0.

Declared in
`GKAchievement.h`

identifier

A string used to uniquely identify the specific achievement the object refers to.

```
@property(nonatomic, retain) NSString *identifier
```

Discussion
The `identifier` property must match the identifier string for an achievement you created for your game on iTunes Connect.

Availability
Available in iOS 4.1 and later.

Declared in
`GKAchievement.h`

lastReportedDate

The last time that progress on the achievement was successfully reported to Game Center. (read-only)

```
@property(nonatomic, retain, readonly) NSDate *lastReportedDate
```

Discussion

On a newly initialized achievement object, this property holds the current date.

Availability

Available in iOS 4.1 and later.

Declared in

`GKAchievement.h`

percentComplete

A percentage value that states how far the player has progressed on this achievement.

```
@property(nonatomic, assign) double percentComplete
```

Discussion

The default value for a newly initialized achievement object is `0.0`. The range of legal values is between `0.0` and `100.0`, inclusive.

Availability

Available in iOS 4.1 and later.

Related Sample Code

GKAchievements

GKTapper

Declared in

`GKAchievement.h`

showsCompletionBanner

A Boolean value that states whether a banner is displayed when the achievement is completed.

```
@property(nonatomic, assign) BOOL showsCompletionBanner
```

Discussion
When an achievement is completed and the value of this property is `YES`, a notification banner is displayed to the player to inform them of the completed achievement. If the value of this property is `NO`, there is no visual indication that the achievement is completed. Your game should set this property to `NO` only when it wants to provide its own visual indicator that the achievement was earned. The default value is `NO`.

Availability
Available in iOS 5.0 and later.

Declared in
`GKAchievement.h`

Class Methods

loadAchievementsWithCompletionHandler:

Loads previously submitted achievement progress for the local player from Game Center.

```
+ (void)loadAchievementsWithCompletionHandler:(void (^)(NSArray *achievements,
NSError *error))completionHandler
```

Parameters
`completionHandler`

A block to be called when the download is completed.

The block receives the following parameters:

achievements

An array of achievement objects that represents all progress reported to Game Center for the local player. If an error occurred, this parameter may be non-`nil`, in which case the array holds whatever achievement information Game Kit was able to fetch.

error

If an error occurred, this object describes the error. If the operation completed successfully, this value is `nil`.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 4.1 and later.

Related Sample Code
GKTapper

Declared in

`GKAchievement.h`

reportAchievements:withCompletionHandler:

Reports progress on an array of achievements.

```
+ (void)reportAchievements:(NSArray *)achievements withCompletionHandler:(void
(^)(NSError *error))completionHandler
```

Parameters

`achievements`

An array of achievement objects.

`completionHandler`

A block to be called after the operation completes.

The block takes the following parameter:

error

If the operation was successful, this value is `nil`; otherwise, this parameter holds an object that describes the problem that occurred.

Discussion

Use this class method whenever you need to submit multiple achievement updates at the same time. Calling this method reports each of the achievements, exactly as if you called the `reportAchievementWithCompletionHandler:` (page 22) method on each achievement object in the array. However, the entire operation can be processed more efficiently using this method, and the completion handler is only called once.

Availability

Available in iOS 6.0 and later.

Declared in
GKAchievement.h

resetAchievementsWithCompletionHandler:

Resets all achievement progress for the local player.

```
+ (void)resetAchievementsWithCompletionHandler:(void (^)(NSError
*error))completionHandler
```

Parameters
completionHandler

A block to be called when the reset action is completed.

The block takes the following parameter:

error

If the operation was successful, this value is nil; otherwise, this parameter holds an object that describes the problem that occurred.

Discussion
Calling this class method deletes all progress towards achievements previously reported for the local player. Hidden achievements that were previously visible are now hidden again.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability
Available in iOS 4.1 and later.

Related Sample Code
GKAchievements
GKTapper

Declared in
GKAchievement.h

Instance Methods

initWithIdentifier:

Initializes a new achievement object.

```
- (id)initWithIdentifier:(NSString *)identifier
```

Parameters

`identifier`

A string that matches the identifier string for an achievement you created for your game on iTunes Connect.

Return Value

An initialized achievement object, or `nil` if an error occurred.

Discussion

Your game initializes a new achievement object only when it has not previously reported progress for that achievement. If your game has previously reported progress on an achievement, you should first retrieve the current progress by calling the `loadAchievementsWithCompletionHandler:` (page 18) class method.

Availability

Available in iOS 4.1 and later.

Related Sample Code

GKAchievements

GKTapper

Declared in

`GKAchievement.h`

issueChallengeToPlayers:message:

Issue an achievement challenge to a list of players.

```
- (void)issueChallengeToPlayers:(NSArray *)playerIDs message:(NSString *)message
```

Parameters

`playerIDs`

The players to be challenged.

`message`

A text message to display to the challenged players.

Discussion
This method should be used only to implement your own custom challenge user interface. You should only issue challenges when the local player directs you to do so.

If the achievement is marked as hidden in iTunes Connect, or if the challenged player has already earned the achievement and it is not marked as replayable, then the challenge is not issued.

Availability
Available in iOS 6.0 and later.

Declared in
`GKChallenge.h`

reportAchievementWithCompletionHandler:

Reports the player's progress to Game Center.

```
- (void)reportAchievementWithCompletionHandler:(void (^)(NSError
*error))completionHandler
```

Parameters
`completionHandler`

A block to be called after the operation completes.

The block takes the following parameter:

error

If the operation was successful, this value is `nil`; otherwise, this parameter holds an object that describes the problem that occurred.

Discussion
When the player makes progress towards completing an achievement, your game communicates the player's progress to Game Center by calling this method. An achievement object is implicitly tied to the local player that was authenticated when the object was created; your game should only report progress when the same local player is still authenticated on the device.

> **Note:** To avoid using network bandwidth unnecessarily, only report an achievement when the player has made more progress towards completing it.

When the progress is successfully reported, the achievement is made visible if it was previously hidden. The `percentComplete` and `lastReportedDate` property values stored on Game Center are updated if the new `percentComplete` value is greater than the value previously stored on Game Center. if the value of the `percentComplete` property was equal to `100.0`, then the achievement is marked as completed and a banner may be shown to the player.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. The background task automatically handles network errors, resending the data until the task completes. When the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability
Available in iOS 4.1 and later.

See Also
`@property hidden` (page 16)
`@property completed` (page 15)
`@property showsCompletionBanner` (page 17)

Related Sample Code
GKAchievements
GKTapper

Declared in
`GKAchievement.h`

selectChallengeablePlayerIDs:withCompletionHandler:

Finds the subset of players that can earn an achievement.

```
- (void)selectChallengeablePlayerIDs:(NSArray *)playerIDs withCompletionHandler:(void
 (^)(NSArray *challengeablePlayerIDs, NSError *error))completionHandler
```

Parameters
`playerIDs`

A list of players you want to check.

`completionHandler`

A block to be called when the download is completed.

The block receives the following parameters:

challengeablePlayerIDs

An array of player identifiers representing the players in the original array that are able to complete the challenge. If an error occurred, this parameter may be non-`nil`, in which case the array holds whatever achievement information Game Kit was able to fetch.

error

If an error occurred, this object describes the error. If the operation completed successfully, this value is `nil`.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 6.0 and later.

Declared in

`GKChallenge.h`

GKAchievementChallenge Class Reference

Inherits from	GKChallenge : NSObject
Conforms to	NSCoding (GKChallenge) NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 6.0 and later.
Declared in	GKChallenge.h
Companion guide	Game Center Programming Guide

Overview

A `GKAchievementChallenge` is a challenge to a player to complete a specific achievement.

Tasks

Obtaining the Achievement

`achievement` (page 25) *property*

The achievement the player must complete. (read-only)

Properties

achievement

The achievement the player must complete. (read-only)

```
@property(nonatomic, readonly, retain) GKAchievement *achievement
```

Availability

Available in iOS 6.0 and later.

Declared in

`GKChallenge.h`

GKAchievementDescription Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKAchievementDescription.h
Companion guide	Game Center Programming Guide

Overview

An `GKAchievementDescription` object holds text and images used to describe an achievement. When you develop your game, you create localized achievement descriptions in iTunes Connect. At runtime, your game retrieves these descriptions from Game Center. Usually your game downloads the achievement descriptions when it wants to present a custom achievement user interface to the player.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication see *Game Center Programming Guide*.

To retrieve a set of achievement description objects for your game, call the `loadAchievementDescriptionsWithCompletionHandler:` (page 33) class method. For performance reasons, achievement images are downloaded separately. To download am achievement description's completion image, call that object's `loadImageWithCompletionHandler:` (page 34) method.

The `incompleteAchievementImage` (page 32) and `placeholderCompletedAchievementImage` (page 33) class methods provide standard images your game can use to present achievement progress to the player.

Tasks

Retrieving Achievement Descriptions

`+ loadAchievementDescriptionsWithCompletionHandler:` (page 33)
Downloads the achievement descriptions from Game Center.

Reading and Writing Achievement Properties

`identifier` (page 30) *property*
A unique string used to identify the achievement. (read-only)

`title` (page 31) *property*
A localized title for the achievement. (read-only)

`unachievedDescription` (page 32) *property*
A localized description of the achievement to be used when the local player has not completed the achievement. (read-only)

`achievedDescription` (page 29) *property*
A localized description to be used after the local player has completed the achievement. (read-only)

`maximumPoints` (page 31) *property*
The number of points the player earns by completing this achievement. (read-only)

`image` (page 30) *property*
An image to display for the completed achievement. (read-only)

`hidden` (page 30) *property*
A Boolean value that states whether this achievement is initially visible to players. (read-only)

`replayable` (page 31) *property*
A Boolean value that states whether this achievement can be earned multiple times. (read-only)

Working with Achievement Images

`+ incompleteAchievementImage` (page 32)
A common image for incomplete achievements.

`+ placeholderCompletedAchievementImage` (page 33)
A common image for completed achievements.

– `loadImageWithCompletionHandler:` (page 34)

Loads the `image` property for a completed achievement.

Retrieving Group Information

`groupIdentifier` (page 29) *property*

The identifier for the group the achievement description is part of. (read-only)

Properties

achievedDescription

A localized description to be used after the local player has completed the achievement. (read-only)

```
@property(nonatomic, retain, readonly) NSString *achievedDescription
```

Availability
Available in iOS 4.1 and later.

Declared in
`GKAchievementDescription.h`

groupIdentifier

The identifier for the group the achievement description is part of. (read-only)

```
@property(nonatomic, retain, readonly) NSString *groupIdentifier
```

Discussion
If your game is configured to be part of a game group in iTunes Connect, this property holds the identifier you assigned to the achievement in the game group. If the game is not part of a game group, this property is `nil`.

Availability
Available in iOS 6.0 and later.

Declared in
`GKAchievementDescription.h`

hidden

A Boolean value that states whether this achievement is initially visible to players. (read-only)

```
@property(nonatomic, getter=isHidden, assign, readonly) BOOL hidden
```

Discussion
If the value of this property is `NO`, this achievement is always visible to the player. If `YES`, the achievement is not displayed in any of the standard achievement user interface screens. It remains hidden until the first time your game reports progress towards completing this achievement.

Availability
Available in iOS 4.1 and later.

Declared in
`GKAchievementDescription.h`

identifier

A unique string used to identify the achievement. (read-only)

```
@property(nonatomic, retain, readonly) NSString *identifier
```

Discussion
The `GKAchievementDescription` property holds the identifier string you created for the achievement on iTunes Connect.

Availability
Available in iOS 4.1 and later.

Declared in
`GKAchievementDescription.h`

image

An image to display for the completed achievement. (read-only)

```
@property(nonatomic, retain, readonly) UIImage *image
```

Discussion
The value of this property is undefined until after the image is loaded. See `loadImageWithCompletionHandler:` (page 34).

Availability
Available in iOS 4.1 and later.

Declared in
`GKAchievementDescription.h`

maximumPoints

The number of points the player earns by completing this achievement. (read-only)

```
@property(nonatomic, assign, readonly) NSInteger maximumPoints
```

Availability
Available in iOS 4.1 and later.

Declared in
`GKAchievementDescription.h`

replayable

A Boolean value that states whether this achievement can be earned multiple times. (read-only)

```
@property(nonatomic, getter=isReplayable, assign, readonly) BOOL replayable
```

Discussion
If the value of this property is `NO`, then the achievement may only be earned once. After the achievement is earned, Game Center ignores any further progress submitted for it. If the value of this property is `YES`, then the achievement is considered earned each time your game reports progress to Game Center that completes the achievement. This means that any appropriate banners are displayed to the player again, challenges based on the achievement are completed, and so on.

Availability
Available in iOS 6.0 and later.

Declared in
`GKAchievementDescription.h`

title

A localized title for the achievement. (read-only)

```
@property(nonatomic, retain, readonly) NSString *title
```

Availability
Available in iOS 4.1 and later.

Declared in
`GKAchievementDescription.h`

unachievedDescription

A localized description of the achievement to be used when the local player has not completed the achievement. (read-only)

```
@property(nonatomic, retain, readonly) NSString *unachievedDescription
```

Availability
Available in iOS 4.1 and later.

Declared in
`GKAchievementDescription.h`

Class Methods

incompleteAchievementImage

A common image for incomplete achievements.

```
+ (UIImage *)incompleteAchievementImage
```

Return Value
An image object.

Discussion
On OS X, this class method returns an `NSImage` object but otherwise works identically.

Availability
Available in iOS 4.1 and later.

Declared in
`GKAchievementDescription.h`

loadAchievementDescriptionsWithCompletionHandler:

Downloads the achievement descriptions from Game Center.

```
+ (void)loadAchievementDescriptionsWithCompletionHandler:(void (^)(NSArray
*descriptions, NSError *error))completionHandler
```

Parameters

`completionHandler`

A block to be called when the download is completed.

The block receives the following parameters:

descriptions

An array of description objects for the achievements in your game. If an error occurred, this value may be non-`nil`. In this case, the array holds whatever descriptions were downloaded by Game Kit before the error occurred.

error

If an error occurred, this error object describes the error. If the operation completed successfully, this value is `nil`.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 4.1 and later.

Declared in

`GKAchievementDescription.h`

placeholderCompletedAchievementImage

A common image for completed achievements.

```
+ (UIImage *)placeholderCompletedAchievementImage
```

Return Value

An image object.

Discussion

When an achievement is completed, your game can display this image until the custom image for an achievement finishes loading.

On OS X, this class method returns an `NSImage` object but otherwise works identically.

Availability

Available in iOS 4.1 and later.

See Also

– `loadImageWithCompletionHandler:` (page 34)

Declared in

`GKAchievementDescription.h`

Instance Methods

loadImageWithCompletionHandler:

Loads the `image` property for a completed achievement.

```
– (void)loadImageWithCompletionHandler:(void (^)(UIImage *image, NSError
*error))completionHandler
```

Parameters

`completionHandler`

A block to be called when the download is completed.

The block receives the following parameters:

image

The downloaded image. If an error occurred, this value is `nil`.

error

If an error occurred, this error object describes the error. If the operation completed successfully, this value is `nil`.

Discussion

Your game should call `loadImageWithCompletionHandler:` for each achievement the user has completed. Your game should display the placeholder image until the image is successfully downloaded. After the block is called, the description's `image` property holds the same image object that is returned to the block.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

On OS X, this class method returns an `NSImage` object but otherwise works identically.

Availability
Available in iOS 4.1 and later.

See Also
`+ placeholderCompletedAchievementImage` (page 33)

Declared in
`GKAchievementDescription.h`

GKAchievementViewController Class Reference

Inherits from	GKGameCenterViewController : UINavigationController : UIViewController : UIResponder : NSObject
Conforms to	NSCoding (UIViewController) UIAppearanceContainer (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKAchievementViewController.h
Companion guide	Game Center Programming Guide
Related sample code	GKAchievements GKTapper

Overview

An `GKAchievementViewController` object provides a standard user interface to display achievement progress for the local player. If the `GKGameCenterViewController` class is available, you should use it instead.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication, see *Game Center Programming Guide*.

To show achievements for the local player, initialize a new `GKAchievementViewController` object and set the delegate. Then present the new view controller and wait for the delegate to be called. Once the delegate is called, dismiss the view controller.

On iOS, you present and dismiss the view controller from another view controller in your game, using the methods provided by the `UIViewController` class. On OS X, you use the `GKDialogController` class to present and dismiss the view controller in a window.

Subclassing Notes

The `GKAchievementViewController` class is not intended to be subclassed.

Tasks

Setting the Delegate

`achievementDelegate` (page 37) *property*

The achievement view controller's delegate.

Properties

achievementDelegate

The achievement view controller's delegate.

```
@property(nonatomic, assign) id<GKAchievementViewControllerDelegate> achievementDelegate
```

Discussion

Your game must set the delegate before presenting the view controller.

Availability

Available in iOS 4.1 and later.

Related Sample Code
GKTapper

Declared in

`GKAchievementViewController.h`

GKChallenge Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 6.0 and later.
Declared in	GKChallenge.h
Companion guide	Game Center Programming Guide

Overview

A `GKChallenge` object represents a challenge issued by a player to another player.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication see *Game Center Programming Guide* .

Players use the Game Center app to issue and view challenges. However, your game can also customize its challenge behaviors in a number of ways:

- You can load the list of challenges issued to the local player by calling the `loadReceivedChallengesWithCompletionHandler:` (page 42) class method. For example, you might do this to display the challenges in your game's user interface.
- Your app can issue challenges using a `GKScore` or `GKAchievement` object. Your game should only issue challenges when the local player initiates the action in your user interface.
- Your game can be notified when new challenge events are received. See `GKChallengeEventHandlerDelegate`.

Subclassing Notes

You never subclass the `GKChallenge` class directly. However, subclasses of `GKChallenge` represent specific kinds of challenges. Two challenge types exist:

- A `GKScoreChallenge` is a challenge to beat a score the local player earned in a leaderboard.
- A `GKAchievementChallenge` is a challenge to complete an achievement that the local player has already completed.

Tasks

Retrieving the List of Challenges to the Local Player

`+ loadReceivedChallengesWithCompletionHandler:` (page 42)
Loads the list of outstanding challenges.

Examining Details About a Challenge

`issueDate` (page 40) *property*
The date the challenge was issued. (read-only)

`issuingPlayerID` (page 40) *property*
The player identifier for the player who issued the challenge. (read-only)

`receivingPlayerID` (page 41) *property*
The player identifier for the player who received the challenge. (read-only)

`message` (page 41) *property*
A text message that describes the challenge. (read-only)

`state` (page 41) *property*
The current state of the challenge. (read-only)

`completionDate` (page 40) *property*
The date the challenge was completed. (read-only)

Declining a Challenge

`- decline` (page 42)
Declines a challenge.

Properties

completionDate

The date the challenge was completed. (read-only)

```
@property(nonatomic, readonly, retain) NSDate *completionDate
```

Discussion
If the challenge is not complete, this value is `nil`.

Availability
Available in iOS 6.0 and later.

Declared in
`GKChallenge.h`

issueDate

The date the challenge was issued. (read-only)

```
@property(nonatomic, readonly, retain) NSDate *issueDate
```

Availability
Available in iOS 6.0 and later.

Declared in
`GKChallenge.h`

issuingPlayerID

The player identifier for the player who issued the challenge. (read-only)

```
@property(nonatomic, readonly, copy) NSString *issuingPlayerID
```

Availability
Available in iOS 6.0 and later.

Declared in
`GKChallenge.h`

message

A text message that describes the challenge. (read-only)

```
@property(nonatomic, readonly, copy) NSString *message
```

Availability
Available in iOS 6.0 and later.

Declared in
GKChallenge.h

receivingPlayerID

The player identifier for the player who received the challenge. (read-only)

```
@property(nonatomic, readonly, copy) NSString *receivingPlayerID
```

Availability
Available in iOS 6.0 and later.

Declared in
GKChallenge.h

state

The current state of the challenge. (read-only)

```
@property(nonatomic, readonly, assign) GKChallengeState state
```

Discussion
See "Challenge State" (page 43) for possible values.

Availability
Available in iOS 6.0 and later.

Declared in
GKChallenge.h

Class Methods

loadReceivedChallengesWithCompletionHandler:

Loads the list of outstanding challenges.

```
+ (void)loadReceivedChallengesWithCompletionHandler:(void (^)(NSArray *challenges,
 NSError *error))completionHandler
```

Parameters

`completionHandler`

A block to be called when the download is completed.

The block receives the following parameters:

challenges

An array of challenge objects that represents all challenges made to the local player. If an error occurred, this parameter may be non-`nil`, in which case the array holds whatever challenge information Game Kit was able to fetch.

error

If an error occurred, this object describes the error. If the operation completed successfully, this value is `nil`.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 6.0 and later.

Declared in

`GKChallenge.h`

Instance Methods

decline

Declines a challenge.

```
- (void)decline
```

Discussion
If your game implements a custom user interface to display challenges, it should include controls that allow a player to decline a challenge. If the player uses your user interface to decline a challenge, call this method.

Availability
Available in iOS 6.0 and later.

Declared in
`GKChallenge.h`

Constants

Challenge State

Possible states that a challenge can live in.

```
typedef enum GKChallengeState {     GKChallengeStateInvalid = 0,
   GKChallengeStatePending = 1,
   GKChallengeStateCompleted = 2,
   GKChallengeStateDeclined = 3,
};
```

Constants

`GKChallengeStateInvalid`

An error occurred. The state of this challenge is not valid.

Available in iOS 6.0 and later.

Declared in `GKChallenge.h`.

`GKChallengeStatePending`

The challenge has been issued, but is not yet completed nor declined.

Available in iOS 6.0 and later.

Declared in `GKChallenge.h`.

`GKChallengeStateCompleted`

The receiving player successfully completed the challenge.

Available in iOS 6.0 and later.

Declared in `GKChallenge.h`.

`GKChallengeStateDeclined`

The receiving player declined the challenge.

Available in iOS 6.0 and later.

Declared in `GKChallenge.h`.

GKChallengeEventHandler Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 6.0 and later.
Declared in	GKChallengeEventHandler.h
Companion guide	Game Center Programming Guide

Overview

The `GKChallengeEventHandler` class is used to respond to events related to challenges sent or received by the local player.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication see *Game Center Programming Guide*.

To use it, call the `challengeEventHandler` (page 46) class method to get the singleton instance and assign an object that implements the `GKChallengeEventHandlerDelegate` protocol to its `delegate` (page 46) property. You should assign a challenge event handler immediately after the local player has been authenticated, because your game may have been launched in response to a challenge notification being received by the player.

Tasks

Retrieving the Shared Instance

`+ challengeEventHandler` (page 46)

Returns the shared instance of the event handler

Getting and Setting the Delegate

`delegate` (page 46) *property*

The delegate for the event handler.

Properties

delegate

The delegate for the event handler.

```
@property(nonatomic, assign) id<GKChallengeEventHandlerDelegate> delegate
```

Discussion

Only access this property from your game's main thread.

Availability

Available in iOS 6.0 and later.

Declared in

`GKChallengeEventHandler.h`

Class Methods

challengeEventHandler

Returns the shared instance of the event handler

```
+ (GKChallengeEventHandler *)challengeEventHandler
```

Return Value

An event handler object.

Discussion

Your game never directly creates a `GKChallengeEventHandler` object. Instead, retrieve the shared instance using this class method.

Availability

Available in iOS 6.0 and later.

Declared in

`GKChallengeEventHandler.h`

GKFriendRequestComposeViewController Class Reference

Inherits from	UINavigationController : UIViewController : UIResponder : NSObject
Conforms to	NSCoding (UIViewController) UIAppearanceContainer (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.2 and later.
Declared in	GKFriendRequestComposeViewController.h
Companion guide	Game Center Programming Guide

Overview

Your game uses the `GKFriendRequestComposeViewController` class to present a screen that allows the local player to send friend requests to other players.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication see *Game Center Programming Guide*.

To show a friend request, initialize a new `GKFriendRequestComposeViewController` object and set the delegate. Optionally, you can customize the request by adding a text message or a list of recipients. Then, present the new view controller and wait for the delegate to be called. Once the delegate is called, dismiss the view controller.

On iOS, you present and dismiss the view controller from another view controller in your game, using the methods provided by the `UIViewController` class. On OS X, you use the `GKDialogController` class to present and dismiss the view controller.

Subclassing Notes

The `GKFriendRequestComposeViewController` class is not intended to be subclassed.

Tasks

Determining the Maximum Number of Recipients

`+ maxNumberOfRecipients` (page 50)

Returns the maximum number of recipients permitted in a single request.

Delegate

`composeViewDelegate` (page 49) *property*

The view controller's delegate

Adding Recipients

`– addRecipientsWithEmailAddresses:` (page 50)

Adds recipients based on their email addresses..

`– addRecipientsWithPlayerIDs:` (page 51)

Adds recipients based on their Game Center player identifiers.

Setting an Invitation Message

`– setMessage:` (page 51)

Sets the text message included in the friend invitation.

Properties

composeViewDelegate

The view controller's delegate

```
@property(nonatomic, assign) id<GKFriendRequestComposeViewControllerDelegate>
composeViewDelegate
```

Discussion
Before displaying the friend request, your game must set a delegate.

Availability
Available in iOS 4.2 and later.

Declared in
`GKFriendRequestComposeViewController.h`

Class Methods

maxNumberOfRecipients

Returns the maximum number of recipients permitted in a single request.

```
+ (NSUInteger)maxNumberOfRecipients
```

Return Value
The maximum number of recipients.

Discussion
If you add more recipients than the value returned from this method, Game Kit throws an exception.

Availability
Available in iOS 4.2 and later.

Declared in
`GKFriendRequestComposeViewController.h`

Instance Methods

addRecipientsWithEmailAddresses:

Adds recipients based on their email addresses..

```
- (void)addRecipientsWithEmailAddresses:(NSArray *)emailAddresses
```

Parameters

`emailAddresses`

An array with one or more `NSString` objects, each containing an email address.

Discussion

If you do not add at least once recipient, the recipients field is selected when the view controller is presented so that the player can type a list of recipients.

Availability

Available in iOS 4.2 and later.

See Also

– `addRecipientsWithPlayerIDs:` (page 51)

Declared in

`GKFriendRequestComposeViewController.h`

addRecipientsWithPlayerIDs:

Adds recipients based on their Game Center player identifiers.

```
- (void)addRecipientsWithPlayerIDs:(NSArray *)playerIDs
```

Parameters

`playerIDs`

An array with one or more `NSString` objects, each containing an player identifier.

Discussion

If you do not add at least once recipient, the recipients field is selected when the view controller is presented so that the player can type a list of recipients.

Availability

Available in iOS 4.2 and later.

See Also

– `addRecipientsWithEmailAddresses:` (page 50)

Declared in

`GKFriendRequestComposeViewController.h`

setMessage:

Sets the text message included in the friend invitation.

```
- (void)setMessage:(NSString *)message
```

Parameters

`message`

The text message.

Discussion

If you do not set a text message, Game Kit sets a default text message.

Availability

Available in iOS 4.2 and later.

Declared in

`GKFriendRequestComposeViewController.h`

GKGameCenterViewController Class Reference

Inherits from	UINavigationController : UIViewController : UIResponder : NSObject
Conforms to	NSCoding (UIViewController) UIAppearanceContainer (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 6.0 and later.
Declared in	GKGameCenterViewController.h
Companion guide	Game Center Programming Guide

Overview

The GKGameCenterViewController class aggregates many common Game Center features into a single user interface. It replaces `GKAchievementViewController` and `GKLeaderboardViewController` as the preferred way to show Game Center content in your game.

Important: Your application must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your application receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication, see *Game Center Programming Guide*.

To display the Game Center screen, initialize a new GKGameCenterViewController object and set its delegate. Optionally, you can choose to configure the view controller further to specify which content is initially displayed. Then present the view controller. Your delegate is called when the user dismisses the screen.

Your game should pause other activities before presenting the Game Center user interface.

Tasks

Configuring the Game Center View Controller's Delegate

`gameCenterDelegate` (page 54) *property*
The view controller's delegate.

Configuring the Game Center Controller's Content

`viewState` (page 56) *property*
The content displayed by the Game Center controller.

`leaderboardCategory` (page 54) *property*
The named leaderboard that is displayed by the view controller.

`leaderboardTimeScope` (page 55) *property*
A time filter used to restrict which scores are displayed to the player.

Properties

gameCenterDelegate

The view controller's delegate.

```
@property(nonatomic, assign) id<GKGameCenterControllerDelegate> gameCenterDelegate
```

Discussion
Before displaying the view controller, your game must set a delegate.

Availability
Available in iOS 6.0 and later.

Declared in
`GKGameCenterViewController.h`

leaderboardCategory

The named leaderboard that is displayed by the view controller.

```
@property(nonatomic, retain) NSString *leaderboardCategory
```

Discussion
The category property must either be `nil` or it must match a category identifier you defined when you created your leaderboards on iTunes Connect. If `nil`, the view displays scores for the aggregate leaderboard. Default is `nil`.

When the leaderboard is presented, the value of this property determines which leaderboard content is displayed to the player. As the player changes which leaderboard content they view, the `leaderboardCategory` property is automatically updated. For example, to preserve the player's selections, you can read the `leaderboardCategory` property after the screen is dismissed, and set that value the next time you initialize the view controller.

Availability
Available in iOS 6.0 and later.

Declared in
`GKGameCenterViewController.h`

leaderboardTimeScope

A time filter used to restrict which scores are displayed to the player.

```
@property(nonatomic, assign) GKLeaderboardTimeScope leaderboardTimeScope
```

Discussion
This property determines which tab view of the scores screen is displayed to the player. The default value is `GKLeaderboardTimeScopeAllTime` (page 74), which shows the best score each player has earned. For more information on time scopes, see *GKLeaderboard Class Reference* .

When the leaderboard is presented, the value of this property determines the initial tab that is displayed to the player. As the player changes which tab they view, the `leaderboardTimeScope` property is automatically updated. For example, to preserve the player's selections, you can read the `leaderboardTimeScope` property after the screen is dismissed, and set that value the next time you initialize the view controller.

Availability
Available in iOS 6.0 and later.

Declared in
`GKGameCenterViewController.h`

viewState

The content displayed by the Game Center controller.

```
@property(nonatomic, assign) GKGameCenterControllerViewState viewState
```

Discussion
See "Game Center Controller View State" (page 56) for possible values. When you first present the Game Center view controller, the content displayed by the view controller is determined by this property. If the player navigates to different content, the view state is automatically updated. For example, to preserve the player's selections, you can read the `viewState` property after the screen is dismissed, and set that value the next time you initialize the view controller.

Availability
Available in iOS 6.0 and later.

Declared in
`GKGameCenterViewController.h`

Constants

Game Center Controller View State

Possible values for the `viewState` (page 56) property.

```
typedef enum /*: NSInteger */ {
   GKGameCenterViewControllerStateDefault = -1,
   GKGameCenterViewControllerStateLeaderboards ,
   GKGameCenterViewControllerStateAchievements,
   GKGameCenterViewControllerStateChallenges,
} GKGameCenterViewControllerState;
```

Constants
`GKGameCenterViewControllerStateDefault`

Indicates that the view controller should present the default screen.

Available in iOS 6.0 and later.

Declared in `GKGameCenterViewController.h`.

`GKGameCenterViewControllerStateLeaderboards`

Indicates that the view controller presents leaderboard content. The `leaderboardCategory` (page 54) and `leaderboardTimeScope` (page 55) properties affect the appearance of this view state.

Available in iOS 6.0 and later.

Declared in `GKGameCenterViewController.h`.

`GKGameCenterViewControllerStateAchievements`

Indicates that the view controller presents achievements content.

Available in iOS 6.0 and later.

Declared in `GKGameCenterViewController.h`.

`GKGameCenterViewControllerStateChallenges`

Indicates that the view controller presents challenges content.

Available in iOS 6.0 and later.

Declared in `GKGameCenterViewController.h`.

GKInvite Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKMatchmaker.h
Companion guide	Game Center Programming Guide

Overview

A `GKInvite` object represents a matchmaking invitation sent by another player to the local player. Your game never directly creates `GKInvite` objects. Instead, these objects are created by Game Kit and delivered to your game's matchmaking event handler. See *GKMatchmaker Class Reference*.

The properties of the invitation object describe the match the local player is being invited to join.

Tasks

Reading Invitation Properties

`hosted` (page 59) *property*
A Boolean value that states whether the game is hosted on your servers. (read-only)

`inviter` (page 59) *property*
The identifier for the player who sent the invitation. (read-only)

`playerAttributes` (page 59) *property*
The player attributes for the match. (read-only)

`playerGroup` (page 60) *property*
The player group for the match. (read-only)

Properties

hosted

A Boolean value that states whether the game is hosted on your servers. (read-only)

```
@property(nonatomic, readonly, getter=isHosted) BOOL hosted
```

Discussion
If the value of the `hosted` property is `YES`, this is a match hosted on your own server. If the value is `NO`, this is a peer-to-peer match using Game Center. The default is `NO`.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

inviter

The identifier for the player who sent the invitation. (read-only)

```
@property(nonatomic, readonly, retain) NSString *inviter
```

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

playerAttributes

The player attributes for the match. (read-only)

```
@property(nonatomic, readonly) uint32_t playerAttributes
```

Discussion
The value of this property matches the `playerAttributes` (page 121) property of the original match request used to create the match.

Availability
Available in iOS 6.0 and later.

Declared in
`GKMatchmaker.h`

playerGroup

The player group for the match. (read-only)

```
@property(nonatomic, readonly) NSUInteger playerGroup
```

Discussion
The value of this property matches the `playerGroup` property of the original match request used to create the match.

Availability
Available in iOS 6.0 and later.

Declared in
`GKMatchmaker.h`

GKLeaderboard Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKLeaderboard.h
Companion guide	Game Center Programming Guide
Related sample code	GKTapper

Overview

A `GKLeaderboard` object is used to read data from a leaderboard stored on Game Center. Your game uses `GKLeaderboard` objects when it wants to retrieve localized information about a specific leaderboard or to retrieve scores from a leaderboard. Typically, you do this when you want the data needed to implement your own custom leaderboard user interface.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication, see *Game Center Programming Guide*.

During the development process, you create the leaderboards for your game on iTunes Connect.

To retrieve information about the available leaderboards, use the `loadLeaderboardsWithCompletionHandler:` (page 69) class method.

To request score data from Game Center, your game allocates and initializes a `GKLeaderboard` object, configures its search properties, and then calls the object's `loadScoresWithCompletionHandler:` (page 72) method. Game Kit retrieves the score data from Game Center and calls your completion handler.

The search properties are used to filter the score data returned to your game:

- The `category` property allows you to choose which leaderboard on iTunes Connect is searched.
- The `playerScope` property allows you to restrict the search to a local player's friends. Optionally, you can also explicitly initialize a leaderboard object to search for scores for a specific group of players.
- The `timeScope` property allows you to filter based on when the score was earned.
- The `range` property allows you to pick scores within a specific range. For example, the range `[1,10]` returns the best ten scores.

The algorithm used by the `GKLeaderboard` object is as follows:

1. Start with the set of all possible scores.
2. Discard any scores that do not match the `playerScope`, `timeScope` and `category` properties.
3. For each player, keep the best score that player has earned and discard the rest.
4. Sort the scores from best to worst.
5. Use the range property to determine which scores are returned.

Tasks

Determining the Available Leaderboards to Display

`+ loadLeaderboardsWithCompletionHandler:` (page 69)
Loads the list of leaderboards from Game Center

`+ loadCategoriesWithCompletionHandler:` (page 68)
Loads the list of leaderboard categories along with their corresponding localized titles. (Deprecated. Use the `loadLeaderboardsWithCompletionHandler:` (page 69) method instead.)

Initialization

`– init` (page 71)
Initializes a default leaderboard request.

`– initWithPlayerIDs:` (page 72)
Initializes a leaderboard request to retrieve the scores of a specific group of players.

Customizing the Leaderboard Request

`playerScope` (page 66) *property*

A filter used to restrict the search to a subset of the players on Game Center.

`range` (page 66) *property*

The numerical score rankings to return from the search.

`timeScope` (page 67) *property*

A filter used to restrict the search to scores that were posted within a specific period of time.

`category` (page 64) *property*

The named leaderboard to retrieve information from.

Retrieving High Scores

– `loadScoresWithCompletionHandler:` (page 72)

Retrieves a set of scores from Game Center.

`loading` (page 64) *property*

A Boolean value that indicates whether the leaderboard object is retrieving scores. (read-only)

`title` (page 67) *property*

The localized title for the leaderboard. (read-only)

`scores` (page 67) *property*

The list of scores returned by the search. (read-only)

`localPlayerScore` (page 65) *property*

The score earned by the local player. (read-only)

`maxRange` (page 65) *property*

The size of the leaderboard. (read-only)

Changing a Local Player's Default Leaderboard

+ `setDefaultLeaderboard:withCompletionHandler:` (page 70)

Sets the default leaderboard for the local player.

Retrieving Group Information

`groupIdentifier` (page 64) *property*

The identifier for the group the leaderboard is part of.

Properties

category

The named leaderboard to retrieve information from.

```
@property(nonatomic, retain) NSString *category
```

Discussion

If non-`nil`, Game Center only returns scores from the matching leaderboard. If `nil`, all scores previously reported by the game are searched. Default is `nil`.

Availability

Available in iOS 4.1 and later.

Related Sample Code

GKTapper

Declared in

`GKLeaderboard.h`

groupIdentifier

The identifier for the group the leaderboard is part of.

```
@property(nonatomic, retain) NSString *groupIdentifier
```

Discussion

If your game was configured to be part of a group in iTunes Connect, this property holds the identifier you assigned to the group.

Availability

Available in iOS 6.0 and later.

Declared in

`GKLeaderboard.h`

loading

A Boolean value that indicates whether the leaderboard object is retrieving scores. (read-only)

```
@property(readonly, getter=isLoading) BOOL loading
```

Discussion
The value of the `loading` property is `YES` if the leaderboard object has any pending requests for scores.

Availability
Available in iOS 4.1 and later.

Declared in
`GKLeaderboard.h`

localPlayerScore

The score earned by the local player. (read-only)

```
@property(nonatomic, readonly, retain) GKScore *localPlayerScore
```

Discussion
This property is invalid until a call to `loadScoresWithCompletionHandler:` is completed. Afterward, it contains a score object representing the local player's score on the leaderboard given the filters you applied to the query.

Availability
Available in iOS 4.1 and later.

Related Sample Code
GKTapper

Declared in
`GKLeaderboard.h`

maxRange

The size of the leaderboard. (read-only)

```
@property(nonatomic, readonly, assign) NSUInteger maxRange
```

Discussion
This property is invalid until a call to `loadScoresWithCompletionHandler:` is completed. Afterward, it contains the total number of entries available to return to your game given the filters you applied to the query.

Availability
Available in iOS 4.1 and later.

Declared in
GKLeaderboard.h

playerScope

A filter used to restrict the search to a subset of the players on Game Center.

```
@property(nonatomic, assign) GKLeaderboardPlayerScope playerScope
```

Discussion
The `playerScope` property is ignored if the leaderboard request was initialized using the `initWithPlayerIDs:` method. Otherwise, the `playerScope` property determines which players are included in the request for high scores. The default is `GKLeaderboardPlayerScopeGlobal`. See `Leaderboard Player Scope` (page 74) for more information.

Availability
Available in iOS 4.1 and later.

Declared in
GKLeaderboard.h

range

The numerical score rankings to return from the search.

```
@property(nonatomic, assign) NSRange range
```

Discussion
The `range` property is ignored if the leaderboard request was initialized using the `initWithPlayerIDs:` method. Otherwise, the `range` property is used to filter which scores are returned to your game. For example, if you specified a range of `[1,10]`, after the search is complete, your game receives the best ten scores. The default range is `[1,25]`.

The minimum index is `1`. The maximum length is `100`.

Availability
Available in iOS 4.1 and later.

Related Sample Code
GKTapper

Declared in
GKLeaderboard.h

scores

The list of scores returned by the search. (read-only)

```
@property(nonatomic, readonly, retain) NSArray *scores
```

Discussion

This property is invalid until a call to `loadScoresWithCompletionHandler:` is complete. Afterward, it contains the same score objects that were returned to the completion handler.

Availability

Available in iOS 4.1 and later.

Related Sample Code

GKTapper

Declared in

`GKLeaderboard.h`

timeScope

A filter used to restrict the search to scores that were posted within a specific period of time.

```
@property(nonatomic, assign) GKLeaderboardTimeScope timeScope
```

Discussion

This property determines how far back in time to look for scores. The default value is `GKLeaderboardTimeScopeAllTime` (page 74). See `Leaderboard Time Scope` (page 73) for more information.

Availability

Available in iOS 4.1 and later.

Related Sample Code

GKTapper

Declared in

`GKLeaderboard.h`

title

The localized title for the leaderboard. (read-only)

```
@property(nonatomic, readonly, retain) NSString *title
```

Discussion
If you initialized a new leaderboard object, this property is invalid until a call to `loadScoresWithCompletionHandler:` is complete. Afterward, it contains the localized title for the leaderboard identified by the `category` property.

Availability
Available in iOS 4.1 and later.

Declared in
`GKLeaderboard.h`

Class Methods

loadCategoriesWithCompletionHandler:

Loads the list of leaderboard categories along with their corresponding localized titles. (Deprecated in iOS 6.0. Use the `loadLeaderboardsWithCompletionHandler:` (page 69) method instead.)

```
+ (void)loadCategoriesWithCompletionHandler:(void (^)(NSArray *categories, NSArray
 *titles, NSError *error))completionHandler
```

Parameters

`completionHandler`

A block that is called when the categories have been retrieved from the server.

The block receives the following parameters:

categories

An array of `NSString` objects that provides the categories to your game. If an error occurred, this value may be non-`nil`. In this case, the array holds whatever data Game Kit was able to download before the error occurred.

titles

An array of `NSString` objects that provides localized titles for each category. If an error occurred, this value may be non-`nil`. In this case, the array holds whatever data Game Kit was able to download before the error occurred.

error

If an error occurred, this error object describes the error. If the operation completed successfully, the value is `nil`.

Discussion

You use this class method to retrieve the category identifiers and titles you configured for your leaderboards on iTunes Connect. To create a leaderboard query that targets a particular category, set the `category` (page 64) property to one of the strings returned by this method.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 4.1 and later.

Deprecated in iOS 6.0.

Declared in

`GKLeaderboard.h`

loadLeaderboardsWithCompletionHandler:

Loads the list of leaderboards from Game Center

```
+ (void)loadLeaderboardsWithCompletionHandler:(void (^)(NSArray *leaderboards,
NSError *error))completionHandler
```

Parameters

`completionHandler`

A block that is called when the categories have been retrieved from the server.

The block receives the following parameters:

leaderboards

An array of `GKLeaderboard` objects that provides the leaderboards for your game. If an error occurred, this value may be non-`nil`. In this case, the array holds whatever data Game Kit was able to download before the error occurred.

error

If an error occurred, this error object describes the error. If the operation completed successfully, the value is `nil`.

Discussion

Use this class method to retrieve the list of leaderboards you configured on iTunes Connect. Use the properties of each leaderboard object, especially the `category` (page 64) and `title` (page 67) properties, to learn more about the leaderboard.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 6.0 and later.

Declared in

`GKLeaderboard.h`

setDefaultLeaderboard:withCompletionHandler:

Sets the default leaderboard for the local player.

```
+ (void)setDefaultLeaderboard:(NSString *)categoryID withCompletionHandler:(void
(^)(NSError *error))completionHandler
```

Parameters

`categoryID`

The named leaderboard that should be the new default leaderboard for the local player.

`completionHandler`

A block to be called after the scores are retrieved from the server.

The block receives the following parameter:

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

The default leaderboard is used whenever your game uses a `GKScore` object to report a score to Game Center without explicitly setting the score object's `category` (page 142) property. The default leaderboard is normally set in iTunes Connect. However, your game can use this class method to override the default leaderboard that appears for the local player. This information is stored for each player on Game Center.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

If an error occurs and was a network error, your game should periodically resend the request until it completes.

Availability

Available in iOS 5.0 and later.

Declared in

`GKLeaderboard.h`

Instance Methods

init

Initializes a default leaderboard request.

```
- (id)init
```

Return Value

An initialized leaderboard request.

Discussion
A leaderboard object initialized with this method uses the `playerScope`, `timeScope`, and `range` properties to search Game Center for scores.

Availability
Available in iOS 4.1 and later.

Declared in
`GKLeaderboard.h`

initWithPlayerIDs:

Initializes a leaderboard request to retrieve the scores of a specific group of players.

```
- (id)initWithPlayerIDs:(NSArray *)playerIDs
```

Parameters
`playerIDs`
An array of `NSString` objects that holds the player identifier strings of the players to retrieve.

Return Value
An initialized leaderboard request.

Discussion
A leaderboard object initialized with this method ignores the `playerScope` and `range` properties. Instead, it retrieves scores for the specific list of players whose identifiers are included in the `playerIDs` parameter.

Availability
Available in iOS 4.1 and later.

Declared in
`GKLeaderboard.h`

loadScoresWithCompletionHandler:

Retrieves a set of scores from Game Center.

```
- (void)loadScoresWithCompletionHandler:(void (^)(NSArray *scores, NSError
*error))completionHandler
```

Parameters

`completionHandler`

A block to be called after the scores are retrieved from the server.

The block receives the following parameters:

scores

An array of score objects that hold the requested scores. If an error occurred, this value may be non-`nil`. In this case, the array holds whatever score data could be retrieved from Game Center before the error occurred.

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

You can call this method multiple times; each call represents a different query against the scores stored on Game Center. If you post multiple load operations using the same leaderboard object, any properties that are updated by loading scores reflect the last query that completed. The order that achievement queries are processed is arbitrary.

Availability

Available in iOS 4.1 and later.

Related Sample Code
GKTapper

Declared in

`GKLeaderboard.h`

Constants

Leaderboard Time Scope

The period of time to which a player's best score is restricted.

```
enum {
```

```
   GKLeaderboardTimeScopeToday = 0,
   GKLeaderboardTimeScopeWeek,
   GKLeaderboardTimeScopeAllTime
};
typedef NSInteger GKLeaderboardTimeScope;
```

Constants

`GKLeaderboardTimeScopeToday`

Each player is restricted to scores recorded in the past 24 hours.

Available in iOS 4.1 and later.

Declared in `GKLeaderboard.h`.

`GKLeaderboardTimeScopeWeek`

Each player is restricted to scores recorded in the past week.

Available in iOS 4.1 and later.

Declared in `GKLeaderboard.h`.

`GKLeaderboardTimeScopeAllTime`

Each player's best score is returned.

Available in iOS 4.1 and later.

Declared in `GKLeaderboard.h`.

Availability

Available in iOS 4.1 and later.

Declared in

`GKLeaderboard.h`

Leaderboard Player Scope

The scope of players to be searched for scores.

```
enum {
   GKLeaderboardPlayerScopeGlobal = 0,
   GKLeaderboardPlayerScopeFriendsOnly
};
typedef NSInteger GKLeaderboardPlayerScope;
```

Constants

`GKLeaderboardPlayerScopeGlobal`

All players on Game Center should be considered when generating the list of scores.

Available in iOS 4.1 and later.

Declared in `GKLeaderboard.h`.

`GKLeaderboardPlayerScopeFriendsOnly`

Only friends of the local player should be considered when generating the list of scores.

Available in iOS 4.1 and later.

Declared in `GKLeaderboard.h`.

Availability

Available in iOS 4.1 and later.

Declared in

`GKLeaderboard.h`

GKLeaderboardViewController Class Reference

Inherits from	GKGameCenterViewController : UINavigationController : UIViewController : UIResponder : NSObject
Conforms to	NSCoding (UIViewController) UIAppearanceContainer (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKLeaderboardViewController.h
Companion guide	Game Center Programming Guide
Related sample code	GKLeaderboards GKTapper

Overview

The `GKLeaderboardViewController` class provides a standard user interface that displays leaderboard scores to the player. If the `GKGameCenterViewController` class is available, you should use it instead.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication, see *Game Center Programming Guide*.

To show a leaderboard screen, initialize a new `GKLeaderboardViewController` object and set the delegate. Optionally, you can configure the view controller to display specific data to the player. Then, present the new view controller and wait for the delegate to be called. Once the delegate is called, dismiss the view controller.

On iOS, you present and dismiss the view controller from another view controller in your game, using the methods provided by the `UIViewController` class. On OS X, you use the `GKDialogController` class to present and dismiss the view controller.

Your game should pause other activities before presenting the leaderboard.

Subclassing Notes

The `GKLeaderboardViewController` class is not intended to be subclassed.

Tasks

Configuring the Leaderboard View Controller

`category` (page 77) *property*
The named leaderboard that is displayed by the view controller.

`leaderboardDelegate` (page 78) *property*
The view controller's delegate.

`timeScope` (page 78) *property*
A time filter used to restrict which scores are displayed to the player.

Properties

category

The named leaderboard that is displayed by the view controller.

```
@property(nonatomic, retain) NSString *category
```

Discussion
The `category` property must either be `nil` or it must match a leaderboard identifier for a leaderboard you defined in iTunes Connect. If `nil`, the view displays scores for the default leaderboard. Default is `nil`.

When the view controller is presented, the initial leaderboard shown is based on the value of this property. If the player changes which leaderboard they are viewing, the `category` property is automatically updated. For example, you can read the `category` property after the screen is dismissed, and set that value the next time you initialize a new leaderboard view controller.

Availability
Available in iOS 4.1 and later.

Related Sample Code
GKTapper

Declared in
`GKLeaderboardViewController.h`

leaderboardDelegate

The view controller's delegate.

```
@property(nonatomic, assign) id<GKLeaderboardViewControllerDelegate> leaderboardDelegate
```

Discussion
Before displaying the leaderboard, you must set a delegate.

Availability
Available in iOS 4.1 and later.

Related Sample Code
GKTapper

Declared in
`GKLeaderboardViewController.h`

timeScope

A time filter used to restrict which scores are displayed to the player.

```
@property(nonatomic, assign) GKLeaderboardTimeScope timeScope
```

Discussion
This property determines which tab view is displayed to the player. The default value is `GKLeaderboardTimeScopeAllTime` (page 74), which shows the best score each player has earned. For more information on time scopes, see *GKLeaderboard Class Reference*.

If the player changes which tab they view, the `timeScope` property is automatically updated. For example, you can read the `timeScope` property after the view controller is dismissed, and set that value the next time you initialize a new leaderboard view controller.

Availability
Available in iOS 4.1 and later.

Related Sample Code
GKTapper

Declared in
`GKLeaderboardViewController.h`

GKLocalPlayer Class Reference

Inherits from	GKPlayer : NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKLocalPlayer.h
Companion guide	Game Center Programming Guide
Related sample code	GKAchievements GKAuthentication GKLeaderboards GKTapper

Overview

The `GKLocalPlayer` class is a special subclass of `GKPlayer` that represents the authenticated player running your game on the device. At any given time, only one player may be authenticated on the device; this player must log out before another player can log in.

Your game must authenticate the local player before using any Game Center features. Authenticating the player ensures that the player has created an account and is connected to Game Center. To authenticate the local player, retrieve the shared instance of the local player by calling the `localPlayer` (page 83) class method and set its `authenticateHandler` (page 81) property.

You can see whether the local player is authenticated by reading the `authenticated` (page 81) property. If `authenticated` (page 81) is `YES`, then the local player's other properties are valid, and you can call other Game Center methods.

Call the `loadFriendsWithCompletionHandler:` (page 86) method to retrieve the player identifiers for the local player's friends.

Tasks

Accessing the Shared Local Player

`+ localPlayer` (page 83)
Retrieves the shared instance of the local player.

Authentication

`authenticateHandler` (page 81) *property*
A handler called to process an authentication-related event.

`authenticated` (page 81) *property*
A Boolean value that indicates whether a local player is currently signed in to Game Center. (read-only)

`– authenticateWithCompletionHandler:` (page 84)
Authenticates the local player on the device. (Deprecated. Set the `authenticateHandler` (page 81) property instead.)

Accessing Friends

`– loadFriendsWithCompletionHandler:` (page 86)
Retrieves a list of player identifiers for the local player's friends.

`friends` (page 82) *property*
A list of player identifiers for the local player's friends. (read-only)

Determining If the Player Is Underage

`underage` (page 83) *property*
A Boolean value that declares whether the local player is underage. (read-only)

Working with the Default Leaderboard Category

`– loadDefaultLeaderboardCategoryIDWithCompletionHandler:` (page 85)
Loads the category identifier for the local player's default leaderboard.

– `setDefaultLeaderboardCategoryID:completionHandler:` (page 87)
Sets the category identifier for the local player's default leaderboard.

Properties

authenticated

A Boolean value that indicates whether a local player is currently signed in to Game Center. (read-only)

```
@property(nonatomic, readonly, getter=isAuthenticated) BOOL authenticated
```

Availability
Available in iOS 4.1 and later.

Declared in
`GKLocalPlayer.h`

authenticateHandler

A handler called to process an authentication-related event.

```
@property(nonatomic, copy) void(^authenticateHandler)(UIViewController *viewController,
NSError *error)
```

Parameters
`viewController`
This parameter is `nil` if the authentication process is complete. Otherwise, it contains a view controller that your game should display to the player.

`error`
This parameter contains an error object that describes any error that occurred.

Discussion
Your game should authenticate the player as early as possible after launching, ideally as soon as you can present a user interface to the player. For example, your game may be launched because the player accepted an invitation to join a match or to take a turn in a turn-based match, so you want your game to authenticate the player and process the match invitation as quickly as possible. After you set a handler, authentication begins automatically and is repeated when your game moves to the background and then back to the foreground.

During the authentication process, Game Kit calls your handler one or more times to handle specific authentication events. Your handler must handle three kinds of events:

- If the device does not have an authenticated player, Game Kit passes a view controller to your authenticate handler. When presented, this view controller displays the authentication user interface. Your game should pause other activities that require user interaction (such as your game loop), present this view controller and then return. When the player finishes interacting with it, the view controller is dismissed automatically.
- If the authentication process succeeded, the`GKLocalPlayer` (page 79) singleton object's `authenticated` (page 81) property is set to `YES` and the object's other properties are set to match those of the connected player.
- If the authentication process failed, the `GKLocalPlayer` (page 79) singleton object's `authenticated` (page 81) property is set to `NO` and the object's other properties are cleared.

Each time the authentication handler is called, the data stored in the local player singleton object may have changed. A new player may have logged into the device or the player may have simply logged out from Game Center. Because of both of these possibilities, your authentication handler must be prepared to update any other objects that assume that a particular player is logged in. For more information, see "Authenticating the Local Player in a Multitasking Application" in *Game Center Programming Guide*.

Availability
Available in iOS 6.0 and later.

Declared in
`GKLocalPlayer.h`

friends

A list of player identifiers for the local player's friends. (read-only)

```
@property(nonatomic, readonly, retain) NSArray *friends
```

Discussion
This property is invalid until a call to `loadFriendsWithCompletionHandler:` succeeds.

Availability
Available in iOS 4.1 and later.

Declared in
`GKLocalPlayer.h`

underage

A Boolean value that declares whether the local player is underage. (read-only)

```
@property(nonatomic, readonly, getter=isUnderage) BOOL underage
```

Discussion
Some Game Center features are disabled if the local player is underage. Your game can also test this property if it wants to disable some of its own features based on the player's age.

Availability
Available in iOS 4.1 and later.

Declared in
`GKLocalPlayer.h`

Class Methods

localPlayer

Retrieves the shared instance of the local player.

```
+ (GKLocalPlayer *)localPlayer
```

Return Value
The local player object.

Discussion
You never directly create a local player object. Instead, you retrieve the singleton object by calling this class method.

Availability
Available in iOS 4.1 and later.

Related Sample Code
GKAchievements
GKAuthentication
GKLeaderboards
GKTapper

Declared in
`GKLocalPlayer.h`

Instance Methods

authenticateWithCompletionHandler:

Authenticates the local player on the device. (Deprecated in iOS 6.0. Set the `authenticateHandler` *(page 81) property instead.)*

```
- (void)authenticateWithCompletionHandler:(void (^)(NSError *error))completionHandler
```

Parameters

`completionHandler`

A block to be called when the player has authenticated or when an error occurs.

The block takes the following parameter:

error

This parameter is `nil` if the player successfully authenticated. Otherwise, it contains an error object that describes the error that occurred.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Your game should authenticate the player as early as possible after launching, ideally as soon as you can present a user interface to the player. For example, your game may be launched because the player accepted an invitation to join a match or to take a turn in a turn-based match, so you want your game to authenticate the player and process the match invitation as quickly as possible.

If there is not an authenticated player on the device when your game calls this method, Game Kit displays a user interface that allows the player to sign in with their credentials (or to create a new account if he or she has never used Game Center). Your game should pause other activities that require user interaction (such as a real time game loop) before attempting to authenticate the local player.

Game Kit maintains a strong reference to your completion handler even after successfully authenticating a local player. If your game moves into the background, Game Kit automatically authenticates the player again whenever your game moves back to the foreground. Game Kit calls your same completion handler each time it authenticates the local player. Be mindful that in block programming, any Objective-C object referenced inside a block is also strongly referenced by the block until the block is released. Because Game Kit maintains a strong reference to your completion handler until your game terminates, any objects referenced from within your authentication handler are also held indefinitely.

Each time the completion handler is called, the data stored in the local player singleton object may have changed. A new player may have logged into the device or the player may have simply logged out from Game Center. Because of both of these possibilities, your authentication handler must be prepared to update any other objects that assume that a particular player is logged in. For more information, see "Authenticating the Local Player in a Multitasking Application" in *Game Center Programming Guide*.

Availability
Available in iOS 4.1 and later.

Deprecated in iOS 6.0.

Related Sample Code
GKAchievements
GKAuthentication
GKLeaderboards
GKTapper

Declared in
`GKLocalPlayer.h`

loadDefaultLeaderboardCategoryIDWithCompletionHandler:

Loads the category identifier for the local player's default leaderboard.

```
- (void)loadDefaultLeaderboardCategoryIDWithCompletionHandler:(void (^)(NSString
*categoryID, NSError *error))completionHandler
```

Parameters
`completionHandler`

A block to be called when the request completes.

The block receives the following parameters:

categoryID

The category ID string for the local player's default leaderboard.

error

If an error occurred, this parameter holds an error object that explains the error. Otherwise, the value of this parameter is `nil`.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 6.0 and later.

See Also

– `setDefaultLeaderboardCategoryID:completionHandler:` (page 87)

Declared in

`GKLocalPlayer.h`

loadFriendsWithCompletionHandler:

Retrieves a list of player identifiers for the local player's friends.

```
– (void)loadFriendsWithCompletionHandler:(void (^)(NSArray *friends, NSError
*error))completionHandler
```

Parameters

`completionHandler`

A block to be called when the request completes.

The block receives the following parameters:

friends

An array of player identifiers for the players that are friends of the local player. If an error occurred, this value can be non-`nil`. In that case, the array contains the data that Game Kit was able to download before the error occurred.

error

If an error occurred, this parameter holds an error object that explains the error. Otherwise, the value of this parameter is `nil`.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Once this call is completed, the `friends` property of the shared local player object is set to the same list of players returned in the completion handler.

Availability
Available in iOS 4.1 and later.

See Also
`@property friends` (page 82)

Declared in
`GKLocalPlayer.h`

setDefaultLeaderboardCategoryID:completionHandler:

Sets the category identifier for the local player's default leaderboard.

```
- (void)setDefaultLeaderboardCategoryID:(NSString *)categoryID completionHandler:(void
 (^)(NSError *error))completionHandler
```

Parameters

`categoryID`

The category ID string for one of your game's leaderboards.

`completionHandler`

A block to be called when the request completes.

The block receives the following parameter:

error

If an error occurred, this parameter holds an error object that explains the error. Otherwise, the value of this parameter is `nil`.

Discussion
When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

The default leaderboard is configured in iTunes Connect as part of configuring your game's leaderboards. All players normally start with this leaderboard as the default leaderboard. Calling this method changes the default leaderboard only for the local player.

Availability
Available in iOS 6.0 and later.

See Also

– `loadDefaultLeaderboardCategoryIDWithCompletionHandler:` (page 85)

Declared in

`GKLocalPlayer.h`

Notifications

GKPlayerAuthenticationDidChangeNotificationName

Posted after the `authenticated` (page 81) property of the shared local player object changes.

Availability

Available in iOS 4.1 and later.

Declared in

`GKLocalPlayer.h`

GKMatch Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKMatch.h
Companion guide	Game Center Programming Guide

Overview

A `GKMatch` object provides a peer-to-peer network between a group of devices that are connected to Game Center. Matches provide transmit both voice and game data. Your game never directly allocates `GKMatch` objects. Instead, it uses the `GKMatchmaker` class to programmatically find a match with other interested players or a `GKMatchmakerViewController` object to display a user interface to the player.

After your game receives a match object, set its delegate and then wait until the other participants are connected to the match. You can read the `expectedPlayerCount` (page 91) property to determine how many players have not connected to the match.

Each device in the match is identified by the player identifier for the player authenticated on that device. Your game transmits its own data to other players by calling either the `sendDataToAllPlayers:withDataMode:error:` (page 95) method or the `sendData:toPlayers:withDataMode:error:` (page 94) method. To allow voice chat, call `voiceChatWithName:` (page 95) to create one or more voice channels.

When you are finished with the match, call the match's `disconnect` (page 93) method.

Tasks

Getting and Setting the Delegate

`delegate` (page 91) *property*
The delegate for the match.

Working with Other Players

`playerIDs` (page 91) *property*
The player identifiers for the players in the match. (read-only)

`expectedPlayerCount` (page 91) *property*
The remaining number of players who have not yet connected to the match. (read-only)

Sending Data to Other Players

– `sendData:toPlayers:withDataMode:error:` (page 94)
Transmits data to a list of connected players.

– `sendDataToAllPlayers:withDataMode:error:` (page 95)
Transmits data to all players connected to the match.

– `chooseBestHostPlayerWithCompletionHandler:` (page 92)
Determines the best player in the game to act as the server for a client-server match.

Joining a Voice Chat

– `voiceChatWithName:` (page 95)
Joins a voice channel.

Finishing the Match

– `disconnect` (page 93)
Disconnects the local player from the match.

– `rematchWithCompletionHandler:` (page 93)
Create a new match with the list of players from an existing match.

Properties

delegate

The delegate for the match.

```
@property(nonatomic, assign) id<GKMatchDelegate> delegate
```

Discussion
You must set a delegate to receive data from other members of the match.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatch.h`

expectedPlayerCount

The remaining number of players who have not yet connected to the match. (read-only)

```
@property(nonatomic, readonly) NSUInteger expectedPlayerCount
```

Discussion
The value of this property is decremented whenever a player connects to the match. When its value reaches `0`, all expected players are connected, and your game can begin the match.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatch.h`

playerIDs

The player identifiers for the players in the match. (read-only)

```
@property(nonatomic, readonly) NSArray *playerIDs
```

Discussion
The `playerIDs` property initially includes the player identifiers for any players already connected to the match; the array may initially be empty. As each new player connects to the match, that player's identifier is added to the array.

Availability
Available in iOS 4.1 and later.

See Also
@property expectedPlayerCount (page 91)

Declared in
GKMatch.h

Instance Methods

chooseBestHostPlayerWithCompletionHandler:

Determines the best player in the game to act as the server for a client-server match.

```
- (void)chooseBestHostPlayerWithCompletionHandler:(void (^)(NSString
*playerID))completionHandler
```

Parameters
`completionHandler`

A block to be called after the score is reported.

The block receives the following parameter:

playerID

The player identifier for the player with the best estimated network performance, or `nil` if a player could not currently be determined.

Discussion
Calling this method causes Game Kit to attempt to estimate which player has the best overall network connection using a variety of metrics such as bandwidth, latency and network reliability. Typically, you call this method when your game implements a client-server model on top of the match's peer-to-peer connection. See "Designing Your Network Game" in *Game Center Programming Guide* .

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability
Available in iOS 6.0 and later.

Declared in
`GKMatch.h`

disconnect

Disconnects the local player from the match.

```
- (void)disconnect
```

Discussion
Your game should call `disconnect` before removing the last strong reference to the match object. Calling `disconnect` notifies other players that you have left the match.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatch.h`

rematchWithCompletionHandler:

Create a new match with the list of players from an existing match.

```
- (void)rematchWithCompletionHandler:(void (^)(GKMatch *match, NSError
*error))completionHandler
```

Parameters
`completionHandler`

A block to be called after the match is created.

The block receives the following parameter:

match

The new match. If an error occurred, this parameter's value is `nil`.

error

If an error occurred, this parameter holds an error object that describes the problem. If the match was successfully recreated, this parameter's value is `nil`.

Discussion
Calling this method uses auto-matching to recreate a previous match. A new match with the same set of players is created and returned. If your game attempts to recreate matches using this method, each instance of your game on each device should call this method.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability
Available in iOS 6.0 and later.

Declared in
`GKMatch.h`

sendData:toPlayers:withDataMode:error:

Transmits data to a list of connected players.

```
- (BOOL)sendData:(NSData *)data toPlayers:(NSArray *)*)playerIDs
withDataMode:(GKMatchSendDataMode)mode error:(NSError **)error
```

Parameters

`data`

The bytes to be sent.

`players`

An array containing the identifier strings for the list of players who should receive the data.

`mode`

The mechanism used to send the data.

`error`

If the data could not be queued, on return, this parameter holds an `NSError` object describing the error.

Return Value
`YES` if the data was successfully queued for transmission; `NO` if the match was unable to queue the data.

Discussion
The match queues the data and transmits it when the network becomes available.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatch.h`

sendDataToAllPlayers:withDataMode:error:

Transmits data to all players connected to the match.

```
- (BOOL)sendDataToAllPlayers:(NSData *)data withDataMode:(GKMatchSendDataMode)mode
error:(NSError **)error
```

Parameters

`data`

The bytes to be sent.

`mode`

The mechanism used to send the data.

`error`

If the data could not be queued, on return, this parameter holds an `NSError` object describing the error.

Return Value

`YES` if the data was successfully queued for transmission; `NO` if the match was unable to queue the data.

Discussion

The match queues the data and transmits it when the network becomes available.

Availability

Available in iOS 4.1 and later.

Declared in

`GKMatch.h`

voiceChatWithName:

Joins a voice channel.

```
- (GKVoiceChat *)voiceChatWithName:(NSString *)name
```

Parameters

`name`

The channel to join.

Return Value

An autoreleased voice chat object for the voice channel, or `nil` if an error occurred.

Discussion
Calling this method joins a voice channel, creating it if necessary. Your game should keep a strong reference to the voice chat object until the player leaves the channel. All participants who join a channel with the same name are connected to each other.

A single match can have multiple voice chat channels, and any player in the match can join multiple channels simultaneously. For example, a team-based game might create a channel for each team, and a single channel that includes all of the players.

Voice chat objects are dependent on the network connection provided by the match. When the player disconnects from the match, all voice channels associated with that match stop working. Typically, you should exit any voice channels and release any strong references to the channels before disconnecting from the match.

Parental controls may prevent a player from joining a voice chat. If the player is not permitted to join the voice channel, a `nil` object is returned to your application.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatch.h`

Constants

Data Transmission Modes

The mechanism used to transmit data to other players.

```
enum {
   GKMatchSendDataReliable,
   GKMatchSendDataUnreliable
};
typedef NSInteger GKMatchSendDataMode;
```

Constants

`GKMatchSendDataReliable`

The data is sent continuously until it is successfully received by the intended recipients or the connection times out. Use this when you need to guarantee delivery and speed is not critical.

Reliable transmissions are delivered in the order they were sent.

Available in iOS 4.1 and later.

Declared in `GKMatch.h`.

`GKMatchSendDataUnreliable`

The data is sent once and is not sent again if a transmission error occurs. Use this for small packets of data that must arrive quickly to be useful to the recipient.

Data transmitted unreliably may be received out of order by recipients. Typically, you build your own game-specific error handling on top of this mechanism.

Available in iOS 4.1 and later.

Declared in `GKMatch.h`.

Availability

Available in iOS 4.1 and later.

Declared in

`GKMatch.h`

GKMatchmaker Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKMatchmaker.h
Companion guide	Game Center Programming Guide

Overview

The `GKMatchmaker` class is used to programmatically create matches to other players and to receive match invitations sent by other players.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication, see *Game Center Programming Guide*.

Matches can be either **peer-to-peer** or **hosted**. A peer-to-peer match is fully supported in Game Kit by the `GKMatch` class. A `GKMatch` object provides all of the network connections between the devices and routes the network data through Game Center if necessary. In contrast, a hosted match uses matchmaking to find the players for a match, but your game implements its own networking between the participants, routing through your own server if necessary.

> **Note:** Hosted matches are intended for developers who already have a networking game service but who want to use Game Center to find players for a match. Because you need to design and implement your own networking code, you can define your own protocols to connect to your server.

To receive invitations from other players, your game must provide an invitation handler. After your game successfully authenticates the local player, it should immediately set the `inviteHandler` (page 100) property. The invite handler is called immediately if your game was launched in response to a push notification.

To programmatically search for other players, start by creating a `GKMatchRequest` object that describes the match you are interested in. Then, call the shared matchmaker's `findMatchForRequest:withCompletionHandler:` (page 103) method to create a peer-to-peer match, or its `findPlayersForHostedMatchRequest:withCompletionHandler:` (page 104) method to create a hosted match. In either case, Game Center matches players into a match and calls your completion handler. If the match does not have enough players (for example, you invited a specific list of players and some declined the invitation), you can create another match request and call the `addPlayersToMatch:matchRequest:completionHandler:` (page 102) method to add more participants to the match. Once the match is complete, call the `finishMatchmakingForMatch:` (page 105) method.

If you implement programmatic matchmaking and invite specific players to the match, you should also implement an invitee response handler. This handler is notified as invitations are processed, allowing you to update your game's custom user interface to display which players are connected to the match.

Tasks

Retrieving the Shared Matchmaker

`+ sharedMatchmaker` (page 101)
Returns the singleton matchmaker instance.

Receiving Invitations From Other Players

`inviteHandler` (page 100) *property*
A block to be called when an invitation to join a match is accepted by the local player.

`– matchForInvite:completionHandler:` (page 106)
Creates a match from an accepted invitation.

Matching Players

– `findMatchForRequest:withCompletionHandler:` (page 103)
Initiates a request to find players for a peer-to-peer match.

– `findPlayersForHostedMatchRequest:withCompletionHandler:` (page 104)
Initiates a request to find players for a hosted match.

– `addPlayersToMatch:matchRequest:completionHandler:` (page 102)
Adds players to an existing match.

– `finishMatchmakingForMatch:` (page 105)
Informs Game Center that programmatic matchmaking has finished.

– `cancel` (page 103)
Cancels a pending matchmaking request.

– `cancelInviteToPlayer:` (page 103)
Cancels a pending invitation to another player.

– `queryPlayerGroupActivity:withCompletionHandler:` (page 108)
Queries Game Center for the activity in a player group.

– `queryActivityWithCompletionHandler:` (page 107)
Initiates a search for activity in all player groups.

Looking For Nearby Players

– `startBrowsingForNearbyPlayersWithReachableHandler:` (page 109)
Enables the matchmaking process to find nearby players.

– `stopBrowsingForNearbyPlayers` (page 109)
Ends the search for nearby players.

Properties

inviteHandler

A block to be called when an invitation to join a match is accepted by the local player.

```
@property(nonatomic, copy) void(^inviteHandler)(GKInvite *acceptedInvite, NSArray
*playersToInvite)
```

Discussion
The block takes the following parameters:

acceptedInvite
The invitation accepted by the player.

playersToInvite
A list of player identifiers for additional players to invite into the game..

Your block should respond to the invitation in one of two ways:

- Display the standard user interface by initializing a new `GKMatchmakerViewController` object, passing the invitation object and the list of player identifiers as parameters.
- Create a match programmatically by calling the `matchForInvite:completionHandler:` (page 106) method on the shared matchmaker instance.

If your game receives an invitation while it is already running, it should transition to multiplayer play. It should clean up any existing content, such as ending the current match the player is playing, and then process the invitation.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

Class Methods

sharedMatchmaker

Returns the singleton matchmaker instance.

```
+ (GKMatchmaker *)sharedMatchmaker
```

Return Value
The shared matchmaker instance.

Discussion
Games do not create a `GKMatchmaker` object. Instead, they retrieve the shared singleton by calling this method.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

Instance Methods

addPlayersToMatch:matchRequest:completionHandler:

Adds players to an existing match.

```
- (void)addPlayersToMatch:(GKMatch *)match matchRequest:(GKMatchRequest *)matchRequest
 completionHandler:(void (^)(NSError *error))completionHandler
```

Parameters

`match`

A previously created match.

`matchRequest`

The parameters for the new match request.

`completionHandler`

A block to be called when matchmaking completes.

The block takes the following parameter:

error

If matchmaking was successful, this parameter contains `nil`. Otherwise, this parameter holds an error object that describes the error that occurred.

Discussion
This method updates an existing match object by adding additional players.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a different thread than the thread originally used to invoke the method. This means that the code in your block needs to be thread safe.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

cancel

Cancels a pending matchmaking request.

```
- (void)cancel
```

Discussion
The completion handler receives a callback with a `GKErrorCancelled` error.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

cancelInviteToPlayer:

Cancels a pending invitation to another player.

```
- (void)cancelInviteToPlayer:(NSString *)playerID
```

Parameters
`playerID`
The player identifier for a player previously invited to the match.

Availability
Available in iOS 6.0 and later.

Declared in
`GKMatchmaker.h`

findMatchForRequest:withCompletionHandler:

Initiates a request to find players for a peer-to-peer match.

```
- (void)findMatchForRequest:(GKMatchRequest *)request withCompletionHandler:(void
(^)(GKMatch *match, NSError *error))completionHandler
```

Parameters

`request`

The configuration for the desired match.

`completionHandler`

A block to be called when the match has been created. This block receives the following parameters:

match

If matchmaking was successful, this parameter contains the created match. Otherwise, this parameter is `nil`.

error

If matchmaking was successful, this parameter contains `nil`. Otherwise, this parameter holds an error object that describes the error that occurred.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a different thread than the thread originally used to invoke the method. This means that the code in your block needs to be thread safe.

On iOS 6, if the match request's `playersToInvite` (page 122) property is non-`NIL`, Game Center sends invitations only out to the players listed in the property. If the `playersToInvite` (page 122) property is `NIL`, then it searches for any waiting players that match the request. If your game wants to perform programmatic matchmaking for the remaining slots, it should call the `addPlayersToMatch:matchRequest:completionHandler:` (page 102) method with a match request whose `playersToInvite` (page 122) property is `NIL`.

Prior to iOS 6, the match request's `playersToInvite` (page 122) property is ignored and this method only searches for available players.

Availability

Available in iOS 4.1 and later.

See Also

– `cancel` (page 103)

Declared in

`GKMatchmaker.h`

findPlayersForHostedMatchRequest:withCompletionHandler:

Initiates a request to find players for a hosted match.

```
- (void)findPlayersForHostedMatchRequest:(GKMatchRequest *)request
withCompletionHandler:(void (^)(NSArray *playerIDs, NSError *error))completionHandler
```

Parameters

`request`

The configuration for the desired match.

`completionHandler`

A block to be called when the match has been created. This block receives the following parameters:

players

If matchmaking was successful, this parameter contains an array of players to connect into the match. Otherwise, this parameter is `nil`.

error

If matchmaking was successful, this parameter contains `nil`. Otherwise, this parameter holds an error object that describes the error that occurred.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a different thread than the thread originally used to invoke the method. This means that the code in your block needs to be thread safe. When your completion handler is called, your game should connect those players to your own server.

On iOS 6, if the match request's `playersToInvite` (page 122) property is non-`NIL`, Game Center sends invitations only out to the players listed in the property. If the `playersToInvite` (page 122) property is `NIL`, then it searches for any waiting players that match the request. Prior to iOS 6, the match request's `playersToInvite` (page 122) property is ignored and this method only searches for available players.

Availability

Available in iOS 4.1 and later.

See Also

– `cancel` (page 103)

Declared in

`GKMatchmaker.h`

finishMatchmakingForMatch:

Informs Game Center that programmatic matchmaking has finished.

```
- (void)finishMatchmakingForMatch:(GKMatch *)match
```

Parameters

`match`

The match that has completed the matchmaking process.

Discussion

If your game uses programmatic matchmaking, it makes a series of calls to the `findMatchForRequest:withCompletionHandler:` (page 103) and `addPlayersToMatch:matchRequest:completionHandler:` (page 102) methods to fill a match with players. When the match has the proper number of players, call the `finishMatchmakingForMatch:` method before starting the match.

Availability

Available in iOS 6.0 and later.

Declared in

`GKMatchmaker.h`

matchForInvite:completionHandler:

Creates a match from an accepted invitation.

```
- (void)matchForInvite:(GKInvite *)invite completionHandler:(void (^)(GKMatch *match,
 NSError *error))completionHandler
```

Parameters

`invite`

The invitation accepted by the player.

`completionHandler`

A block to be called when the match has been created. This block receives the following parameters:

match

If the match was successfully created, this parameter contains the created match. Otherwise, this parameter is `nil`.

error

If the match was successfully created, this parameter contains `nil`. Otherwise, this parameter holds an error object that describes the error that occurred.

Discussion

When using this method to create a match, your game should display its own user interface to inform the player that he or she has been connected to a match.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a different thread than the thread originally used to invoke the method. This means that the code in your block needs to be thread safe.

Availability

Available in iOS 6.0 and later.

See Also

`@property inviteHandler` (page 100)

Declared in

`GKMatchmaker.h`

queryActivityWithCompletionHandler:

Initiates a search for activity in all player groups.

```
- (void)queryActivityWithCompletionHandler:(void (^)(NSInteger activity, NSError
*error))completionHandler
```

Parameters

`completionHandler`

A block that takes the following parameters:

activity

The amount of activity in the player group.

error

If the search completed successfully, this parameter is `nil`; otherwise, this parameter holds an error object that describes the error that occurred.

Discussion

A query allows your game to see how many players have recently searched for a match, across all player groups.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a different thread than the thread originally used to invoke the method. This means that the code in your block needs to be thread safe.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

queryPlayerGroupActivity:withCompletionHandler:

Queries Game Center for the activity in a player group.

```
- (void)queryPlayerGroupActivity:(NSUInteger)playerGroup withCompletionHandler:(void
 (^)(NSInteger activity, NSError *error))completionHandler
```

Parameters

`playerGroup`

A number that uniquely identifies a subset of players of your game.

`completionHandler`

A block that is called when the search completes. The block takes the following parameters:

activity

The amount of activity in the player group.

error

If the search completed successfully, this parameter is `nil`; otherwise, this parameter holds an error object that describes the error that occurred.

Discussion
A query allows your game to see how many players have recently searched for a match. As a result, you can present a user interface that shows the relative activity in each player group. For example, if one group sees less activity than others, you might display a warning so that players are aware that finding a match may take longer.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a different thread than the thread originally used to invoke the method. This means that the code in your block needs to be thread safe.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

startBrowsingForNearbyPlayersWithReachableHandler:

Enables the matchmaking process to find nearby players.

```
- (void)startBrowsingForNearbyPlayersWithReachableHandler:(void (^)(NSString
*playerID, BOOL reachable))reachableHandler
```

Parameters

`reachableHandler`

A block called when the reachability for a player changes. The block takes the following parameters:

playerID

The player identifier for the player whose reachability status has changed.

reachable

`YES` if a new player has been discovered locally, `NO` if a previously discovered player has disappeared.

Discussion

You only use this method when you are implementing programmatic matchmaking. After enabling browsing for nearby players, use the responses to populate your user interface with information about nearby players. If a player wants to invite a player to a game, add that player's player identifier to a match request and call either the `findMatchForRequest:withCompletionHandler:` (page 103) to create a match or `addPlayersToMatch:matchRequest:completionHandler:` (page 102) method to update a match.

Availability

Available in iOS 6.0 and later.

Declared in

`GKMatchmaker.h`

stopBrowsingForNearbyPlayers

Ends the search for nearby players.

```
- (void)stopBrowsingForNearbyPlayers
```

Availability

Available in iOS 6.0 and later.

Declared in

`GKMatchmaker.h`

GKMatchmakerViewController Class Reference

Inherits from	UINavigationController : UIViewController : UIResponder : NSObject
Conforms to	NSCoding (UIViewController) UIAppearanceContainer (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKMatchmakerViewController.h
Companion guide	Game Center Programming Guide

Overview

The `GKMatchmakerViewController` class is used to present a standard user interface to the player. This interface allows them to invite friends to a match or to allow Game Center to fill the remaining players needed for a match.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication, see *Game Center Programming Guide*.

To show a matchmaking screen, initialize a new `GKMatchmakerViewController` object and set the delegate. Configure the view controller's other properties to match your specific needs, then present the new view controller. and wait for the delegate to be called. The view controller's delegate is notified when the matchmaking process is completed or canceled. In either situation, you dismiss the view controller.

If the user is creating the match, your game initializes the matchmaker view controller by creating a `GKMatchRequest` object that describes the desired match. This match request is passed to the `initWithMatchRequest:` (page 115) method. When this view controller is displayed, the local player can invite other players into the match.

If your game receives an invitation from another player, it receives a `GKInvite` object representing the match the player was invited to. You initialize the matchmaker view controller by passing the `GKInvite` object received from Game Kit to the `initWithInvite:` (page 114) method. When this view controller is presented to the player, the player joins the existing match, but is not allowed to invite others to the match.

On iOS, you present and dismiss the view controller from another view controller in your game, using the methods provided by the `UIViewController` class. On OS X, you use the `GKDialogController` class to present and dismiss the view controller.

Tasks

Initializing a Matchmaker View Controller

– `initWithInvite:` (page 114)
Initializes a matchmaker view controller to respond to an invitation received from another player.

– `initWithMatchRequest:` (page 115)
Initializes a matchmaker view controller to create a new match.

Getting and Setting the Delegate

`matchmakerDelegate` (page 113) *property*
The delegate for the matchmaker view controller.

Matchmaker View Controller Properties

`defaultInvitationMessage` (page 112) *property*
The default invitation message used to initialize an invitation.

`hosted` (page 112) *property*
A Boolean value that indicates whether the match is hosted or peer-to-peer.

`matchRequest` (page 113) *property*
The configuration for the desired match. (read-only)

Adding Players to an Existing Match

– `addPlayersToMatch:` (page 114)

Implementing Hosted Matches

– `setHostedPlayer:connected:` (page 115)

Updates a player's status on the view to show that the player has connected or disconnected from your server.

– `setHostedPlayerReady:` (page 116)

Informs the controller that a player has joined a hosted match. (Deprecated. Deprecated. Use `setHostedPlayer:connected:` (page 115) instead.)

Properties

defaultInvitationMessage

The default invitation message used to initialize an invitation.

```
@property(nonatomic, copy) NSString *defaultInvitationMessage
```

Discussion

Your game sets this property to change the default invitation text displayed when the local player creates a new invitation. The local player may edit the text before sending the invitation.

Availability

Available in iOS 5.0 and later.

Declared in

`GKMatchmakerViewController.h`

hosted

A Boolean value that indicates whether the match is hosted or peer-to-peer.

```
@property(nonatomic, assign, getter=isHosted) BOOL hosted
```

Discussion
The value of the `hosted` property determines which methods of the delegate are called when the match is complete. If `YES`, this is a hosted match, and the delegate's `matchmakerViewController:didFindPlayers` method is to provide the list of players to your game. If `NO`, this is a peer-to-peer match, and `matchmakerViewController:didCreateMatch` is called with a `GKMatch` object. The default value is `NO`.

Hosted matches require you to provide a server that hosts the participants in the match. For more information on implementing hosted matches, see "Multiplayer".

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmakerViewController.h`

matchmakerDelegate

The delegate for the matchmaker view controller.

```
@property(nonatomic, assign) id<GKMatchmakerViewControllerDelegate> matchmakerDelegate
```

Discussion
A delegate is required to receive feedback when the match is created.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmakerViewController.h`

matchRequest

The configuration for the desired match. (read-only)

```
@property(nonatomic, readonly, retain) GKMatchRequest *matchRequest
```

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmakerViewController.h`

Instance Methods

addPlayersToMatch:

```
- (void)addPlayersToMatch:(GKMatch *)match
```

Parameters

`match`

An existing match that you want to add players to.

Discussion

Your game calls this method prior to presenting the view controller to the player. Calling this method instructs the view controller to add new players to the provided match rather than creating a new match.

When called, this method sets the delegate on the match to `nil` and updates the view controller's user interface to display the players already connected to the match.

> **Important:** Only one device connected to the match should call this method.

Availability

Available in iOS 5.0 and later.

Declared in

`GKMatchmakerViewController.h`

initWithInvite:

Initializes a matchmaker view controller to respond to an invitation received from another player.

```
- (id)initWithInvite:(GKInvite *)invite
```

Parameters

`invite`

The invitation received from the other player.

Return Value

An initialized matchmaker view controller object. If an error occurred, `NULL` is returned.

Discussion

The user is allowed to join the match that the user was invited to, but is not allowed to invite others to the match.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmakerViewController.h`

initWithMatchRequest:

Initializes a matchmaker view controller to create a new match.

```
- (id)initWithMatchRequest:(GKMatchRequest *)request
```

Parameters
`request`
A request containing the characteristics for the desired match.

Return Value
An initialized matchmaker view controller object. If an error occurred, `NULL` is returned.

Discussion
Your game uses the `initWithMatchRequest:` method when it wants the local user to create a new match.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmakerViewController.h`

setHostedPlayer:connected:

Updates a player's status on the view to show that the player has connected or disconnected from your server.

```
- (void)setHostedPlayer:(NSString *)playerID connected:(BOOL)connected
```

Parameters
`playerID`
The identifier string for a player that connected to the external server.

`connected`
A Boolean value that states whether the player is connected to the hosted match.

Discussion

When setting up a hosted match, each device should instantiate a matchmaker view controller and display it to the player. Then, when a new player connects to your server, your server should notify all participating devices already connected to your server. Each participating device should then call this method to update that player's status in the matchmaking interface. Similarly, if a player disconnects from the server, your server should inform each device so that the devices can update their user interface.

Availability

Available in iOS 5.0 and later.

Declared in

`GKMatchmakerViewController.h`

setHostedPlayerReady:

Informs the controller that a player has joined a hosted match. (Deprecated in iOS 5.0. Deprecated. Use `setHostedPlayer:connected:` *(page 115) instead.)*

```
- (void)setHostedPlayerReady:(NSString *)playerID
```

Parameters

`player`

The identifier string for a player that connected to the external server.

Discussion

In a hosted match, when a new player connects to the server, your server should inform all participating devices connected to the match. Each participating device must separately call this method to update its matchmaking user interface.

Availability

Available in iOS 4.1 and later.

Deprecated in iOS 5.0.

Declared in

`GKMatchmakerViewController.h`

GKMatchRequest Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKMatchmaker.h
Companion guide	Game Center Programming Guide

Overview

A `GKMatchRequest` object is used to specify the parameters for a new live or turn-based match. You initialize a match request object, then pass it to another object to actually create the match. The kind of object you pass it to depends on which kind of match you want and whether you want to display the standard matchmaking user interface. See Table 16-1 (page 117).

Table 16-1 The class of object that receives the match request object.

	Live Match	Turn-based Match
Creating a match using the standard user interface	`GKMatchmakerView-Controller`	`GKTurnBasedMatch`
Creating a match programmatically for a custom user interface	`GKMatchmaker`	`GKTurnBased-MatchmakerView-Controller`

Tasks

Determining the Number of Players Allowed in the Game

`+ maxPlayersAllowedForMatchOfType:` (page 122)
Returns the maximum number of players allowed in the match request for a given match type.

`maxPlayers` (page 120) *property*
The maximum number of players that may join the match.

`minPlayers` (page 120) *property*
The minimum number of players that may join the match.

`defaultNumberOfPlayers` (page 119) *property*
The default number of players for the match.

Setting an Invite Message

`inviteMessage` (page 120) *property*
The string displayed on another player's device when invited to join a match.

Creating Subsets of Players

`playerGroup` (page 121) *property*
A number identifying a subset of players allowed to join the match.

`playerAttributes` (page 121) *property*
A mask that specifies the role that the local player would like to play in the game.

Inviting an Initial Group of Players

`playersToInvite` (page 122) *property*
A list of player identifers for players to invite to the match.

`inviteeResponseHandler` (page 119) *property*
A block to be called when a response from an invited player is returned to your game.

Properties

defaultNumberOfPlayers

The default number of players for the match.

```
@property(nonatomic, assign) NSUInteger defaultNumberOfPlayers
```

Discussion

If this property is not set, then the default number of players is equal to the value stored in the `maxPlayers` (page 120) property. The default number of players determines the number of invitees shown in the standard matchmaking user interface. The player can choose to override this to add or remove slots.

Availability

Available in iOS 6.0 and later.

Declared in

`GKMatchmaker.h`

inviteeResponseHandler

A block to be called when a response from an invited player is returned to your game.

```
@property(nonatomic, copy) void(^inviteeResponseHandler)(NSString *playerID, GKInviteeResponse
 response)
```

Discussion

The block takes the following parameters:

playerID

The identifier for the player.

GKInviteeResponse

The nature of the response. See "Invitee Responses" (page 124).

An invitee response handler is called whenever you programmatically invite specific players to join a match. It is called once for each player invited to the match. Typically, your game uses the responses to update the custom user interface. For example, you want the player to be able to perform any of the following tasks:

- Start the match.
- Invite an additional set of specific players.
- Use matchmaking to fill the remaining match slots.

Availability
Available in iOS 6.0 and later.

Declared in
GKMatchmaker.h

inviteMessage

The string displayed on another player's device when invited to join a match.

```
@property(nonatomic, copy) NSString *inviteMessage
```

Availability
Available in iOS 6.0 and later.

Declared in
GKMatchmaker.h

maxPlayers

The maximum number of players that may join the match.

```
@property(nonatomic, assign) NSUInteger maxPlayers
```

Discussion
The maximum number of players must be equal or greater than the minimum number of players. The maximum number of players may be no more than value returned by the `maxPlayersAllowedForMatchOfType:` (page 122) method for the kind of match you plan to create.

Availability
Available in iOS 4.1 and later.

Declared in
GKMatchmaker.h

minPlayers

The minimum number of players that may join the match.

```
@property(nonatomic, assign) NSUInteger minPlayers
```

Discussion
The minimum number of players must be at least 2.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

playerAttributes

A mask that specifies the role that the local player would like to play in the game.

```
@property(nonatomic, assign) uint32_t playerAttributes
```

Discussion
If this value is `0` (the default), this property is ignored. If the value is nonzero, then automatching uses the value as a mask that restricts the role the player can play in the group. Automatching with player attributes matches new players into the game so that the bitwise OR of the masks of all the players in the resulting match equals `0xFFFFFFFF`.

For more information, see *Game Center Programming Guide* .

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

playerGroup

A number identifying a subset of players allowed to join the match.

```
@property(nonatomic, assign) NSUInteger playerGroup
```

Discussion
If your game sets the `playerGroup` property, only players whose requests share the same `playerGroup` value are automatched by Game Center. The value of a player group is arbitrary. For example, you could define different `playerGroup` values to implement any of the following filters:

- A game could restrict players based on skill level.

- A game that provides multiple game modes could use it to filter players into the specific game they want to play.
- A game that provides bonus content through in-app purchase could match players who own the same content with each other.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

playersToInvite

A list of player identifers for players to invite to the match.

```
@property(nonatomic, retain) NSArray *playersToInvite
```

Discussion
The property holds an array of `NSString` objects, each of which is an identifier for a player on Game Center. If the value of the property is non-`nil`, when you use the request to create a match, Game Center invites those players to the match. If `nil` (the default), no players are invited. The exact behavior for matchmaking depends on the kind of match being created and the class used to create the match. For more information, see *Game Center Programming Guide* .

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatchmaker.h`

Class Methods

maxPlayersAllowedForMatchOfType:

Returns the maximum number of players allowed in the match request for a given match type.

```
+ (NSUInteger)maxPlayersAllowedForMatchOfType:(GKMatchType)matchType
```

Parameters

`matchType`

The kind of match. See "Match Type" (page 123).

Return Value

The maximum number of allowed players for that type of match.

Availability

Available in iOS 6.0 and later.

Declared in

`GKMatchmaker.h`

Constants

Match Type

The kind of match being created.

```
enum {
   GKMatchTypePeerToPeer,
   GKMatchTypeHosted,
   GKMatchTypeTurnBased
};
typedef NSUInteger GKMatchType;
```

Constants

`GKMatchTypePeerToPeer`

A peer-to-peer match hosted by Game Center. It is represented by a `GKMatch` object.

Available in iOS 6.0 and later.

Declared in `GKMatchmaker.h`.

`GKMatchTypeHosted`

A match hosted on your private server.

Available in iOS 6.0 and later.

Declared in `GKMatchmaker.h`.

`GKMatchTypeTurnBased`

A turn-based match hosted by Game Center. It is represented by a `GKTurnBasedMatch` object.

Available in iOS 6.0 and later.

Declared in `GKMatchmaker.h`.

Invitee Responses

Possible responses from an invitation to a remote player.

```
enum {
   GKInviteeResponseAccepted           = 0,
   GKInviteeResponseDeclined           = 1,
   GKInviteeResponseFailed             = 2,
   GKInviteeResponseIncompatible       = 3,
   GKInviteeResponseUnableToConnect    = 4,
   GKInviteeResponseNoAnswer           = 5,
};
typedef NSInteger GKInviteeResponse;
```

Constants

`GKInviteeResponseAccepted`

The player accepted the invitation.

Available in iOS 6.0 and later.

Declared in `GKMatchmaker.h`.

`GKInviteeResponseDeclined`

The player rejected the invitation.

Available in iOS 6.0 and later.

Declared in `GKMatchmaker.h`.

`GKInviteeResponseFailed`

The invitation was unable to be delivered.

Available in iOS 6.0 and later.

Declared in `GKMatchmaker.h`.

`GKInviteeResponseIncompatible`

The invitee is not running a compatible version of your game.

Available in iOS 6.0 and later.

Declared in `GKMatchmaker.h`.

`GKInviteeResponseUnableToConnect`

The invitee could not be contacted.

Available in iOS 6.0 and later.

Declared in `GKMatchmaker.h`.

`GKInviteeResponseNoAnswer`

The invitation timed out without an answer.

Available in iOS 6.0 and later.

Declared in `GKMatchmaker.h`.

GKNotificationBanner Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 5.0 and later.
Declared in	GKNotificationBanner.h
Companion guide	Game Center Programming Guide

Overview

The `GKNotificationBanner` class allows your game to display a notification banner that displays text to the player. The behavior of this banner is identical to other banners used by Game Kit.

Important: Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication, see *Game Center Programming Guide*.

Tasks

Displaying a Notification Banner

`+ showBannerWithTitle:message:completionHandler:` (page 126)

Displays a banner to the player.

`+ showBannerWithTitle:message:duration:completionHandler:` (page 126)

Displays a banner to the player for a specified period of time.

Class Methods

showBannerWithTitle:message:completionHandler:

Displays a banner to the player.

```
+ (void)showBannerWithTitle:(NSString *)title message:(NSString *)message
completionHandler:(void (^)(void))completionHandler
```

Parameters

`title`

The title of the banner.

`message`

A secondary message to be displayed.

`completionHandler`

A block to be called after the score is reported.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 5.0 and later.

Declared in

`GKNotificationBanner.h`

showBannerWithTitle:message:duration:completionHandler:

Displays a banner to the player for a specified period of time.

```
+ (void)showBannerWithTitle:(NSString *)title message:(NSString *)message
duration:(NSTimeInterval)duration completionHandler:(void (^)(void))completionHandler
```

Parameters

`title`

The title of the banner.

`message`

A secondary message to be displayed.

`duration`

The amount of time that the banner should be displayed to the player.

`completionHandler`

A block to be called after the score is reported.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 6.0 and later.

Declared in

`GKNotificationBanner.h`

GKPeerPickerController Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 3.0 and later.
Declared in	GKPeerPickerController.h
Companion guide	Game Center Programming Guide
Related sample code	GKTank

Overview

The `GKPeerPickerController` class provides a standard user interface to allow one iOS device to discover and connect to another. The result is a configured `GKSession` object connecting the two devices. To use a `GKPeerPickerController` object, your application creates the controller, adds a delegate, configures the allowed connection types, and then shows the peer picker. The delegate is called as the user makes selections within the peer picker interface.

In iOS 3.0, the peer picker can be configured to select between Bluetooth and Internet connections.

> **Important:** Although users can select Internet connections in the peer picker, `GKPeerPickerController` has no user interface for configuring them. If your application configures the peer picker to allow Internet connections, your application must also dismiss the peer picker and present its own interface to configure an Internet connection.

On iOS 3.0, your application should release the peer picker object after it dismisses the peer picker dialog. On iOS 3.1 or later, your application may release the peer picker after it is shown to the user. If you do this, the peer picker controller is automatically deallocated after the dialog is dismissed.

Tasks

Setting and Getting the Delegate

`delegate` (page 130) *property*
The delegate of the peer picker controller.

Displaying the Picker Dialog

– `show` (page 131)
Displays the peer picker dialog to the user.

– `dismiss` (page 131)
Hides the peer picker dialog.

`visible` (page 130) *property*
A Boolean value that indicates whether the picker dialog is visible. (read-only)

Configuring Connectivity Options

`connectionTypesMask` (page 129) *property*
A mask that determines the types of connections a dialog presents to the user.

Properties

connectionTypesMask

A mask that determines the types of connections a dialog presents to the user.

```
@property(nonatomic, assign) GKPeerPickerConnectionType connectionTypesMask
```

Discussion
Your application configures the connection types it allows before showing the peer picker. If your application allows more than one connection type, the peer picker offers the user a choice of which type of connection to use. The default value for the mask is `GKPeerPickerConnectionTypeNearby` (page 132).

Important: In iOS 3.0, `GKPeerPickerConnectionTypeNearby` (page 132) is required to be one of the allowed connection types. An exception is thrown if your application does not include it.

Availability
Available in iOS 3.0 and later.

Declared in
`GKPeerPickerController.h`

delegate

The delegate of the peer picker controller.

```
@property(nonatomic, assign) id<GKPeerPickerControllerDelegate> delegate
```

Discussion
The delegate must adopt the `GKPeerPickerControllerDelegate` formal protocol.

Availability
Available in iOS 3.0 and later.

Related Sample Code
GKTank

Declared in
`GKPeerPickerController.h`

visible

A Boolean value that indicates whether the picker dialog is visible. (read-only)

```
@property(readonly, getter=isVisible) BOOL visible
```

Availability
Available in iOS 3.0 and later.

Declared in
`GKPeerPickerController.h`

Instance Methods

dismiss

Hides the peer picker dialog.

```
- (void)dismiss
```

Discussion

The controller's delegate is responsible for dismissing the peer picker when it is no longer needed.

On iOS 3.1 or later, the peer picker is retained when it is shown, and autoreleased when it is dismissed.

Availability

Available in iOS 3.0 and later.

Related Sample Code

GKTank

Declared in

`GKPeerPickerController.h`

show

Displays the peer picker dialog to the user.

```
- (void)show
```

Discussion

On iOS 3.1 or later, the peer picker is retained when it is shown, and autoreleased when it is dismissed.

Availability

Available in iOS 3.0 and later.

Related Sample Code

GKTank

Declared in

`GKPeerPickerController.h`

Constants

GKPeerPickerConnectionType

Network connections available to the peer picker dialog.

```
enum {
   GKPeerPickerConnectionTypeOnline = 1 << 0,
   GKPeerPickerConnectionTypeNearby = 1 << 1
};
typedef NSUInteger GKPeerPickerConnectionType;
```

Constants

`GKPeerPickerConnectionTypeOnline`

An Internet-based connection.

Available in iOS 3.0 and later.

Declared in `GKPeerPickerController.h`.

`GKPeerPickerConnectionTypeNearby`

A Bluetooth connection to a device.

Available in iOS 3.0 and later.

Declared in `GKPeerPickerController.h`.

GKPlayer Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKPlayer.h
Companion guide	Game Center Programming Guide
Related sample code	GKTapper

Overview

`GKPlayer` objects provide information about a player on Game Center.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication see *Game Center Programming Guide*.

Every player account on Game Center is permanently assigned a unique **player identifier** string. Your game should use this string to store per-player information or to disambiguate between players. In most cases, Game Kit classes return player identifier strings to your game. For example, in a multiplayer match, the `GKMatch` object's `playerIDs` (page 91) property contains an array of the player identifiers for all the players connected to the match.

To load information about a set of players, your game calls the `loadPlayersForIdentifiers:withCompletionHandler:` (page 136) method, passing in an array of identifier strings. Your completion handler is called after the player data is downloaded from Game Center. For performance and resource reasons, player objects returned by the `loadPlayersForIdentifiers:withCompletionHandler:` (page 136) method do not include player photos. To load the photo associated with a player, call the player object's `loadPhotoForSize:withCompletionHandler:` (page 137) method.

Tasks

Loading Player Details

`+ loadPlayersForIdentifiers:withCompletionHandler:` (page 136)
Loads information from Game Center about a list of players.

Identifying the Player

`playerID` (page 136) *property*
A string assigned by Game Center to uniquely identify a player. (read-only)

Player Details

`alias` (page 134) *property*
A string chosen by the player to identify themselves to other players. (read-only)

`displayName` (page 135) *property*
A string to display for the player. (read-only)

`isFriend` (page 135) *property*
A Boolean value that indicates whether this player is a friend of the local player. (read-only)

Player Photos

`- loadPhotoForSize:withCompletionHandler:` (page 137)
Loads a photo of this player from Game Center.

Properties

alias

A string chosen by the player to identify themselves to other players. (read-only)

```
@property(nonatomic, readonly, copy) NSString *alias
```

Discussion
A player's alias is used when a player is not a friend of the local player. Typically, you never display the alias string directly in your user interface. Instead use the `displayName` (page 135) property.

Availability
Available in iOS 4.1 and later.

Declared in
`GKPlayer.h`

displayName

A string to display for the player. (read-only)

```
@property(nonatomic, readonly) NSString *displayName
```

Discussion
The display name for a player depends on whether the player is a friend of the local player authenticated on the device. If the player is a friend of the local player, then the display name is the actual name of the player. If the player is not a friend, then the display name is the player's alias.

Availability
Available in iOS 6.0 and later.

Declared in
`GKPlayer.h`

isFriend

A Boolean value that indicates whether this player is a friend of the local player. (read-only)

```
@property(nonatomic, readonly) BOOL isFriend
```

Discussion
Players use the Game Center app to declare other players as friends.

Availability
Available in iOS 4.1 and later.

Declared in
`GKPlayer.h`

playerID

A string assigned by Game Center to uniquely identify a player. (read-only)

```
@property(nonatomic, readonly, retain) NSString *playerID
```

Discussion
The player identifier should never be displayed to the player. Use it only as a way to identify a particular player.

Do not make assumptions about the contents of the player identifier string. Its format and length are subject to change.

Availability
Available in iOS 4.1 and later.

Declared in
`GKPlayer.h`

Class Methods

loadPlayersForIdentifiers:withCompletionHandler:

Loads information from Game Center about a list of players.

```
+ (void)loadPlayersForIdentifiers:(NSArray *)identifiers withCompletionHandler:(void
 (^)(NSArray *players, NSError *error))completionHandler
```

Parameters

`identifiers`

An array of `NSString` objects, each a unique identifier for a Game Center player.

`completionHandler`

A block to be called when the player data is retrieved from Game Center.

The block receives the following parameters:

players

An array of `GKPlayer` objects, one per identifier. If an error occurred, this may be non-`nil`. In that case, the array holds whatever data Game Kit was able to retrieve for the requested players.

error

If an error occurred, this error object describes the error. If the operation completed successfully, this is `nil`.

Discussion
When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability
Available in iOS 4.1 and later.

Related Sample Code
GKTapper

Declared in
`GKPlayer.h`

Instance Methods

loadPhotoForSize:withCompletionHandler:

Loads a photo of this player from Game Center.

```
- (void)loadPhotoForSize:(GKPhotoSize)size withCompletionHandler:(void (^)(UIImage *photo, NSError *error))completionHandler
```

Parameters

`size`

A constant that determines the size of the photo to load.

`completionHandler`

A block to be called when the player data is retrieved from Game Center.

The block receives the following parameters:

photo

An image for the player. If an error occurred, this may still be non-`nil`. In this case, the image reflects an image cached by Game Kit on the device.

error

If an error occurred, this error object describes the error. If the operation completed successfully, this is `nil`.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

> **Important:** The size of the image returned to your game is dependent on both the constant you provided in the initial request and the user interface idiom of the device your game is running on.

> **Note:** On OS X, the returned image is an `NSImage` object.

Availability

Available in iOS 5.0 and later.

Declared in

`GKPlayer.h`

Constants

Player Photo Sizes

The size of a photo loaded by Game Center.

```
enum {
   GKPhotoSizeSmall = 0,
   GKPhotoSizeNormal,
};
typedef NSInteger GKPhotoSize;
```

Constants

`GKPhotoSizeSmall`

Load a small photo.

Available in iOS 5.0 and later.

Declared in `GKPlayer.h`.

`GKPhotoSizeNormal`

Load a normal sized photo.

Available in iOS 5.0 and later.

Declared in `GKPlayer.h`.

Notifications

GKPlayerDidChangeNotificationName

Posted when a player object's data changes.

Availability
Available in iOS 4.1 and later.

Declared in
`GKPlayer.h`

GKScore Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKScore.h GKScore+Sharing.h
Companion guide	Game Center Programming Guide
Related sample code	GKLeaderboards GKTapper

Overview

A `GKScore` class holds information for a score that was earned by the player. Your game creates `GKScore` objects to post scores to a leaderboard on Game Center. When your game retrieves score information from a leaderboard, those scores are returned as `GKScore` objects.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication, see *Game Center Programming Guide*.

To report a score to Game Center, your game allocates and initializes a new object, sets the `value` property to the score the player earned, and then calls the `reportScoreWithCompletionHandler:` (page 148) method. The mechanism your game uses to calculate scores is up to you to design; scores are only compared within your game.

Tasks

Initializing a Score Object

– `initWithCategory:` (page 146)
Returns an initialized score object.

Score Properties

`playerID` (page 144) *property*
The player identifier for the player that earned the score. (read-only)

`category` (page 142) *property*
The leaderboard that this score belongs to.

`date` (page 143) *property*
The date and time when the score was earned. (read-only)

`value` (page 145) *property*
The score earned by the player.

`context` (page 142) *property*
An integer value used by your game.

`formattedValue` (page 143) *property*
Returns the player's score as a localized string. (read-only)

`rank` (page 144) *property*
The position of the score in the results of a leaderboard search. (read-only)

Reporting a New Score

+ `reportScores:withCompletionHandler:` (page 146)
Reports a list of scores to Game Center

– `reportScoreWithCompletionHandler:` (page 148)
Reports a score to Game Center.

Changing the Default Leaderboard

`shouldSetDefaultLeaderboard` (page 145) *property*

A Boolean value that indicates whether this score should also update the default leaderboard.

Issuing a Score Challenge

`– issueChallengeToPlayers:message:` (page 147)

Issues a score challenge to a set of players.

Properties

category

The leaderboard that this score belongs to.

```
@property(nonatomic, retain) NSString *category
```

Discussion

The category string must match an identifier for a leaderboard you created on iTunes Connect.

Availability

Available in iOS 4.1 and later.

Declared in

`GKScore.h`

context

An integer value used by your game.

```
@property(nonatomic, assign) uint64_t context
```

Discussion

The `context` (page 142) property is stored and returned to your game, but is otherwise ignored by Game Center. It allows your game to associate an arbitrary 64-bit unsigned integer value with the score data reported to Game Center. You decide how this integer value is interpreted by your game. For example, you might use the

`context` (page 142) property to store flags that provide game-specific details about a player's score, or you might use the context as a key to other data stored on the device or on your own server. The context is most useful when your game displays a custom leaderboard user interface.

Availability
Available in iOS 5.0 and later.

Declared in
`GKScore.h`

date

The date and time when the score was earned. (read-only)

```
@property(nonatomic, readonly, retain) NSDate *date
```

Discussion
When you initialize the new score object, the `date` property is automatically set to the current date and time.

Availability
Available in iOS 4.1 and later.

Declared in
`GKScore.h`

formattedValue

Returns the player's score as a localized string. (read-only)

```
@property(nonatomic, readonly, retain) NSString *formattedValue
```

Discussion
This property is invalid on a newly initialized score object. On a score returned from Game Kit, it contains a formatted string based on the player's score. You determine how a score is formatted when you define the leaderboard on iTunes Connect.

Never convert the `value` property into a string directly; always use this method to receive the formatted string.

Availability
Available in iOS 4.1 and later.

See Also
`@property value` (page 145)

Related Sample Code
GKTapper

Declared in
`GKScore.h`

playerID

The player identifier for the player that earned the score. (read-only)

```
@property(nonatomic, readonly, retain) NSString *playerID
```

Discussion
When you initialize a new score object, the `playerID` property is set to the identifier for the local player. If the score object was returned to your game by loading scores from Game Center, the `playerID` property identifies the player who recorded that score.

Availability
Available in iOS 4.1 and later.

Declared in
`GKScore.h`

rank

The position of the score in the results of a leaderboard search. (read-only)

```
@property(nonatomic, readonly, assign) NSInteger rank
```

Discussion
The value of this property is only valid on score objects returned from Game Center. The `rank` property represents the position of the score in the returned results, with `1` being the best score, `2` being the second best, and so on.

Availability
Available in iOS 4.1 and later.

Declared in
`GKScore.h`

shouldSetDefaultLeaderboard

A Boolean value that indicates whether this score should also update the default leaderboard.

```
@property(nonatomic, assign) BOOL shouldSetDefaultLeaderboard
```

Discussion

If the value of this property is `YES`, when the score is reported to Game Center, Game Center also updates the local player's default leaderboard to match the value stored in the `category` (page 142) property of the score object. This matches the behavior of the `GKLeaderboard` class's `setDefaultLeaderboard:withCompletionHandler:` (page 70) class method. If the value of this property is `NO`, the default leaderboard is not changed by reporting the score. The default value of this property is `NO`.

Availability

Available in iOS 5.0 and later.

Declared in

`GKScore.h`

value

The score earned by the player.

```
@property(nonatomic, assign) int64_t value
```

Discussion

You can use any algorithm you want to calculate scores in your game. Your game must set the `value` property before reporting a score, otherwise an error is returned.

The value provided by a score object is interpreted by Game Center only when formatted for display. You determine how your scores are formatted when you define the leaderboard on iTunes Connect.

Availability

Available in iOS 4.1 and later.

See Also

`@property formattedValue` (page 143)

Related Sample Code

GKLeaderboards

GKTapper

Declared in

`GKScore.h`

Class Methods

reportScores:withCompletionHandler:

Reports a list of scores to Game Center

```
+ (void)reportScores:(NSArray *)scores withCompletionHandler:(void (^)(NSError
*error))completionHandler
```

Parameters

`scores`

An array of score objects to report to Game Center.

`completionHandler`

A block to be called after the score is reported.

The block receives the following parameter:

error

If an error occurred, this parameter holds an error object that describes the problem. If the score was successfully reported, this parameter's value is `nil`.

Discussion

Use this class method whenever you need to submit multiple scores at the same time. Calling this method reports each of the scores, exactly as if you called the `reportScoreWithCompletionHandler:` (page 148) method on each score object in the array. However, the entire operation can typically be processed more efficiently using this method, and the completion handler is only called once.

Availability

Available in iOS 6.0 and later.

Declared in

`GKScore.h`

Instance Methods

initWithCategory:

Returns an initialized score object.

```
- (id)initWithCategory:(NSString *)category
```

Parameters

`category`

An identifier for a specific leaderboard you've configured on iTunes Connect. Must not be `nil`.

Return Value

An initialized score object, or `nil` if an error occurred.

Discussion

Your game explicitly allocates and initializes a score object when it needs to report a new score to Game Center.

Availability

Available in iOS 4.1 and later.

Related Sample Code
GKLeaderboards

GKTapper

Declared in

`GKScore.h`

issueChallengeToPlayers:message:

Issues a score challenge to a set of players.

```
- (void)issueChallengeToPlayers:(NSArray *)playerIDs message:(NSString *)message
```

Parameters

`playerIDs`

An array of player identifiers for the players to challenge.

`message`

A text message to display to the players.

Discussion

Your game should only issue a challenge request in direct response to a player action. That is, your game should provide a user interface that allows the player to choose to issue a challenge, and only issue a challenge when the player wishes to do so.

Availability

Available in iOS 6.0 and later.

Declared in

`GKChallenge.h`

reportScoreWithCompletionHandler:

Reports a score to Game Center.

```
– (void)reportScoreWithCompletionHandler:(void (^)(NSError *error))completionHandler
```

Parameters

`completionHandler`

A block to be called after the score is reported.

The block receives the following parameter:

error

If an error occurred, this parameter holds an error object that describes the problem. If the score was successfully reported, this parameter's value is `nil`.

Discussion

The `value` property must be set before calling this method.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 4.1 and later.

Related Sample Code

GKLeaderboards

GKTapper

Declared in

`GKScore.h`

GKScoreChallenge Class Reference

Inherits from	GKChallenge : NSObject
Conforms to	NSCoding (GKChallenge) NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 6.0 and later.
Declared in	GKChallenge.h
Companion guide	Game Center Programming Guide

Overview

A `GKScoreChallenge` object represents a challenge based on a score in a leaderboard. To complete the challenge, the player must score an equal or better score than the score used to create the challenge. When a player beats a score challenge, a new score challenge is automatically issued to the player that issued the challenge unless there is already a pending score challenge that requires a better score.

Tasks

Obtaining the Score to Beat

`score` (page 150) *property*

The score to beat. (read-only)

Properties

score

The score to beat. (read-only)

```
@property(nonatomic, readonly, retain) GKScore *score
```

Availability
Available in iOS 6.0 and later.

Declared in
`GKChallenge.h`

GKSession Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 3.0 and later.
Declared in	GKSession.h
Companion guide	Game Center Programming Guide
Related sample code	GKRocket GKTank

Overview

A `GKSession` object provides the ability to discover and connect to nearby iOS devices using Bluetooth or Wi-fi.

Sessions primarily work with **peers**. A peer is any iOS device made visible by creating and configuring a `GKSession` object. Each peer is identified by a unique identifier, called a peer id (`peerID`) string. Your application can use a `peerID` string to obtain a user-readable name for a remote peer and to attempt to connect to that peer. Similarly, your session's peer ID is visible to other nearby peers. After a connection is established, your application uses the remote peer's ID to address data packets that it wants to send.

Peers discover other peers by using a unique string to identify the service they implement, called a session ID (`sessionID`). Sessions can be configured to broadcast a session ID (as a **server**), to search for other peers advertising with that session ID (as a **client**), or to act as both a server and a client simultaneously (as a **peer**.

Your application controls the behavior of a session through a delegate that implements the `GKSessionDelegate` protocol. The delegate is called when remote peers are discovered, when those peers attempt to connect to the session, and when the state of a remote peer changes. Your application also provides a data handler to the session so that the session can forward data it receives from remote peers. The data handler can be a separate object or the same object as the delegate.

`GKSession` methods are thread-safe and may be called from any thread. However, the session always calls its delegate on the main thread.

Tasks

Creating a Session

– `initWithSessionID:displayName:sessionMode:` (page 161)
Initializes and returns a newly allocated session.

Setting and Getting the Delegate

`delegate` (page 154) *property*
The delegate of the session object.

Searching for Other Peers

`available` (page 154) *property*
A Boolean value that determines whether or not the session wants to connect to new peers.

Obtaining Information About Other Peers

– `peersWithConnectionState:` (page 162)
Returns a list of peers in the specified connection state.

– `displayNameForPeer:` (page 161)
Returns a user-readable name for a peer.

Connecting to a Remote Peer

– `connectToPeer:withTimeout:` (page 158)
Creates a connection to another iOS device.

– `cancelConnectToPeer:` (page 158)
Cancels a pending request to connect to another iOS device.

Receiving Connections from a Remote Peer

– `acceptConnectionFromPeer:error:` (page 157)
Called by the delegate to accept a connection request received from a remote peer.

– `denyConnectionFromPeer:` (page 159)
Called by the delegate to reject a connection request received from a remote peer.

Working with Connected Peers

– `setDataReceiveHandler:withContext:` (page 164)
Sets the object that handles data received from other peers connected to the session.

– `sendData:toPeers:withDataMode:error:` (page 163)
Transmits a collection of bytes to a list of connected peers.

– `sendDataToAllPeers:withDataMode:error:` (page 164)
Transmits a collection of bytes to all connected peers.

`disconnectTimeout` (page 155) *property*
A time interval that expresses how long the session waits before it disconnects a nonresponsive peer.

– `disconnectFromAllPeers` (page 160)
Disconnects the session from all connected peers.

– `disconnectPeerFromAllPeers:` (page 160)
Disconnects a connected peer from all peers connected to the session.

Information About the Session

`displayName` (page 155) *property*
The name of the user. (read-only)

`peerID` (page 156) *property*
A string that identifies your session to other peers. (read-only)

`sessionID` (page 156) *property*
A string used to filter the list of peers who are allowed to see your session. (read-only)

`sessionMode` (page 156) *property*
The mode the session uses to find other peers. (read-only)

Properties

available

A Boolean value that determines whether or not the session wants to connect to new peers.

```
@property(getter=isAvailable) BOOL available
```

Discussion

When `available` is `YES`, the session is visible to other peers based on its `sessionMode` (page 156) property. When `available` is set to `NO`, it remains connected to peers, but is no longer visible to nonconnected peers. The default is `NO`.

Typically, your application configures the session object with a delegate and data receiver, and then sets `available` (page 154) to `YES`. When the delegate finishes connecting to peers, it should set the session's `available` (page 154) property to `NO`.

Availability

Available in iOS 3.0 and later.

Related Sample Code

GKRocket

GKTank

Declared in

`GKSession.h`

delegate

The delegate of the session object.

```
@property(assign) id<GKSessionDelegate> delegate
```

Discussion

A session's delegate is responsible for observing changes to other peers running with the same session ID. Your application must set a delegate before making your session known to other peers.

Availability

Available in iOS 3.0 and later.

See Also

`GKSessionDelegate`

Related Sample Code
GKRocket
GKTank

Declared in
`GKSession.h`

disconnectTimeout

A time interval that expresses how long the session waits before it disconnects a nonresponsive peer.

```
@property(assign) NSTimeInterval disconnectTimeout
```

Discussion
The timeout is the waiting period before disconnecting a peer from the session. If a peer is disconnected, the delegate's `session:peer:didChangeState:` (page 263) method is called.

Availability
Available in iOS 3.0 and later.

Declared in
`GKSession.h`

displayName

The name of the user. (read-only)

```
@property(readonly) NSString *displayName
```

Discussion
The display name is transmitted to visible peers so that they can present a human-readable name for your session.

Availability
Available in iOS 3.0 and later.

See Also
– `displayNameForPeer:` (page 161)

Declared in
`GKSession.h`

peerID

A string that identifies your session to other peers. (read-only)

```
@property(readonly) NSString *peerID
```

Availability
Available in iOS 3.0 and later.

Related Sample Code
GKRocket

Declared in
`GKSession.h`

sessionID

A string used to filter the list of peers who are allowed to see your session. (read-only)

```
@property(readonly) NSString *sessionID
```

Discussion
The session ID is used by sessions configured as servers to advertise itself to other peers and by sessions configured as clients to search for compatible servers. The session ID should be the short name of an approved Bonjour service type.

Availability
Available in iOS 3.0 and later.

Declared in
`GKSession.h`

sessionMode

The mode the session uses to find other peers. (read-only)

```
@property(readonly) GKSessionMode sessionMode
```

Discussion
The session mode changes the behavior of the session when `available` (page 154) is set to `YES`.

- A `GKSessionModeServer` (page 166) session advertises itself to local devices using its session ID.

- A `GKSessionModeClient` (page 166) session searches for local devices advertising the same session ID. As it discovers available and compatible peers, it calls the delegate's `session:peer:didChangeState:` (page 263) method.
- A `GKSessionModePeer` (page 166) session both advertises as a server and searches as a client.

Availability
Available in iOS 3.0 and later.

See Also
`@property available` (page 154)

Declared in
`GKSession.h`

Instance Methods

acceptConnectionFromPeer:error:

Called by the delegate to accept a connection request received from a remote peer.

```
- (BOOL)acceptConnectionFromPeer:(NSString *)peerID error:(NSError **)error
```

Parameters

`peerID`
The string identifying the peer that initiated the connection to the session.

`error`
If an error occurred when connecting the peer, upon return contains an `NSError` object that explains the problem.

Return Value
`YES` if a connection was established to the remote peer; `NO` if an error occurred.

Discussion
When your session acts as a server, client peers can discover your session and attempt to connect to it. When a client attempts to connect to the session, the delegate's `session:didReceiveConnectionRequestFromPeer:` (page 262) method is called to decide whether the peer should be connected. Your application calls this method to accept the request, or `denyConnectionFromPeer:` (page 159) to reject it.

Availability
Available in iOS 3.0 and later.

See Also
– `denyConnectionFromPeer:` (page 159)

Declared in
`GKSession.h`

cancelConnectToPeer:

Cancels a pending request to connect to another iOS device.

```
- (void)cancelConnectToPeer:(NSString *)peerID
```

Parameters
`peerID`

The string identifying the peer you previously requested to connect to.

Discussion
Your application previously called `connectToPeer:withTimeout:` (page 158) to create a connection to another iOS device. When your application cancels the connection attempt, both delegates' `session:connectionWithPeerFailed:withError:` (page 261) methods are called.

If your application already connected to the peer, your application should call `disconnectFromAllPeers` (page 160) instead.

Availability
Available in iOS 3.0 and later.

See Also
– `connectToPeer:withTimeout:` (page 158)

Related Sample Code
GKRocket

Declared in
`GKSession.h`

connectToPeer:withTimeout:

Creates a connection to another iOS device.

```
- (void)connectToPeer:(NSString *)peerID withTimeout:(NSTimeInterval)timeout
```

Parameters

`peerID`

The string that identifies the peer to connect to.

`timeout`

The amount of time to wait before canceling the connection attempt.

Discussion

When your application is acting as a client, it calls this method to connect to an available peer it discovered. When your application calls this method, a request is transmitted to the remote peer, who chooses whether to accept or reject the connection request.

If the connection to the remote peer is successful, the delegate's `session:peer:didChangeState:` (page 263) method is called for each peer it successfully connected to. If the connection fails or your application cancels the connection attempt, the session calls the delegate's `session:connectionWithPeerFailed:withError:` (page 261) method.

Availability

Available in iOS 3.0 and later.

See Also

– `cancelConnectToPeer:` (page 158)

Related Sample Code
GKRocket

Declared in

`GKSession.h`

denyConnectionFromPeer:

Called by the delegate to reject a connection request received from a remote peer.

```
- (void)denyConnectionFromPeer:(NSString *)peerID
```

Parameters

`peerID`

The string identifying the peer that initiated the connection to the session.

Discussion

When your session acts as a server, client peers can discover your session and attempt to connect to it. When a client attempts to connect to the session, the delegate's `session:didReceiveConnectionRequestFromPeer:` (page 262) method is called to decide whether the peer should be connected. Your application calls this method to reject the request or `acceptConnectionFromPeer:error:` (page 157) to accept it.

Availability

Available in iOS 3.0 and later.

See Also

– `acceptConnectionFromPeer:error:` (page 157)

Related Sample Code

GKRocket

Declared in

`GKSession.h`

disconnectFromAllPeers

Disconnects the session from all connected peers.

```
- (void)disconnectFromAllPeers
```

Availability

Available in iOS 3.0 and later.

Related Sample Code

GKRocket

GKTank

Declared in

`GKSession.h`

disconnectPeerFromAllPeers:

Disconnects a connected peer from all peers connected to the session.

```
- (void)disconnectPeerFromAllPeers:(NSString *)peerID
```

Parameters

`peerID`

A string identifying the peer to disconnect.

Availability

Available in iOS 3.0 and later.

Declared in

`GKSession.h`

displayNameForPeer:

Returns a user-readable name for a peer.

```
- (NSString *)displayNameForPeer:(NSString *)peerID
```

Parameters

`peerID`

A string that uniquely identifies a peer.

Return Value

The name for the peer, or `nil` if `peerID` is not associated with a visible peer.

Discussion

The display name is used to populate your user interface with the names of other peers visible to the session.

Availability

Available in iOS 3.0 and later.

See Also

`@property displayName` (page 155)

Related Sample Code

GKRocket

GKTank

Declared in

`GKSession.h`

initWithSessionID:displayName:sessionMode:

Initializes and returns a newly allocated session.

```
- (id)initWithSessionID:(NSString *)sessionID displayName:(NSString *)name
sessionMode:(GKSessionMode)mode
```

Parameters

`sessionID`

A unique string that identifies your application. Your `sessionID` should be the short name of an approved Bonjour service type. If `nil`, the session uses the application's bundle identifier to create a `sessionID` string.

`name`

A string identifying the user to display to other peers. If `nil`, the session uses the device name.

`mode`

The mode the session should run in. See "Session Modes" (page 166) for possible values.

Return Value

An initialized session object, or `nil` if an error occurred.

Discussion

Only sessions running with the same `sessionID` are visible to your session.

Availability

Available in iOS 3.0 and later.

Related Sample Code
GKRocket

GKTank

Declared in

`GKSession.h`

peersWithConnectionState:

Returns a list of peers in the specified connection state.

```
- (NSArray *)peersWithConnectionState:(GKPeerConnectionState)state
```

Parameters

`state`

The connection state to search for. See "Connection States" (page 167) for possible values.

Return Value

An array of `NSString` objects with a `peerID` string for each peer visible to the session that is currently in the specified connection state. If there are no peers in the specified connection state, this method returns `nil`.

Availability
Available in iOS 3.0 and later.

Declared in
GKSession.h

sendData:toPeers:withDataMode:error:

Transmits a collection of bytes to a list of connected peers.

```
- (BOOL)sendData:(NSData *)data toPeers:(NSArray *)peers
withDataMode:(GKSendDataMode)mode error:(NSError **)error
```

Parameters
`data`

The bytes to be sent.

`peers`

An array of `NSString` objects identifying the peers that should receive the data.

`mode`

The mechanism used to send the data.

`error`

If the data could not be queued, an `NSError` object describing the error.

Return Value
`YES` if the data was successfully queued for transmission; `NO` if the session object was unable to queue the data.

Discussion
The session queues the data and transmits it in the order it was queued. Data transmitted unreliably may be received out of order by the other peers.

Availability
Available in iOS 3.0 and later.

See Also
`– sendDataToAllPeers:withDataMode:error:` (page 164)

Related Sample Code
GKTank

Declared in
GKSession.h

sendDataToAllPeers:withDataMode:error:

Transmits a collection of bytes to all connected peers.

```
- (BOOL)sendDataToAllPeers:(NSData *)data withDataMode:(GKSendDataMode)mode
error:(NSError **)error
```

Parameters

`data`

The bytes to be sent.

`mode`

The mechanism used to send the data.

`error`

If the data could not be queued, an `NSError` object describing the error.

Return Value

`YES` if the data was successfully queued for transmission; `NO` if the session object was unable to queue the data.

Discussion

The session queues the data and transmits it when the network is free.

Availability

Available in iOS 3.0 and later.

See Also

– `sendData:toPeers:withDataMode:error:` (page 163)

Declared in

`GKSession.h`

setDataReceiveHandler:withContext:

Sets the object that handles data received from other peers connected to the session.

```
- (void)setDataReceiveHandler:(id)handler withContext:(void *)context
```

Parameters

`handler`

The object you want the session to call when it receives data from other peers.

`context`

Arbitrary data to be passed to each invocation of the handler.

Discussion

The handler must implement a method with the following signature:

```
- (void) receiveData:(NSData *)data fromPeer:(NSString *)peer inSession:
(GKSession *)session context:(void *)context;
```

where *data* contains the bytes received from a remote peer, *peer* is a string that identifies the peer, *session* is the session that received the data, and *context* is the same context that was passed into the original call to `setDataReceiveHandler:withContext:` (page 164).

Important: Data received from other peers should be treated as *untrusted* data. Be sure to validate the data you receive from the session and write your code carefully to avoid security vulnerabilities. See the *Secure Coding Guide* for more information.

Availability

Available in iOS 3.0 and later.

Related Sample Code

GKRocket

GKTank

Declared in

`GKSession.h`

Constants

Data Transmission Modes

The mechanism used to transmit data to other peers.

```
typedef enum {
   GKSendDataReliable,
   GKSendDataUnreliable,
} GKSendDataMode;
```

Constants

`GKSendDataReliable`

The data is sent continuously until it is successfully received by the intended recipients or the connection times out.

Reliable transmissions are delivered in the order they were sent. Use this when you need to guarantee delivery.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKSendDataUnreliable`

The data is sent once and is not sent again if a transmission error occurred.

Data transmitted unreliably may be received out of order by recipients. Use this for small packets of data that must arrive quickly to be useful to the recipient.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

Session Modes

Modes that determine how a session interacts with other peers.

```
typedef enum {
   GKSessionModeServer,
   GKSessionModeClient,
   GKSessionModePeer,
} GKSessionMode;
```

Constants

`GKSessionModeServer`

A server advertises itself to local devices using its `sessionID` (page 156) property.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKSessionModeClient`

A client searches for servers advertising the same `sessionID` (page 156) property.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKSessionModePeer`

A peer advertises like a server and searches like a client.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

Connection States

The state of a peer known to the session.

```
typedef enum {
   GKPeerStateAvailable,
   GKPeerStateUnavailable,
   GKPeerStateConnected,
   GKPeerStateDisconnected,
   GKPeerStateConnecting
} GKPeerConnectionState;
```

Constants

`GKPeerStateAvailable`

A peer not connected to the session, but one that the session can connect to.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKPeerStateUnavailable`

A peer that is no longer interested in receiving connections.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKPeerStateConnected`

A peer connected to the session.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKPeerStateDisconnected`

A peer that disconnected from the session.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKPeerStateConnecting`

A peer attempting to connect to the session.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

Discussion

States are not mutually exclusive. For example, a peer can be available for other peers to discover while it is attempting to connect to another peer.

The Session Error Domain

The `GKSession` error domain.

```
NSString * const GKSessionErrorDomain;
```

Constants

`GKSessionErrorDomain`

An error occurred in `GKSession`.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

GKSession Errors

Error codes for the `GKSession` error domain.

```
typedef enum {
   GKSessionInvalidParameterError = 30500,
   GKSessionPeerNotFoundError = 30501,
   GKSessionDeclinedError = 30502,
   GKSessionTimedOutError = 30503,
   GKSessionCancelledError = 30504,
   GKSessionConnectionFailedError = 30505,
   GKSessionConnectionClosedError = 30506,
   GKSessionDataTooBigError = 30507,
   GKSessionNotConnectedError = 30508,
   GKSessionCannotEnableError = 30509,
   GKSessionInProgressError = 30510,
   GKSessionConnectivityError = 30201,
   GKSessionTransportError = 30202,
   GKSessionInternalError = 30203,
   GKSessionUnknownError = 30204,
   GKSessionSystemError = 30205
} GKSessionError;
```

Constants

`GKSessionInvalidParameterError`

A parameter had an unexpected value.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionPeerNotFoundError`

A peer with the specified `peerID` string could not be found.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionDeclinedError`

The peer your application tried to connect to refused the connection.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionTimedOutError`

The operation could not be completed in the specified timeout period.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionCancelledError`

A peer that invited the session to connect to them canceled the connection request.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionConnectionFailedError`

The attempt to establish a connection with another peer failed.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionConnectionClosedError`

The connection to another peer closed unexpectedly.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionDataTooBigError`

The data your application attempted to send was too large for the session to transmit in a single call.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionNotConnectedError`

Reserved for future use.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionCannotEnableError`

Bluetooth is not currently available.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionInProgressError`

The peer your application attempted to connect to has already requested a connection to your session.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionConnectivityError`

An error occurred in the `GKSession` object's connection code.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionTransportError`

An error occurred in the `GKSession` object's transport code.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionInternalError`

An serious error occurred inside `GKSession`.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionUnknownError`

Reserved for when the error does not fit in another category above.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

`GKSessionSystemError`

An error occurred outside of the `GKSession` object's control, such as memory allocation.

Available in iOS 3.0 and later.

Declared in `GKSessionError.h`.

GKTurnBasedEventHandler Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 5.0 and later.
Declared in	GKTurnBasedMatch.h
Companion guide	Game Center Programming Guide

Overview

The `GKTurnBasedEventHandler` class is used to respond to important messages related to turn-based matches. To use it, call the `sharedTurnBasedEventHandler` (page 172) class method to get the singleton instance and assign an object that implements the `GKTurnBasedEventHandlerDelegate` protocol to its `delegate` (page 172) property.

Important: Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication see *Game Center Programming Guide* .

Subclassing Notes

This class may not be subclassed.

Tasks

Retrieving the Shared Instance

`+ sharedTurnBasedEventHandler` (page 172)
Returns the shared instance of the event handler.

Getting and Setting the Delegate

`delegate` (page 172) *property*
The delegate for the event handler.

Properties

delegate

The delegate for the event handler.

```
@property(nonatomic, assign) NSObject<GKTurnBasedEventHandlerDelegate> *delegate
```

Discussion
If your game implements turn-based matches, it should set the delegate immediately after the local player is successfully authenticated. You want to set the delegate immediately because your game may have been launched specifically to handle a turn-based match event.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatch.h`

Class Methods

sharedTurnBasedEventHandler

Returns the shared instance of the event handler.

```
+ (GKTurnBasedEventHandler *)sharedTurnBasedEventHandler
```

Return Value
An event handler object.

Discussion
Your game never directly creates a `GKTurnBasedEventHandler` object. Instead, retrieve the shared instance using this class method.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatch.h`

GKTurnBasedMatch Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 5.0 and later.
Declared in	GKTurnBasedMatch.h
Companion guide	Game Center Programming Guide

Overview

The `GKTurnBasedMatch` class allows your game to implement turn-based matches between sets of players on Game Center. A turn-based match uses a store-and-forward approach to share data between the participants. When a player participating in the match performs actions that advance the state of the match, your game describes the new state of the match and decides which player acts next. The next player to act is notified by a push notification. Later, when the next player launches your game, you download the match data from Game Center and continue the match. Players take turns acting (based on whatever internal logic your game implements) until the match ends. One advantage of turn-based matches is that a player may participate in multiple matches simultaneously.

Important: Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication see *Game Center Programming Guide* .

Your game never directly creates `GKTurnBasedMatch` objects, because a match always represents an existing match that is stored by Game Center. Instead, match objects are created for your game by Game Kit. A match object may hold a newly created match (for a match that has not started yet) or a match that is already under way. Here are ways your game acquires match objects:

- To allow the player to see a standard user interface for turn-based matches, use the `GKTurnBasedMatchmakerViewController` class.

- To programmatically find a new match for the player to join, call the `findMatchForRequest:withCompletionHandler:` (page 182) class method.
- To programmatically load a collection of match objects for matches the local player is already participating in, call the `loadMatchesWithCompletionHandler:` (page 183) class method.
- To receive notifications when it is the local player's turn in a match, attach a delegate to the `GKTurnBasedEventHandler` singleton object.

A match object contains a few essential properties:

- The `participants` (page 181) property contains an array of `GKTurnBasedParticipant` objects. Each identifies a player in the match along with the current status of that player.
- The `currentParticipant` (page 179) property contains the `GKTurnBasedParticipant` object for the player must act to advance the state of the match.
- The `matchData` (page 179) property holds binary data that describes the current state of the match. The format of the data contained in this object is something you design when you create your game; Game Kit only manages storing and forwarding of this data.

When a player views a particular match, your game should take the match object and call its `loadMatchDataWithCompletionHandler:` (page 190) method to retrieve the match data. After the match data is downloaded from Game Center, your game parses the data and displays the game's user interface. A player may want to view the match even when it is another player's turn to act; your game should expect this and allow the player to view the state of the match without allowing them to act. When the current player views the match, your game should both show them the state of the match and provide a user interface to allow the player to take a turn. The current player is expected to perform actions that change the state of the match. These actions are based on the internal logic of your game's rules. The player continues to act until he or she takes an action that requires another person in the match to act in order to advance the match. For example, in a game like Chess, moving a piece on the board and passing control to the other participant happen simultaneously. A more complex game might allow the current player to take multiple actions before the match reaches a state in which control is ceded to another player.

If the player takes an action that is irrevocable (but control is not yet passed to another player), you update the match data by calling the `saveCurrentTurnWithMatchData:completionHandler:` (page 195) method.

Once the player takes an action that requires another player to act, your game should determine which player needs to act next. Then, it should encode the state of the match into an `NSData` object and call the match object's `endTurnWithNextParticipants:turnTimeout:matchData:completionHandler:` (page 189) method to set the next player to act. The next player is automatically informed that the match is waiting on his or her actions in order to continue the match.

As the match progresses, players may leave the match. For example, this situation might occur if your game determines that this player has been eliminated from the match. Or your game might offer a user interface to allow players to resign from matches. If the local player resigns from the match and is also the match's current player, your game must call the match object's `participantQuitInTurnWithOutcome:nextParticipant:matchData:completionHandler:` (page 190) method. As it does when the player takes a turn, your game passes in the state of the match and the next player to act. Your game also provides a **match outcome** — essentially, a numerical value that indicates the reason why that player left the match. If your game allows players to resign from matches when it is not that player's turn, your game calls the match object's `participantQuitOutOfTurnWithOutcome:withCompletionHandler:` (page 193) method to allow the player to leave the match.

Eventually, a match ends. When your game logic dictates that a match should end, your game calls the match object's `endMatchInTurnWithMatchData:completionHandler:` (page 187) method. Your game must ensure that all participants in the match have a valid match outcome before ending the match.

Even after a match ends, its data is still stored on Game Center; this allows players to view previous victories and defeats. A player may choose to delete the match using the standard user interface, but your game can also programmatically remove a match by calling the match object's `removeWithCompletionHandler:` (page 194) method. In all cases, the decision to delete a match should be made by a player; the `removeWithCompletionHandler:` (page 194) method exists to allow your game to present a custom user interface for viewing and working with matches.

Subclassing Notes

The `GKTurnBasedMatch` class may not be subclassed.

Tasks

Retrieving Existing Matches

`+ loadMatchesWithCompletionHandler:` (page 183)

Loads the turn-based matches involving the local player and creates a match object for each match.

`+ loadMatchWithID:withCompletionHandler:` (page 184)

Loads a specific match.

Creating a New Match

`+ findMatchForRequest:withCompletionHandler:` (page 182)
Programmatically searches for a new match to join.

`– acceptInviteWithCompletionHandler:` (page 185)
Programmatically accept an invitation to a turn-based match.

`– declineInviteWithCompletionHandler:` (page 186)
Programmatically decline an invitation to a turn-based match.

`– rematchWithCompletionHandler:` (page 193)
Create a new turn-based match with the same participants as an existing match.

Retrieving Information About the Match

`creationDate` (page 179) *property*
The date that the match was created. (read-only)

`currentParticipant` (page 179) *property*
The participant whose turn it is to act next. (read-only)

`matchID` (page 180) *property*
A string that uniquely identifies the match. (read-only)

`message` (page 181) *property*
A message displayed to all players in the match.

`participants` (page 181) *property*
Information about the players participating in the match. (read-only)

`status` (page 182) *property*
The current state of the match. (read-only)

`matchDataMaximumSize` (page 180) *property*
Returns the limit the Game Center servers place on the size of the match data. (read-only)

`matchData` (page 179) *property*
Game-specific data that reflects the details of the match. (read-only)

Retrieving the Match's Custom Data

`– loadMatchDataWithCompletionHandler:` (page 190)
Loads the game-specific data associated with a match.

Handling the Current Player's Turn

– `saveCurrentTurnWithMatchData:completionHandler:` (page 195)
Update the match data without advancing the game to another player.

– `endTurnWithNextParticipants:turnTimeout:matchData:completionHandler:` (page 189)
Updates the data stored on Game Center for the current match.

– `endTurnWithNextParticipant:matchData:completionHandler:` (page 188)
Updates the data stored on Game Center for the current match. (Deprecated. Use `endTurnWithNextParticipants:turnTimeout:matchData:completionHandler:` (page 189) instead.)

Leaving a Match

– `participantQuitInTurnWithOutcome:nextParticipants:turnTimeout:matchData:completionHandler:` (page 191)
Resigns the current player from the match without ending the match.

– `participantQuitOutOfTurnWithOutcome:withCompletionHandler:` (page 193) Deprecated in iOS 5.0
Resigns the player from the match when that player is not the current player. This action does not end the match

– `participantQuitInTurnWithOutcome:nextParticipant:matchData:completionHandler:` (page 190) Deprecated in iOS 6.0
Resigns the current player from the match without ending the match. (Deprecated. Use `participantQuitInTurnWithOutcome:nextParticipants:turnTimeout:matchData:completionHandler:` (page 191) instead.)

Ending a Match

– `endMatchInTurnWithMatchData:completionHandler:` (page 187)
Ends the match.

Deleting a Match From Game Center

– `removeWithCompletionHandler:` (page 194)
Programmatically removes a match from Game Center.

Properties

creationDate

The date that the match was created. (read-only)

```
@property(nonatomic, readonly, retain) NSDate *creationDate
```

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatch.h`

currentParticipant

The participant whose turn it is to act next. (read-only)

```
@property(nonatomic, readonly, retain) GKTurnBasedParticipant *currentParticipant
```

Discussion
The current participant is the only participant that is allowed to update the match data.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatch.h`

matchData

Game-specific data that reflects the details of the match. (read-only)

```
@property(nonatomic, readonly, retain) NSData *matchData
```

Discussion
Although Game Center knows who is participating in the match and who is expected to act next, it does not know anything about your game's internal logic. Your game provides the match data and all the programming logic required to interpret it. This data should include the current state of the game and provide any necessary details about what actions the current player is expected to take. It can also be helpful for your game to record information about recent moves made by other players. The game can then replay those moves visually for the player to show exactly how the match reached the state it is in now.

Your game never directly updates the match state associated with this property. Instead, when the data is updated to reflect the actions of the current player, your game serializes the updated state into memory and calls one of the match's instance methods that transmit the updated state to Game Center.

The value of this property is `nil` until after your game calls the `loadMatchDataWithCompletionHandler:` (page 190) method and the load task is complete. After this task completes, the `matchData` (page 179) property holds the data that the last player to act transmitted to Game Center.

Availability
Available in iOS 5.0 and later.

See Also
– `endTurnWithNextParticipant:matchData:completionHandler:` (page 188)
– `participantQuitInTurnWithOutcome:nextParticipant:matchData:completionHandler:` (page 190)

Declared in
`GKTurnBasedMatch.h`

matchDataMaximumSize

Returns the limit the Game Center servers place on the size of the match data. (read-only)

```
@property(nonatomic, readonly) NSUInteger matchDataMaximumSize
```

Discussion
Game Kit returns an error if your game sends updated data larger than this value.

Availability
Available in iOS 6.0 and later.

Declared in
`GKTurnBasedMatch.h`

matchID

A string that uniquely identifies the match. (read-only)

```
@property(nonatomic, readonly, retain) NSString *matchID
```

Discussion
This string is not intended to be displayed to players. Your game should use this string whenever it needs to identify a specific match. For example, if you want your game to store additional information on a device or in iCloud, it might store it in a database using the match ID as a key.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatch.h`

message

A message displayed to all players in the match.

```
@property(nonatomic, readwrite, copy) NSString *message
```

Discussion
The `message` property is displayed by the standard user interface; this allows your game to use the message to inform players of the current state of the match.

> **Important:** This property can be changed only by an instance of your game associated with the current player. If an instance of your game associated with another player in the match attempts to write to this property, an exception is thrown.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatch.h`

participants

Information about the players participating in the match. (read-only)

```
@property(nonatomic, readonly, retain) NSArray *participants
```

Discussion
The elements of this array are `GKTurnBasedParticipant` objects representing each participant in the match. Your game uses these objects to retrieve more information about the participants in the match. Your game also uses one of the objects in this array as a parameter whenever it calls a method that sets a different participant to act in the match.

The size of the array and the order in which the participants appear in the array are set when the match is first created, and never changes. When a match is first created, some participants may not hold actual players yet. Game Center searches for a player to fill that spot in the match only after your game sets that participant as the current player.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatch.h`

status

The current state of the match. (read-only)

```
@property(nonatomic, readonly) GKTurnBasedMatchStatus status
```

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatch.h`

Class Methods

findMatchForRequest:withCompletionHandler:

Programmatically searches for a new match to join.

```
+ (void)findMatchForRequest:(GKMatchRequest *)request withCompletionHandler:(void
(^)(GKTurnBasedMatch *match, NSError *error))completionHandler
```

Parameters

`request`

A match request that specifies the properties that the new match must fulfill.

`completionHandler`

A block to be called after the match is successfully created.

The block receives the following parameters:

match

A newly initialized match object that contains a list of players for the match. If an error occurred, this value is `nil`.

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

When this method is called, it creates a background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

This method may either create a new match or it may place the player into an existing match that needs a new player to advance the match further. Regardless of how the player is placed in the match, the local player is always the current participant in the returned match. Your game should immediately display the match in its user interface and allow the player to take a turn.

Availability

Available in iOS 5.0 and later.

Declared in

`GKTurnBasedMatch.h`

loadMatchesWithCompletionHandler:

Loads the turn-based matches involving the local player and creates a match object for each match.

```
+ (void)loadMatchesWithCompletionHandler:(void (^)(NSArray *matches, NSError
*error))completionHandler
```

Parameters

`completionHandler`

A block to be called after the matches are retrieved from the server.

The block receives the following parameters:

matches

An array of match objects for matches that the local player is playing in, or `nil` if there are no matches to load. If an error occurred, this value may be non-`nil`. In this case, the array holds whatever match data could be retrieved from Game Center before the error occurred.

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 5.0 and later.

Declared in

`GKTurnBasedMatch.h`

loadMatchWithID:withCompletionHandler:

Loads a specific match.

```
+ (void)loadMatchWithID:(NSString *)matchID withCompletionHandler:(void
(^)(GKTurnBasedMatch *match, NSError *error))completionHandler
```

Parameters

`matchID`

The identifier for the turn-based match.

`completionHandler`

A block to be called after the match is retrieved from the server.

The block receives the following parameters:

match

If the operation completed successfully, this parameter holds the match. If an error occurred, the value is `nil`.

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 6.0 and later.

Declared in

`GKTurnBasedMatch.h`

Instance Methods

acceptInviteWithCompletionHandler:

Programmatically accept an invitation to a turn-based match.

```
- (void)acceptInviteWithCompletionHandler:(void (^)(GKTurnBasedMatch *match, NSError
 *error))completionHandler
```

Parameters

`completionHandler`

A block to be called after the match is successfully created.

The block receives the following parameters:

match

A newly initialized match object that contains a list of players for the match. If an error occurred, this value is `nil`.

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

When this method is called, it creates a background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 6.0 and later.

Declared in

`GKTurnBasedMatch.h`

declineInviteWithCompletionHandler:

Programmatically decline an invitation to a turn-based match.

```
- (void)declineInviteWithCompletionHandler:(void (^)(NSError *error))completionHandler
```

Parameters

`completionHandler`

A block to be called after the match is successfully created.

The block receives the following parameter:

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

When this method is called, it creates a background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 6.0 and later.

Declared in

`GKTurnBasedMatch.h`

endMatchInTurnWithMatchData:completionHandler:

Ends the match.

```
- (void)endMatchInTurnWithMatchData:(NSData *)matchData completionHandler:(void
(^)(NSError *error))completionHandler
```

Parameters

`matchData`

A serialized blob of data reflecting the end state for the match.

`completionHandler`

A block to be called after the match is successfully ended.

The block receives the following parameters:

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

Calling this method ends the match for all players. This method may only be called by the current participant. Before your game calls this method, the `matchOutcome` property on each participant object stored in the `participants` (page 181) property must have been set to a value other than `GKTurnBasedMatchOutcomeNone` (page 205).

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability
Available in iOS 5.0 and later.

Declared in
GKTurnBasedMatch.h

endTurnWithNextParticipant:matchData:completionHandler:

Updates the data stored on Game Center for the current match. (Deprecated in iOS 6.0. Use `endTurnWithNextParticipants:turnTimeout:matchData:completionHandler:` *(page 189) instead.)*

```
- (void)endTurnWithNextParticipant:(GKTurnBasedParticipant *)nextParticipant
matchData:(NSData *)matchData completionHandler:(void (^)(NSError
*error))completionHandler
```

Parameters
`nextParticipant`

The next player in the match who needs to take an action. It must be one of the object's stored in the match's `participants` (page 181) property.

`matchData`

A serialized blob of data reflecting the game-specific state for the match.

`completionHandler`

A block to be called after the data is uploaded to Game Center.

The block receives the following parameters:

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion
When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability
Available in iOS 5.0 and later.

Deprecated in iOS 6.0.

Declared in
GKTurnBasedMatch.h

endTurnWithNextParticipants:turnTimeout:matchData:completionHandler:

Updates the data stored on Game Center for the current match.

```
- (void)endTurnWithNextParticipants:(NSArray *)nextParticipants
turnTimeout:(NSTimeInterval)timeout matchData:(NSData *)matchData
completionHandler:(void (^)(NSError *error))completionHandler
```

Parameters

`nextParticipants`

An array of participant objects reflecting the order in which the players should act next. Each object in the array must be one of the objects stored in the match's `participants` (page 181) property.

`timeout`

The length of time the next player has to complete their turn.

`matchData`

A serialized blob of data reflecting the game-specific state for the match.

`completionHandler`

A block to be called after the data is uploaded to Game Center.

The block receives the following parameters:

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion
If the next player to act does not take their turn in the specified interval, the next player in the array receives a notification to act. This process continues until a player takes a turn or the last player in the list is notified.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability
Available in iOS 6.0 and later.

Declared in
GKTurnBasedMatch.h

loadMatchDataWithCompletionHandler:

Loads the game-specific data associated with a match.

```
- (void)loadMatchDataWithCompletionHandler:(void (^)(NSData *matchData, NSError
*error))completionHandler
```

Parameters

`completionHandler`

A block to be called after the scores are retrieved from the server.

The block receives the following parameters:

matchData

The data stored on Game Center that reflects the current state of the match. If an error occurred, this value is `nil`.

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 5.0 and later.

Declared in

`GKTurnBasedMatch.h`

participantQuitInTurnWithOutcome:nextParticipant:matchData:completionHandler:

Resigns the current player from the match without ending the match. (Deprecated in iOS 6.0. Use `participantQuitInTurnWithOutcome:nextParticipants:turnTimeout:matchData:completionHandler:` *(page 191) instead.)*

```
- (void)participantQuitInTurnWithOutcome:(GKTurnBasedMatchOutcome)matchOutcome
nextParticipant:(GKTurnBasedParticipant *)nextPlayer matchData:(NSData *)matchData
 completionHandler:(void (^)(NSError *error))completionHandler
```

Parameters

`matchOutcome`

The end outcome of the current player in the match.

`nextParticipant`

The next player in the match who needs to take an action. It must be one of the object's stored in the match's `participants` (page 181) property.

`matchData`

A serialized blob of data reflecting the game-specific state for the match.

`completionHandler`

A block to be called after the data is uploaded to the server.

The block receives the following parameters:

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

Your game calls this method on an instance of your game that is processing the current player's turn, but that player has left the match. For example, the player may have willingly resigned from the match or that player may have been eliminated by the other players (based on your game's internal logic).

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 5.0 and later.

Deprecated in iOS 6.0.

Declared in

`GKTurnBasedMatch.h`

participantQuitInTurnWithOutcome:nextParticipants:turnTimeout:matchData:completionHandler:

Resigns the current player from the match without ending the match.

```
- (void)participantQuitInTurnWithOutcome:(GKTurnBasedMatchOutcome)matchOutcome
nextParticipants:(NSArray *)nextParticipants turnTimeout:(NSTimeInterval)timeout
matchData:(NSData *)matchData completionHandler:(void (^)(NSError
*error))completionHandler
```

Parameters

`matchOutcome`

The end outcome of the current player in the match.

`nextParticipants`

An array of participant objects reflecting the order in which the players should act next. Each object in the array must be one of the objects stored in the match's `participants` (page 181) property.

`timeout`

The length of time the next player has to complete their turn.

`matchData`

A serialized blob of data reflecting the game-specific state for the match.

`completionHandler`

A block to be called after the data is uploaded to the server.

The block receives the following parameters:

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

Your game calls this method on an instance of your game that is processing the current player's turn, but that player has left the match. For example, the player may have willingly resigned from the match or that player may have been eliminated by the other players (based on your game's internal logic).

If the next player to act does not take their turn in the specified interval, the next player in the array receives a notification to act. This process continues until a player takes a turn or the last player in the list is notified.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 6.0 and later.

Declared in
GKTurnBasedMatch.h

participantQuitOutOfTurnWithOutcome:withCompletionHandler:

Resigns the player from the match when that player is not the current player. This action does not end the match

```
- (void)participantQuitOutOfTurnWithOutcome:(GKTurnBasedMatchOutcome)matchOutcome
withCompletionHandler:(void (^)(NSError *error))completionHandler
```

Parameters

`matchOutcome`

The end outcome of the current player in the match.

`completionHandler`

A block to be called after the player's status is updated on Game Center.

The block receives the following parameters:

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion
If the local player decided they wanted to resign from the match but is not the current participant in the match, your game calls this method.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability
Available in iOS 5.0 and later.

Declared in
GKTurnBasedMatch.h

rematchWithCompletionHandler:

Create a new turn-based match with the same participants as an existing match.

```
- (void)rematchWithCompletionHandler:(void (^)(GKTurnBasedMatch *match, NSError
*error))completionHandler
```

Parameters

`completionHandler`

A block to be called after the match is successfully created.

The block receives the following parameters:

match

A newly initialized match object that contains a list of players for the match. If an error occurred, this value is `nil`.

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

When this method is called, it creates a background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 6.0 and later.

Declared in

`GKTurnBasedMatch.h`

removeWithCompletionHandler:

Programmatically removes a match from Game Center.

```
- (void)removeWithCompletionHandler:(void (^)(NSError *error))completionHandler
```

Parameters

`completionHandler`

A block to be called after the scores are retrieved from the server.

The block receives the following parameters:

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

Even after a player's participation in a match ends, the data associated with the match continues to be stored on Game Center. Storing the data on Game Center allows the player to continue to watch the match's progress, or even see the final state of the match when it ends. However, players may also want to delete matches that they have finished playing. If you choose not to use the standard matchmaker user interface, your game should offer the ability to delete a finished match from Game Center. When a player chooses to delete a match from Game Center, call this method. It is a programming error to call this method on a match that has the local player as an active participant.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

When the task completes, the match is no longer visible to the local player whose device made the call. Other players involved in the match still see the match.

Availability

Available in iOS 5.0 and later.

Declared in

`GKTurnBasedMatch.h`

saveCurrentTurnWithMatchData:completionHandler:

Update the match data without advancing the game to another player.

```
- (void)saveCurrentTurnWithMatchData:(NSData *)matchData completionHandler:(void
(^)(NSError *error))completionHandler
```

Parameters

`matchData`

A serialized blob of data reflecting the game-specific state for the match.

`completionHandler`

A block to be called after the data is uploaded to Game Center.

The block receives the following parameters:

error

If an error occurred, this error object describes the error. If the operation was completed successfully, the value is `nil`.

Discussion

This method updates the match data stored on Game Center. Call this method when the current player takes an action that advances the state of the match but does not end the player's turn. For example, if your game has a fog-of-war mechanic, you might call this method when the player revealed new information on the map.

When this method is called, it creates a new background task to handle the request. The method then returns control to your game. Later, when the task is complete, Game Kit calls your completion handler. Keep in mind that the completion handler may be called on a thread other than the one originally used to invoke the method. This means that the code in your block needs to be thread-safe.

Availability

Available in iOS 6.0 and later.

Declared in

`GKTurnBasedMatch.h`

Constants

GKTurnBasedMatchStatus

The different states that a match can enter.

```
enum {
   GKTurnBasedMatchStatusUnknown = 0,
   GKTurnBasedMatchStatusOpen = 1,
   GKTurnBasedMatchStatusEnded = 2,
   GKTurnBasedMatchStatusMatching = 3
};
typedef NSInteger GKTurnBasedMatchStatus;
```

Constants

`GKTurnBasedMatchStatusUnknown`

The match is in an unexpected state.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchStatusOpen`

The match is currently being played.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchStatusEnded`

The match has been completed.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchStatusMatching`

Game Center is still searching for other players to join the match.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

Turn Timeouts

Common values for turn timeouts.

```
extern NSTimeInterval GKTurnTimeoutDefault;
extern NSTimeInterval GKTurnTimeoutNone;
```

Constants

`GKTurnTimeoutDefault`

Indicates that the player has one week to take a turn.

Available in iOS 6.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnTimeoutNone`

Indicates that the player's turn never times out.

Available in iOS 6.0 and later.

Declared in `GKTurnBasedMatch.h`.

GKTurnBasedMatchmakerViewController Class Reference

Inherits from	UINavigationController : UIViewController : UIResponder : NSObject
Conforms to	NSCoding (UIViewController) UIAppearanceContainer (UIViewController) NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 5.0 and later.
Declared in	GKTurnBasedMatchmakerViewController.h
Companion guide	Game Center Programming Guide

Overview

The `GKTurnBasedMatchmakerViewController` class displays a user interface that allows players to manage the turn-based matches that they are participating in.

> **Important:** Your game must authenticate a local player before you can use any Game Center classes. If there is no authenticated player, your game receives a `GKErrorNotAuthenticated` (page 282) error. For more information on authentication see *Game Center Programming Guide*.

To show a turn-based matching user interface, first allocate and initialize a `GKMatchRequest` object that describes the desired match. Then, use the match request to initialize a new `GKTurnBasedMatchmakerViewController` object. Set the view controller's delegate, present the view controller, and wait for the delegate to be called. The view controller's delegate is notified when the matchmaking process is completed or canceled. Once the delegate is called, dismiss the view controller.

On iOS, you present and dismiss the view controller from another view controller in your game, using the methods provided by the `UIViewController` class. On OS X, you use the `GKDialogController` class to present and dismiss the view controller.

Tasks

Displaying a UI For Turn-Based Matches

– `initWithMatchRequest:` (page 200)
Initializes a new matchmaker view controller.

`turnBasedMatchmakerDelegate` (page 199) *property*
The view controller's delegate.

`showExistingMatches` (page 199) *property*
A Boolean value that determines whether the view controller shows existing matches.

Properties

showExistingMatches

A Boolean value that determines whether the view controller shows existing matches.

```
@property(nonatomic, readwrite, assign) BOOL showExistingMatches
```

Discussion
If the value of this property is `YES`, the view controller shows matches that are already in progress. If the value of this property is `NO`, the view controller only offers the ability to create new matches. The default value is `YES`.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatchmakerViewController.h`

turnBasedMatchmakerDelegate

The view controller's delegate.

```
@property(nonatomic, readwrite, assign) id<GKTurnBasedMatchmakerViewControllerDelegate>
turnBasedMatchmakerDelegate
```

Discussion
Your game must implement the delegate protocol on an object and assign that object to this property before presenting the view controller.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatchmakerViewController.h`

Instance Methods

initWithMatchRequest:

Initializes a new matchmaker view controller.

```
- (id)initWithMatchRequest:(GKMatchRequest *)request
```

Parameters
`request`
A match request with parameters for the match.

Return Value
An initialized matchmaker view controller. If an error occurred, this method returns `NIL`.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatchmakerViewController.h`

GKTurnBasedParticipant Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 5.0 and later.
Declared in	GKTurnBasedMatch.h
Companion guide	Game Center Programming Guide

Overview

A `GKTurnBasedParticipant` object stores information for a participant in a turn-based match. Your game never creates objects of this class directly; instead it retrieves an array of `GKTurnBasedParticipant` objects from a `GKTurnBasedMatch` object.

Most information stored in a `GKTurnBasedParticipant` object is read-only, and is provided by Game Kit to assist you in implementing your game logic. However, the `matchOutcome` (page 202) property is quite important; before your game ends a match, it must set the `matchOutcome` property for every participant in the match.

Subclassing Notes

This class may not be subclassed.

Tasks

Participant Information

`playerID` (page 203) *property*

The player identifier for this participant. (read-only)

`lastTurnDate` (page 202) *property*

The date and time that this participant last took a turn in the game. (read-only)

`timeoutDate` (page 204) *property*

The date and time that the participant's turn times out. (read-only)

`status` (page 203) *property*

The current status of the participant. (read-only)

Setting the Match Outcome

`matchOutcome` (page 202) *property*

The end-state of this participant in the match.

Properties

lastTurnDate

The date and time that this participant last took a turn in the game. (read-only)

```
@property(nonatomic, readonly, retain) NSDate *lastTurnDate
```

Discussion

The value of this property is invalid until the participant first takes a turn in the match.

Availability

Available in iOS 5.0 and later.

Declared in

`GKTurnBasedMatch.h`

matchOutcome

The end-state of this participant in the match.

```
@property(nonatomic, assign) GKTurnBasedMatchOutcome matchOutcome
```

Discussion
Initially, this property holds `GKTurnBasedMatchOutcomeNone` (page 205). Before your game can end a match, it must set the match outcome to a value that reflects the outcome of this participant when he or she left the match. See "Match Outcomes" (page 204). Optionally, it may also use an OR operation to include a custom match outcome for your specific game. Game Center does not use the custom value; it exists to allow your game to store its own information about the end of the match. The custom value must fit in the range provided by the `GKTurnBasedMatchOutcomeCustomRange` (page 206) constant.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatch.h`

playerID

The player identifier for this participant. (read-only)

```
@property(nonatomic, readonly, retain) NSString *playerID
```

Discussion
The value of this property may be `nil` if this slot in the match has not yet been filled by an actual player.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatch.h`

status

The current status of the participant. (read-only)

```
@property(nonatomic, readonly) GKTurnBasedParticipantStatus status
```

Discussion
This property is updated by Game Kit to reflect the

Availability
Available in iOS 5.0 and later.

Declared in
GKTurnBasedMatch.h

timeoutDate

The date and time that the participant's turn times out. (read-only)

```
@property(nonatomic, readonly, retain) NSDate *timeoutDate
```

Discussion
If a timeout was set when the turn state was advanced, this property holds when the player's turn expires. Otherwise, this property is `nil`.

Availability
Available in iOS 6.0 and later.

Declared in
GKTurnBasedMatch.h

Constants

Match Outcomes

The state the participant was in when they left the match.

```
enum {
   GKTurnBasedMatchOutcomeNone = 0,
   GKTurnBasedMatchOutcomeQuit = 1,
   GKTurnBasedMatchOutcomeWon = 2,
   GKTurnBasedMatchOutcomeLost = 3,
   GKTurnBasedMatchOutcomeTied = 4,
   GKTurnBasedMatchOutcomeTimeExpired = 5,
   GKTurnBasedMatchOutcomeFirst = 6,
   GKTurnBasedMatchOutcomeSecond = 7,
   GKTurnBasedMatchOutcomeThird = 8,
   GKTurnBasedMatchOutcomeFourth = 9,
   GKTurnBasedMatchOutcomeCustomRange = 0x00FF0000
};
typedef NSInteger GKTurnBasedMatchOutcome;
```

Constants

`GKTurnBasedMatchOutcomeNone`

The participant's outcome has not been set yet (typically because the match is still in progress).

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchOutcomeQuit`

The participant forfeited the match.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchOutcomeWon`

The participant won the match.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchOutcomeLost`

The participant lost the match.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchOutcomeTied`

The participant tied the match.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchOutcomeTimeExpired`

The participant was ejected from the match because he or she did not act in a timely fashion.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchOutcomeFirst`

The participant finished first.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchOutcomeSecond`

The participant finished second.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchOutcomeThird`

The participant finished third.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchOutcomeFourth`

The participant finished fourth.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedMatchOutcomeCustomRange`

A mask used to allow your game to provide its own custom outcome. Any custom value must fit inside the mask.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

GKTurnBasedParticipantStatus

The state the participant is in during the match.

```
enum {      GKTurnBasedParticipantStatusUnknown = 0,
   GKTurnBasedParticipantStatusInvited = 1,
   GKTurnBasedParticipantStatusDeclined = 2,
   GKTurnBasedParticipantStatusMatching = 3,
   GKTurnBasedParticipantStatusActive = 4,
   GKTurnBasedParticipantStatusDone = 5,
};
typedef NSInteger GKTurnBasedParticipantStatus;
```

Constants

`GKTurnBasedParticipantStatusUnknown`

The participant is in an unexpected state.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedParticipantStatusInvited`

The participant was invited to the match, but has not responded to the invitation.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedParticipantStatusDeclined`

The participant declined the invitation to join the match. When a participant declines an invitation to join a match, the match is automatically terminated.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedParticipantStatusMatching`

The participant represents an unfilled position in the match that Game Center promises to fill when needed. When you make this participant the next person to take a turn in the match, Game Center fills the position and updates the `status` (page 203) and `playerID` (page 203) properties.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedParticipantStatusActive`

The participant has joined the match and is an active player in it.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

`GKTurnBasedParticipantStatusDone`

The participant has exited the match. Your game sets the `matchOutcome` (page 202) property to state why the participant left the match.

Available in iOS 5.0 and later.

Declared in `GKTurnBasedMatch.h`.

GKVoiceChat Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKVoiceChat.h
Companion guide	Game Center Programming Guide

Overview

A `GKVoiceChat` object provides a voice channel that allows a set of players in a match to speak with each other.

In an iOS game, before you can use voice chat, your game must configure an audio session that allows for both play and recording (`kAudioSessionCategory_PlayAndRecord`). For more information on audio sessions, see *Audio Session Programming Guide*.

You use the `GKMatch` object to create a voice chat channel by calling its `voiceChatWithName:` (page 95) method, passing in a string that identifies the name of the channel you want to join. The name is never displayed by Game Kit, and you are free to name the channels however you like. You can create multiple channels for the same match. For example, if the match has multiple teams, you might create a separate channel for each team, and an additional channel that holds all of the players in the match.

To connect a player to a channel, call the voice chat object's `start` (page 213) method. After a channel is started, it receives voice data from other players already in the channel. To allow the player to speak in a channel, set the voice chat object's `active` (page 210) property to `YES`. A player can listen to multiple channels simultaneously but may only speak on one channel at any given time.

Your game uses the `volume` (page 211) property to set the volume level for an entire channel, and the `setMute:forPlayer:` (page 212) method to selectively mute other players in the channel.

If there is insufficient bandwidth over wi-fi to maintain a voice chat, Game Kit may disconnect individual players from the channel or disband the channel entirely.

Tasks

Determining Whether Voice Chat Is Available

`+ isVoIPAllowed` (page 212)
Returns whether voice chat may be used on the device.

Starting and Stopping Voice Chat

`– start` (page 213)
Starts communication with other participants in a channel.

`– stop` (page 214)
Ends communication with the channel.

Transmitting to Other Players

`active` (page 210) *property*
A Boolean value that states whether the channel is sampling the microphone.

Receiving Updates About Other Participants

`playerStateUpdateHandler` (page 211) *property*
A block that is called when a participant changes state.

Controlling Chat Volume

`– setMute:forPlayer:` (page 212)
Mutes a participant in the chat.

`volume` (page 211) *property*
The volume level for the voice channel.

Channel Properties

`name` (page 210) *property*

The name of the voice chat (read-only)

`playerIDs` (page 211) *property* Deprecated in iOS 6.0

An array of player identifiers for the players connected to the channel. (read-only)

Properties

active

A Boolean value that states whether the channel is sampling the microphone.

```
@property(nonatomic, assign, getter=isActive) BOOL active
```

Discussion
When `active` is `YES`, the voice chat samples the microphone and transmits the voice data to other players connected to the channel. Default value is `NO`.

Only one `GKVoiceChat` object is allowed to sample the microphone at any given time. When your game sets the `active` property to `YES` on a voice chat object, the previous voice chat object that owned the microphone (if there was one) sets its `active` property to `NO`.

Availability
Available in iOS 4.1 and later.

Declared in
`GKVoiceChat.h`

name

The name of the voice chat (read-only)

```
@property(nonatomic, readonly) NSString *name
```

Availability
Available in iOS 4.1 and later.

Declared in
`GKVoiceChat.h`

playerIDs

An array of player identifiers for the players connected to the channel. (read-only)

```
@property(nonatomic, readonly) NSArray *playerIDs
```

Availability
Available in iOS 5.0 and later.

Declared in
GKVoiceChat.h

playerStateUpdateHandler

A block that is called when a participant changes state.

```
@property(nonatomic, copy) void(^playerStateUpdateHandler)(NSString *playerID,
GKVoiceChatPlayerState state)
```

Discussion
Your game sets this property with a block to be called when the state of any participant in the chat changes (including the local player). The block receives the following parameters:

player

The player identifier for the player whose status changed.

state

The new state of the player. See `Player Voice Chat States` (page 214).

Availability
Available in iOS 4.1 and later.

Declared in
GKVoiceChat.h

volume

The volume level for the voice channel.

```
@property(nonatomic, assign) float volume
```

Discussion
All voice data received from other participants is mixed and then scaled by the `volume` property. The `volume` property has a range between `0.0` and `1.0`, inclusive. A volume level of `0.0` means the entire channel is muted; a value of `1.0` plays voice samples at full volume. The default value is `1.0`.

Availability
Available in iOS 4.1 and later.

Declared in
`GKVoiceChat.h`

Class Methods

isVoIPAllowed

Returns whether voice chat may be used on the device.

```
+ (BOOL)isVoIPAllowed
```

Return Value
`YES` if voice chat is available to the game.

Discussion
Some countries or phone carriers may restrict the availability of voice over IP services. Before creating a `GKVoiceChat` object, your game should first check to see whether voice over IP is permitted on the device.

Availability
Available in iOS 4.1 and later.

Declared in
`GKVoiceChat.h`

Instance Methods

setMute:forPlayer:

Mutes a participant in the chat.

```
- (void)setMute:(BOOL)isMuted forPlayer:(NSString *)player
```

Parameters

`isMuted`

Determines whether the player is to be muted or not.

`player`

The player identifier string for a player in the match.

Discussion

When a player is muted, the local player does not hear voice data transmitted by that player.

Availability

Available in iOS 4.1 and later.

Declared in

`GKVoiceChat.h`

start

Starts communication with other participants in a channel.

```
- (void)start
```

Discussion

When `start` is called, the voice chat object connects to the channel and notifies other players connected to the channel that the local player joined the chat. While connected, voice data from other players is played automatically. If the player is connected to the channel and the voice chat object's `active` (page 210) property is `YES`, then the microphone is sampled and the data sent to the channel.

A device only connects to a channel when the device has a microphone and is connected via wi-fi. However, your game may configure and start a voice chat channel when the device is not currently capable of using voice chat. If conditions change to allow voice chat—for example, the device connects to a wi-fi network—the `GKVoiceChat` object then automatically connects to the channel.

Availability

Available in iOS 4.1 and later.

See Also

`@property playerStateUpdateHandler` (page 211)

Declared in

`GKVoiceChat.h`

stop

Ends communication with the channel.

```
– (void)stop
```

Discussion
When `stop` is called, the voice chat object disconnects from the channel. You should call `stop` on a channel before the voice chat object is disposed.

Availability
Available in iOS 4.1 and later.

See Also
`@property playerStateUpdateHandler` (page 211)

Declared in
`GKVoiceChat.h`

Constants

Player Voice Chat States

The states returned to your game about other players in a voice chat.

```
enum {
   GKVoiceChatPlayerConnected,
   GKVoiceChatPlayerDisconnected,
   GKVoiceChatPlayerSpeaking,
   GKVoiceChatPlayerSilent,
   GKVoiceChatPlayerConnecting
};
typedef NSInteger GKVoiceChatPlayerState;
```

Constants

`GKVoiceChatPlayerConnected`

The player connected to the channel.

Available in iOS 4.1 and later.

Declared in `GKVoiceChat.h`.

`GKVoiceChatPlayerDisconnected`

The player left the channel.

Available in iOS 4.1 and later.

Declared in `GKVoiceChat.h`.

`GKVoiceChatPlayerSpeaking`

The player began speaking.

Available in iOS 4.1 and later.

Declared in `GKVoiceChat.h`.

`GKVoiceChatPlayerSilent`

The player stopped speaking.

Available in iOS 4.1 and later.

Declared in `GKVoiceChat.h`.

`GKVoiceChatPlayerConnecting`

The player is connecting to the channel, but is not yet connected.

Available in iOS 6.0 and later.

Declared in `GKVoiceChat.h`.

Availability

Available in iOS 4.1 and later.

Declared in

`GKVoiceChat.h`

GKVoiceChatService Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 3.0 and later.
Declared in	GKVoiceChatService.h
Companion guide	Game Center Programming Guide
Related sample code	GKRocket

Overview

The `GKVoiceChatService` class allows your application to connect two iOS devices into a voice chat.

Before you can use voice chat, your application must configure an audio session that allows for both play and recording (`kAudioSessionCategory_PlayAndRecord`). For more information on audio sessions, see *Audio Session Programming Guide*.

The voice chat service uses a `client` implemented by your application to find and connect to other participants. Each participant in the chat is identified by a unique **participant identifier** string. The client provides a participant identifier for the local user and translates other participant identifiers into connections to other users. The format and mechanism used to translate participant identifiers into network connections is defined by the client. See *Game Center Programming Guide* for a more complete discussion.

Your application can configure the voice chat service to control the volume level of both local and remote participants and to detect when someone is speaking.

To use the voice chat service, your application retrieves the default service and attaches a client to it, then either connects to another participant or waits for them to start a connection.

Tasks

Determining Whether Voice Chat Is Available

`+ isVoIPAllowed` (page 222)
Returns whether voice chat is allowed to be used on the device.

Getting the Shared Voice Chat Service

`+ defaultVoiceChatService` (page 221) Deprecated in iOS 6.0
Retrieves the singleton chat service.

Setting the Client

`client` (page 218) *property* Deprecated in iOS 6.0
An object that the voice chat service uses to communicate with remote participants.

Establishing a Voice Chat

`– startVoiceChatWithParticipantID:error:` (page 225)
Sends a request to another participant to join the voice chat.

Adjusting Audio Properties

`microphoneMuted` (page 220) *property*
A Boolean value that determines whether the user's microphone is muted.

`remoteParticipantVolume` (page 221) *property* Available in iOS 4.1 through iOS 5.1
A float that scales the volume of all remote participants.

Monitoring the Audio Level

`inputMeteringEnabled` (page 219) *property*
A Boolean value that indicates whether the microphone's sound level is being monitored.

`inputMeterLevel` (page 219) *property*

The volume, in decibels (db), being received by the microphone. (read-only)

`outputMeteringEnabled` (page 220) *property*

A Boolean value that indicates whether the voice level of remote participants is monitored.

`outputMeterLevel` (page 220) *property* Deprecated in iOS 6.0 Deprecated in iOS 6.0

The volume, in decibels (db), being received from all other participants. (read-only)

Ending a Voice Chat

– `stopVoiceChatWithParticipantID:` (page 226)

Ends a previously established voice chat.

Methods Called by the Client

– `acceptCallID:error:` (page 222)

Accepts a request from a remote user to establish a voice chat.

– `denyCallID:` (page 223)

Rejects a request to establish a voice chat.

– `receivedData:fromParticipantID:` (page 224)

Called by the client to deliver new data received from a remote participant.

– `receivedRealTimeData:fromParticipantID:` (page 224)

Called by the client to deliver voice data received from a remote participant..

Properties

client

An object that the voice chat service uses to communicate with remote participants.

```
@property(assign) id<GKVoiceChatClient> client
```

Discussion

The client's chief responsibility is to provide a network connection that the voice chat service can use to connect to another participant.

Availability
Available in iOS 3.0 and later.

Declared in
GKVoiceChatService.h

inputMeteringEnabled

A Boolean value that indicates whether the microphone's sound level is being monitored.

```
@property(nonatomic, getter=isInputMeteringEnabled) BOOL inputMeteringEnabled
```

Discussion
If YES, your application can read the inputMeterLevel (page 219) property to monitor the sound level of the microphone. If NO, the value of the inputMeterLevel (page 219) property is undefined. Default is NO. When your application doesn't need to monitor the microphone, it should set this property to NO to improve performance.

Availability
Available in iOS 3.0 and later.

See Also
@property inputMeterLevel (page 219)

Declared in
GKVoiceChatService.h

inputMeterLevel

The volume, in decibels (db), being received by the microphone. (read-only)

```
@property(readonly) float inputMeterLevel
```

Discussion
The value of this property is undefined if inputMeteringEnabled (page 219) is set to NO.

Availability
Available in iOS 3.0 and later.

See Also
@property inputMeteringEnabled (page 219)

Related Sample Code
GKRocket

Declared in
GKVoiceChatService.h

microphoneMuted

A Boolean value that determines whether the user's microphone is muted.

```
@property(nonatomic, getter=isMicrophoneMuted) BOOL microphoneMuted
```

Discussion
YES if the user's microphone is turned off; NO if the user's speech is being transmitted to remote participants. The default is NO.

Availability
Available in iOS 3.0 and later.

Declared in
GKVoiceChatService.h

outputMeteringEnabled

A Boolean value that indicates whether the voice level of remote participants is monitored.

```
@property(nonatomic, getter=isOutputMeteringEnabled) BOOL outputMeteringEnabled
```

Discussion
If YES, your application can read the `outputMeterLevel` (page 220) property to monitor sound level of remote participants. If NO, the value of the `outputMeterLevel` (page 220) property is undefined. Default is NO. When your application doesn't need to monitor remote participants, it should set this property to NO to improve performance.

Availability
Available in iOS 3.0 and later.

Declared in
GKVoiceChatService.h

outputMeterLevel

The volume, in decibels (db), being received from all other participants. (read-only)

```
@property(readonly) float outputMeterLevel
```

Discussion
The value of this property is undefined if `outputMeteringEnabled` (page 220) is set to `NO`.

The volume level is the aggregate volume of all remote participants, modified by the `remoteParticipantVolume` (page 221) property.

Availability
Available in iOS 3.0 and later.

Related Sample Code
GKRocket

Declared in
`GKVoiceChatService.h`

remoteParticipantVolume

A float that scales the volume of all remote participants.

```
@property(nonatomic) float remoteParticipantVolume
```

Discussion
The value should be between `0.0` (muted) and `1.0` (full volume). The default is `1.0`.

Availability
Available in iOS 3.0 and later.

Declared in
`GKVoiceChatService.h`

Class Methods

defaultVoiceChatService

Retrieves the singleton chat service.

```
+ (GKVoiceChatService *)defaultVoiceChatService
```

Return Value
The chat service.

Availability
Available in iOS 3.0 and later.

Related Sample Code
GKRocket

Declared in
`GKVoiceChatService.h`

isVoIPAllowed

Returns whether voice chat is allowed to be used on the device.

```
+ (BOOL)isVoIPAllowed
```

Return Value
`YES` if voice chat is available to the application.

Discussion
Some countries or phone carriers may restrict the availability of voice over IP services. Before retrieving the shared voice chat service object, your application should check to see whether voice chat is available.

Availability
Available in iOS 4.1 and later.

Declared in
`GKVoiceChatService.h`

Instance Methods

acceptCallID:error:

Accepts a request from a remote user to establish a voice chat.

```
- (BOOL)acceptCallID:(NSInteger)callID error:(NSError **)error
```

Parameters
`callID`

An integer that identifies the connection request.

`error`

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

`YES` if the connection was established; otherwise `NO`.

Discussion

When a remote user requests a voice chat, the voice chat service calls the client's `voiceChatService:didReceiveInvitationFromParticipantID:callID:` (page 275) method. The client calls this method to accept the request or `denyCallID:` (page 223) to reject it.

Availability

Available in iOS 3.0 and later.

Declared in

`GKVoiceChatService.h`

denyCallID:

Rejects a request to establish a voice chat.

```
- (void)denyCallID:(NSInteger)callID
```

Parameters

`callID`

An integer that identifies the connection request.

Discussion

When a remote user requests a voice chat, the voice chat service calls the client's `voiceChatService:didReceiveInvitationFromParticipantID:callID:` (page 275) method. The client calls this method to reject the request or `acceptCallID:error:` (page 222) to accept it.

Availability

Available in iOS 3.0 and later.

Related Sample Code
GKRocket

Declared in

`GKVoiceChatService.h`

receivedData:fromParticipantID:

Called by the client to deliver new data received from a remote participant.

```
- (void)receivedData:(NSData *)arbitraryData fromParticipantID:(NSString
*)participantID
```

Parameters

`arbitraryData`

The data received from a participant.

`participantID`

A string that uniquely identifies the participant who sent the data.

Discussion

The voice chat service uses a network connection provided by the client to exchange information between the participants. When the client receives information intended for the voice chat service, it should call this method to transfer it.

Availability

Available in iOS 3.0 and later.

See Also

`voiceChatService:sendData:toParticipantID:` (page 277)

Related Sample Code

GKRocket

Declared in

`GKVoiceChatService.h`

receivedRealTimeData:fromParticipantID:

Called by the client to deliver voice data received from a remote participant..

```
- (void)receivedRealTimeData:(NSData *)audio fromParticipantID:(NSString
*)participantID
```

Parameters

`audio`

The audio data that was received from the other participant.

`participantID`

A string that uniquely identifies the speaking participant.

Discussion

The voice chat service uses a network connection provided by the client to exchange information between the participants. When the client receives information intended for the voice chat service, it should call this method to transfer it.

Availability

Available in iOS 3.0 and later.

See Also

`voiceChatService:sendRealTimeData:toParticipantID:` (page 278)

Declared in

`GKVoiceChatService.h`

startVoiceChatWithParticipantID:error:

Sends a request to another participant to join the voice chat.

```
- (BOOL)startVoiceChatWithParticipantID:(NSString *)participantID error:(NSError **)error
```

Parameters

`participantID`

A string that uniquely identifies the participant to connect to.

`error`

If an error occurs, upon return contains an `NSError` object that describes the problem. Pass `NULL` if you do not want error information.

Return Value

`YES` if the connection was successfully created.

Discussion

The voice chat service calls the client's `voiceChatService:sendData:toParticipantID:` (page 277) method to send the connection request to the remote participant.

Availability

Available in iOS 3.0 and later.

Declared in

`GKVoiceChatService.h`

stopVoiceChatWithParticipantID:

Ends a previously established voice chat.

```
- (void)stopVoiceChatWithParticipantID:(NSString *)participantID
```

Parameters

`participantID`

A string that uniquely identifies the participant in the chat.

Discussion

When this method is called, the client's `voiceChatService:didStopWithParticipantID:error:` (page 277) method is called.

Availability

Available in iOS 3.0 and later.

Related Sample Code
GKRocket

Declared in
`GKVoiceChatService.h`

Constants

Voice Chat Service Error Domain

The `GKVoiceChatService` error domain.

```
NSString * const GKVoiceChatServiceErrorDomain;
```

Constants

`GKVoiceChatServiceErrorDomain`

An error occurred in `GKVoiceChatService`.

Available in iOS 3.0 and later.

Declared in `GKVoiceChatService.h`.

Voice Chat Service Errors

Error codes for the `GKVoiceChatService` error domain.

```
typedef enum {
   GKVoiceChatServiceInternalError = 32000,
   GKVoiceChatServiceNoRemotePacketsError = 32001,
   GKVoiceChatServiceUnableToConnectError = 32002,
   GKVoiceChatServiceRemoteParticipantHangupError = 32003,
   GKVoiceChatServiceInvalidCallIDError = 32004,
   GKVoiceChatServiceAudioUnavailableError = 32005,
   GKVoiceChatServiceUninitializedClientError = 32006,
   GKVoiceChatServiceClientMissingRequiredMethodsError = 32007,
   GKVoiceChatServiceRemoteParticipantBusyError = 32008,
   GKVoiceChatServiceRemoteParticipantCancelledError = 32009,
   GKVoiceChatServiceRemoteParticipantResponseInvalidError = 32010,
   GKVoiceChatServiceRemoteParticipantDeclinedInviteError = 32011,
   GKVoiceChatServiceMethodCurrentlyInvalidError = 32012,
   GKVoiceChatServiceNetworkConfigurationError = 32013,
   GKVoiceChatServiceUnsupportedRemoteVersionError = 32014,
   GKVoiceChatServiceOutOfMemoryError = 32015,
   GKVoiceChatServiceInvalidParameterError = 32016
} GKVoiceChatServiceError;
```

Constants

`GKVoiceChatServiceInternalError`

A serious error occurred inside the voice chat service.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceNoRemotePacketsError`

The voice chat service stopped receiving packets from the remote participant.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceUnableToConnectError`

The voice chat service was unable to establish a connection with another user.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceRemoteParticipantHangupError`

The remote participant in a voice chat stopped the chat.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceInvalidCallIDError`

The voice chat service didn't recognize the call identifier.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceAudioUnavailableError`

The audio hardware is unavailable to the voice chat service.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceUninitializedClientError`

The application did not set a client before calling voice chat service methods.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceClientMissingRequiredMethodsError`

The voice chat service did not find an expected method defined by the client.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceRemoteParticipantBusyError`

The remote participant is already connected to a voice chat.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceRemoteParticipantCancelledError`

A remote participant attempted to start a voice chat, then canceled.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceRemoteParticipantResponseInvalidError`

Invalid data was received from a remote participant.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceRemoteParticipantDeclinedInviteError`

A remote participant declined an invitation.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceMethodCurrentlyInvalidError`

A method on the voice chat service was called when it was not allowed to be called (for example, attempting to connect when the voice chat service was already connected).

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceNetworkConfigurationError`

The voice chat service had problems accessing the network.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceUnsupportedRemoteVersionError`

The other participant is running a different version of the voice chat service.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceOutOfMemoryError`

The voice chat service was unable to allocate memory required to operate.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

`GKVoiceChatServiceInvalidParameterError`

A parameter had an unrecognized value.

Available in iOS 3.0 and later.

Declared in `GKPublicConstants.h`.

Protocols

GKAchievementViewControllerDelegate Protocol Reference

Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKAchievementViewController.h
Companion guide	Game Center Programming Guide
Related sample code	GKAchievements GKTapper

Overview

An object implementing the `GKAchievementViewControllerDelegate` protocol is called when the user dismisses the achievements view controller. Typically, this protocol is implemented by the object in your game that originally displayed the achievements user interface.

Tasks

Responding to a Dismiss Action

– `achievementViewControllerDidFinish:` (page 231) *required method*

Called when the user dismisses the achievements user interface. (required)

Instance Methods

achievementViewControllerDidFinish:

Called when the user dismisses the achievements user interface. (required)

```
– (void)achievementViewControllerDidFinish:(GKAchievementViewController
*)viewController
```

Parameters

`viewController`

The achievement view controller whose interface was dismissed by the player.

Discussion

Your delegate should dismiss the view controller. If your game paused any gameplay or other activities, it can restart those services in this method.

Availability

Available in iOS 4.1 and later.

Declared in

`GKAchievementViewController.h`

GKChallengeEventHandlerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 6.0 and later.
Declared in	GKChallengeEventHandler.h
Companion guide	Game Center Programming Guide

Overview

You implement the `GKChallengeEventHandlerDelegate` delegate to control how challenges are displayed in your game.

By default, Game Kit briefly displays a banner over your game when any of the following events occur:

- The local player receives a challenge.
- The local player completes a challenge.
- A remote player completes a challenge issued by the local player.

Your event handler can override or extend this behavior:

- It can prevent a banner from being displayed.
- It can be notified when a player taps in a banner.
- It can handle the events directly.

Tasks

Detecting When a User Taps a Banner

– `localPlayerDidSelectChallenge:` (page 235)
Called when the local player selects a challenge banner displayed by Game Kit.

Responding When a New Challenge Is Received

– `localPlayerDidReceiveChallenge:` (page 235)
Called when the local player receives a new challenge.

– `shouldShowBannerForLocallyReceivedChallenge:` (page 237)
Called to determine whether a banner should be shown when the local player receives a challenge.

Responding to Challenges Completed By the Local Player

– `localPlayerDidCompleteChallenge:` (page 234)
Called when the local player completes a challenge.

– `shouldShowBannerForLocallyCompletedChallenge:` (page 236)
Called to determine whether a banner should be shown when the local player completes a challenge.

Responding to Challenges Issued by the Local Player

– `remotePlayerDidCompleteChallenge:` (page 236)
Called when a remote player completes a challenge issued by the local player.

– `shouldShowBannerForRemotelyCompletedChallenge:` (page 237)
Called to determine whether a banner should be shown when a remote player completes a challenge.

Instance Methods

localPlayerDidCompleteChallenge:

Called when the local player completes a challenge.

```
- (void)localPlayerDidCompleteChallenge:(GKChallenge *)challenge
```

Parameters

`challenge`

The completed challenge.

Availability

Available in iOS 6.0 and later.

Declared in

`GKChallengeEventHandler.h`

localPlayerDidReceiveChallenge:

Called when the local player receives a new challenge.

```
- (void)localPlayerDidReceiveChallenge:(GKChallenge *)challenge
```

Parameters

`challenge`

The received challenge.

Availability

Available in iOS 6.0 and later.

Declared in

`GKChallengeEventHandler.h`

localPlayerDidSelectChallenge:

Called when the local player selects a challenge banner displayed by Game Kit.

```
- (void)localPlayerDidSelectChallenge:(GKChallenge *)challenge
```

Parameters

`challenge`

The selected challenge.

Discussion

Availability

Available in iOS 6.0 and later.

Declared in
`GKChallengeEventHandler.h`

remotePlayerDidCompleteChallenge:

Called when a remote player completes a challenge issued by the local player.

```
- (void)remotePlayerDidCompleteChallenge:(GKChallenge *)challenge
```

Parameters
`challenge`
The completed challenge.

Availability
Available in iOS 6.0 and later.

Declared in
`GKChallengeEventHandler.h`

shouldShowBannerForLocallyCompletedChallenge:

Called to determine whether a banner should be shown when the local player completes a challenge.

```
- (BOOL)shouldShowBannerForLocallyCompletedChallenge:(GKChallenge *)challenge
```

Parameters
`challenge`
The completed challenge.

Return Value
Your delegate should return `YES` if it wants a banner to be displayed. Otherwise it should return `NO`.

Discussion
If you do not implement this method, a banner is always shown.

Availability
Available in iOS 6.0 and later.

Declared in
`GKChallengeEventHandler.h`

shouldShowBannerForLocallyReceivedChallenge:

Called to determine whether a banner should be shown when the local player receives a challenge.

```
– (BOOL)shouldShowBannerForLocallyReceivedChallenge:(GKChallenge *)challenge
```

Parameters

`challenge`

The received challenge.

Return Value

Your delegate should return `YES` if it wants a banner to be displayed. Otherwise it should return `NO`.

Discussion

If you do not implement this method, a banner is always shown.

Availability

Available in iOS 6.0 and later.

Declared in

`GKChallengeEventHandler.h`

shouldShowBannerForRemotelyCompletedChallenge:

Called to determine whether a banner should be shown when a remote player completes a challenge.

```
– (BOOL)shouldShowBannerForRemotelyCompletedChallenge:(GKChallenge *)challenge
```

Parameters

`challenge`

The completed challenge.

Return Value

Your delegate should return `YES` if it wants a banner to be displayed. Otherwise it should return `NO`.

Discussion

If you do not implement this method, a banner is always shown.

Availability

Available in iOS 6.0 and later.

Declared in

`GKChallengeEventHandler.h`

GKFriendRequestComposeViewControllerDelegate Protocol Reference

Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.2 and later.
Declared in	GKFriendRequestComposeViewController.h
Companion guide	Game Center Programming Guide

Overview

The `GKFriendRequestComposeViewControllerDelegate` protocol is implemented by delegates of the `GKFriendRequestComposeViewController` class. The delegate is called when the player dismisses the friend request.

Tasks

Responding to User Events

– `friendRequestComposeViewControllerDidFinish:` (page 238) *required method*

Called when the player dismisses the request. (required)

Instance Methods

friendRequestComposeViewControllerDidFinish:

Called when the player dismisses the request. (required)

```
– (void)friendRequestComposeViewControllerDidFinish:(GKFriendRequestComposeViewController
 *)viewController
```

Parameters

`viewController`

The friend request view controller that was dismissed by the player.

Discussion

Your delegate should dismiss the view controller. If your game paused any gameplay or other activities, it can restart those services in this method.

Availability

Available in iOS 4.2 and later.

Declared in

`GKFriendRequestComposeViewController.h`

GKGameCenterControllerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 6.0 and later.
Declared in	GKGameCenterViewController.h
Companion guide	Game Center Programming Guide

Overview

The GKGameCenterControllerDelegate protocol is implemented by delegates of the `GKGameCenterViewController` class. The delegate is called when the player dismisses the Game Center user interface.

Tasks

Handling User Actions

– `gameCenterViewControllerDidFinish:` (page 240) *required method*

Called when the player is done interacting with the view controller's content. (required)

Instance Methods

gameCenterViewControllerDidFinish:

Called when the player is done interacting with the view controller's content. (required)

```
– (void)gameCenterViewControllerDidFinish:(GKGameCenterViewController
*)gameCenterViewController
```

Parameters

`gameCenterViewController`

The view controller the player finished interacting with.

Discussion

Your delegate should dismiss the Game Center view controller. If your game paused any gameplay or other activities, it can restart those services in this method.

Availability

Available in iOS 6.0 and later.

Declared in

`GKGameCenterViewController.h`

GKLeaderboardViewControllerDelegate Protocol Reference

Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKLeaderboardViewController.h
Companion guide	Game Center Programming Guide
Related sample code	GKLeaderboards GKTapper

Overview

The `GKLeaderboardViewControllerDelegate` protocol is implemented by delegates of the `GKLeaderboardViewController` class. The delegate is called when the player dismisses the leaderboard.

Tasks

Handling User Actions

`– leaderboardViewControllerDidFinish:` (page 242) *required method*

Called when the leaderboard view is dismissed. (required)

Instance Methods

leaderboardViewControllerDidFinish:

Called when the leaderboard view is dismissed. (required)

```
– (void)leaderboardViewControllerDidFinish:(GKLeaderboardViewController
*)viewController
```

Parameters

`viewController`

The leaderboard view controller that was dismissed by the player.

Discussion

Your delegate should dismiss the view controller. If your game paused any gameplay or other activities, it can restart those services in this method.

Availability

Available in iOS 4.1 and later.

Declared in

`GKLeaderboardViewController.h`

GKMatchDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKMatch.h
Companion guide	Game Center Programming Guide

Overview

The `GKMatchDelegate` protocol is implemented to receive status updates and network data from players connected to a `GKMatch` object.

Tasks

Receiving Data from Other Players

– `match:didReceiveData:fromPlayer:` (page 246) *required method*

Called when data is received from a player. (required)

Receiving State Notifications About Other Players

– `match:player:didChangeState:` (page 247)

Called when a player connects to or disconnects from the match.

Handling Errors

– `match:connectionWithPlayerFailed:withError:` (page 245)
Called when the match fails to connect to a player.

– `match:didFailWithError:` (page 246)
Called when the match cannot connect to any other players.

Reinviting a Player

– `match:shouldReinvitePlayer:` (page 247)
Called when a player in a two-player match was disconnected.

Instance Methods

match:connectionWithPlayerFailed:withError:

Called when the match fails to connect to a player. (Available in iOS 4.1 through iOS 5.1.)

```
- (void)match:(GKMatch *)match connectionWithPlayerFailed:(NSString *)playerID
withError:(NSError *)error
```

Parameters

`match`
The match that received the error.

`player`
The identifier for the player whose connection failed.

`error`
The error that occurred.

Discussion

This method is called if the match was unable to send a transmission to another player in the match.

Availability

Available in iOS 4.1 through iOS 5.1.

Declared in

`GKMatch.h`

match:didFailWithError:

Called when the match cannot connect to any other players.

```
– (void)match:(GKMatch *)match didFailWithError:(NSError *)error
```

Parameters

`match`

The match that received the error.

`error`

The error that occurred.

Discussion

This method is called if the match cannot connect to any other players associated with the match. It usually means a serious networking error has occurred.

Availability

Available in iOS 4.1 and later.

Declared in

`GKMatch.h`

match:didReceiveData:fromPlayer:

Called when data is received from a player. (required)

```
– (void)match:(GKMatch *)match didReceiveData:(NSData *)data fromPlayer:(NSString
*)playerID
```

Parameters

`match`

The match that received the data.

`data`

The bytes sent by the player.

`player`

The string identifier for the player that sent the data.

Discussion

Your game defines its own format for data packets it transmits and receives over the network.

Important: Data received from other players should be treated as *untrusted* data. Be sure to validate the data you receive from the match and write your code carefully to avoid security vulnerabilities. See the Secure Coding Guide for more information.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatch.h`

match:player:didChangeState:

Called when a player connects to or disconnects from the match.

```
- (void)match:(GKMatch *)match player:(NSString *)playerID
didChangeState:(GKPlayerConnectionState)state
```

Parameters

`match`
The match that the player is connected to.

`player`
The identifier for the player whose state changed.

`state`
The state the player moved to.

Discussion
Your game implements this method to be notified when players connect to or disconnect from the match.

Availability
Available in iOS 4.1 and later.

Declared in
`GKMatch.h`

match:shouldReinvitePlayer:

Called when a player in a two-player match was disconnected.

```
- (BOOL)match:(GKMatch *)match shouldReinvitePlayer:(NSString *)playerID
```

Parameters

`match`

The match that lost the player.

`playerID`

The identifier for the player whose connection failed.

Return Value

Your game should return `YES` if it wants Game Kit to attempt to reconnect the player, `NO` if it wants to terminate the match.

Discussion

Occasionally, players may get disconnected from a match. If your game implements this method in the match delegate and the match only contains two players, Game Kit calls this method after a player gets disconnected. If your delegate allows Game Kit to reconnect to the other player, it reconnects the other player. Your `match:player:didChangeState:` (page 247) method is called when the other player is reconnected.

Availability

Available in iOS 5.0 and later.

Declared in

`GKMatch.h`

Constants

Player Connection States

The state of another player in the match.

```
enum {
   GKPlayerStateUnknown,
   GKPlayerStateConnected,
   GKPlayerStateDisconnected
};
typedef NSInteger GKPlayerConnectionState;
```

Constants

`GKPlayerStateUnknown`

The player is in an indeterminate state and cannot receive data.

Available in iOS 4.1 and later.

Declared in `GKMatch.h`.

`GKPlayerStateConnected`

The player is connected to the match and can receive data.

Available in iOS 4.1 and later.

Declared in `GKMatch.h`.

`GKPlayerStateDisconnected`

The player is disconnected from the match and cannot receive data.

Available in iOS 4.1 and later.

Declared in `GKMatch.h`.

Availability

Available in iOS 4.1 and later.

Declared in

`GKMatch.h`

GKMatchmakerViewControllerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 4.1 and later.
Declared in	GKMatchmakerViewController.h
Companion guide	Game Center Programming Guide

Overview

A class implements the `GKMatchmakerViewControllerDelegate` protocol to receive notifications from a `GKMatchmakerViewController` object. The delegate is called if a new match has been successfully created, if the user cancels matchmaking, and if an error occurs. In all three cases, the delegate should dismiss the view controller.

Tasks

Completing a Match Request

– `matchmakerViewController:didFindMatch:` (page 251)
Called when a peer-to-peer match is found.

– `matchmakerViewController:didFindPlayers:` (page 252)
Called when a hosted match is found.

Handling Cancellations

– `matchmakerViewControllerWasCancelled:` (page 253) *required method*
Called when the user cancels the matchmaking request (required)

Handling Errors

Hosted Matches

Instance Methods

matchmakerViewController:didFailWithError:

Called when the view controller encounters an unrecoverable error. (required)

```
– (void)matchmakerViewController:(GKMatchmakerViewController *)viewController
didFailWithError:(NSError *)error
```

Parameters

`viewController`

The view controller that received the error.

`error`

An error object that describes the error.

Availability

Available in iOS 4.1 and later.

Declared in

`GKMatchmakerViewController.h`

matchmakerViewController:didFindMatch:

Called when a peer-to-peer match is found.

```
– (void)matchmakerViewController:(GKMatchmakerViewController *)viewController
didFindMatch:(GKMatch *)match
```

Parameters

`viewController`

The view controller that performed the matchmaking.

`match`

A completed match.

Discussion

This method is called when the view controller's `hosted` property is `NO`. Although optional in the protocol, if your game attaches a delegate to the view controller for a peer-to-peer match, the view controller expects your game to provide an implementation of this method.

Availability

Available in iOS 4.1 and later.

See Also

– `matchmakerViewController:didFindPlayers:` (page 252)

Declared in

`GKMatchmakerViewController.h`

matchmakerViewController:didFindPlayers:

Called when a hosted match is found.

```
– (void)matchmakerViewController:(GKMatchmakerViewController *)viewController
didFindPlayers:(NSArray *)playerIDs
```

Parameters

`viewController`

The view controller that performed the matchmaking.

`players`

An array of identifier strings for the matched players.

Discussion

This method is called when the view controller's `hosted` property is `YES`. Although optional in the protocol, if your game attaches a delegate to the view controller for a hosted match, the view controller expects your game to provide an implementation of this method.

The view controller returns the list of players to your game by calling this method. Your game is responsible for connecting these players to your own server and then using that server to relay messages between the players.

Availability
Available in iOS 4.1 and later.

See Also
`– matchmakerViewController:didFindMatch:` (page 251)

Declared in
`GKMatchmakerViewController.h`

matchmakerViewController:didReceiveAcceptFromHostedPlayer:

Called when a player in a hosted match accepts the invitation.

```
– (void)matchmakerViewController:(GKMatchmakerViewController *)viewController
didReceiveAcceptFromHostedPlayer:(NSString *)playerID
```

Parameters
`viewController`

The view controller that accepted the invitation..

`playerID`

The identifier of the accepting player.

Discussion
After a player accepts an invitation, that player's device should connect to your server. Once the connection is established, your game should call the view controller's `setHostedPlayer:connected:` (page 115) method to update the player's connection status.

Availability
Available in iOS 5.0 and later.

Declared in
`GKMatchmakerViewController.h`

matchmakerViewControllerWasCancelled:

Called when the user cancels the matchmaking request (required)

```
– (void)matchmakerViewControllerWasCancelled:(GKMatchmakerViewController
*)viewController
```

Parameters

`viewController`

The view controller that received the cancellation.

Availability

Available in iOS 4.1 and later.

Declared in

`GKMatchmakerViewController.h`

GKPeerPickerControllerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 3.0 and later.
Declared in	GKPeerPickerController.h
Companion guide	Game Center Programming Guide
Related sample code	GKTank

Overview

The GKPeerPickerControllerDelegate (page 255) protocol is implemented on an object to customize the behavior of a `GKPeerPickerController` object. The delegate is called by the peer picker to create a session object and to respond as the session is configured by the controller.

Tasks

Creating a Session for the Peer Picker

– `peerPickerController:didSelectConnectionType:` (page 256)
Tells the delegate that the user selected a connection type.

– `peerPickerController:sessionForConnectionType:` (page 257)
Asks the delegate to return a session for the specified connection type.

Responding to Connection Messages

– `peerPickerController:didConnectPeer:toSession:` (page 256)
Tells the delegate that the controller connected a peer to the session.

Responding When the User Cancels the Connection Attempt

– `peerPickerControllerDidCancel:` (page 258)
Tells the delegate that the user canceled the connection attempt.

Instance Methods

peerPickerController:didConnectPeer:toSession:

Tells the delegate that the controller connected a peer to the session.

```
– (void)peerPickerController:(GKPeerPickerController *)picker
didConnectPeer:(NSString *)peerID toSession:(GKSession *)session
```

Parameters

`picker`
The controller that connected the peer.

`peerID`
The identification string for the peer that connected to the session.

`session`
The session that the peer is connected to.

Discussion

Once a peer is connected to the session, your application should take ownership of the session, dismiss the peer picker, and then use the session to communicate with the other peer.

> **Important:** Although optional in the protocol, Game Kit expects your application to implement this method.

Availability

Available in iOS 3.0 and later.

Declared in

`GKPeerPickerController.h`

peerPickerController:didSelectConnectionType:

Tells the delegate that the user selected a connection type.

```
- (void)peerPickerController:(GKPeerPickerController *)picker
didSelectConnectionType:(GKPeerPickerConnectionType)type
```

Parameters

`picker`

The controller for the peer picker dialog.

`type`

The type of network connection chosen by the user.

Discussion

If the peer picker is configured to allow users to choose between multiple connection types, this method is called when users select the connection type they want to use. Your delegate implements this method if you want to override the behavior for a particular connection type.

Important: In iOS 3.0, the peer picker can configure Bluetooth connections (`GKPeerPickerConnectionTypeNearby` (page 132)). If the user chooses an Internet connection (`GKPeerPickerConnectionTypeOnline` (page 132)), your delegate should dismiss the dialog and present its own user interface to configure the Internet connection:

```
- (void)peerPickerController:(GKPeerPickerController *)picker
didSelectConnectionType:(GKPeerPickerConnectionType)type {
    if(type == GKPeerPickerConnectionTypeOnline) {
        [picker dismiss];
        [picker autorelease];
// Display your own user interface here.
}
```

Availability

Available in iOS 3.0 and later.

Declared in

`GKPeerPickerController.h`

peerPickerController:sessionForConnectionType:

Asks the delegate to return a session for the specified connection type.

```
- (GKSession *)peerPickerController:(GKPeerPickerController *)picker
sessionForConnectionType:(GKPeerPickerConnectionType)type
```

Parameters

`picker`

The controller requesting the session.

`type`

The type of connection the controller wants to configure.

Discussion

Your delegate is responsible for providing a `GKSession` to use to find and connect to other devices. When the peer picker needs a session, it calls this method. Your application can either create a new session or return a previously created session to the peer picker. The session that your application returns to the peer picker must advertise itself as a peer (`GKSessionModePeer` (page 166)).

If your delegate does not implement this method and the user selected a network of type `GKPeerPickerConnectionTypeNearby` (page 132), the peer controller allocates a new session that advertises itself as a peer (`GKSessionModePeer`) with the default `sessionID` and `displayName` parameters.

Special Considerations

In iOS 3.0, your delegate receives requests only for networks of type `GKPeerPickerConnectionTypeNearby` (page 132).

Availability

Available in iOS 3.0 and later.

Declared in

`GKPeerPickerController.h`

peerPickerControllerDidCancel:

Tells the delegate that the user canceled the connection attempt.

```
- (void)peerPickerControllerDidCancel:(GKPeerPickerController *)picker
```

Parameters

`picker`

The controller for the peer picker dialog.

Discussion

After this method returns, the controller dismisses the picker interface.

Important: Although optional in the protocol, Game Kit expects your application to implement this method.

Availability

Available in iOS 3.0 and later.

Declared in

`GKPeerPickerController.h`

GKSessionDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 3.0 and later.
Declared in	GKSession.h
Companion guide	Game Center Programming Guide
Related sample code	GKRocket GKTank

Overview

An object implements the `GKSessionDelegate` protocol to control the behavior of a `GKSession` object. The delegate is called when other visible peers change their state relative to the session. It is also called to determine whether another peer is allowed to connect to the session.

Tasks

Observing Changes to Peers

– `session:peer:didChangeState:` (page 263) *required method*

Received by the delegate when a peer changes state. (required)

Connection Requests from Other Peers

– `session:didReceiveConnectionRequestFromPeer:` (page 262) *required method*

Received by the delegate when a remote peer wants to create a connection to the session. (required)

Connection Errors

– `session:connectionWithPeerFailed:withError:` (page 261) *required method*

Received by the delegate when an attempt to connect to another peer failed. (required)

– `session:didFailWithError:` (page 261) *required method*

Sent to the delegate when a serious error has occurred in the session. (required)

Instance Methods

session:connectionWithPeerFailed:withError:

Received by the delegate when an attempt to connect to another peer failed. (required)

```
– (void)session:(GKSession *)session connectionWithPeerFailed:(NSString *)peerID
withError:(NSError *)error
```

Parameters

`session`

The session that received the message.

`peerID`

A string that uniquely identifies the peer.

`error`

The error that occurred.

Discussion

The `error` parameter can be used to inform the user of why the connection failed.

> **Important:** If a `GKPeerPickerController` object is being used to configure the session, the controller handles this message automatically. Your delegate can ignore it if the peer picker dialog is in use.

Availability

Available in iOS 3.0 and later.

Declared in

`GKPublicProtocols.h`

session:didFailWithError:

Sent to the delegate when a serious error has occurred in the session. (required)

```
- (void)session:(GKSession *)session didFailWithError:(NSError *)error
```

Parameters

`session`

The session that failed.

`error`

The error that occurred.

Discussion

This method is called when a serious internal error occurred in the session. Your application should disconnect the session from other peers and release the session.

Availability

Available in iOS 3.0 and later.

Declared in

`GKPublicProtocols.h`

session:didReceiveConnectionRequestFromPeer:

Received by the delegate when a remote peer wants to create a connection to the session. (required)

```
- (void)session:(GKSession *)session didReceiveConnectionRequestFromPeer:(NSString
*)peerID
```

Parameters

`session`

The session that received the request.

`peerID`

A string that uniquely identifies the peer.

Discussion

The delegate should call the session's `acceptConnectionFromPeer:error:` (page 157) method if it wants to accept the connection or the `denyConnectionFromPeer:` (page 159) method if it wants to refuse the connection.

Important: If a `GKPeerPickerController` object is being used to configure the session, the controller handles this message automatically. Your delegate can ignore it if the peer picker dialog is in use. If your application is not using a `GKPeerPickerController` object to configure the session, your delegate must implement this method as described above.

Availability
Available in iOS 3.0 and later.

Declared in
`GKPublicProtocols.h`

session:peer:didChangeState:

Received by the delegate when a peer changes state. (required)

```
– (void)session:(GKSession *)session peer:(NSString *)peerID
didChangeState:(GKPeerConnectionState)state
```

Parameters

`session`
The session that received the update.

`peerID`
A string that identifies the peer.

`state`
The state the peer changed to.

Discussion
A session calls this method whenever a visible peer changes it state relative to itself. The action your delegate should take depends on what state the peer moved to.

- When a peer first becomes visible to the session, it appears with a state of `GKPeerStateAvailable`. Your application should show this peer in its user interface. If the peer changes its state to `GKPeerStateUnavailable`, it no longer accepts connection requests and your application should remove it from the user interface.
- The delegate should ignore `GKPeerStateConnecting` changes and implement the `session:didReceiveConnectionRequestFromPeer:` (page 262) method instead.
- When a peer is connected (`GKPeerStateConnected`), your application may send data to the peer and receive data from the peer. If a connection to a peer is lost or if the peer deliberately disconnects (`GKPeerStateDisconnected`), your application should stop sending messages to this peer.

Important: If a `GKPeerPickerController` object is being used to configure the session, the controller handles updates for the `GKPeerStateAvailable`, `GKPeerStateUnavailable`, and `GKPeerStateConnected` states. Your delegate can ignore state changes if the peer picker dialog is in use.

Availability
Available in iOS 3.0 and later.

Declared in
`GKPublicProtocols.h`

GKTurnBasedEventHandlerDelegate Protocol Reference

Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 5.0 and later.
Declared in	GKTurnBasedMatch.h
Companion guide	Game Center Programming Guide

Overview

The `GKTurnBasedEventHandlerDelegate` (page 265) protocol is implemented by an object to receive notifications events for turn-based matches.

Tasks

Receiving Turn-based Events

– `handleInviteFromGameCenter:` (page 266)
Sent to the delegate when the local player receives an invitation to join a new turn-based match.

– `handleTurnEventForMatch:` (page 267)
Sent to the delegate when it is the local player's turn to act in a turn-based match. (Deprecated. Implement `handleTurnEventForMatch:didBecomeActive:` (page 267) instead.)

– `handleTurnEventForMatch:didBecomeActive:` (page 267)
Sent to the delegate when it is the local player's turn to act in a turn-based match.

– `handleMatchEnded:` (page 266)
Sent to the delegate when a match the local player is participating in has ended.

Instance Methods

handleInviteFromGameCenter:

Sent to the delegate when the local player receives an invitation to join a new turn-based match.

```
– (void)handleInviteFromGameCenter:(NSArray *)playersToInvite
```

Parameters

`playersToInvite`

An array of player identifiers for the players to initially invite to the game.

Discussion

When your delegate receives this message, your game should create a new `GKMatchRequest` object and assign the `playersToInvite` parameter to the match request's `playersToInvite` property. Then, your game can either call the `GKTurnBasedMatch` class method `findMatchForRequest:withCompletionHandler:` (page 182) to find a match programmatically or it can use the request to instantiate a new `GKTurnBasedMatchmakerViewController` object to show a user interface to the player.

Availability

Available in iOS 5.0 and later.

Declared in

`GKTurnBasedMatch.h`

handleMatchEnded:

Sent to the delegate when a match the local player is participating in has ended.

```
– (void)handleMatchEnded:(GKTurnBasedMatch *)match
```

Parameters

`match`

The match that just ended.

Discussion

When your delegate receives this message, it should display the match's final results to the player and allow the player the option of saving or removing the match data from Game Center.

Availability

Available in iOS 5.0 and later.

Declared in
GKTurnBasedMatch.h

handleTurnEventForMatch:

Sent to the delegate when it is the local player's turn to act in a turn-based match. (Deprecated in iOS 6.0. Implement `handleTurnEventForMatch:didBecomeActive:` *(page 267) instead.)*

```
- (void)handleTurnEventForMatch:(GKTurnBasedMatch *)match
```

Parameters
`match`
A match object containing the current state of the match.

Discussion
When your delegate receives this message, the player has accepted a push notification for a match already in progress. Your game should end whatever task it was performing and switch to the match information provided by the match object.

Availability
Available in iOS 5.0 and later.

Deprecated in iOS 6.0.

Declared in
GKTurnBasedMatch.h

handleTurnEventForMatch:didBecomeActive:

Sent to the delegate when it is the local player's turn to act in a turn-based match.

```
- (void)handleTurnEventForMatch:(GKTurnBasedMatch *)match
didBecomeActive:(BOOL)didBecomeActive
```

Parameters
`match`
A match object containing the current state of the match.

`didBecomeActive`
`YES` if the game was launched or brought to the foreground to handle the event.

Discussion

When your delegate receives this message, the player has accepted a push notification for a match already in progress. Your game should end whatever task it was performing and switch to the match information provided by the match object.

Availability

Available in iOS 6.0 and later.

Declared in

`GKTurnBasedMatch.h`

GKTurnBasedMatchmakerViewControllerDelegate Protocol Reference

Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 5.0 and later.
Declared in	GKTurnBasedMatchmakerViewController.h
Companion guide	Game Center Programming Guide

Overview

Your game implements the `GKTurnBasedMatchmakerViewControllerDelegate` (page 269) protocol on an object to respond to events generated by a `GKTurnBasedMatchmakerViewController` object.

Tasks

Events

– `turnBasedMatchmakerViewController:didFindMatch:` (page 270) *required method*

Called when the player selected a match to view. (required)

– `turnBasedMatchmakerViewController:playerQuitForMatch:` (page 271) *required method*

Called when a player chooses to quit the match. (required)

– `turnBasedMatchmakerViewController:didFailWithError:` (page 270) *required method*

Called when an error occurs. (required)

– `turnBasedMatchmakerViewControllerWasCancelled:` (page 271) *required method*

Called when the player cancels matchmaking. (required)

Instance Methods

turnBasedMatchmakerViewController:didFailWithError:

Called when an error occurs. (required)

```
– (void)turnBasedMatchmakerViewController:(GKTurnBasedMatchmakerViewController
*)viewController didFailWithError:(NSError *)error
```

Parameters

`viewController`

The view controller that received an error.

`error`

An error object that describes the error.

Discussion

Your game should dismiss the view controller.

Availability

Available in iOS 5.0 and later.

Declared in

`GKTurnBasedMatchmakerViewController.h`

turnBasedMatchmakerViewController:didFindMatch:

Called when the player selected a match to view. (required)

```
– (void)turnBasedMatchmakerViewController:(GKTurnBasedMatchmakerViewController
*)viewController didFindMatch:(GKTurnBasedMatch *)match
```

Parameters

`viewController`

The view controller that found a match.

`match`

The match that the player selected.

Discussion

Your game should dismiss the view controller and use the match object to show the current state of the match to the player.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatchmakerViewController.h`

turnBasedMatchmakerViewController:playerQuitForMatch:

Called when a player chooses to quit the match. (required)

```
- (void)turnBasedMatchmakerViewController:(GKTurnBasedMatchmakerViewController
*)viewController playerQuitForMatch:(GKTurnBasedMatch *)match
```

Parameters
`viewController`
The view controller that the player interacted with.

`match`
The match the player has chosen to quit.

Discussion
When this method is called, the player is the current participant in the match, but that player has chosen to resign the match instead of taking a turn. Your game should dismiss the view controller, set an outcome for the player, and then call the match's
`participantQuitInTurnWithOutcome:nextParticipant:matchData:completionHandler:` (page 190) method.

Availability
Available in iOS 5.0 and later.

Declared in
`GKTurnBasedMatchmakerViewController.h`

turnBasedMatchmakerViewControllerWasCancelled:

Called when the player cancels matchmaking. (required)

```
-
(void)turnBasedMatchmakerViewControllerWasCancelled:(GKTurnBasedMatchmakerViewController
 *)viewController
```

Parameters
`viewController`
The view controller that the player canceled.

Discussion

Your game should dismiss the view controller.

Availability

Available in iOS 5.0 and later.

Declared in

`GKTurnBasedMatchmakerViewController.h`

GKVoiceChatClient Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/GameKit.framework
Availability	Available in iOS 3.0 and later.
Declared in	GKVoiceChatService.h
Companion guide	Game Center Programming Guide

Overview

The `GKVoiceChatClient` protocol is implemented to control the behavior of the `GKVoiceChatService` object. The voice chat client has a number of responsibilities:

- Provides a network connection that the voice chat service uses to send and receive configuration data with other participants. If this network connection is shared with other application data, the client must also disambiguate between chat configuration data and application data.
- Provides a participant ID that identifies the user to remote participants in the chat.
- Defines how a remote user's participant ID translates into a network connection to that user.
- Accepts or rejects requests from remote participants to join the voice chat.

Tasks

Getting Information About the Participant

– `participantID` (page 274) *required method*

Returns a string that uniquely identifies the local user. (required)

Sending Data to Other Participants

– `voiceChatService:sendData:toParticipantID:` (page 277) *required method*

A request for the client to send data to a participant. (required)

– `voiceChatService:sendRealTimeData:toParticipantID:` (page 278)

Asks the client to send data to a participant that must get there quickly.

Accepting Invitations from Remote Participants

– `voiceChatService:didReceiveInvitationFromParticipantID:callID:` (page 275)

Asks the client to accept or reject an invitation from a remote participant.

Responding to Changes in Other Participants

– `voiceChatService:didStartWithParticipantID:` (page 276)

Received by the client when a voice chat with another participant is established.

– `voiceChatService:didNotStartWithParticipantID:error:` (page 275)

Received by the client when an attempt to establish a voice chat with another participant failed.

– `voiceChatService:didStopWithParticipantID:error:` (page 277)

Received by the client when a previously established voice chat has ended.

Instance Methods

participantID

Returns a string that uniquely identifies the local user. (required)

```
- (NSString *)participantID
```

Return Value

A string that can be used by other participants to connect to the local user.

Discussion

The client decides the format and meaning of the participant identifier. For more information, see the *Game Center Programming Guide*.

Availability
Available in iOS 3.0 and later.

Declared in
`GKPublicProtocols.h`

voiceChatService:didNotStartWithParticipantID:error:

Received by the client when an attempt to establish a voice chat with another participant failed.

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
didNotStartWithParticipantID:(NSString *)participantID error:(NSError *)error
```

Parameters
`voiceChatService`
The voice chat service that was establishing the connection.

`participantID`
A string that uniquely identifies the other user.

`error`
The error that prevented the voice chat from being established.

Discussion
Your application can implement this method to notify the user that an error occurred when establishing a connection.

Availability
Available in iOS 3.0 and later.

Declared in
`GKPublicProtocols.h`

voiceChatService:didReceiveInvitationFromParticipantID:callID:

Asks the client to accept or reject an invitation from a remote participant.

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
didReceiveInvitationFromParticipantID:(NSString *)participantID
callID:(NSInteger)callID
```

Parameters
`voiceChatService`
The service that received the request.

`participantID`

A string that uniquely identifies the other user.

`callID`

An integer that uniquely identifies the request.

Discussion

If this method is not implemented by the client, the voice chat service automatically accept requests from other participants.

This method should call the service's `acceptCallID:error:` (page 222) method if it wants to accept the request or the `denyCallID:` (page 223) to reject it.

Availability

Available in iOS 3.0 and later.

Declared in

`GKPublicProtocols.h`

voiceChatService:didStartWithParticipantID:

Received by the client when a voice chat with another participant is established.

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
didStartWithParticipantID:(NSString *)participantID
```

Parameters

`voiceChatService`

The voice chat service that initiated the connection.

`participantID`

A string that uniquely identifies the other user.

Discussion

Your client can use this method to update the user interface to show that a connection has been established.

Availability

Available in iOS 3.0 and later.

Declared in

`GKPublicProtocols.h`

voiceChatService:didStopWithParticipantID:error:

Received by the client when a previously established voice chat has ended.

```
– (void)voiceChatService:(GKVoiceChatService *)voiceChatService
didStopWithParticipantID:(NSString *)participantID error:(NSError *)error
```

Parameters

`voiceChatService`

The voice chat that maintained the connection.

`participantID`

A string that uniquely identifies the user who disconnected.

`error`

The error that caused the chat to end.

Discussion

Your application can implement this method to notify the user that an established voice connection has ended. This may occur when another participant ends the chat or if the network connection was lost.

Availability

Available in iOS 3.0 and later.

Declared in

`GKPublicProtocols.h`

voiceChatService:sendData:toParticipantID:

A request for the client to send data to a participant. (required)

```
– (void)voiceChatService:(GKVoiceChatService *)voiceChatService sendData:(NSData
*)data toParticipantID:(NSString *)participantID
```

Parameters

`voiceChatService`

The service that requested the transmission.

`data`

The data to send.

`participantID`

A string that uniquely identifies the participant to send the data to.

Discussion

An implementation of this method must reliably transmit the data to the participant identified by `participantID`. When the client on the other end receives the data, it should forward it to the voice chat service by calling the service's `receivedData:fromParticipantID:` (page 224) method.

Availability

Available in iOS 3.0 and later.

Declared in

`GKPublicProtocols.h`

voiceChatService:sendRealTimeData:toParticipantID:

Asks the client to send data to a participant that must get there quickly.

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
sendRealTimeData:(NSData *)data toParticipantID:(NSString *)participantID
```

Parameters

`voiceChatService`

The service that requested the transmission.

`data`

The data to send.

`participantID`

A string that uniquely identifies the participant to send the data to.

Discussion

An implementation of this method maps the `participantID` string to a known participant and transmits the data to them. Data transmitted by this method can be sent unreliably. When the client on the other end receives this data, it should forward it to the voice chat service by calling the service's `receivedRealTimeData:fromParticipantID:` (page 224) method.

Availability

Available in iOS 3.0 and later.

Declared in

`GKPublicProtocols.h`

Constants

Game Kit Constants Reference

Framework	GameKit/GKError.h
Declared in	GKError.h
Companion guide	Game Center Programming Guide

Overview

This document describes the constants defined in the Game Kit framework that are not described in a document for an individual class.

Constants

GKErrorDomain

The Game Kit framework error domain.

```
NSString * const GKErrorDomain;
```

Constants

`GKErrorDomain`

An error occurred in Game Kit.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

Discussion

`GKSession` and `GKVoiceChatService` define their own error domains.

Game Kit Errors

Error codes for the Game Kit error domain.

```
enum {
   GKErrorUnknown = 1,
   GKErrorCancelled = 2,
   GKErrorCommunicationsFailure = 3,
   GKErrorUserDenied = 4,
   GKErrorInvalidCredentials = 5,
   GKErrorNotAuthenticated = 6,
   GKErrorAuthenticationInProgress = 7,
   GKErrorInvalidPlayer = 8,
   GKErrorScoreNotSet = 9,
   GKErrorParentalControlsBlocked = 10,
   GKErrorPlayerStatusExceedsMaximumLength = 11,
   GKErrorPlayerStatusInvalid = 12,
   GKErrorMatchRequestInvalid = 13,
   GKErrorUnderage = 14,
   GKErrorGameUnrecognized = 15,
   GKErrorNotSupported = 16,
   GKErrorInvalidParameter = 17,
   GKErrorUnexpectedConnection = 18,
   GKErrorChallengeInvalid = 19,
   GKErrorTurnBasedMatchDataTooLarge = 20,
   GKErrorTurnBasedTooManySessions = 21,
   GKErrorTurnBasedInvalidParticipant = 22,
   GKErrorTurnBasedInvalidTurn = 23,
   GKErrorTurnBasedInvalidState = 24,
};
typedef NSInteger GKErrorCode;
```

Constants

`GKErrorUnknown`

An unexpected error occurred.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorCancelled`

The requested operation was canceled.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorCommunicationsFailure`

An error occurred when communicating with Game Center.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorUserDenied`

The operation was denied by the user.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorInvalidCredentials`

The operation failed because the player's user name or password or both are incorrect.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorNotAuthenticated`

The local player has not been authenticated.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorAuthenticationInProgress`

The local player is currently authenticating.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorInvalidPlayer`

A player object or identifier is invalid.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorScoreNotSet`

A score value was not set before attempting to post the score.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorParentalControlsBlocked`

The feature has been blocked by the user.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorPlayerStatusExceedsMaximumLength`

The player's status message is too long.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorPlayerStatusInvalid`

The player's status message is invalid.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorMatchRequestInvalid`

The match request's properties are impossible to fulfill. For example, the minimum number of players cannot be larger than the maximum number of players.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorUnderage`

The feature is disabled because the local player is underage.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorGameUnrecognized`

Game Center does not recognize the application that made the request. Make sure the bundle identifier is set properly for the application.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorNotSupported`

The device does not support Game Center.

Available in iOS 4.1 and later.

Declared in `GKError.h`.

`GKErrorInvalidParameter`

One or more of the parameters was incorrect.

For example, this error code may be returned if your application attempts to post a score and provides a category string that does not match a category you configured for your leaderboards on iTunes Connect.

Available in iOS 4.2 and later.

Declared in `GKError.h`.

`GKErrorUnexpectedConnection`

An unexpected player has connected to a match.

Available in iOS 5.0 and later.

Declared in `GKError.h`.

`GKErrorChallengeInvalid`

The challenge was invalid.

Available in iOS 6.0 and later.

Declared in `GKError.h`.

`GKErrorTurnBasedMatchDataTooLarge`

Your game submitted data that exceeded the maximum size that Game Center permits for a turn-based game.

Available in iOS 6.0 and later.

Declared in `GKError.h`.

`GKErrorTurnBasedTooManySessions`

The requested operation could not be completed because it would exceed the maximum number of sessions.

Available in iOS 6.0 and later.

Declared in `GKError.h`.

`GKErrorTurnBasedInvalidParticipant`

One of the participant objects you provided was invalid.

Available in iOS 6.0 and later.

Declared in `GKError.h`.

`GKErrorTurnBasedInvalidTurn`

The requested operation could not be completed because the specified participant does not have the required turn state."

Available in iOS 6.0 and later.

Declared in `GKError.h`.

`GKErrorTurnBasedInvalidState`

The requested operation could not be completed because the session is in an invalid state.

Available in iOS 6.0 and later.

Declared in `GKError.h`.

Document Revision History

This table describes the changes to *Game Kit Framework Reference*.

Date	Notes
2012-09-19	Includes new classes added in iOS 6.
2012-07-23	Added Game Kit Framework Reference to OS X library.
2011-10-12	Updated for iOS 5.
2010-11-15	Added the GKFriendRequestComposeViewController class.
2010-08-26	Updated to current seeded API.
2009-05-26	Updated to add new iOS 4.0 classes. Minor edits.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010