GameSpy Technology C SDK Help

Table of Contents

1 GameSpy SDK Help

Modules

Name	Description
ATLAS (see page 13)	
Auth Service (see page 45)	
CD Key (see page 52)	
HTTP (see page 58)	
Nat Negotiation (see page 80)	
Presence and Messaging (see page 87)	
Query and Reporting (see page 161)	
Sake (see page 176)	
Server Browsing (see page 208)	
Transport (see page 240)	

Available Services Check

ATLAS

API Documentation

Module

ATLAS (see page 13)

Functions

Functions

	Name	Description
	scCreateMatchlessSession (see page 15)	This is a variation of scCreateSession (see page 16) that creates a "matchless" session. A matchless session isn't sanity-checked against other session data for arbitration of any statistics. It essentially turns off detection for cheating. As data is sent in, it will be immediately applied to the stats instead of being applied when the match has finished. A "matchless" game may be more commonly known as non-arbitrated or unranked.
	scCreateReport (see page 15)	Creates a new ATLAS (see page 13) report for the game session.
	scCreateSession (see page 16)	Requests that the ATLAS (see page 13) service create a new game session and keep track of the session that is about to start.
	scDestroyReport (see page 17)	Used to clean up and free the report object after it has been submitted.

	scGetConnectionId (see page 17)	Used to obtain the unique connection ID for the current game session.
	scGetSessionId (see page 18)	Gets the unique identifier (GUID) for the current game session.
	scInitialize (see page 18)	Initializes the ATLAS (see page 13) SDK, providing a valid ATLAS (see page 13) SDK object.
	scReportAddByteValue (see page 19)	Adds a byte value to the report for a specific key.
	scReportAddFloatValue (see page 19)	Adds a float value to the report for a specific key.
	scReportAddIntValue (see page 20)	Adds an integer value to the report for a specific key.
	scReportAddShortValue (see page 20)	Adds a short value to the report for a specific key.
	scReportAddStringValue (see page 21)	Adds a string value to the report for a specific key.
	scReportBeginGlobalData (see page 22)	Tells the ATLAS (see page 13) SDK to start writing global data to the report.
	scReportBeginNewPlayer (see page 22)	Add a new player to the report.
	scReportBeginNewTeam (see page 23)	Adds a new team to the report.
	scReportBeginPlayerData (see page 23)	Tells the ATLAS (see page 13) SDK to start writing player data to the report.
	scReportBeginTeamData (see page 24)	Tells the ATLAS (see page 13) SDK to start writing player data to the report.
	scReportEnd (see page 24)	Denotes the end of a report for the report specified.
	scReportSetAsMatchless (see page 25)	Called after creating the report to set it as a matchless report; this is needed if the report is being submitted to a "matchless" game session. A "matchless" game may be more commonly known as non-arbitrated or unranked.
	scReportSetPlayerData (see page 25)	Sets initial player data in the report specified.
	scReportSetTeamData (see page 26)	Sets the initial team data in the report specified.
	scSetReportIntention (see page 26)	Called to tell ATLAS (see page 13) the type of report that the player or host will send.
	scSetSessionId (see page 27)	Used to set the session ID for the current game session.
	scShutdown (see page 28)	Closes down the specified ATLAS (see page 13) SDK object and frees memory.
	scSubmitReport (see page 28)	Initiates the submission of a report.
	scThink (see page 29)	Called to complete pending operations for functions with callbacks.
	scReportAddInt64Value (see page 29)	Adds a 64-bit integer value to the report for a specific key.
	scAddQueryParameterToList (see page 30)	Adds a parameter name and value to the parameter list.
	scCheckBanList (see page 31)	Checks if a given gsid account is whitelisted or not.
	scCreateQueryParameterList (see page 31)	Creates a SCQueryParameterList to use with query functions.
	scDestroyGameStatsQueryResponse (see page 32)	Destroys a SCGameStatsQueryResponse (see page 43) object.
	scDestroyPlayerStatsQueryResponse (see page 32)	Destroys a SCPlayerStatsQueryResponse (see page 44) object.
	scDestroyQueryParameterList (see page 33)	Destroys a SCQueryParameterList object.
	scDestroyTeamStatsQueryResponse (see page 33)	Destroys a scDestroyTeamStatsQueryResponse object.

◆	scRunGameStatsQuery (see page 34)	Performs a game stat query.
◆	scRunPlayerStatsQuery (see page 34)	Performs a player stat query.
◆	scRunTeamStatsQuery (see page 35)	Performs a team stat query.

scCreateMatchlessSession Function

Summary

This is a variation of scCreateSession Function (see page 16) that creates a "matchless" session. A matchless session isn't sanity-checked against other session data for arbitration of any statistics. It essentially turns off detection for cheating. As data is sent in, it will be immediately applied to the stats instead of being applied when the match has finished. A "matchless" game may be more commonly known as non-arbitrated or unranked.

C++

```
COMMON_API SCResult SC_CALL scCreateMatchlessSession(
    SCInterfacePtr theInterface,
    const GSLoginCertificate * certificate,
    const GSLoginPrivateData * privateData,
    SCCreateSessionCallback callback,
    gsi_time timeoutMs,
    void * userData
);
```

Parameters

Parameters	Description
SCInterfacePtr theInterface	[in] A valid ATLAS (see page 13) SDK object.
const GSLoginCertificate * certificate	[in] Certificate obtained from the GameSpy AuthService.
const GSLoginPrivateData * privateData	[in] Private Data obtained from the GameSpy AuthService.
SCCreateSessionCallback callback	[in] The callback function to call when this request completes.
gsi_time timeoutMs	[in] The amount of time in milliseconds to wait for this operation to complete before timing out.
void * userData	[in] User data for use in callbacks. Note that it is a constant pointer in the callback.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

Reports sent for matchless sessions should be marked as such using "scReportSetAsMatchless (see page 25)".

See Also

scReportSetAsMatchless (see page 25), SCCreateSessionCallback (see page 37), scInitialize (see page 18), scThink (see page 29)

scCreateReport Function

Summary

Creates a new ATLAS (see page 13) report for the game session.

C++

```
COMMON_API SCResult SC_CALL scCreateReport(
    const SCInterfacePtr theInterface,
    gsi_u32 theRulesetVersion,
    gsi_u32 thePlayerCount,
    gsi_u32 theTeamCount,
    SCReportPtr * theReportOut
);
```

Parameters

Parameters	Description
const SCInterfacePtr theInterface	[in] A valid ATLAS (see page 13) SDK object.
gsi_u32 theRulesetVersion	[in] Ruleset version of the report as specified on the Web Admin Panel.
gsi_u32 thePlayerCount	[in] Player count for allocating enough resources and verification purposes.
gsi_u32 theTeamCount	[in] Team count for allocating enough resources and verification purposes.
SCReportPtr * theReportOut	[out] The pointer to the new SCReport object.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

You must call CreateSession and SetReportIntention at some point before calling this function. This function should be called after a game session has ended. The player count and team count are more accurate at that point for dedicated server games. This function should also be called before calling any scReport* function. The header version can be obtained from the ATLAS (see page 13) Web Admin Panel where the the keys are created.

See Also

scCreateSession (see page 16), scSetReportIntention (see page 26), scReportBeginGlobalData (see page 22), scReportBeginPlayerData (see page 23), scReportBeginTeamData (see page 24), scReportBeginNewPlayer (see page 22), scReportSetPlayerData (see page 25), scReportBeginNewTeam (see page 23), scReportSetTeamData (see page 26), scReportAddIntValue (see page 20), scReportAddStringValue (see page 21)

scCreateSession Function

Summary

Requests that the ATLAS (see page 13) service create a new game session and keep track of the session that is about to start.

C++

```
COMMON_API SCResult SC_CALL scCreateSession(
    SCInterfacePtr theInterface,
    const GSLoginCertificate * certificate,
    const GSLoginPrivateData * privateData,
    SCCreateSessionCallback callback,
    gsi_time timeoutMs,
    void * userData
);
```

Parameters

Parameters	Description
SCInterfacePtr theInterface	[in] A valid ATLAS (see page 13) SDK object
const GSLoginCertificate * certificate	[in] A valid certificate obtained from the GameSpy AuthService.
const GSLoginPrivateData * privateData	[in] Valid private data obtained from the GameSpy AuthService.
SCCreateSessionCallback callback	[in] The callback function to call when this request completes.
gsi_time timeoutMs	[in] The amount of time in milliseconds to wait for this operation to complete before timing out.

void * userData	[in] User data for use in callbacks. Note that it is a constant pointer in the callback.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

The certificate and private data may be NULL if the local client is an unauthenticated dedicated server. The function should be called by the host after initializing the SDK, and obtaining a certificate and private data from the authentication service. The competition service creates and sends a session ID to the host. The callback passed in will get called even if the request failed.

See Also

SCCreateSessionCallback (see page 37), scInitialize (see page 18), scThink (see page 29)

scDestroyReport Function

Summary

Used to clean up and free the report object after it has been submitted.

C++

```
COMMON_API SCResult SC_CALL scDestroyReport(
    SCReportPtr theReport
);
```

Parameters

Parameters	Description
SCReportPtr theReport	[in] The pointer to a created SC Report Object.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

This should be called regardless of whether or not the report was submitted successfully. It should only be used if the report object contains a valid pointer from a successful call to scCreateReport (see page 15).

See Also

scCreateReport (see page 15)

scGetConnectionId Function

Summary

Used to obtain the unique connection ID for the current game session.

C++

```
COMMON_API const char * scGetConnectionId(
    const SCInterfacePtr theInterface
);
```

Parameters

Parameters	Description
const SCInterfacePtr theInterface	[in] A valid ATLAS (see page 13) SDK object.

Returns

The connection GUID for the current game session.

Remarks

The connection id identifies a single player in a game session. It may be possible to have different connection ids during the same session, since players can come and leave sessions.

scGetSessionId Function

Summary

Gets the unique identifier (GUID) for the current game session.

C++

```
COMMON_API const char * scGetSessionId(
    const SCInterfacePtr theInterface
);
```

Parameters

Parameters	Description
const SCInterfacePtr theInterface	[in] A valid ATLAS (see page 13) SDK object.

Returns

The session GUID for the current game session.

Remarks

The session GUID identifies a single game session between players. After the host creates a session, the host calls this function to obtain the session GUID. The host then sends the session GUID to all other participants in the game session.

See Also

scSetSessionId (see page 27), scCreateSession (see page 16)

scInitialize Function

Summary

Initializes the ATLAS (see page 13) SDK, providing a valid ATLAS (see page 13) SDK object.

C++

```
COMMON_API SCResult SC_CALL scInitialize(
    int gameId,
    SCInterfacePtr * theInterfaceOut
);
```

Parameters

Parameters	Description
int gameId	[in] The GameSpy Game Id issued to identify a game.
SCInterfacePtr * theInterfaceOut	[out] A pointer to a new ATLAS (see page 13) SDK object.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

The function must be called in order to get a valid ATLAS (see page 13) SDK (SCInterface) object. Most other interface functions require an initialized interface object when being called. Note that if the standard GameSpy Availability Check was not performed prior to this call, the SDK will return SCResult_NO_AVAILABILITY_CHECK.

See Also

scShutdown (↗ see page 28)

scReportAddByteValue Function

Summary

Adds a byte value to the report for a specific key.

C++

```
COMMON_API SCResult SC_CALL scReportAddByteValue(
    const SCReportPtr theReportData,
    gsi_u16 theKeyId,
    gsi_i8 theValue
);
```

Parameters

Parameters	Description
const SCReportPtr theReportData	[in] A valid SC Report object.
gsi_u16 theKeyId	[in] Key Identifier for reporting data.
gsi_i8 theValue	[in] 8 bit Byte value representation of the data.

Returns

An SCResult (↗ see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

The host or player can call this function to add either global, player-, or team-specific data. A report needs to be created before calling this function. For global keys, this function can only be called after starting global data. For players or teams, a new player or team needs to be added.

See Also

scCreateReport (↗ see page 15), scReportBeginGlobalData (↗ see page 22), scReportBeginPlayerData (↗ see page 23), scReportBeginTeamData (↗ see page 24), scReportBeginNewPlayer (↗ see page 22), scReportSetPlayerData (↗ see page 25), scReportBeginNewTeam (↗ see page 23), scReportSetTeamData (↗ see page 26)

scReportAddFloatValue Function

Summary

Adds a float value to the report for a specific key.

C++

```
COMMON_API SCResult SC_CALL scReportAddFloatValue(
    const SCReportPtr theReportData,
    gsi_u16 theKeyId,
    float theValue
);
```

Parameters

Parameters	Description
const SCReportPtr theReportData	[in] A valid SC Report object.
gsi_u16 theKeyId	[in] Key Identifier for reporting data.
float theValue	[in] 32 bit Float value representation of the data.

Returns

An SCResult (↗ see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

The host or player can call this function to add either global, player-, or team-specific data. A report needs to be created before calling this function. For global keys, this function can only be called after starting global data. For players or teams, a new player or team needs to be added.

See Also

scCreateReport (see page 15), scReportBeginGlobalData (see page 22), scReportBeginPlayerData (see page 23), scReportBeginTeamData (see page 24), scReportBeginNewPlayer (see page 22), scReportSetPlayerData (see page 25), scReportBeginNewTeam (see page 23), scReportSetTeamData (see page 26)

scReportAddIntValue Function

Summary

Adds an integer value to the report for a specific key.

C++

```
COMMON_API SCResult SC_CALL scReportAddIntValue(
    SCReportPtr theReportData,
    gsi_u16 theKeyId,
    gsi_i32 theValue
);
```

Parameters

Parameters	Description
SCReportPtr theReportData	[in] A valid SC Report object.
gsi_u16 theKeyId	[in] Key Identifier for reporting data.
gsi_i32 theValue	[in] 32 bit Integer value representation of the data.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

The host or player can call this function to add either global, player-, or team-specific data. A report needs to be created before calling this function. For global keys, this function can only be called after starting global data. For players or teams, a new player or team needs to be added.

See Also

scCreateReport (see page 15), scReportBeginGlobalData (see page 22), scReportBeginPlayerData (see page 23), scReportBeginTeamData (see page 24), scReportBeginNewPlayer (see page 22), scReportSetPlayerData (see page 25), scReportBeginNewTeam (see page 23), scReportSetTeamData (see page 26)

scReportAddShortValue Function

Summary

Adds a short value to the report for a specific key.

C++

```
COMMON_API SCResult SC_CALL scReportAddShortValue(
    const SCReportPtr theReportData,
    gsi_u16 theKeyId,
    gsi_i16 theValue
);
```

Parameters

Parameters	Description
const SCReportPtr theReportData	[in] A valid SC Report object.
gsi_u16 theKeyId	[in] Key Identifier for reporting data.
gsi_i16 theValue	[in] 16 bit Short value representation of the data.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

The host or player can call this function to add either global, player-, or team-specific data. A report needs to be created before calling this function. For global keys, this function can only be called after starting global data. For players or teams, a new player or team needs to be added.

See Also

scCreateReport (see page 15), scReportBeginGlobalData (see page 22), scReportBeginPlayerData (see page 23), scReportBeginTeamData (see page 24), scReportBeginNewPlayer (see page 22), scReportSetPlayerData (see page 25), scReportBeginNewTeam (see page 23), scReportSetTeamData (see page 26)

scReportAddStringValue Function

Summary

Adds a string value to the report for a specific key.

C++

```
COMMON_API SCResult SC_CALL scReportAddStringValue(
    const SCReportPtr theReportData,
    gsi_u16 theKeyId,
    const gsi_char * theValue
);
```

Parameters

Parameters	Description
const SCReportPtr theReportData	[in] A valid SC Report object.
gsi_u16 theKeyId	[in] The string key's identifier.
const gsi_char * theValue	[in] The string value.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

The host or player can call this function to add either global, player-, or team-specific data. A report needs to be created before calling this function. For global keys, this function can only be called after starting global data. For players or teams, a new player or team needs to be added.

See Also

scCreateReport (see page 15), scReportBeginGlobalData (see page 22), scReportBeginPlayerData (see page 23), scReportBeginTeamData (see page 24), scReportBeginNewPlayer (see page 22), scReportSetPlayerData (see page 25), scReportBeginNewTeam (see page 23), scReportSetTeamData (see page 26)

scReportBeginGlobalData Function

Summary

Tells the ATLAS (see page 13) SDK to start writing global data to the report.

C++

```
COMMON_API SCResult SC_CALL scReportBeginGlobalData(
    SCReportPtr theReportData
);
```

Parameters

Parameters	Description
SCReportPtr theReportData	[ref] A valid SC Report Object.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

After creating a report, this function should be called prior to writing global game data. Keys and values are reported using the individual key-value functions.

ATLAS (see page 13) reports must include Global data first using scReportBeginGlobalData, then Player data using scReportBeginPlayerData (see page 23), and finally Team data using scReportBeginTeamData (see page 24). All three must be included in this order, even if there is no data to report for one or more of these sections.

See Also

scCreateReport (see page 15), scReportAddIntValue (see page 20), scReportAddStringValue (see page 21)

scReportBeginNewPlayer Function

Summary

Add a new player to the report.

C++

```
COMMON_API SCResult SC_CALL scReportBeginNewPlayer(
    SCReportPtr theReportData
);
```

Parameters

Parameters	Description
SCReportPtr theReportData	[in] A valid SC Report Object.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

This function adds new player data to the report.

See Also

scCreateReport (see page 15), scReportBeginPlayerData (see page 23), scReportSetPlayerData (see page 25), scReportAddIntValue (see page 20), scReportAddStringValue (see page 21)

scReportBeginNewTeam Function

Summary

Adds a new team to the report.

C++

```
COMMON_API SCResult SC_CALL scReportBeginNewTeam(
    SCReportPtr theReportData
);
```

Parameters

Parameters	Description
SCReportPtr theReportData	[in] A valid SC Report Object.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

After the beginning of any team data is set, this function can be called to start a new team. After this function has been called, the game can start adding team data to the report.

See Also

scCreateReport (see page 15), scReportBeginTeamData (see page 24), scReportSetPlayerData (see page 25), scReportAddIntValue (see page 20), scReportAddStringValue (see page 21)

scReportBeginPlayerData Function

Summary

Tells the ATLAS (see page 13) SDK to start writing player data to the report.

C++

```
COMMON_API SCResult SC_CALL scReportBeginPlayerData(
    SCReportPtr theReportData
);
```

Parameters

Parameters	Description
SCReportPtr theReportData	[in] A valid SC Report Object.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

Use this function to mark the start of player data. The game can start adding each player and its specific data after this is called.

ATLAS (see page 13) reports must include Global data first using scReportBeginGlobalData (see page 22), then Player data using scReportBeginPlayerData, and finally Team data using scReportBeginTeamData (see page 24). All three must be included in this order, even if there is no data to report for one or more of these sections.

See Also

scCreateReport (see page 15), scReportBeginNewPlayer (see page 22), scReportSetPlayerData (see page 25), scReportAddIntValue (see page 20), scReportAddStringValue (see page 21)

scReportBeginTeamData Function

Summary

Tells the ATLAS (see page 13) SDK to start writing player data to the report.

C++

```
COMMON_API SCResult SC_CALL scReportBeginTeamData(
    SCReportPtr theReportData
);
```

Parameters

Parameters	Description
SCReportPtr theReportData	[in] A valid SC Report Object

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

Use this function to mark the start of team data. The game can start adding each team and its specific data after this is called.

Team data must come after global data and before player data.

ATLAS (see page 13) reports must include Global data first using scReportBeginGlobalData (see page 22), then Player data using scReportBeginPlayerData (see page 23), and finally Team data using scReportBeginTeamData. All three must be included in this order, even if there is no data to report for one or more of these sections.

See Also

scCreateReport (see page 15), scReportBeginGlobalData (see page 22), scReportBeginPlayerData (see page 23) scReportBeginNewTeam (see page 23), scReportSetTeamData (see page 26), scReportAddIntValue (see page 20), scReportAddStringValue (see page 21)

scReportEnd Function

Summary

Denotes the end of a report for the report specified.

C++

```
COMMON_API SCResult SC_CALL scReportEnd(
    SCReportPtr theReport,
    gsi_bool isAuth,
    SCGameStatus theStatus
);
```

Parameters

Parameters	Description
SCReportPtr theReport	[in] A valid SC Report Object.
gsi_bool isAuth	[in] Authoritative report.
SCGameStatus theStatus	[in] Final Status of the reported game.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

Used to set the end of a report. The report must have been properly created and have some data. Any report being submitted requires that function be called before the submission. Incomplete reports will be discarded.

See Also

scCreateReport (see page 15), scSubmitReport (see page 28), SCGameStatus (see page 39)

scReportSetAsMatchless Function

Summary

Called after creating the report to set it as a matchless report; this is needed if the report is being submitted to a "matchless" game session. A "matchless" game may be more commonly known as non-arbitrated or unranked.

C++

```
COMMON_API SCResult SC_CALL scReportSetAsMatchless(
    SCReportPtr theReport
);
```

Parameters

Parameters	Description
SCReportPtr theReport	[ref] A valid SC Report Object.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

This should not be used for a non-matchless session report.

See Also

scCreateMatchlessSession (see page 15), scCreateReport (see page 15)

scReportSetPlayerData Function

Summary

Sets initial player data in the report specified.

C++

```
COMMON_API SCResult SC_CALL scReportSetPlayerData(
    SCReportPtr theReport,
    gsi_u32 thePlayerIndex,
    const gsi_u8 thePlayerConnectionId[SC_CONNECTION_GUID_SIZE],
    gsi_u32 thePlayerTeamId,
    SCGameResult result,
    gsi_u32 theProfileId,
    const GSLoginCertificate * certificate,
    const gsi_u8 theAuthData[16]
);
```

Parameters

Parameters	Description
SCReportPtr theReport	[ref] A valid SC Report Object.
gsi_u32 thePlayerIndex	[in] Index of the player (0 - Number of players).
const gsi_u8 thePlayerConnectionId[SC_CONNECTION_GUID_SIZE]	[in] Connection GUID generated by the ATLAS (see page 13) service and retrieved using scGetConnectionId (see page 17)().
SCGameResult result	[in] Standard SC Game result.
gsi_u32 theProfileId	[in] Profile ID of the player.

const GSLoginCertificate * certificate	[in] Certificate obtained from the GameSpy AuthService. Note: This parameter is currently unused.
const gsi_u8 theAuthData[16]	[in] Authentication data.
thePlayerTeamIndex	[in] Team index of the player, if that player is on a team.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

A report must have been created prior to using this function. Each player must have a valid login certificate from the authentication service also. This function should be called after a new player is added to the report. Any key-value pairs that need to be added should be done after calling this function.

See Also

scCreateReport (see page 15), scReportBeginPlayerData (see page 23), scReportBeginNewPlayer (see page 22), scReportAddIntValue (see page 20), scReportAddStringValue (see page 21)

scReportSetTeamData Function

Summary

Sets the initial team data in the report specified.

C++

```
COMMON_API SCResult SC_CALL scReportSetTeamData(
    SCReportPtr theReport,
    gsi_u32 theTeamId,
    SCGameResult result
);
```

Parameters

Parameters	Description
SCReportPtr theReport	[in] A valid SC Report Object.
SCGameResult result	[in] The team's result (e.g. win, loss, draw).
theTeamIndex	[in] The index of the team being reported.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

A report must have been created prior to using this function. This function should be called after a new team is added to the report. Any key-value pairs that need to be added should be done after calling this function.

See Also

scCreateReport (see page 15), scReportBeginTeamData (see page 24), scReportBeginNewTeam (see page 23), scReportAddIntValue (see page 20), scReportAddStringValue (see page 21)

scSetReportIntention Function

Summary

Called to tell ATLAS (see page 13) the type of report that the player or host will send.

C++

```
COMMON_API SCResult SC_CALL scSetReportIntention(
    const SCInterfacePtr theInterface,
    const gsi_u8 theConnectionId[SC_CONNECTION_GUID_SIZE],
    gsi_bool isAuthoritative,
    const GSLoginCertificate * certificate,
    const GSLoginPrivateData * privateData,
    SCSetReportIntentionCallback callback,
    gsi_time timeoutMs,
    void * userData
);
```

Parameters

Parameters	Description
const SCInterfacePtr theInterface	[ref] A valid ATLAS (see page 13) SDK object.
const gsi_u8 theConnectionId[SC_CONNECTION_GUID_SIZE]	[in] The player's connection id. Set to NULL unless the player is rejoining a session he previously left.
gsi_bool isAuthoritative	[in] Flag set if the snapshot being reported will be authoritative.
const GSLoginCertificate * certificate	[ref] Certificate obtained from the authentication web service.
const GSLoginPrivateData * privateData	[ref] Private data obtained from the authentication web service.
SCSetReportIntentionCallback callback	[ref] The callback function to call when this request completes.
gsi_time timeoutMs	[in] The amount of time in milliseconds to wait for this operation to complete before timing out.
void * userData	[ref] Application data that may be used in the callback.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

The should be called by both the host and client before sending a report.

The host should have created a session before calling this. It allows the server to know ahead of time what type of report will be sent. Reports submitted without an intention will be discarded.

Notes

The theConnectionId argument should be set to NULL unless the player is rejoining a session he previously left.

See Also

scCreateSession (see page 16), SCSetReportIntentionCallback (see page 37), scSubmitReport (see page 28)

scSetSessionId Function

Summary

Used to set the session ID for the current game session.

C++

```
COMMON_API SCResult SC_CALL scSetSessionId(
    const SCInterfacePtr theInterface,
    const gsi_u8 theSessionId[SC_SESSION_GUID_SIZE]
);
```

Parameters

Parameters	Description
const SCInterfacePtr theInterface	[in] A valid ATLAS (see page 13) SDK object
const gsi_u8 theSessionId[SC_SESSION_GUID_SIZE]	[in] The session GUID of length SC_SESSION_GUID_SIZE

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return

SCResult_NO_ERROR if the request completed successfully.

Remarks

The session ID identifies a single game session between players. Players should use the scGetSessionId (see page 18) function in order to obtain the session ID. This should not be called if a session has not yet been created.

See Also

scGetSessionId (see page 18), scCreateSession (see page 16)

scShutdown Function

Summary

Closes down the specified ATLAS (see page 13) SDK object and frees memory.

C++

```
COMMON_API SCResult SC_CALL scShutdown(
    SCInterfacePtr theInterface
);
```

Parameters

Parameters	Description
SCInterfacePtr theInterface	[in] A valid ATLAS (see page 13) SDK object

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

In order to clean up all resources used by the SDK, this interface function must be called once operations with this SDK are complete. Do not call this function if you plan to continue reporting stats.

See Also

scInitialize (see page 18)

scSubmitReport Function

Summary

Initiates the submission of a report.

C++

```
COMMON_API SCResult SC_CALL scSubmitReport(
    const SCInterfacePtr theInterface,
    const SCReportPtr theReport,
    gsi_bool isAuthoritative,
    const GSLoginCertificate * certificate,
    const GSLoginPrivateData * privateData,
    SCSubmitReportCallback callback,
    gsi_time timeoutMs,
    void * userData
);
```

Parameters

Parameters	Description
const SCInterfacePtr theInterface	[in] A valid ATLAS (see page 13) SDK object.
const SCReportPtr theReport	[in] A valid SC Report object.
gsi_bool isAuthoritative	[in] Flag to tell if the snapshot is authoritative.
const GSLoginCertificate * certificate	[in] Certificate obtained from the GameSpy AuthService.

const GSLoginPrivateData * privateData	[in] Private Data obtained from the GameSpy AuthService.
SCSubmitReportCallback callback	[in] The callback function to call when this request completes.
gsi_time timeoutMs	[in] The amount of time in milliseconds to wait for this operation to complete before timing out.
void * userData	[in] Application data that may be used in the callback.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

Once the report has been completed with a call to scReportEnd (see page 24), the player or host can call this function to submit a report. The certificate and private data are both required to submit a report. Incomplete reports will be discarded. The callback passed in will tell the game the result of the operation.

See Also

scInitialize (see page 18), scCreateSession (see page 16), scSetReportIntention (see page 26), scReportEnd (see page 24), SCSubmitReportCallback (see page 38), scThink (see page 29)

scThink Function

Summary

Called to complete pending operations for functions with callbacks.

C++

```
COMMON_API SCResult SC_CALL scThink(
    SCInterfacePtr theInterface
);
```

Parameters

Parameters	Description
SCInterfacePtr theInterface	[in] A valid ATLAS (see page 13) SDK object.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

This function should be called with a valid interface object. It will take care of pending requests that have been made by the interface functions.

See Also

scInitialize (see page 18), scCreateSession (see page 16), scSetReportIntention (see page 26), scSubmitReport (see page 28)

scReportAddInt64Value Function

Summary

Adds a 64-bit integer value to the report for a specific key.

C++

```
COMMON_API SCResult SC_CALL scReportAddInt64Value(
    SCReportPtr theReportData,
    gsi_u16 theKeyId,
    gsi_i64 theValue
);
```

Parameters

Parameters	Description
SCReportPtr theReportData	[in] A valid SC Report object.
gsi_u16 theKeyId	[in] Key Identifier for reporting data.
gsi_i64 theValue	[in] 64 bit Integer value representation of the data.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

The host or player can call this function to add either global, player-, or team-specific data. A report needs to be created before calling this function. For global keys, this function can only be called after starting global data. For players or teams, a new player or team needs to be added.

See Also

scCreateReport (see page 15), scReportBeginGlobalData (see page 22), scReportBeginPlayerData (see page 23), scReportBeginTeamData (see page 24), scReportBeginNewPlayer (see page 22), scReportSetPlayerData (see page 25), scReportBeginNewTeam (see page 23), scReportSetTeamData (see page 26)

scAddQueryParameterToList Function

Summary

Adds a parameter name and value to the parameter list.

C++

```
COMMON_API SCResult SC_CALL scAddQueryParameterToList(
    SCQueryParameterListPtr queryParams,
    const gsi_char * name,
    const gsi_char * value
);
```

Parameters

Parameters	Description
SCQueryParameterListPtr queryParams	[out] The pointer to the query parameters created.
const gsi_char * name	[in] The name of the parameter
const gsi_char * value	[in] The value of the parameter

Returns

Enum value used to indicate the specific result of the call. This will return SCResult_NO_ERROR if the call completed successfully.

Remarks

Parameters need to be added to a SCQueryParameterList before passing a SCQueryParameterList to a query function. This function should be used to add those parameters while not going over the number of parameters pre-allocated in scCreateQueryParameterList (see page 31). Keep in mind that incorrect parameter names or values are not going to be handled here. The query function will inform the user appropriately.

See Also

scCreateQueryParameterList (see page 31), scDestroyQueryParameterList (see page 33)

scCheckBanList Function

Summary

Checks if a given gsid account is whitelisted or not.

C++

```
COMMON_API SCResult SC_CALL scCheckBanList(
    SCInterfacePtr theInterface,
    const GSLoginCertificate * certificate,
    const GSLoginPrivateData * privateData,
    gsi_u32 hostProfileId,
    SCPlatform hostPlatform,
    SCCheckBanListCallback callback,
    gsi_time timeoutMs,
    void * userData
);
```

Parameters

Parameters	Description
SCInterfacePtr theInterface	[in] A valid ATLAS (see page 13) SDK object
const GSLoginCertificate * certificate	[in] A valid certificate obtained from the GameSpy AuthService.
const GSLoginPrivateData * privateData	[in] Valid private data obtained from the GameSpy AuthService.
gsi_u32 hostProfileId	[in] Profileid of the host
SCPlatform hostPlatform	[in] Platform of the host.
SCCheckBanListCallback callback	[in] The callback function to call when this request completes.
gsi_time timeoutMs	[in] The amount of time in milliseconds to wait for this operation to complete before timing out.
void * userData	[in] User data for use in callbacks. Note that it is a constant pointer in the callback.

Returns

An SCResult (see page 40) enum value used to indicate the specific result of the request. This will return SCResult_NO_ERROR if the request completed successfully.

Remarks

Only applicable if whitelisting is enabled for your title. See Also

scCreateQueryParameterList Function

Summary

Creates a SCQueryParameterList to use with query functions.

C++

```
COMMON_API SCResult SC_CALL scCreateQueryParameterList(
    SCQueryParameterListPtr * queryParams,
    gsi_u32 queryParamsCount
);
```

Parameters

Parameters	Description
SCQueryParameterListPtr * queryParams	[out] The pointer to the query parameters created. This should point to NULL when passed.
gsi_u32 queryParamsCount	[in] The number of parameters that will be added.

Returns

Enum value used to indicate the specific result of the call. This will return SCResult_NO_ERROR if the call completed successfully.

Remarks

This needs to be called before performing any query function other than game queries. Any invalid input to the function will return SCResult_INVALID_PARAMETERS. The user must pass in a queryParamsCount large enough to add the parameters required by the query (as listed on the Web Admin Panel). The function will return a failure code if the queryParams points to a non-NULL object. When the query has completed, it is the user's responsibility to clean up the parameter list.

See Also

scAddQueryParameterToList (see page 30), scDestroyQueryParameterList (see page 33)

scDestroyGameStatsQueryResponse Function

Summary

Destroys a SCGameStatsQueryResponse Structure (see page 43) object.

C++

```
COMMON_API SCResult SC_CALL scDestroyGameStatsQueryResponse(
    SCGameStatsQueryResponse ** response
);
```

Parameters

Parameters	Description
SCGameStatsQueryResponse ** response	[out] The pointer to the response passed into the callback.

Returns

Enum value used to indicate the specific result of the call. This will return SCResult_NO_ERROR if the call completed successfully.

Remarks

Use this function to destroy and NULL out an SCGameStatsQueryResponse (see page 43). This call will only free a non-NULL response.

See Also

SCPlayerStatQueryCallback

scDestroyPlayerStatsQueryResponse Function

Summary

Destroys a SCPlayerStatsQueryResponse Structure (see page 44) object.

C++

```
COMMON_API SCResult SC_CALL scDestroyPlayerStatsQueryResponse(
    SCPlayerStatsQueryResponse ** response
);
```

Parameters

Parameters	Description
SCPlayerStatsQueryResponse ** response	[out] The pointer to the response passed into the callback.

Returns

Enum value used to indicate the specific result of the call. This will return SCResult_NO_ERROR if the call completed successfully.

Remarks

Use this function to destroy and NULL out an SCPlayerStatsQueryResponse (see page 44). This call will only free a non-NULL response.

See Also

SCPlayerStatQueryCallback

scDestroyQueryParameterList Function

Summary

Destroys a SCQueryParameterList object.

C++

```
COMMON_API SCResult SC_CALL scDestroyQueryParameterList(
    SCQueryParameterListPtr * queryParams
);
```

Parameters

Parameters	Description
SCQueryParameterListPtr * queryParams	[out] The pointer to the query parameters created.

Returns

Enum value used to indicate the specific result of the call. This will return SCResult_NO_ERROR if the call completed successfully.

Remarks

This function must be called after either a query is complete or the SCQueryParameterList is no longer needed. The function will free data only if queryParams points to non-NULL data. The parameter will also be nulled out for the user.

See Also

scCreateQueryParameterList (see page 31)

scDestroyTeamStatsQueryResponse Function

Summary

Destroys a scDestroyTeamStatsQueryResponse object.

C++

```
COMMON_API SCResult SC_CALL scDestroyTeamStatsQueryResponse(
    SCTeamStatsQueryResponse ** response
);
```

Parameters

Parameters	Description
SCTeamStatsQueryResponse ** response	[out] The pointer to the response passed into the callback.

Returns

Enum value used to indicate the specific result of the call. This will return SCResult_NO_ERROR if the call completed successfully.

Remarks

Use this function to destroy and NULL out an scDestroyTeamStatsQueryResponse. This call will only free a non-NULL response.

See Also

SCTeamStatsQueryCallback

scRunGameStatsQuery Function

Summary

Performs a game stat query.

C++

```
COMMON_API SCResult SC_CALL scRunGameStatsQuery(
    SCInterfacePtr interfacePtr,
    const GSLoginCertificate * certificate,
    const GSLoginPrivateData * privData,
    int ruleSetVersion,
    const char queryId[GS_GUID_SIZE],
    SCQueryParameterListPtr queryParameters,
    SCGameStatsQueryCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
const GSLoginCertificate * certificate	[in] Pointer to GSLoginCertificate (see page 50) obtained from authservice.
const GSLoginPrivateData * privData	[in] Pointer to GSLoginPrivateData (see page 51) obtained from authservice.
int ruleSetVersion	[in] Ruleset version number created by the user on the Web Admin Panel.
const char queryId[GS_GUID_SIZE]	[in] A guid generated after building a query on the Web Admin Panel; you can use the variable defined in the auto-generated header file rather than pass the guid manually.
SCQueryParameterListPtr queryParameters	[in] SCQueryParameterListPtr that was built with parameters obtained from the Web Admin Panel.
SCGameStatsQueryCallback callback	[in] User-supplied callback that must be valid in order to receive stats.
void * userData	[in/out] Application data that may be used in the callback.
theInterface	[in] A valid SCInterfacePtr.

Returns

Enum value used to indicate the specific result of the call. This will return SCResult_NO_ERROR if the call completed successfully.

Remarks

Queries must first be built using the Web Admin Panel for ATLAS (see page 13). Once a query has been generated, you can take advantage of this generic function for game stats. The user must pass in valid parameters or the following error will be returned: SCResult_INVALID_PARAMETERS. The user should be calling the scThink (see page 29) function in order for this request to complete.

See Also

scCreateQueryParameterList (see page 31)

scRunPlayerStatsQuery Function

Summary

Performs a player stat query.

C++

```
COMMON_API SCResult SC_CALL scRunPlayerStatsQuery(
    SCInterfacePtr theInterface,
    const GSLoginCertificate * certificate,
    const GSLoginPrivateData * privData,
    int ruleSetVersion,
    const char queryId[GS_GUID_SIZE],
    const SCQueryParameterListPtr queryParameters,
    SCPlayerStatsQueryCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SCInterfacePtr theInterface	[in] A valid SCInterfacePtr.
const GSLoginCertificate * certificate	[in] Pointer to GSLoginCertificate (see page 50) obtained from authservice.
const GSLoginPrivateData * privData	[in] Pointer to GSLoginPrivateData (see page 51) obtained from authservice.
int ruleSetVersion	[in] Ruleset version number created by the user on the Web Admin Panel.
const char queryId[GS_GUID_SIZE]	[in] A guid generated after building a query on the Web Admin Panel; you can use the variable defined in the auto-generated header file rather than pass the guid manually.
const SCQueryParameterListPtr queryParameters	[in] SCQueryParameterListPtr that was built with parameters obtained from the Web Admin Panel.
SCPlayerStatsQueryCallback callback	[in] User-supplied callback that must be valid in order to receive stats.
void * userData	[in/out] Application data that may be used in the callback.

Returns

Enum value used to indicate the specific result of the call. This will return SCResult_NO_ERROR if the call completed successfully.

Remarks

Queries must first be built using the Web Admin Panel for ATLAS (see page 13). Once a query has been generated, you can take advantage of this generic function for player stats. This function can be used for both single and multiple players or for a ranked leaderboard. The user must pass in valid parameters or the following error will be returned: SCResult_INVALID_PARAMETERS. The user should be calling the scThink (see page 29) function in order for this request to complete.

See Also

scCreateQueryParameterList (see page 31)

scRunTeamStatsQuery Function

Summary

Performs a team stat query.

C++

```
COMMON_API SCResult SC_CALL scRunTeamStatsQuery(
    SCInterfacePtr theInterface,
    const GSLoginCertificate * certificate,
    const GSLoginPrivateData * privData,
    int ruleSetVersion,
    const char queryId[GS_GUID_SIZE],
    const SCQueryParameterListPtr queryParameters,
    SCTeamStatsQueryCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SCInterfacePtr theInterface	[in] A valid SCInterfacePtr.
const GSLoginCertificate * certificate	[in] Pointer to GSLoginCertificate (see page 50) obtained from authservice.
const GSLoginPrivateData * privData	[in] Pointer to GSLoginPrivateData (see page 51) obtained from authservice.
int ruleSetVersion	[in] Ruleset version number created by the user on the Web Admin Panel.
const char queryId[GS_GUID_SIZE]	[in] A guid generated after building a query on the Web Admin Panel; you can use the variable defined in the auto-generated header file rather than pass the guid manually.
const SCQueryParameterListPtr queryParameters	[in] SCQueryParameterListPtr which was built with parameters obtained from the Web Admin Panel.
SCTeamStatsQueryCallback callback	[in] User-supplied callback that must be valid in order to receive stats.
void * userData	[in/out] Application data that may be used in the callback.

Returns

Enum value used to indicate the specific result of the call. This will return SCResult_NO_ERROR if the call completed successfully.

Remarks

Queries must first be built using the Web Admin Panel for ATLAS (see page 13). Once a query has been generated, you can take advantage of this generic function for player stats. This function can be used for both single and multiple players. The user must pass in valid parameters or the following error will be returned: SCResult_INVALID_PARAMETERS.

See Also

scCreateQueryParameterList (see page 31), SCTeamStatsQueryCallback

Callbacks

Types

Name	Description
SCCheckBanListCallback (see page 36)	Called when scCheckBanList (see page 31) has completed.
SCCreateSessionCallback (see page 37)	Called when scCreateSession (see page 16) has completed.
SCSetReportIntentionCallback (see page 37)	Called when scReportIntention has completed.
SCSubmitReportCallback (see page 38)	Called when scSubmitReport (see page 28) completes.

SCCheckBanListCallback Type

Summary

Called when scCheckBanList Function (see page 31) has completed.

C++

```
typedef void (* SCCheckBanListCallback)(const SCInterfacePtr theInterface, GHTTPResult
httpResult, SCResult result, void * userData, int resultProfileId, int resultPlatformId,
gsi_bool resultProfileBannedHost);
```

Parameters

Parameters	Description
theInterface	[in] The pointer to the SC Interface object. The game usually also has a copy of this.
httpResult	[in] Http result from creating a session.
result	[in] SC Result telling the application what happened when checking the game's ban list.
theUserData	[in] Constant pointer to user data.
resultProfileId	[in] The profileid that was checked.
resultPlatformId	[in] The platformid that was checked.
resultProfileBannedHost	[in] The ban status.

Remarks

Only applicable if whitelisting is enabled for your title. See Also

SCCreateSessionCallback Type

Summary

Called when scCreateSession Function (see page 16) has completed.

C++

```
typedef void (* SCCreateSessionCallback)(const SCInterfacePtr theInterface, GHTTPResult
httpResult, SCResult result, void * userData);
```

Parameters

Parameters	Description
theInterface	[in] A pointer to the ATLAS (see page 13) SDK object. The game usually also has a copy of this.
httpResult	[in] The HTTP (see page 58) result of the most recent HTTP (see page 58) transaction.
result	[in] The SDK result (SCResult (see page 40)) of this request.
userData	[in] A constant pointer to user-provided data.

Remarks

Called when a game session is created. The results will determine if the game session was successfully created. If there were any errors, the result will be set to the specific error code. Otherwise, the result will be set to SCResult_NO_ERROR. See SCResult (see page 40) for error codes.

See Also

scCreateSession (see page 16), SCResult (see page 40)

SCSetReportIntentionCallback Type

Summary

Called when scReportIntention has completed.

C++

```
typedef void (* SCSetReportIntentionCallback)(const SCInterfacePtr theInterface,
GHTTPResult httpResult, SCResult result, void * userData);
```

Parameters

Parameters	Description
theInterface	[in] A pointer to the ATLAS (see page 13) SDK object. The game usually also has a copy of this.
httpResult	[in] The HTTP (see page 58) result of the most recent HTTP (see page 58) transaction.
result	[in] The SDK result (SCResult (see page 40)) of this request.
userData	[in] A constant pointer to user-provided data.

Remarks

Called when a host or client reporting its intention is complete. The results will determine if the game session was successfully created. If there were any errors, the result will be set to the specific error code. Otherwise, the result will be set to SCResult_NO_ERROR.

See SCResult (see page 40) for error codes.

See Also

scSetReportIntention (see page 26), SCResult (see page 40)

SCSubmitReportCallback Type

Summary

Called when scSubmitReport Function (see page 28) completes.

C++

```
typedef void (* SCSubmitReportCallback)(const SCInterfacePtr theInterface, GHTTPResult
httpResult, SCResult result, void * userData);
```

Parameters

Parameters	Description
theInterface	[in] A pointer to the ATLAS (see page 13) SDK object. The game usually also has a copy of this.
httpResult	[in] The HTTP (see page 58) result of the most recent HTTP (see page 58) transaction.
result	[in] The SDK result (SCResult (see page 40)) of this request.
userData	[in] A constant pointer to user-provided data.

Remarks

After the SDK submits the report, the backend will send back results that will be available in this callback. If there were any errors, the result will be set to the specific error code. Otherwise, the result will be set to SCResult_NO_ERROR.

See SCResult (see page 40) for error codes.

See Also

scSubmitReport (see page 28), SCResult (see page 40)

Enumerations

Enumerations

Name	Description
SCGameResult (see page 39)	Used when submitting a report for a game session to indicate the result of the game session for the current player or team.

SCGameStatus (see page 39)	Indicates how the game session ended and is declared when ending a report.
SCResult (see page 40)	SCResult is used to indicate error and failure conditions in the ATLAS (see page 13) SDK.

SCGameResult Enumeration

Summary

Used when submitting a report for a game session to indicate the result of the game session for the current player or team.

C++

```
typedef enum {
  SCGameResult_WIN,
  SCGameResult_LOSS,
  SCGameResult_DRAW,
  SCGameResult_DISCONNECT,
  SCGameResult_DESYNC,
  SCGameResult_NONE,
  SCGameResultMax
} SCGameResult;
```

Members

Members	Description
SCGameResult_WIN	The game session resulted in a win for the current player or team.
SCGameResult_LOSS	The game session resulted in a loss for the current player or team.
SCGameResult_DRAW	The game session resulted in a draw for the current player or team.
SCGameResult_DISCONNECT	The current player or team disconnected during the game session.
SCGameResult_DESYNC	The current player or team lost sync during the game session.
SCGameResult_NONE	There was no result from the game session for the current player or team.
SCGameResultMax	The upper bound of game result codes.

Remarks

Can be used for both player and a team.

SCGameStatus Enumeration

Summary

Indicates how the game session ended and is declared when ending a report.

C++

```
typedef enum {
  SCGameStatus_COMPLETE,
  SCGameStatus_PARTIAL,
  SCGameStatus_BROKEN,
  SCGameStatusMax
} SCGameStatus;
```

Members

Members	Description
SCGameStatus_COMPLETE	The game session came to the expected end without interruption (disconnects, desyncs). This status indicates that game results are available for all players.
SCGameStatus_PARTIAL	Although the game session came to the expected end, one or more players unexpectedly quit or were disconnected. Game results should explicitly report which players were disconnected to be used during normalization for possible penalty metrics.
SCGameStatus_BROKEN	The game session did not reach the expected end point and is incomplete. This should be reported when there has been an event detected that makes the end result indeterminate.
SCGameStatusMax	The upper bound of game status codes.

Remarks

For SCGameStatus reporting, the game should do the following:

- As long as the game finished properly, and no one disconnected during the course of play, then all players in the game session should submit SCGameStatus_COMPLETE reports. If any members disconnected during play, but the game was finished completely, then all players in the session should submit SCGameStatus_PARTIAL reports indicating that disconnects occurred. For any players who do not complete the session, a SCGameStatus_BROKEN report should be submitted.
- Thus, if the game did not completely finish, all players will submit broken reports. The only case that will trigger an invalid report is if reports for the same game describe status as both SCGameStatus_COMPLETEand SCGameStatus_PARTIAL.
- Since COMPLETE indicates that all players finished the game without a disconnect and PARTIAL indicates that disconnects occured, at no time should a game report both COMPLETE and PARTIAL -- this will be seen as an exploit and invalidate the report.

SCResult Enumeration

Summary

SCResult is used to indicate error and failure conditions in the ATLAS (see page 13) SDK.

C++

```
typedef enum {
  SCResult_NO_ERROR = 0,
  SCResult_NO_AVAILABILITY_CHECK,
  SCResult_INVALID_PARAMETERS,
  SCResult_NOT_INITIALIZED,
  SCResult_CORE_NOT_INITIALIZED,
  SCResult_OUT_OF_MEMORY,
  SCResult_CALLBACK_PENDING,
  SCResult_HTTP_ERROR,
  SCResult_UNKNOWN_RESPONSE,
  SCResult_RESPONSE_INVALID,
  SCResult_INVALID_DATATYPE,
  SCResult_REPORT_INCOMPLETE,
  SCResult_REPORT_INVALID,
  SCResult_SUBMISSION_FAILED,
  SCResult_QUERY_DISABLED,
  SCResult_QUERY_TYPE_MISMATCH,
  SCResult_QUERY_INVALID,
  SCResult_QUERY_PARAMS_MISSING,
  SCResult_QUERY_PARAMS_TOO_MANY,
  SCResult_QUERY_PARAM_TYPE_INVALID,
  SCResult_UNKNOWN_ERROR,
  SCResult_INVALID_GAMEID,
  SCResult_INVALID_SESSIONTOKEN,
  SCResult_SESSIONTOKEN_EXPIRED,
  SCResultMax
} SCResult;
```

Members

Members	Description
SCResult_NO_ERROR = 0	No error has occurred.
SCResult_NO_AVAILABILITY_CHECK	The standard GameSpy Availability Check was not performed prior to initialization.
SCResult_INVALID_PARAMETERS	Parameters passed to interface function were invalid.
SCResult_NOT_INITIALIZED	The SDK was not initialized.
SCResult_CORE_NOT_INITIALIZED	The GameSpy Core SDK was initialized by the application.
SCResult_OUT_OF_MEMORY	The SDK could not allocate memory for its resources.
SCResult_CALLBACK_PENDING	Rhe operation is still pending.
SCResult_HTTP_ERROR	Error if the backend fails to respond with correct HTTP.
SCResult_UNKNOWN_RESPONSE	Error if the SDK cannot understand the result.
SCResult_RESPONSE_INVALID	Error if the SDK cannot read the response from the backend.
SCResult_INVALID_DATATYPE	Error if an invalid datatype is received.
SCResult_REPORT_INCOMPLETE	The report was incomplete.
SCResult_REPORT_INVALID	Part or all of report is invalid.
SCResult_SUBMISSION_FAILED	Submission of report failed.
SCResult_QUERY_DISABLED	Error occurs if the query id is disabled on the Web Admin Panel.
SCResult_QUERY_TYPE_MISMATCH	Error occurs if the query id passed is used for the wrong query type (e.g., query id is passed into a game stats query instead of player stats query.
SCResult_QUERY_INVALID	Error occurs if the query id is invalid or not found. The text message will provide details.
SCResult_QUERY_PARAMS_MISSING	Error occurs if a parameter or parameters for the specified query are missing.
SCResult_QUERY_PARAMS_TOO_MANY	Error occurs if the number of params exceeds the expected number
SCResult_QUERY_PARAM_TYPE_INVALID	Error occurs if a parameter value that is passed does not match what the query expects (e.g., if the query expects a number string and a non-numeric string is passed in).
SCResult_UNKNOWN_ERROR	Error unknown to SDK
SCResult_INVALID_GAMEID	Make sure GameID is properly set with wsSetCredentials.
SCResult_INVALID_SESSIONTOKEN	Make sure wsSetCredentials was called with valid credentials and you have logged in via AuthService.
SCResult_SESSIONTOKEN_EXPIRED	Re-login via AuthService to refresh your 'session'.
SCResultMax	Total number of result codes that can be returned.

Remarks

Results of a call to an interface function or operation. It can be used to see if the initial call to a function completed without error. The callback that is passed to interface functions will also have a value that is of this type. The application can check this value for failures.

Structures

Structures

	Name	Description
◆	SCTeamStatsQueryResponse (see page 42)	Response received from backend for team stats query. This object is comprised of stats for the requested category or categories for each team in the return set.
◆	SCGameStatsCategory (see page 42)	Category object that contains a name and global game stats. These objects will be a part of a SCGameStatsQueryResponse (see page 43) object that will be created by the SDK and passed to the user-defined callback.
◆	SCGameStatsQueryResponse (see page 43)	Response received from backend for game stats query. This object is comprised of stats for the requested category or categories.
◆	SCPlayer (see page 43)	User-facing object that contains player data. Each player will have a profile id and some stats.
◆	SCPlayerStatsCategory (see page 43)	Category object that contains a name and a set of players. These objects will be a part of a SCPlayerStatsQueryResponse (see page 44) object which will be created by the SDK and passed to the user-defined callback.
◆	SCPlayerStatsQueryResponse (see page 44)	Response received from backend for player stats query. This object is comprised of stats for the requested category or categories for each player in the return set.
◆	SCStat (see page 44)	User-facing object that contains a stat name, value, and type. The name and type of the stat correlates to that on the backend administration site. The type will be one of the values defined in the SCStatDataType.
◆	SCTeam (see page 44)	User-facing object that contains team data. Each team will have a team id and some stats.
◆	SCTeamStatsCategory (see page 45)	Category object that contains a name and a set of teams. These objects will be a part of a SCTeamStatsQueryResponse (see page 42) object which will be created by the SDK and passed to the user-defined callback.

SCTeamStatsQueryResponse Structure

Summary

Response received from backend for team stats query. This object is comprised of stats for the requested category or categories for each team in the return set.

C++

```
struct SCTeamStatsQueryResponse {
  gsi_u32 mCategoriesCount;
  SCTeamStatsCategory * mCategories;
};
```

See Also

SCTeamStatQueryCallback, SCTeamStatsCategory (see page 45)

SCGameStatsCategory Structure

Summary

Category object that contains a name and global game stats. These objects will be a part of a SCGameStatsQueryResponse Structure (see page 43) object that will be created by the SDK and passed to the user-defined callback.

C++

```
struct SCGameStatsCategory {
  gsi_char * mName;
  gsi_u32 mStatsCount;
  SCStat * mStats;
};
```

Members

Members	Description
gsi_char * mName;	Name of the category.
gsi_u32 mStatsCount;	Player count for mPlayers below.
SCStat * mStats;	Array of SCPlayer objects.

See Also

SCGameStatsQueryResponse (see page 43), SCStat (see page 44)

SCGameStatsQueryResponse Structure

Summary

Response received from backend for game stats query. This object is comprised of stats for the requested category or categories.

C++

```
struct SCGameStatsQueryResponse {
  gsi_u32 mCategoriesCount;
  SCGameStatsCategory * mCategories;
};
```

See Also

SCPlayerStatQueryCallback, SCGameStatCategory

SCPlayer Structure

Summary

User-facing object that contains player data. Each player will have a profile id and some stats.

C++

```
struct SCPlayer {
  gsi_u32 mProfileId;
  gsi_u32 mStatsCount;
  SCStat * mStats;
};
```

Members

Members	Description
gsi_u32 mProfileId;	Profile ID associated with the current player.
gsi_u32 mStatsCount;	Stats count for mStats below.
SCStat * mStats;	Array of SCStat objects.

See Also

SCPlayerStatsCategory (see page 43), SCStat (see page 44)

SCPlayerStatsCategory Structure

Summary

Category object that contains a name and a set of players. These objects will be a part of a SCPlayerStatsQueryResponse Structure (see page 44) object which will be created by the SDK and passed to the user-defined callback.

C++

```
struct SCPlayerStatsCategory {
    gsi_char * mName;
    gsi_u32 mPlayersCount;
    SCPlayer * mPlayers;
  };
```

Members

Members	Description
gsi_char * mName;	Name of the category.
gsi_u32 mPlayersCount;	Player count for mPlayers below.
SCPlayer * mPlayers;	Array of SCPlayer objects.

See Also

SCPlayerStatsQueryResponse (see page 44), SCPlayer (see page 43)

SCPlayerStatsQueryResponse Structure

Summary

Response received from backend for player stats query. This object is comprised of stats for the requested category or categories for each player in the return set.

C++

```
struct SCPlayerStatsQueryResponse {
  gsi_u32 mCategoriesCount;
  SCPlayerStatsCategory * mCategories;
};
```

See Also

SCPlayerStatQueryCallback, SCPlayerStatsCategory (see page 43)

SCStat Structure

Summary

User-facing object that contains a stat name, value, and type. The name and type of the stat correlates to that on the backend administration site. The type will be one of the values defined in the SCStatDataType.

C++

```
struct SCStat {
  gsi_char * mName;
  SCDataType mStatType;
  gsi_char * mValue;
};
```

Members

Members	Description
gsi_char * mName;	Name for this stat.
SCDataType mStatType;	Type for this stat.
gsi_char * mValue;	Value for this stat.

See Also

SCGameStatsCategory (see page 42), SCPlayer (see page 43), SCTeam (see page 44)

SCTeam Structure

Summary

User-facing object that contains team data. Each team will have a team id and some stats.

C++

```
struct SCTeam {
  gsi_u32 mTeamId;
  gsi_u32 mStatsCount;
  SCStat * mStats;
};
```

Members

Members	Description
gsi_u32 mTeamId;	Team ID associated with the current player.
gsi_u32 mStatsCount;	Stats count for mStats below.
SCStat * mStats;	Array of SCStat objects.

See Also

SCTeamStatsCategory (see page 45), SCStat (see page 44)

SCTeamStatsCategory Structure

Summary

Category object that contains a name and a set of teams. These objects will be a part of a SCTeamStatsQueryResponse Structure (see page 42) object which will be created by the SDK and passed to the user-defined callback.

C++

```
struct SCTeamStatsCategory {
  gsi_char * mName;
  gsi_u32 mTeamsCount;
  SCTeam * mTeams;
};
```

Members

Members	Description
gsi_char * mName;	Name of the category.
gsi_u32 mTeamsCount;	Player count for mPlayers below.
SCTeam * mTeams;	Array of SCPlayer objects.

See Also

SCTeamStatsQueryResponse (see page 42), SCTeam (see page 44)

Auth Service

API Documentation

Module

Auth Service (see page 45)

Functions

Functions

	Name	Description
	wsLoginSonyCert (see page 46)	Login for the PS3 or PSP system; authenticates the NP account and creates a corresponding GP (see page 87) 'shadow account'.
	wsLoginProfile (see page 47)	Login using the full GameSpy Presence login information, requiring email, password, and profile name.
	wsLoginRemoteAuth (see page 47)	Login by authenticating with a partner system and then use that authentication information with the GameSpy system.
	wsLoginUnique (see page 48)	Login using a subset of the GameSpy Presence login information, requiring only a unique nick and password.
	wsLoginValueString (see page 49)	Given a WSLoginValue, this returns a meaningful string describing the login result.
	wsSetGameCredentials (see page 49)	Set your Access Key and Secret Key, which will be passed with every web service call for authentication and usage/metric tracking.
	wsGetBuddyList (see page 49)	Grabs the names and profileids for a user's buddy list using an authenticated login certificate.
	wsLoginCertIsValid (see page 50)	Certificate Utilities, for use after obtaining a certificate.

wsLoginSonyCert Function

Summary

Login for the PS3 or PSP system; authenticates the NP account and creates a corresponding Presence and Messaging (see page 87) 'shadow account'.

C++

```
COMMON_API WSLoginValue wsLoginSonyCert(
    int gameId,
    int partnerCode,
    int namespaceId,
    const gsi_u8 * ps3cert,
    int certLen,
    WSLoginSonyCertCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
int gameId	[in] The game id for your title.
int partnerCode	[in] The partner code.
int namespaceId	[in] The namespace ID.
int certLen	[in] The length of the npTicket.
WSLoginSonyCertCallback callback	[in] Pointer to a function that will be called by the SDK to report the result of the authentication request.
void * userData	[in] A pointer to data that will be supplied to the callback function.
ps3Cert	[in] The npTicket obtained from the Sony SDK.

Returns

WSLoginValue: If successful, the value WSLogin_Success will be returned. Otherwise, a code specific to the error encountered will be returned.

Notes

The GameSpy SDK core must be initialized first using gsCoreInitialize before using this function.

wsLoginProfile Function

Summary

Login using the full GameSpy Presence login information, requiring email, password, and profile name.

C++

```
COMMON_API WSLoginValue wsLoginProfile(
    int gameId,
    int partnerCode,
    int namespaceId,
    const gsi_char * profileNick,
    const gsi_char * email,
    const gsi_char * password,
    const gsi_char * cdkeyhash,
    WSLoginCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
int gameId	[in] The game id.
int partnerCode	[in] The partner code.
int namespaceId	[in] The namespace ID.
const gsi_char * profileNick	[in] The nickname associated with the user's GameSpy account.
const gsi_char * email	[in] The email associated with user's GameSpy account.
const gsi_char * password	[in] The password for the user's GameSpy account.
const gsi_char * cdkeyhash	[in] CD key hash.
WSLoginCallback callback	[in] Pointer to a function that will be called by the SDK to report the result of the authentication request.
void * userData	[in] A pointer to data that will be supplied to the callback function.

Returns

WSLoginValue: If successful, the value WSLogin_Success will be returned. Otherwise, a code specific to the error encountered will be returned.

Notes

The GameSpy SDK Core must be initialized first using gsCoreInitialize before using this function.

wsLoginRemoteAuth Function

Summary

Login by authenticating with a partner system and then use that authentication information with the GameSpy system.

C++

```
COMMON_API WSLoginValue wsLoginRemoteAuth(
    int gameId,
    int partnerCode,
    int namespaceId,
    const gsi_char authtoken[WS_LOGIN_AUTHTOKEN_LEN],
    const gsi_char partnerChallenge[WS_LOGIN_PARTNERCHALLENGE_LEN],
    WSLoginCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
int gameId	[in] The game id.
int partnerCode	[in] The partner code.
int namespaceId	[in] The namespace ID.
const gsi_char authtoken[WS_LOGIN_AUTHTOKEN_LEN]	[in] The authentication token.
const gsi_char partnerChallenge[WS_LOGIN_PARTNERCHALLENGE_LEN]	[in] The partner challenge.
WSLoginCallback callback	[in] Pointer to a function that will be called by the SDK to report the result of the authentication request.
void * userData	[in] A pointer to data that will be supplied to the callback function.

Returns

WSLoginValue: If successful, the value WSLogin_Success will be returned. Otherwise, a code specific to the error encountered will be returned.

Notes

The GameSpy SDK core must be initialized first using gsCoreInitialize before using this function.

wsLoginUnique Function

Summary

Login using a subset of the GameSpy Presence login information, requiring only a unique nick and password.

C++

```
COMMON_API WSLoginValue wsLoginUnique(
    int gameId,
    int partnerCode,
    int namespaceId,
    const gsi_char * uniqueNick,
    const gsi_char * password,
    const gsi_char * cdkeyhash,
    WSLoginCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
int gameId	[in] The game id.
int partnerCode	[in] The partner code.
int namespaceId	[in] The namespace ID.
const gsi_char * uniqueNick	[in] The unique nickname associated with the user's GameSpy account.
const gsi_char * password	[in] The password for the user's GameSpy account.
const gsi_char * cdkeyhash	[in] CD key hash.
WSLoginCallback callback	[in] Pointer to a function that will be called by the SDK to report the result of the authentication request.
void * userData	[in] A pointer to data that will be supplied to the callback function.

Returns

WSLoginValue: If successful, the value WSLogin_Success will be returned. Otherwise, a code specific to the error encountered will be returned.

Notes

The GameSpy SDK core must be initialized first using gsCoreInitialize before using this function.

wsLoginValueString Function

Summary

Given a WSLoginValue, this returns a meaningful string describing the login result.

C++

```
COMMON_API const char* wsLoginValueString(
    int loginValue
);
```

Parameters

Parameters	Description
int loginValue	[in] The login value

Returns

A null-byte terminated character string describing the login result corresponding to the value given.

Notes

The returned string is read-only and must not be modified.

wsSetGameCredentials Function

Summary

Set your Access Key and Secret Key, which will be passed with every web service call for authentication and usage/metric tracking.

C++

```
COMMON_API void wsSetGameCredentials(
    const char * accessKey,
    const int gameId,
    const char * secretKey
);
```

Parameters

Parameters	Description
const char * accessKey	[in] The Access Key provided via the GameSpy Developer portal.
const char * secretKey	[in] The Secret Key provided by the GameSpy Developer portal.

Notes

This must be called prior to calling any wsLogin function.

wsGetBuddyList Function

Summary

Grabs the names and profileids for a user's buddy list using an authenticated login certificate.

C++

```
COMMON_API GSTask * wsGetBuddyList(
    int gameId,
    const GSLoginCertificate * certificate,
    const GSLoginPrivateData * privData,
    WSGetBuddyListCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
int gameId	[in] The game id for your title.
const GSLoginCertificate * certificate	[in] The user's authenticated login certificate.
const GSLoginPrivateData * privData	[in] The private data returned with authentication.
WSGetBuddyListCallback callback	[in] Pointer to a function that will be called by the SDK to report the result of the request.
void * userData	[in] A pointer to data that will be supplied to the callback function.

Returns

WSRequest: If successful, the created GSTask will be returned, which can be used to cancel the request with gsiCoreCancelTask if necessary. Otherwise, NULL will be returned.

Notes

The GameSpy SDK core must be initialized first using gsCoreInitialize before using this function.

See Also

WSGetBuddyListCallback, gsiCoreCancelTask

wsLoginCertIsValid Function

C++

```
COMMON_API gsi_bool wsLoginCertIsValid(
    const GSLoginCertificate * cert
);
```

Structures

Structures

	Name	Description
◆	GSLoginCertificate (see page 50)	A user's login certificate, signed by the GameSpy AuthService. The certificate is public and may be freely passed around. Avoid use of pointer members so that the structure may be easily copied.
◆	GSLoginCertificatePrivate (see page 51)	Private information for the owner of the certificate only: **NOTE:** Be careful! Private key information must be kept secret.
◆	WSLoginResponse (see page 51)	CERTIFICATE login callback format:
	GSLoginPrivateData (see page 51)	Private information for the owner of the certificate only: **NOTE:** Be careful! Private key information must be kept secret.
◆	WSLoginSonyCertResponse (see page 51)	PS3 login callback format

GSLoginCertificate Structure

C++

```
struct GSLoginCertificate {
  gsi_bool mIsValid;
  gsi_u32 mLength;
  gsi_u32 mVersion;
```

```
    gsi_u32 mPartnerCode;
    gsi_u32 mNamespaceId;
    gsi_u32 mUserId;
    gsi_u32 mProfileId;
    gsi_u32 mExpireTime;
    gsi_char mProfileNick[WS_LOGIN_NICK_LEN];
    gsi_char mUniqueNick[WS_LOGIN_UNIQUENICK_LEN];
    gsi_char mCdKeyHash[WS_LOGIN_KEYHASH_LEN];
    gsCryptRSAKey mPeerPublicKey;
    gsi_u8 mSignature[GS_CRYPT_RSA_BYTE_SIZE];
    gsi_u8 mServerData[WS_LOGIN_SERVERDATA_LEN];
    gsi_char mTimestamp[WS_LOGIN_TIMESTAMP_LEN];
  };
```

Members

Members	Description
gsi_u32 mPartnerCode;	Also called the Account space.
gsi_char mCdKeyHash[WS_LOGIN_KEYHASH_LEN];	hexstr - bigendian
gsi_u8 mSignature[GS_CRYPT_RSA_BYTE_SIZE];	binary - bigendian
gsi_u8 mServerData[WS_LOGIN_SERVERDATA_LEN];	binary - bigendian

GSLoginCertificatePrivate Structure

C++

```
  struct GSLoginCertificatePrivate {
    gsCryptRSAKey mPeerPrivateKey;
    char mKeyHash[GS_CRYPT_MD5_HASHSIZE];
  };
```

WSLoginResponse Structure

C++

```
  struct WSLoginResponse {
    WSLoginValue mLoginResult;
    WSLoginValue mResponseCode;
    GSLoginCertificate mCertificate;
    GSLoginPrivateData mPrivateData;
    void * mUserData;
  };
```

Members

Members	Description
WSLoginValue mLoginResult;	SDK high-level result (e.g., LoginFailed).
WSLoginValue mResponseCode;	Server's result code (e.g., BadPassword).
GSLoginCertificate mCertificate;	Show this to others (proves: "Bill is a valid user").
GSLoginPrivateData mPrivateData;	Keep this secret! (proves: "I am Bill").

GSLoginPrivateData Structure

C++

```
  typedef struct GSLoginCertificatePrivate {
    gsCryptRSAKey mPeerPrivateKey;
    char mKeyHash[GS_CRYPT_MD5_HASHSIZE];
  } GSLoginPrivateData;
```

WSLoginSonyCertResponse Structure

C++

```
  struct WSLoginSonyCertResponse {
    WSLoginValue mLoginResult;
    WSLoginValue mResponseCode;
    char mRemoteAuthToken[WS_LOGIN_AUTHTOKEN_LEN];
    char mPartnerChallenge[WS_LOGIN_PARTNERCHALLENGE_LEN];
    void * mUserData;
  };
```

Members

Members	Description
WSLoginValue mLoginResult;	SDK high-level result (e.g., LoginFailed).
WSLoginValue mResponseCode;	Server's result code (e.g., BadPassword).
char mRemoteAuthToken[WS_LOGIN_AUTHTOKEN_LEN];	Show this to others.
char mPartnerChallenge[WS_LOGIN_PARTNERCHALLENGE_LEN];	Keep this secret (it's a "password" for the token).

CD Key

API Documentation

Module

CD Key (see page 52)

Functions

Functions

	Name	Description
	gcd_authenticate_user (see page 52)	Creates a new client and sends a request for authorization to the validation server.
	gcd_disconnect_all (see page 53)	Calls gcd_disconnect_user (see page 53) for each user still online.
	gcd_disconnect_user (see page 53)	Notify the validation server that a user has disconnected.
	gcd_getkeyhash (see page 54)	Returns the key hash for the given user.
	gcd_init (see page 54)	Initializes the Server API and creates the sockets and structures.
	gcd_process_reauth (see page 54)	Used to respond to a re-authentication request made by the validation server to prove that the client is still on.
	gcd_shutdown (see page 55)	Release the socket and send disconnect messages to the validation server for any clients still on the server.
	gcd_think (see page 55)	Processes any pending data from the validation server and calls the callback to indicate whether or not a client was authorized.
	gcd_compute_response (see page 55)	Calculates a response to a challenge string.
	gcd_init_qr2 (see page 56)	Initializes the Server API and integrates the networking of the CDKey SDK with the Query & Reporting 2 SDK.

gcd_authenticate_user Function

Summary

Creates a new client and sends a request for authorization to the validation server.

C++

```
COMMON_API void gcd_authenticate_user(
    int gameid,
    int localid,
    unsigned int userip,
    const char * challenge,
    const char * response,
    AuthCallBackFn authfn,
    RefreshAuthCallBackFn refreshfn,
    void * instance
);
```

Parameters

Parameters	Description
int gameid	[in] The game ID issued for your game.
int localid	[in] A unique int used to identify each client on the server. No two clients should have the same localid.
unsigned int userip	[in] The client's IP address, preferably in network byte order.
const char * challenge	[in] The challenge string that was sent to the client. Should be no more than 32 characters.
const char * response	[in] The response that the client received.
AuthCallBackFn authfn	[in] A callback that is called when the user is either authorized or rejected.
RefreshAuthCallBackFn refreshfn	[in] A callback called when the server needs to re-authorize a client on the local host.
void * instance	[in] Optional free-format user data for use by the callback.

Remarks

If host self-authorization is being used, the recommended way of implementing host authentication is through the qr2_register_publicaddress_callback (see page 169). We recommend this implementation due to an issue with port forwarding on the host's end that can block communication from the CD Key service.

See Also

qr2_register_publicaddress_callback (see page 169)

gcd_disconnect_all Function

Summary

Calls gcd_disconnect_user Function (see page 53) for each user still online.

C++

```
COMMON_API void gcd_disconnect_all(
    int gameid
);
```

Parameters

Parameters	Description
int gameid	[in] The game ID issued for your game.

gcd_disconnect_user Function

Summary

Notify the validation server that a user has disconnected.

C++

```
COMMON_API void gcd_disconnect_user(
    int gameid,
    int localid
);
```

Parameters

Parameters	Description
int gameid	[in] The game ID issued for your game.
int localid	[in] The unique int used to identify the user.

gcd_getkeyhash Function

Summary

Returns the key hash for the given user.

C++

```
COMMON_API char * gcd_getkeyhash(
    int gameid,
    int localid
);
```

Parameters

Parameters	Description
int gameid	[in] The game ID issued for your game.
int localid	[in] The unique int used to identify the user.

Returns

Returns the key hash string, or an empty string if that user is not connected.

Remarks

The hash returned will always be the same for a given user. This makes it useful for banning or tracking of users (used with the Tracking/Stats SDK). Returns an empty string if that user isn't connected.

gcd_init Function

Summary

Initializes the Server API and creates the sockets and structures.

C++

```
COMMON_API int gcd_init(
    int gameid
);
```

Parameters

Parameters	Description
int gameid	[in] The game ID issued for your game.

Returns

Returns 0 if successful. Returns non-zero if there was an error.

Remarks

Should only be called once (unless gcd_shutdown (see page 55) has been called).

gcd_process_reauth Function

Summary

Used to respond to a re-authentication request made by the validation server to prove that the client is still on.

C++

```
COMMON_API void gcd_process_reauth(
    int gameid,
    int localid,
    int hint,
    const char * response
);
```

Parameters

Parameters	Description
int gameid	[in] The game ID used to initialize the SDK with.
int localid	[in] An index of the client.
const char * response	[in] The client's response to the challenge.
skey	[in] The client's session key that came from the validation server.

Remarks

When the Reauthentication callback (passed to gcd_ authenticate user) is called, the host must send the required information to verify that the client is still online, using the CD Key being checked. This should be called after the client has computed a response to the challenge coming from the callback.

gcd_shutdown Function

Summary

Release the socket and send disconnect messages to the validation server for any clients still on the server.

C++

```
COMMON_API void gcd_shutdown();
```

gcd_think Function

Summary

Processes any pending data from the validation server and calls the callback to indicate whether or not a client was authorized.

C++

```
COMMON_API void gcd_think();
```

Remarks

This function should be called at least once every 10-100ms and is guaranteed not to block (although it may make a callback if an authorization response has come in). If your game uses the Query and Reporting SDK, you can place this call in the same area as the call to qr_process_queries.

gcd_compute_response Function

Summary

Calculates a response to a challenge string.

C++

```
COMMON_API void gcd_compute_response(
    char * cdkey,
    char * challenge,
    char response[73],
    CDResponseMethod method
);
```

Parameters

Parameters	Description
char * cdkey	[in] The client's CD key.
char * challenge	[in] The challenge string. Should be no more than 32 characters.
char response[73]	[out] Receives the computed response string.
CDResponseMethod method	[in] Enum listing the response method; set this to either CDResponseMethod_NEWAUTH or CDResponseMethod_REAUTH.

Remarks

When the client receives the challenge string, it should calculate a response using the gcd_compute_response function in the Client API.

gcd_init_qr2 Function

Summary

Initializes the Server API and integrates the networking of the CDKey SDK with the Query & Reporting 2 SDK.

C++

```
COMMON_API int gcd_init_qr2(
    qr2_t qrec,
    int gameid
);
```

Parameters

Parameters	Description
qr2_t qrec	[in] The intialized QR2 SDK object.
int gameid	[in] The game ID issued for your game.

Returns

Returns 0 if successful. Returns non-zero if there was an error.

Remarks

You must initialize the Query & Reporting 2 SDK with qr2_init (see page 163) or qr2_init_socket (see page 164) prior to calling this. If you are using multiple instances of the QR2 SDK, you can pass the specific instance information in via the "qrec" argument. Otherwise, you can simply pass in NULL.

Make sure to use this function instead of the deprecated gcd_init (see page 54)() function. This is mandatory to pass source review certification.

Callbacks

Types

Name	Description
RefreshAuthCallBackFn (see page 56)	Used to reauthenticate a client for the purpose of proving that a client is still online.
AuthCallBackFn (see page 57)	Called when the user's CD key is either authorized or rejected.

RefreshAuthCallBackFn Type

Summary

Used to reauthenticate a client for the purpose of proving that a client is still online.

C++

```
typedef void (* RefreshAuthCallBackFn)(int gameid, int localid, int hint, char * challenge,
void * instance);
```

Parameters

Parameters	Description
gameid	[in] The game ID used to initialize the SDK.
localid	[in] The index of the player.
hint	[in] A session id for a client used for reauthentication; this is the skey passed into gcd_process_reauth (see page 54).
challenge	[in] A challenge string used for reauthentication.
instance	[in] User data passed in gcd_authenticate_user (see page 52).

Remarks

The reauthentication callback will be called any time the validation server attempts to determine that a client is still online. When called, the client index, challenge, and session key will be available. These values must be used to re-authenticate the user. Remember that this process is similar to the primary authentication process, where the only difference is that the validation server provides the challenge and session key (note: the "hint" parameter in this callback is the session key that should be passed as the "skey" value into gcd_process_reauth (see page 54)).

If the user was not authenticated, the errmsg parameter contains a descriptive string of the reason (either CD Key not valid, or CD Key in use).

AuthCallBackFn Type

Summary

Called when the user's CD key is either authorized or rejected.

C++

```
typedef void (* AuthCallBackFn)(int gameid, int localid, int authenticated, char * errmsg,
void * instance);
```

Parameters

Parameters	Description
gameid	[in] The game ID for which authentication is requested.
localid	[in] The id that was passed into gcd_authenticate_user (see page 52).
authenticated	[in] Indicates if the user was authenticated: 1 if authenticated; 0 if not.
errmsg	[in] Error message if user was not authenticated.
instance	[in] The same instance as was passed into the gcd_authenticate_user (see page 52).

Remarks

This function will be called within two seconds of gcd_authenticate_user (see page 52), even if the validation server has not yet responded.

If the authentication failed, one of the following errmsg strings will be received:

"Bad Response": The CD Key was incorrect. Check that the CD Key was correctly typed or passed to the compute response function. Make sure the gcd_authenticate_user (see page 52) is passed the correct values.

"Invalid CD Key": The CD Key is not registered for this game on the GameSpy backend.

"Invalid authentication": Either the CD Key was bad or the response and challenge were bad. Make sure the gcd_authenticate_user (see page 52) is passed the correct values.

"Your CD Key is disabled. Contact customer service.": The specific, provided CD key value has been turned off.

"CD Key in use": The CD Key provided was in use by another player.

"Validation Timeout": The host was not able to reach the CD key validation server. The SDK intentionally authenticates the user in this case, since it would not be desirable to reject players without network connectivity.

See Also

gcd_authenticate_user (see page 52)

Enumerations

Enumerations

Name	Description
CDResponseMethod (see page 58)	Values are passed to the gcd_compute_response (see page 55) function, which needs to be implemented client-side.

CDResponseMethod Enumeration

Summary

Values are passed to the gcd_compute_response Function (see page 55) function, which needs to be implemented client-side.

C++

```
typedef enum {
  CDResponseMethod_NEWAUTH,
  CDResponseMethod_REAUTH
} CDResponseMethod;
```

Members

Members	Description
CDResponseMethod_NEWAUTH	method = 0 for normal auth. Used for primary authentications.
CDResponseMethod_REAUTH	method = 1 for ison proof. Used for re-authentications.

HTTP

API Documentation

Module

HTTP (see page 58)

Functions

Functions

	Name	Description
●	ghttpPostEx (see page 60)	Do an HTTP (see page 58) POST, which can be used to upload data to a web server.

◆	ghttpPostSetAutoFree (see page 61)	Sets a post object's auto-free flag.
◆	ghttpPostSetCallback (see page 61)	Sets the callback for a post object.
◆	ghttpRequestThink (see page 61)	Process one particular HTTP (see page 58) request on Windows.
◆	ghttpSave (see page 62)	Make an HTTP (see page 58) GET request and save the response to disk.
◆	ghttpSaveEx (see page 62)	Make an HTTP (see page 58) GET request and save the response to disk.
◆	ghttpSetMaxRecvTime (see page 63)	Used to throttle based on time, not on bandwidth.
◆	ghttpSetProxy (see page 64)	Sets a proxy server address.
◆	ghttpSetRequestProxy (see page 64)	Sets a proxy server for a specific request.
◆	ghttpSetThrottle (see page 64)	Used to start or stop throttling an existing connection.
◆	ghttpStartup (see page 65)	Initialize the HTTP (see page 58) SDK.
◆	ghttpStream (see page 65)	Make an HTTP (see page 58) GET request and stream in the response without saving it in memory.
◆	ghttpStreamEx (see page 66)	Make an HTTP (see page 58) GET request and stream in the response without saving it in memory.
◆	ghttpThink (see page 67)	Processes all current HTTP (see page 58) requests.
◆	ghttpThrottleSettings (see page 67)	Used to adjust the throttle settings.
◆	ghttpFreePost (see page 67)	Release the specified post object.
◆	ghttpGet (see page 68)	Make an HTTP (see page 58) GET request and save the response to memory.
◆	ghttpGetEx (see page 68)	Make an HTTP (see page 58) GET request and save the response to memory.
◆	ghttpGetHeaders (see page 69)	Get the response headers from an HTTP (see page 58) request.
◆	ghttpGetResponseStatus (see page 70)	Get an HTTP (see page 58) response's status string and status code.
◆	ghttpGetState (see page 70)	Obtain the current state of an HTTP (see page 58) request.
◆	ghttpGetURL (see page 71)	Used to obtain the URL associated with an HTTP (see page 58) request.
◆	ghttpHead (see page 71)	Make an HTTP (see page 58) HEAD request, which will only retrieve the response headers and not the normal response body.
◆	ghttpHeadEx (see page 72)	Make an HTTP (see page 58) HEAD request, which will only retrieve the response headers and not the normal response body.
◆	ghttpNewPost (see page 72)	Creates a new post object, which is used to represent data to send a web server as part of an HTTP (see page 58) request.
◆	ghttpPost (see page 73)	Do an HTTP (see page 58) POST, which can be used to upload data to a web server.
◆	ghttpPostAddFileFromDisk (see page 74)	Adds a disk file to the specified post object.
◆	ghttpPostAddFileFromMemory (see page 74)	Adds a file from memory to the specified post object.
◆	ghttpPostAddString (see page 75)	Adds a string to the specified post object.
◆	ghttpCancelRequest (see page 75)	Cancel an HTTP (see page 58) request in progress.
◆	ghttpCleanup (see page 76)	Clean up and close down the HTTP (see page 58) SDK. Free internally allocated memory.
◆	ghttpCleanupRootCAList (see page 76)	This function is used resetting the root certificate list.

◆	ghttpFreePostAndUpdateConnection (see page 76)	Free a post object.
◆	ghttpResultString (see page 77)	Given a GHTTPResult, returns a meaningful character string describing the result.
◆	ghttpSetRootCAList (see page 77)	This function is used for setting the root certificate list when making https calls, if the SSL Engine used supports the setting of expected server root certificate from validation purposes.

ghttpPostEx Function

Summary

Do an HTTP (see page 58) POST, which can be used to upload data to a web server.

C++

```
COMMON_API GHTTPRequest ghttpPostEx(
    const gsi_char * URL,
    const gsi_char * headers,
    GHTTPPost post,
    GHTTPBool throttle,
    GHTTPBool blocking,
    ghttpProgressCallback progressCallback,
    ghttpCompletedCallback completedCallback,
    void * param
);
```

Parameters

Parameters	Description
const gsi_char * URL	[in] URL
const gsi_char * headers	[in] Optional headers to pass with the request. Can be NULL or "".
GHTTPPost post	[in] The data to be posted.
GHTTPBool throttle	[in] If true, throttle this connection's download speed.
GHTTPBool blocking	[in] If true, this call doesn't return until finished.
ghttpProgressCallback progressCallback	[in] Called whenever new data is received. Can be NULL.
ghttpCompletedCallback completedCallback	[in] Called when the file has finished streaming. Can be NULL.
void * param	[in] Optional free-format user data to send to the callback.

Returns

If less than 0, the request failed and this is a GHTTPRequestError value. Otherwise, it identifies the request.

Remarks

This function is used to post data to a web page, ignoring any possible response body sent by the server (response status and response headers can still be checked).

If you want to post data and receive a response, use ghttpGetEx (see page 68), ghttpSaveEx (see page 62), or ghttpStreamEx (see page 66).

Use ghttpPost (see page 73) for a simpler version of this function.

Use this function to do an HTTP (see page 58) Post, don't try to access a GHTTPPost object directly.

See Also

ghttpGetEx (see page 68), ghttpSaveEx (see page 62), ghttpStreamEx (see page 66), ghttpHeadEx (see page 72), ghttpPost (see page 73)

ghttpPostSetAutoFree Function

Summary

Sets a post object's auto-free flag.

C++

```
COMMON_API void ghttpPostSetAutoFree(
    GHTTPPost post,
    GHTTPBool autoFree
);
```

Parameters

Parameters	Description
GHTTPPost post	[in] Post object
GHTTPBool autoFree	[in] True if object should be auto-freed

Remarks

By default post objects automatically free themselves after posting. To use the same post with more than one request, set auto-free to false, then use ghttpFreePost (see page 67) to free it after every request it's being used in is completed.

Use this function to do an HTTP (see page 58) Post; don't try to access a GHTTPPost object directly.

See Also

ghttpNewPost (see page 72), ghttpFreePost (see page 67), ghttpPost (see page 73)

ghttpPostSetCallback Function

Summary

Sets the callback for a post object.

C++

```
COMMON_API void ghttpPostSetCallback(
    GHTTPPost post,
    ghttpPostCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GHTTPPost post	[in] The post object to set the callback on.
ghttpPostCallback callback	[in] The callback to call when using this post object.
void * param	[in] User data passed to the callback.

See Also

ghttpNewPost (see page 72)

ghttpRequestThink Function

Summary

Process one particular HTTP (see page 58) request on Windows.

C++

```
COMMON_API GHTTPBool ghttpRequestThink(
    GHTTPRequest request
);
```

Parameters

Parameters	Description
GHTTPRequest request	[in] A valid request object to process.

Returns

GHTTPFalse if the request cannot be found.

Remarks

This allows an HTTP (see page 58) request to be processed in a separate thread. This function is only supported on Windows.

See Also

ghttpThink (see page 67)

ghttpSave Function

Summary

Make an HTTP (see page 58) GET request and save the response to disk.

C++

```
COMMON_API GHTTPRequest ghttpSave(
    const gsi_char * URL,
    const gsi_char * filename,
    GHTTPBool blocking,
    ghttpCompletedCallback completedCallback,
    void * param
);
```

Parameters

Parameters	Description
const gsi_char * URL	[in] URL
const gsi_char * filename	[in] The path and name to store the file as locally.
GHTTPBool blocking	[in] If true, this call doesn't return until the file has been received.
ghttpCompletedCallback completedCallback	[in] Called when the file has been received. Can be NULL.
void * param	[in] Optional free-format user data for use by the callback.

Returns

If less than 0, the request failed and this is a GHTTPRequestError value. Otherwise it identifies the request.

Remarks

This function is used to download the contents of a web page directly to disk. The application supplies the path and filename at which to save the response.

Use ghttpSaveEx (see page 62) for extra optional parameters.

See Also

ghttpGet (see page 68), ghttpSaveEx (see page 62), ghttpStream (see page 65), ghttpHead (see page 71), ghttpPost (see page 73)

ghttpSaveEx Function

Summary

Make an HTTP (see page 58) GET request and save the response to disk.

C++

```
COMMON_API GHTTPRequest ghttpSaveEx(
    const gsi_char * URL,
    const gsi_char * filename,
    const gsi_char * headers,
    GHTTPPost post,
    GHTTPBool throttle,
    GHTTPBool blocking,
    ghttpProgressCallback progressCallback,
    ghttpCompletedCallback completedCallback,
    void * param
);
```

Parameters

Parameters	Description
const gsi_char * URL	[in] URL
const gsi_char * filename	[in] The path and name to store the file as locally.
const gsi_char * headers	[in] Optional headers to pass with the request. Can be NULL or "".
GHTTPPost post	[in] Optional data to be posted. Can be NULL.
GHTTPBool throttle	[in] If true, throttle this connection's download speed.
GHTTPBool blocking	[in] If true, this call doesn't return until the file has been received.
ghttpProgressCallback progressCallback	[in] Called periodically with progress updates. Can be NULL.
ghttpCompletedCallback completedCallback	[in] Called when the file has been received. Can be NULL.
void * param	[in] Optional free-format user data to send to the callback.

Returns

If less than 0, the request failed and this is a GHTTPRequestError value. Otherwise it identifies the request.

Remarks

This function is used to download the contents of a web page directly to disk.

The application supplies the path and filename at which to save the response.

Use ghttpSave (see page 62) for a simpler version of this function.

See Also

ghttpGetEx (see page 68), ghttpSave (see page 62), ghttpStreamEx (see page 66), ghttpHeadEx (see page 72), ghttpPostEx (see page 60)

ghttpSetMaxRecvTime Function

Summary

Used to throttle based on time, not on bandwidth.

C++

```
COMMON_API void ghttpSetMaxRecvTime(
    GHTTPRequest request,
    gsi_time maxRecvTime
);
```

Parameters

Parameters	Description
GHTTPRequest request	[in] A valid request object
gsi_time maxRecvTime	[in] Maximum receive time

Remarks

Prevents recv-loop blocking on ultrafast connections without directly limiting transfer rate.

ghttpSetProxy Function

Summary

Sets a proxy server address.

C++

```
COMMON_API GHTTPBool ghttpSetProxy(
    const char * server
);
```

Parameters

Parameters	Description
const char * server	[in] The address of the proxy server.

Returns

GHTTPFalse if the server format is invalid.

Remarks

The address must be of the form "[:port]". If port is omitted, 80 will be used.

If server is NULL or "", no proxy server will be used. This should not be called while there are any current requests.

See Also

ghttpSetRequestProxy (see page 64)

ghttpSetRequestProxy Function

Summary

Sets a proxy server for a specific request.

C++

```
COMMON_API GHTTPBool ghttpSetRequestProxy(
    GHTTPRequest request,
    const char * server
);
```

Parameters

Parameters	Description
GHTTPRequest request	[in] A valid request object
const char * server	[in] The address of the proxy server.

Returns

GHTTPFalse if the server format is invalid or the request is invalid.

Remarks

The address must be of the form "[:port]". If port is omitted, 80 will be used.

If server is NULL or "", no proxy server will be used. This should not be called while there are any current requests.

ghttpSetThrottle Function

Summary

Used to start or stop throttling an existing connection.

C++

```
COMMON_API void ghttpSetThrottle(
    GHTTPRequest request,
    GHTTPBool throttle
);
```

Parameters

Parameters	Description
GHTTPRequest request	[in] A valid request object
GHTTPBool throttle	[in] True or false to enable or disable throttling.

Remarks

This may not be as efficient as starting a request with the desired setting.

See Also

ghttpThrottleSettings (see page 67)

ghttpStartup Function

Summary

Initialize the HTTP (see page 58) SDK.

C++

```
COMMON_API void ghttpStartup();
```

Remarks

Startup/Cleanup is reference counted, so always call ghttpStartup and ghttpCleanup (see page 76) in pairs.

See Also

ghttpCleanup (see page 76)

ghttpStream Function

Summary

Make an HTTP (see page 58) GET request and stream in the response without saving it in memory.

C++

```
COMMON_API GHTTPRequest ghttpStream(
    const gsi_char * URL,
    GHTTPBool blocking,
    ghttpProgressCallback progressCallback,
    ghttpCompletedCallback completedCallback,
    void * param
);
```

Parameters

Parameters	Description
const gsi_char * URL	[in] URL
GHTTPBool blocking	[in] If true, this call doesn't return until the file has finished streaming.
ghttpProgressCallback progressCallback	[in] Called whenever new data is received. Can be NULL.
ghttpCompletedCallback completedCallback	[in] Called when the file has finished streaming. Can be NULL.
void * param	[in] Optional free-format user data to send to the callback.

Returns

If less than 0, the request failed and this is a GHTTPRequestError value. Otherwise, it identifies the request.

Remarks

This function is used to stream in the contents of a web page. The response body is not stored in memory or to disk. It is only passed to the progressCallback as it is received, and the application can do what it wants with the data.

Use ghttpStreamEx (see page 66) for extra optional parameters.

See Also

ghttpGet (see page 68), ghttpSave (see page 62), ghttpStreamEx (see page 66), ghttpHead (see page 71), ghttpPost (see page 73)

ghttpStreamEx Function

Summary

Make an HTTP (see page 58) GET request and stream in the response without saving it in memory.

C++

```
COMMON_API GHTTPRequest ghttpStreamEx(
    const gsi_char * URL,
    const gsi_char * headers,
    GHTTPPost post,
    GHTTPBool throttle,
    GHTTPBool blocking,
    ghttpProgressCallback progressCallback,
    ghttpCompletedCallback completedCallback,
    void * param
);
```

Parameters

Parameters	Description
const gsi_char * URL	[in] URL
const gsi_char * headers	[in] Optional headers to pass with the request. Can be NULL or "".
GHTTPPost post	[in] Optional data to be posted. Can be NULL.
GHTTPBool throttle	[in] If true, throttle this connection's download speed.
GHTTPBool blocking	[in] If true, this call doesn't return until the file has finished streaming.
ghttpProgressCallback progressCallback	[in] Called whenever new data is received. Can be NULL.
ghttpCompletedCallback completedCallback	[in] Called when the file has finished streaming. Can be NULL.
void * param	[in] Optional free-format user data to send to the callback.

Returns

If less than 0, the request failed and this is a GHTTPRequestError value. Otherwise it identifies the request.

Remarks

This function is used to stream in the contents of a web page. The response body is not stored in memory or to disk. It is only passed to the progressCallback as it is received, and the application can do what it wants with the data.

Use ghttpStreamEx for extra optional parameters.

See Also

ghttpGetEx (see page 68), ghttpSaveEx (see page 62), ghttpStream (see page 65), ghttpHeadEx (see page 72), ghttpPostEx (see page 60)

ghttpThink Function

Summary

Processes all current HTTP (see page 58) requests.

C++

```
COMMON_API void ghttpThink();
```

Remarks

Any application that uses GHTTP in non-blocking mode (i.e., that calls ghttp functions with the blocking argument set to GHTTPFalse) needs to call ghttpThink to let the library do any necessary processing. Non-blocking mode should be use as much as possible.

This call will process any current requests and call any callbacks if necessary. It should typically be called as part of the application's main loop. While it can be called as seldom as a few times a second, it should be called closer to 10-20 times a second. If downloading larger files, it may be desirable to call it even more often to ensure that incoming buffers are emptied to make room for more incoming data.

Threads note: Making GHTTP requests concurrently from multiple threads is currently only supported on Windows. When using GHTTP from multiple threads, instead of calling ghttpThink, use ghttpRequestThink (see page 61) for each individual request. This allows that request's callback to be called from within the same thread in which it was started.

See Also

ghttpRequestThink (see page 61)

ghttpThrottleSettings Function

Summary

Used to adjust the throttle settings.

C++

```
COMMON_API void ghttpThrottleSettings(
    int bufferSize,
    gsi_time timeDelay
);
```

Parameters

Parameters	Description
int bufferSize	[in] The number of bytes to get each receive.
gsi_time timeDelay	[in] How often to receive data, in milliseconds.

Remarks

The throttle settings affect any request initiated with throttling, or for which throttling is enabled with ghttpSetThrottle (see page 64).

See Also

ghttpSetThrottle (see page 64)

ghttpFreePost Function

Summary

Release the specified post object.

C++

```
COMMON_API void ghttpFreePost(
    GHTTPPost post
);
```

Parameters

Parameters	Description
GHTTPPost post	[in] Post object created with ghttpNewPost (⧉ see page 72).

Remarks

By default, post objects created with ghttpNewPost (⧉ see page 72) will be automatically freed after being used in a request. However, ghttpPostSetAutoFree (⧉ see page 61) can be used to turn off the post object's auto-free property. This can be useful if a single post object will be used in multiple requests.

You should then use this function to manually free the post object after the last request it has been used in completes.

See Also

ghttpNewPost (⧉ see page 72), ghttpPostSetAutoFree (⧉ see page 61)

ghttpGet Function

Summary

Make an HTTP (⧉ see page 58) GET request and save the response to memory.

C++

```
COMMON_API GHTTPRequest ghttpGet(
    const gsi_char * URL,
    GHTTPBool blocking,
    ghttpCompletedCallback completedCallback,
    void * param
);
```

Parameters

Parameters	Description
const gsi_char * URL	[in] URL
GHTTPBool blocking	[in] If true, this call doesn't return until the file has been received.
ghttpCompletedCallback completedCallback	[in] Called when the file has been received.
void * param	[in] Optional free-format user data for use by the callback.

Returns

If less than 0, the request failed and this is a GHTTPRequestError value. Otherwise it identifies the request.

Remarks

This function is used to download the contents of a web page to memory. The application can provide the memory by supplying a buffer to this function, or the SDK can be allocate the memory internally.

Use ghttpGetEx (⧉ see page 68) for extra optional parameters.

See Also

ghttpGetEx (⧉ see page 68), ghttpSave (⧉ see page 62), ghttpStream (⧉ see page 65), ghttpHead (⧉ see page 71), ghttpPost (⧉ see page 73)

ghttpGetEx Function

Summary

Make an HTTP (⧉ see page 58) GET request and save the response to memory.

C++

```
COMMON_API GHTTPRequest ghttpGetEx(
    const gsi_char * URL,
    const gsi_char * headers,
    char * buffer,
    int bufferSize,
    GHTTPPost post,
    GHTTPBool throttle,
    GHTTPBool blocking,
    ghttpProgressCallback progressCallback,
    ghttpCompletedCallback completedCallback,
    void * param
);
```

Parameters

Parameters	Description
const gsi_char * URL	[in] URL
const gsi_char * headers	[in] Optional headers to pass with the request. Can be NULL or "".
char * buffer	[in] Optional user-supplied buffer. Set to NULL to have one allocated. Must be (size+1) to allow null terminating character.
int bufferSize	[in] The size of the user-supplied buffer in bytes. 0 if buffer is NULL.
GHTTPPost post	[in] Optional data to be posted. Can be NULL.
GHTTPBool throttle	[in] If true, throttle this connection's download speed.
GHTTPBool blocking	[in] If true, this call doesn't return until the file has been received.
ghttpProgressCallback progressCallback	[in] Called periodically with progress updates. Can be NULL.
ghttpCompletedCallback completedCallback	[in] Called when the file has been received. Can be NULL.
void * param	[in] Optional free-format user data to send to the callback.

Returns

If less than 0, the request failed and this is a GHTTPRequestError value. Otherwise it identifies the request.

Remarks

This function is used to download the contents of a web page to memory. The application can provide the memory by supplying a buffer to this function, or the SDK can be allocate the memory internally. Use ghttpGet (see page 68) for a simpler version of this function.

See Also

ghttpGet (see page 68), ghttpSaveEx (see page 62), ghttpStreamEx (see page 66), ghttpHeadEx (see page 72), ghttpPostEx (see page 60)

ghttpGetHeaders Function

Summary

Get the response headers from an HTTP (see page 58) request.

C++

```
COMMON_API const char * ghttpGetHeaders(
    GHTTPRequest request
);
```

Parameters

Parameters	Description
GHTTPRequest request	[in] A valid request object

Returns

The headers returned in the response.

Remarks

Only valid if the request's state is GHTTPReceivingHeaders.

See Also

ghttpGetState (see page 70)

ghttpGetResponseStatus Function

Summary

Get an HTTP (see page 58) response's status string and status code.

C++

```
COMMON_API const char * ghttpGetResponseStatus(
    GHTTPRequest request,
    int * statusCode
);
```

Parameters

Parameters	Description
GHTTPRequest request	[in] A valid request object
int * statusCode	[out] Status code.

Returns

The response's status string.

Remarks

Can only be used if the state has passed GHTTPReceivingStatus. The status string is a user-readable representation of the result of the request.

The status code is a 3 digit number which can be used to get more details on the result of the request.

There are 5 possible values for the first digit:

1xx: Informational

2xx: Success

3xx: Redirection

4xx: Client Error

5xx: Server Error

See RFC2616 (HTTP (see page 58) 1.1) and any follow-up RFCs for more information on specific codes.

See Also

ghttpGetState (see page 70)

ghttpGetState Function

Summary

Obtain the current state of an HTTP (see page 58) request.

C++

```
COMMON_API GHTTPState ghttpGetState(
    GHTTPRequest request
);
```

Parameters

Parameters	Description
GHTTPRequest request	[in] A valid request object

Returns

The state of an HTTP (see page 58) request.

ghttpGetURL Function

Summary

Used to obtain the URL associated with an HTTP (see page 58) request.

C++

```
COMMON_API const char * ghttpGetURL(
    GHTTPRequest request
);
```

Parameters

Parameters	Description
GHTTPRequest request	[in] A valid request object

Returns

The URL associated with the request.

Remarks

If the request has been redirected, this function will return the new URL, not the original URL.

ghttpHead Function

Summary

Make an HTTP (see page 58) HEAD request, which will only retrieve the response headers and not the normal response body.

C++

```
COMMON_API GHTTPRequest ghttpHead(
    const gsi_char * URL,
    GHTTPBool blocking,
    ghttpCompletedCallback completedCallback,
    void * param
);
```

Parameters

Parameters	Description
const gsi_char * URL	[in] URL
GHTTPBool blocking	[in] If true, this call doesn't return until finished.
ghttpCompletedCallback completedCallback	[in] Called when the request has finished.
void * param	[in] Optional free-format user data to send to the callback.

Returns

If less than 0, the request failed and this is a GHTTPRequestError value. Otherwise, it identifies the request.

Remarks

This function is similar to ghttpGet (see page 68), except it only gets the response headers.

This is done by making an HEAD request instead of a GET request, which instructs the HTTP (see page 58) server to leave the body out of the response.

Use ghttpHeadEx (see page 72) for extra optional parameters.

See Also

ghttpGet (see page 68), ghttpSave (see page 62), ghttpStream (see page 65), ghttpHeadEx (see page 72), ghttpPost (see page 73)

ghttpHeadEx Function

Summary

Make an HTTP (see page 58) HEAD request, which will only retrieve the response headers and not the normal response body.

C++

```
COMMON_API GHTTPRequest ghttpHeadEx(
    const gsi_char * URL,
    const gsi_char * headers,
    GHTTPBool throttle,
    GHTTPBool blocking,
    ghttpProgressCallback progressCallback,
    ghttpCompletedCallback completedCallback,
    void * param
);
```

Parameters

Parameters	Description
const gsi_char * URL	[in] URL
const gsi_char * headers	[in] Optional headers to pass with the request. Can be NULL or "".
GHTTPBool throttle	[in] If true, throttle this connection's download speed.
GHTTPBool blocking	[in] If true, this call doesn't return until finished.
ghttpProgressCallback progressCallback	[in] Called whenever new data is received. Can be NULL.
ghttpCompletedCallback completedCallback	[in] Called when the request has finished. Can be NULL.
void * param	[in] Optional free-format user data to send to the callback.

Returns

If less than 0, the request failed and this is a GHTTPRequestError value. Otherwise, it identifies the request.

Remarks

This function is similar to ghttpGetEx (see page 68), except it only gets the response headers.

This is done by making an HEAD request instead of a GET request, which instructs the HTTP (see page 58) server to leave the body out of the response.

Use ghttpHead (see page 71) for a simpler version of this function.

See Also

ghttpGetEx (see page 68), ghttpSaveEx (see page 62), ghttpStreamEx (see page 66), ghttpHead (see page 71), ghttpPostEx (see page 60)

ghttpNewPost Function

Summary

Creates a new post object, which is used to represent data to send a web server as part of an HTTP (see page 58) request.

C++

```
COMMON_API GHTTPPost ghttpNewPost();
```

Returns

The newly created post object, or NULL if it cannot be created.

Remarks

After getting the post object, use the ghttpPostAdd*() functions to add data to the object, and ghttpPostSetCallback (see page 61)() to add a callback to monitor the progress of the data upload.

By default, post objects automatically free themselves after posting. To use the same post with more than one request, set auto-free to false, then use ghttpFreePost (see page 67) to free it after all requests it's being used for are completed.

See Also

ghttpPostAddString (see page 75), ghttpPostAddFileFromDisk (see page 74), ghttpPostAddFileFromMemory (see page 74), ghttpPostSetAutoFree (see page 61), ghttpFreePost (see page 67), ghttpPostSetCallback (see page 61)

ghttpPost Function

Summary

Do an HTTP (see page 58) POST, which can be used to upload data to a web server.

C++

```
GHTTPRequest ghttpPost(
    const gsi_char * URL,
    GHTTPPost post,
    GHTTPBool blocking,
    ghttpCompletedCallback completedCallback,
    void * param
);
```

Parameters

Parameters	Description
const gsi_char * URL	[in] URL
GHTTPPost post	[in] The data to be posted.
GHTTPBool blocking	[in] If true, this call doesn't return until finished.
ghttpCompletedCallback completedCallback	[in] Called when the file has finished streaming. Can be NULL.
void * param	[in] Optional free-format user data to send to the callback.

Returns

If less than 0, the request failed and this is a GHTTPRequestError value. Otherwise it identifies the request.

Remarks

This function is used to post data to a web page, ignoring any possible response body sent by the server (response status and response headers can still be checked). If you want to post data and receive a response, use ghttpGetEx (see page 68), ghttpSaveEx (see page 62), or ghttpStreamEx (see page 66).

Use ghttpPostEx (see page 60) for extra optional parameters.

Use this function to do an HTTP (see page 58) Post, don't try to access a GHTTPPost object directly.

See Also

ghttpGet (see page 68), ghttpGetEx (see page 68), ghttpSave (see page 62), ghttpSaveEx (see page 62), ghttpStream (see page 65), ghttpStreamEx (see page 66), ghttpHead (see page 71), ghttpPostEx (see page 60)

ghttpPostAddFileFromDisk Function

Summary

Adds a disk file to the specified post object.

C++

```
COMMON_API GHTTPBool ghttpPostAddFileFromDisk(
    GHTTPPost post,
    const gsi_char * name,
    const gsi_char * filename,
    const gsi_char * reportFilename,
    const gsi_char * contentType
);
```

Parameters

Parameters	Description
GHTTPPost post	[in] Post object
const gsi_char * name	[in] The name to attach to this file.
const gsi_char * filename	[in] The name (and possibly path) to the file to upload.
const gsi_char * reportFilename	[in] The filename given to the web server.
const gsi_char * contentType	[in] The MIME type for this file.

Returns

GHTTPTrue if the file was added successfully.

Remarks

The reportFilename is what is reported to the server as the filename. If NULL or empty, the filename will be used (including any possible path). The contentType is the MIME type to report for this file. If NULL, "application/octet-stream" is used.

The file isn't read from until the data is actually sent to the server. When uploading files the content type of the overall request (as opposed to the content this of this file) will be "multipart/form-data".

Use this function to do an HTTP (see page 58) Post, don't try to access a GHTTPPost object directly.

See Also

ghttpNewPost (see page 72), ghttpPost (see page 73), ghttpPostAddString (see page 75), ghttpPostAddFileFromMemory (see page 74)

ghttpPostAddFileFromMemory Function

Summary

Adds a file from memory to the specified post object.

C++

```
COMMON_API GHTTPBool ghttpPostAddFileFromMemory(
    GHTTPPost post,
    const gsi_char * name,
    const char * buffer,
    int bufferLen,
    const gsi_char * reportFilename,
    const gsi_char * contentType
);
```

Parameters

Parameters	Description
GHTTPPost post	[in] Post object
const gsi_char * name	[in] The name to attach to this file.
const char * buffer	[in] The data to send.

int bufferLen	[in] The number of bytes of data to send.
const gsi_char * reportFilename	[in] The filename given to the web server.
const gsi_char * contentType	[in] The MIME type for this file.

Returns

GHTTPTrue if the file was added successfully.

Remarks

The reportFilename is what is reported to the server as the filename. It cannot be NULL or empty.

The contentType is the MIME type to report for this file. If NULL, "application/octet-stream" is used.

The data is not copied off in this call. The data pointer is read from as the data is actually sent to the server. The pointer must remain valid during requests. When uploading files the content type of the overall request (as opposed to the content this of this file) will be "multipart/form-data".

Use this function to do an HTTP (see page 58) Post, don't try to access a GHTTPPost object directly.

See Also

ghttpNewPost (see page 72), ghttpPost (see page 73), ghttpPostAddFileFromDisk (see page 74), ghttpPostAddString (see page 75)

ghttpPostAddString Function

Summary

Adds a string to the specified post object.

C++

```
COMMON_API GHTTPBool ghttpPostAddString(
    GHTTPPost post,
    const gsi_char * name,
    const gsi_char * string
);
```

Parameters

Parameters	Description
GHTTPPost post	[in] Post object
const gsi_char * name	[in] The name to attach to this string.
const gsi_char * string	[in] The string to send.

Returns

GHTTPTrue if the string was added successfully.

Remarks

If a post object only contains string, the content type for the upload will be the "application/x-www-form/urlencoded". If any files are added, the content type for the upload will become "multipart/form-data".

Use this function to do an HTTP (see page 58) Post, don't try to access a GHTTPPost object directly.

See Also

ghttpNewPost (see page 72), ghttpPost (see page 73), ghttpPostAddFileFromDisk (see page 74), ghttpPostAddFileFromMemory (see page 74)

ghttpCancelRequest Function

Summary

Cancel an HTTP (see page 58) request in progress.

C++

```
COMMON_API void ghttpCancelRequest(
    GHTTPRequest request
);
```

Parameters

Parameters	Description
GHTTPRequest request	[in] A valid GHTTPRequest (see page 80) object.

Remarks

The GHTTPRequest (see page 80) should not be referenced once this function returns.

ghttpCleanup Function

Summary

Clean up and close down the HTTP (see page 58) SDK. Free internally allocated memory.

C++

```
COMMON_API void ghttpCleanup();
```

Remarks

One call to ghttpCleanup should be made for each call to ghttpStartup (see page 65).

See Also

ghttpStartup (see page 65)

ghttpCleanupRootCAList Function

Summary

This function is used resetting the root certificate list.

C++

```
GHTTPBool ghttpCleanupRootCAList(
    char * url
);
```

Parameters

Parameters	Description
char * url	[in] url used to initialized the certificate.

Returns

GHTTPTrue if the rootCA reset successfully.

Remarks

Currently, it is ONLY used by TWL applications. Others will return failure (GHTTPFalse).

ghttpFreePostAndUpdateConnection Function

Summary

Free a post object.

C++

```
COMMON_API void ghttpFreePostAndUpdateConnection(
    GHTTPRequest requestId,
    GHTTPPost post
);
```

Parameters

Parameters	Description
GHTTPPost post	[in] Post object created with ghttpNewPost (see page 72).

Remarks

By default, post objects created with ghttpNewPost (see page 72) will be automatically freed after being used in a request. However ghttpPostSetAutoFree (see page 61) can be used to turn off the post object's auto-free property. This can be useful if a single post object will be used in multiple requests. You should then use this function to manually free the post object after the last request it has been used in completes.

See Also

ghttpNewPost (see page 72), ghttpPostSetAutoFree (see page 61)

ghttpResultString Function

Summary

Given a GHTTPResult, returns a meaningful character string describing the result.

C++

```
COMMON_API const char* ghttpResultString(
    int result
);
```

Parameters

Parameters	Description
int result	[in] A GHTTPResult value.

Returns

A nul-byte terminated character string describing the HTTP (see page 58) result corresponding to the value given.

Remarks

The returned string is read-only and should not be modified.

See Also

ghttpGetEx (see page 68), ghttpSaveEx (see page 62), ghttpStream (see page 65), ghttpStreamEx (see page 66), ghttpHeadEx (see page 72), ghttpPostEx (see page 60)

ghttpSetRootCAList Function

Summary

This function is used for setting the root certificate list when making https calls, if the SSL Engine used supports the setting of expected server root certificate from validation purposes.

C++

```
GHTTPBool ghttpSetRootCAList(
    char * url,
    void * theRootCA
);
```

Parameters

Parameters	Description
char * url	[in]
void * theRootCA	[in] A void pointer to the certificate data.

Returns

GHTTPTrue if the rootCA set successfully.

Remarks

Currently, it is ONLY used by TWL applications. Others will return failure (GHTTPFalse). The application must allocate/de-allocate the memory passed as theRootCA. This pointer must be valid until the application has completed.

Callbacks

Types

Name	Description
ghttpCompletedCallback (see page 78)	Called when the entire file has been received.
ghttpPostCallback (see page 79)	Called during requests to let the app know how much of the post data has been uploaded.
ghttpProgressCallback (see page 79)	Called with updates on the current state of the request.

ghttpCompletedCallback Type

Summary

Called when the entire file has been received.

C++

```
typedef GHTTPBool (* ghttpCompletedCallback)(GHTTPRequest request, GHTTPResult result, char
* buffer, GHTTPByteCount bufferLen, char * headers, void * param);
```

Parameters

Parameters	Description
request	[in] A valid request object.
result	[in] The result (success or an error).
buffer	[in] The file's bytes (only valid if ghttpGetFile[Ex] was used).
bufferLen	[in] The file's length.
param	[in] Optional free-format user data for use by the callback.

Returns

If ghttpGetFile[Ex] was used, return true to have the buffer freed, false if the app will free the buffer.

Remarks

If ghttpStreamFileEx or ghttpSaveFile[Ex] was used, buffer is NULL, bufferLen is the number of bytes in the file, and the return value is ignored.

If ghttpGetFile[Ex] was used, return true to have the buffer freed, false if the app will free the buffer. If true, the buffer cannot be accessed once the callback returns. If false, the app can use the buffer even after this call returns, but must free it at some later point. There will always be a file, even if there was an error, although for errors it may be an empty file.

The routine ghttpCompletedCallback only returns GHTTPFalse when it is necessary to save the buffer that was passed to the callback for later use. Otherwise (e.g., when the buffer is no longer need or has been copied with the callback), it returns GHTTPTrue.

The GHTTP SDK will free the buffer if the callback returns GHTTPTrue. Overuse of GHTTPFalse can lead to memeory leaks.

See Also

ghttpGet (see page 68), ghttpGetEx (see page 68), ghttpSave (see page 62), ghttpSaveEx (see page 62), ghttpStream (see page 65), ghttpStreamEx (see page 66), ghttpHead (see page 71), ghttpHeadEx (see page 72), ghttpPost (see page 73), ghttpPostEx (see page 60)

ghttpPostCallback Type

Summary

Called during requests to let the app know how much of the post data has been uploaded.

C++

```
typedef void (* ghttpPostCallback)(GHTTPRequest request, int bytesPosted, int totalBytes,
int objectsPosted, int totalObjects, void * param);
```

Parameters

Parameters	Description
request	[in] A valid request object
bytesPosted	[in] The number of bytes of data posted so far.
totalBytes	[in] The total number of bytes being posted.
objectsPosted	[in] The total number of data objects uploaded so far.
totalObjects	[in] The total number of data objects to upload.
param	[in] Optional free-format user data for use by the callback

See Also

ghttpNewPost (see page 72), ghttpPostSetCallback (see page 61)

ghttpProgressCallback Type

Summary

Called with updates on the current state of the request.

C++

```
typedef void (* ghttpProgressCallback)(GHTTPRequest request, GHTTPState state, const char *
buffer, GHTTPByteCount bufferLen, GHTTPByteCount bytesReceived, GHTTPByteCount totalSize,
void * param);
```

Parameters

Parameters	Description
request	[in] A valid request object.
state	[in] The current state of the request.
buffer	[in] The file's bytes so far, NULL if state < GHTTPReceivingFile.
bufferLen	[in] The number of bytes in the buffer, 0 if state < GHTTPReceivingFile.
bytesReceived	[in] The total number of bytes received, 0 if state < GHTTPReceivingFile.
totalSize	[in] The total size of the file, -1 if unknown.
param	[in] Optional free-format user data to send to the callback.

Remarks

The buffer should not be accessed once this callback returns.

If ghttpGetFile[Ex] was used, the buffer contains all of the data that has been received so far, and bufferSize is the total number of bytes received.

If ghttpSaveFile[Ex] was used, the buffer only contains the most recent data that has been received. This same data is saved to the file. The buffer will not be valid after this callback returns.

If ghttpStreamFileEx was used, the buffer only contains the most recent data that has been received. This data will be lost once the callback returns, and should be copied if it needs to be saved. bufferSize is the number of bytes in the current block of data.

See Also

ghttpGetEx (see page 68), ghttpSaveEx (see page 62), ghttpStream (see page 65), ghttpStreamEx (see page 66), ghttpHeadEx (see page 72), ghttpPostEx (see page 60)

Enumerations

Types

Name	Description
GHTTPRequest (see page 80)	A type that represents an HTTP (see page 58) file request.

GHTTPRequest Type

Summary

A type that represents an HTTP (see page 58) file request.

C++

```
typedef int GHTTPRequest;
```

Nat Negotiation

API Documentation

Module

Nat Negotiation (see page 80)

Functions

Functions

	Name	Description
	NNBeginNegotiation (see page 80)	Starts the negotiation process.
	NNBeginNegotiationWithSocket (see page 81)	Starts the negotiation process using the socket provided, which will be shared with the game.
	NNCancel (see page 82)	Cancels a NAT Negotiation request in progress.
	NNFreeNegotiateList (see page 83)	De-allocates the memory used by for the negotiate list when you are done with NAT Negotiation.
	NNStartNatDetection (see page 83)	Starts the NAT detection process.
	NNThink (see page 83)	Processes any negotiation or NAT detection requests that are in progress.

NNBeginNegotiation Function

Summary

Starts the negotiation process.

C++

```
COMMON_API NegotiateError NNBeginNegotiation(
    int cookie,
    int clientindex,
    NegotiateProgressFunc progresscallback,
    NegotiateCompletedFunc completedcallback,
    void * userdata
);
```

Parameters

Parameters	Description
int cookie	[in] Shared cookie value that both players will use so that the NAT Negotiation Server can match them up.
int clientindex	[in] One client must use clientindex 0, the other must use clientindex 1.
NegotiateProgressFunc progresscallback	[in] Callback function that will be called as the state changes.
NegotiateCompletedFunc completedcallback	[in] Callback function that will be called when negotiation is complete.
void * userdata	[in] Pointer for your own use that will be passed into the callback functions.

Returns

ne_noerror if successful; otherwise one of the 'ne_' error values. See Remarks for detail.

Remarks

Possible errors that can be returned when starting a negotiation:

ne_noerror: No error.

ne_allocerror: Memory allocation failed.

ne_socketerror: Socket allocation failed.

ne_dnserror: DNS lookup failed.

This should only be performed when connecting to a server. It should not be used during server browsing as it will create unnecessary load on the GameSpy NAT service.

See Also

NNBeginNegotiationWithSocket (see page 81), NegotiateCompletedFunc (see page 84)

NNBeginNegotiationWithSocket Function

Summary

Starts the negotiation process using the socket provided, which will be shared with the game.

C++

```
COMMON_API NegotiateError NNBeginNegotiationWithSocket(
    SOCKET gamesocket,
    int cookie,
    int clientindex,
    NegotiateProgressFunc progresscallback,
    NegotiateCompletedFunc completedcallback,
    void * userdata
);
```

Parameters

Parameters	Description
SOCKET gamesocket	[in] The socket to be used to start the negotiation
int cookie	[in] Shared cookie value that both players will use so that the NAT Negotiation Server can match them up.
int clientindex	[in] One client must use clientindex 0, the other must use clientindex 1.
NegotiateProgressFunc progresscallback	[in] Callback function that will be called as the state changes.
NegotiateCompletedFunc completedcallback	[in] Callback function that will be called when negotiation is complete.
void * userdata	[in] Pointer for your own use that will be passed into the callback functions.

Returns

Possible errors that can be returned when starting a negotiation:

ne_noerror: No error.

ne_allocerror: Memory allocation failed.

ne_socketerror: Socket allocation failed.

ne_dnserror: DNS lookup failed.

This should only be performed when connecting to a server. It should not be used during server browsing as it will create unnecessary load on the GameSpy NAT service.

Remarks

Incoming traffic is not processed automatically; you will need to read the data off the socket and pass NN packets to NNProcessData.

This should only be performed when connecting to a server. It should not be used during server browsing as it will create unnecessary load on the GameSpy NAT service.

See Also

NNBeginNegotiation (see page 80)

NNCancel Function

Summary

Cancels a NAT Negotiation request in progress.

C++

```
COMMON_API void NNCancel(
    int cookie
);
```

Parameters

Parameters	Description
int cookie	[in] The cookie associated with this negotiation.

NNFreeNegotiateList Function

Summary

De-allocates the memory used by for the negotiate list when you are done with NAT Negotiation.

C++

```
COMMON_API void NNFreeNegotiateList();
```

Remarks

Once you have finished negotiating, the internal SDK memory must be freed using NNFreeNegotiatorList. Any outstanding negotiations will be cancel them. Calling this will NOT close the game sockets -- you are free to continue game communications.

NNStartNatDetection Function

Summary

Starts the NAT detection process.

C++

```
COMMON_API NegotiateError NNStartNatDetection(
    NatDetectionResultsFunc resultscallback
);
```

Parameters

Parameters	Description
NatDetectionResultsFunc resultscallback	[in] Callback function that will be called when NAT detection is complete.

Returns

ne_noerror if successful; otherwise one of the 'ne_' error values. See Remarks for detail.

Remarks

Possible errors that can be returned when starting a negotiation:

ne_noerror: No error.

ne_socketerror: Socket allocation failed.

ne_dnserror: DNS lookup failed.

See Also

NatDetectionResultsFunc (see page 84), NAT

NNThink Function

Summary

Processes any negotiation or NAT detection requests that are in progress.

C++

```
COMMON_API void NNThink();
```

Remarks

After you've begun a negotiation and/or NAT detection, you need to call the NNThink function on regular intervals (recommended: 100ms) to process the connection. You may call NNThink when no negotiations are in progress as well; it will simply return immediately.

Callbacks

Types

Name	Description
NatDetectionResultsFunc (see page 84)	The callback that gets executed from NNStartNatDetection (see page 83) when the detection is complete.
NegotiateCompletedFunc (see page 84)	The callback that gets executed from NNBeginNegotiation (see page 80) when negotiation is complete.
NegotiateProgressFunc (see page 85)	The callback that gets executed from NNBeginNegotiation (see page 80) as negotiation proceeds.

NatDetectionResultsFunc Type

Summary

The callback that gets executed from NNStartNatDetection Function (see page 83) when the detection is complete.

C++

```
typedef void (* NatDetectionResultsFunc)(gsi_bool success, NAT nat);
```

Parameters

Parameters	Description
success	[in] If gsi_true the NAT detection was successful.
nat	[in] When detection is successful, this contains the NAT device's properties.

Remarks

Once your detection callback function is called, check the success parameter.

If it is gsi_false, then the detection could not be completed and should be retried.

If it is gsi_true, then the NAT parameter will contain the properties of the detected NAT device.

See Also

NNStartNatDetection (see page 83), NAT

NegotiateCompletedFunc Type

Summary

The callback that gets executed from NNBeginNegotiation Function (see page 80) when negotiation is complete.

C++

```
typedef void (* NegotiateCompletedFunc)(NegotiateResult result, SOCKET gamesocket,
SOCKADDR_IN *remoteaddr, void *userdata);
```

Parameters

Parameters	Description
result	[in] Indicates the result of the negotiation attempt.
gamesocket	[in] The socket you should use to continue communications with the client.
remoteaddr	[in] The remote address and port you should use to communicate with the new client.
userdata	[in] Data for your own use.

Remarks

Once your completed function is called, you can begin sending data to the other client immediately using the socket and address provided.

Possible values for the value of the result parameter are:

nr_success: Successful negotiation, an open channel has now been established.

nr_deadbeatpartner: Partner did not register with the NAT Negotiation Server.

nr_inittimeout: Unable to communicate with NAT Negotiation Server.

nr_pingtimeout: Unable to communicate directly with partner.

nr_unknownerror: NAT Negotiation server indicated an unknown error condition.

If you used NNBeginNegotiationWithSocket (see page 81) then the socket parameter will be the socket you passed in originally. Otherwise, it will be a new socket allocated by the NAT Negotiation SDK.

Make sure you copy the remoteaddr structure before the callback returns.

This function needs to be handled properly to pass source code review.

See Also

NNBeginNegotiation (see page 80)

NegotiateProgressFunc Type

Summary

The callback that gets executed from NNBeginNegotiation Function (see page 80) as negotiation proceeds.

C++

```
typedef void (* NegotiateProgressFunc)(NegotiateState state, void *userdata);
```

Parameters

Parameters	Description
state	[in] The state of the negotiation at the time of notification.
userdata	[in] Data for your own use.

Remarks

The two times you will get a progress notification is when the NAT Negotiation server acknowledges your connection request (ns_initack), and when the guessed port data has been received from the NAT Negotiation server and direct negotiation with the other client is in progress (ns_connectping).

See Also

NNBeginNegotiation (see page 80)

Enumerations

Enumerations

Name	Description
NegotiateError (see page 86)	Possible error values that can be returned when starting a negotiation.
NegotiateResult (see page 86)	Possible results of the negotiation.
NegotiateState (see page 87)	Possible states for the SDK. The two you will be notified for are ns_initack and ns_connectping.

NegotiateError Enumeration

Summary

Possible error values that can be returned when starting a negotiation.

C++

```
typedef enum {
  ne_noerror,
  ne_allocerror,
  ne_socketerror,
  ne_dnserror
} NegotiateError;
```

Members

Members	Description
ne_noerror	No error.
ne_allocerror	Memory allocation failed.
ne_socketerror	Socket allocation failed.
ne_dnserror	DNS lookup failed.

See Also

NNBeginNegotiation (see page 80)

NegotiateResult Enumeration

Summary

Possible results of the negotiation.

C++

```
typedef enum {
  nr_success,
  nr_deadbeatpartner,
  nr_inittimeout,
  nr_pingtimeout,
  nr_unknownerror,
  nr_noresult
} NegotiateResult;
```

Members

Members	Description
nr_success	Successful negotiation, other parameters can be used to continue communications with the client.
nr_deadbeatpartner	Partner did not register with the NAT Negotiation Server.
nr_inittimeout	Unable to communicate with NAT Negotiation Server.
nr_pingtimeout	Unable to communicate with partner.
nr_unknownerror	The NAT Negotiation server indicated an unknown error condition.
nr_noresult	Initial negotiation status before a result is determined.

See Also

NegotiateCompletedFunc (see page 84)

NegotiateState Enumeration

Summary

Possible states for the SDK. The two you will be notified for are ns_initack and ns_connectping.

C++

```
typedef enum {
  ns_preinitsent,
  ns_preinitack,
  ns_initsent,
  ns_initack,
  ns_connectping,
  ns_finished,
  ns_canceled,
  ns_reportsent,
  ns_reportack
} NegotiateState;
```

Members

Members	Description
ns_preinitack	When the NAT Negotiation server acknowledges your connection.
ns_initsent	Initial connection request has been sent to the server (internal).
ns_initack	The NAT Negotiation server has acknowledged your connection request.
ns_connectping	Direct negotiation with the other client has started.
ns_finished	The negotiation process has completed (internal).
ns_canceled	The negotiation process has been canceled (internal).
ns_reportsent	The negotiation result report has been sent to the server (internal).
ns_reportack	The NAT Negotiation server has acknowledged your result report (internal).

See Also

NegotiateProgressFunc (see page 85)

Presence and Messaging

API Documentation

Module

Presence and Messaging (see page 87)

Functions

Connection Management

Functions

	Name	Description
◆	gpConnect (see page 88)	This function is used to establish a connection to the server. It establishes a connection with an existing profile, which is identified based on the nick and email and is validated by the password.
◆	gpConnectPreAuthenticated (see page 89)	This function is used to establish a connection to the server. It establishes a connection using an authtoken and a partnerchallenge, both obtained from a partner authentication system.
◆	gpConnectUniqueNick (see page 90)	This function is used to establish a connection to the server. It establishes a connection with an existing profile, which is identified based on the uniquenick and is validated by the password.
◆	gpDestroy (see page 91)	This function is used to destroy a connection object.
◆	gpDisconnect (see page 91)	This function is used to establish a connection to the server. It establishes a connection with an existing profile, which is identified based on the uniquenick and is validated by the password.
◆	gpInitialize (see page 92)	This function is used to initialize a connection object.
◆	gpDisable (see page 92)	This function disables a certain state.
◆	gpEnable (see page 93)	This function enables a certain state.
◆	gpGetLoginTicket (see page 93)	Retrieves a connection "token" that may be used by HTTP (see page 58) requests to uniquely identify the player.
◆	gpProcess (see page 94)	This function checks for incoming callback responses from the GP (see page 87) backend and does some necessary processing to maintain an active GPConnection (see page 161). It should be called frequently to maintain GP (see page 87) responsiveness.
◆	gpIsConnected (see page 94)	Determine whether the GPConnection (see page 161) object has established a connection with the server.
◆	gpConnectLoginTicket (see page 95)	This function is used to establish a connection to the server. It establishes a connection with a login ticket passed from some other GameSpy system or SDK.

gpConnect Function

Summary

This function is used to establish a connection to the server. It establishes a connection with an existing profile, which is identified based on the nick and email and is validated by the password.

C++

```
COMMON_API GPResult gpConnect(
    GPConnection * connection,
    const gsi_char nick[GP_NICK_LEN],
    const gsi_char email[GP_EMAIL_LEN],
    const gsi_char password[GP_PASSWORD_LEN],
    GPEnum firewall,
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
const gsi_char nick[GP_NICK_LEN]	[in] The profile nickname.
const gsi_char email[GP_EMAIL_LEN]	[in] The profile email address.
const gsi_char password[GP_PASSWORD_LEN]	[in] The profile password.

GPEnum firewall	[in] GP_FIREWALL or GP_NO_FIREWALL. This option may limit the users ability to transfer files.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING.
GPCallback callback	[in] A user-supplied callback with an arg type of GPConnectResponseArg (see page 140).
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function establishes a connection with the server. If the local machine is behind a firewall, the firewall parameter should be set to GP_FIREWALL so that buddy messages are sent through the server.

gpDisconnect (see page 91) should be called when this connection is ready to be disconnected. When the connection is complete, the callback will be called.

If the user is not given the option of selecting whether or not they're behind a firewall, GP_NO_FIREWALL is passed for the "firewall" parameter.

The SDK will fall back to firewall support if needed. All buddy messages will travel through the server even if a user is not behind a firewall.

If the user is only connecting to GP (see page 87) to get a loginTicket to use with other services (e.g., Sake (see page 176)), it is important that GP (see page 87) is disconnected immediately after the loginTicket is acquired. This is important because it ensures that the user will not mistakenly receive friend requests, messages, or game invites when the game doesn't support any way to handle such items.

Notes

gpConnectW and gpConnectA are UNICODE and ANSI mapped versions of gpConnect. The arguments of gpConnectA are ANSI strings; those of gpConnectW are wide-character strings.

See Also

GPConnectResponseArg (see page 140)

gpConnectPreAuthenticated Function

Summary

This function is used to establish a connection to the server. It establishes a connection using an authtoken and a partnerchallenge, both obtained from a partner authentication system.

C++

```
COMMON_API GPResult gpConnectPreAuthenticated(
    GPConnection * connection,
    const gsi_char authtoken[GP_AUTHTOKEN_LEN],
    const gsi_char partnerchallenge[GP_PARTNERCHALLENGE_LEN],
    GPEnum firewall,
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
const gsi_char authtoken[GP_AUTHTOKEN_LEN]	[in] An authentication token generated by a partner database.
const gsi_char partnerchallenge[GP_PARTNERCHALLENGE_LEN]	[in] The challenge received from the partner database.

GPEnum firewall	[in] GP_FIREWALL or GO_NO_FIREWALL. This option may limit the user's ability to send files.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A user-supplied callback with an arg type of GPConnectResponseArg (see page 140)
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function establishes a connection with the server. gpDisconnect (see page 91) should be called when this connection is ready to be disconnected. When the connection is complete, the callback will be called.

The namespaceID & partnerID parameters passed to gpInitialize (see page 92) will be overwritten in the SDK to their correct values (based on the authtoken/partnerchallenge used) after the callback is called.

Notes

gpConnectPreAuthenticatedW and gpConnectPreAuthenticatedA are UNICODE and ANSI mapped versions of gpConnectPreAuthenticated. The arguments of gpConnectPreAuthenticatedA are ANSI strings; those of gpConnectPreAuthenticatedW are wide-character strings.

See Also

GPConnectResponseArg (see page 140)

gpConnectUniqueNick Function

Summary

This function is used to establish a connection to the server. It establishes a connection with an existing profile, which is identified based on the uniquenick and is validated by the password.

C++

```
COMMON_API GPResult gpConnectUniqueNick(
    GPConnection * connection,
    const gsi_char uniquenick[GP_UNIQUENICK_LEN],
    const gsi_char password[GP_PASSWORD_LEN],
    GPEnum firewall,
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
const gsi_char uniquenick[GP_UNIQUENICK_LEN]	[in] The uniquenick.
const gsi_char password[GP_PASSWORD_LEN]	[in] The profile password.
GPEnum firewall	[in] GP_FIREWALL or GO_NO_FIREWALL. This option may limit the user's ability to send files.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A user-supplied callback with an arg type of GPConnectResponseArg (see page 140).
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function establishes a connection with the server. If the local machine is behind a firewall, the firewall parameter should be set to GP_FIREWALL so that buddy messages are sent through the server.

gpDisconnect (see page 91) should be called when this connection is ready to be disconnected. When the connection is complete, the callback will be called.

Notes

gpConnectUniqueNickW and gpConnectUniqueNickA are UNICODE and ANSI mapped versions of gpConnectUniqueNick. The arguments of gpConnectUniqueNickA are ANSI strings; those of gpConnectUniqueNickW are wide-character strings.

See Also

GPConnectResponseArg (see page 140)

gpDestroy Function

Summary

This function is used to destroy a connection object.

C++

```
void gpDestroy(
    GPConnection * connection
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).

Remarks

This function destroys a connection object. This should be called when a GPConnection (see page 161) object is no longer needed. The object cannot be used after it has been destroyed.

gpDisconnect Function

Summary

This function is used to establish a connection to the server. It establishes a connection with an existing profile, which is identified based on the uniquenick and is validated by the password.

C++

```
COMMON_API void gpDisconnect(
    GPConnection * connection
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).

Remarks

This function should be called to disconnect a connection when it is no longer needed. After this call, connection can be reused for a new connection.

See Also

gpDestroy (see page 91)

gpInitialize Function

Summary

This function is used to initialize a connection object.

C++

```
COMMON_API GPResult gpInitialize(
    GPConnection * connection,
    int productID,
    int namespaceID,
    int partnerID
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GPConnection (see page 161) object.
int productID	[in] The application's product ID.
int namespaceID	[in] The application's namespace ID. This is typically set to the value defined by GSI_NAMESPACE_GAMESPY_DEFAULT.
int partnerID	[in] The application's partner ID. This is typically set to the value defined by GSI_PARTNERID_GAMESPY_DEFAULT.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function initializes a connection object. As long as there are no errors, this object should stay valid until gpDestroy (see page 91) is called. After the object is initialized by this function, callbacks can be set for the connection using gpSetCallback (see page 136) and states such as info-caching can be turned on using gpEnable (see page 93).

Use GSI_NAMESPACE_GAMESPY_DEFAULT as the namespaceID for most normal use. Use namespace 0 for no namespace.

Use GP_PARTNERID_GAMESPY_DEFAULT as the partnerID for most normal use.

See Also

gpSetCallback (see page 136), gpEnable (see page 93), gpDisable (see page 92), gpDestroy (see page 91)

gpDisable Function

Summary

This function disables a certain state.

C++

```
COMMON_API GPResult gpDisable(
    GPConnection * connection,
    GPEnum state
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPEnum state	[in] A GPEnum (see page 151) value representing the state to enable.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to disable ("turn off") states in the connection once gpInitialize (see page 92) has been completed successfully. To enable a state use gpEnable (see page 93). See gpEnable (see page 93) or GPEnum (see page 151) for the available states.

See Also

gpInitialize (see page 92), gpEnable (see page 93)

gpEnable Function

Summary

This function enables a certain state.

C++

```
COMMON_API GPResult gpEnable(
    GPConnection * connection,
    GPEnum state
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPEnum state	[in] A GPEnum (see page 151) value representing the state to enable.

Remarks

This function is used to enable ("turn on") states in the connection once gpInitialize (see page 92) has been completed successfully. To disable a state use gpDisable (see page 92). The following states are available:

GP_INFO_CACHING_BUDDY_AND_BLOCK_ONLY: Buddy and blocked list info caching caches information queried about other users' profiles when they are on the local profile's buddy list or blocked list, potentially improving profile query performance. This is the recommended mode.

GP_INFO_CACHING: General info caching caches information queried about all other users' profiles, potentially improving profile query performance.

GP_SIMULATION: Simulation mode goes through all GameSpy calls until the network layer is reached, but does not actually make the underlying network calls. This can be useful for testing code without hitting the GameSpy backend services.

GP_NP_SYNC: *PS3 only* The NP to GP (see page 87) friend sync is enabled by default and should only be disabled temporarily when using other NP functionality that may cause contention (then re-enabled immediately afterward).

See Also

gpInitialize (see page 92), gpDisable (see page 92)

gpGetLoginTicket Function

Summary

Retrieves a connection "token" that may be used by HTTP (see page 58) requests to uniquely identify the player.

C++

```
COMMON_API GPResult gpGetLoginTicket(
    GPConnection * connection,
    char loginTicket[GP_LOGIN_TICKET_LEN]
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
char loginTicket[GP_LOGIN_TICKET_LEN]	[out] The login ticket.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

Retrieves a connection "token" that may be used by HTTP (see page 58) requests to uniquely identify the player.

gpProcess Function

Summary

This function checks for incoming callback responses from the Presence and Messaging (see page 87) backend and does some necessary processing to maintain an active GPConnection Type (see page 161). It should be called frequently to maintain Presence and Messaging (see page 87) responsiveness.

C++

```
COMMON_API GPResult gpProcess(
    GPConnection * connection
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function does any necessary processing that needs to be done in the connection. This includes checking for buddy messages, checking for buddy status changes, and completing any non-blocking operations. This function should be called frequently, typically in the application's main loop. If an operation is finished during a call to this function, gpProcess will call that operation's registered callback function.

gpIsConnected Function

Summary

Determine whether the GPConnection Type (see page 161) object has established a connection with the server.

C++

```
COMMON_API GPResult gpIsConnected(
    GPConnection * connection,
    GPEnum * connected
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPEnum * connected	[out] The connected state. GP_CONNECTED or GP_NOT_CONNECTED. (See remarks.)

Returns

This function returns GP_NO_ERROR on success. Otherwise, a valid GPResult (see page 160) is returned.

Remarks

If the connection parameter has not been initialized with gpInitialize (see page 92), the connected parameter will be invalid and the return value will be GP_PARAMETER_ERROR.

gpConnectLoginTicket Function

Summary

This function is used to establish a connection to the server. It establishes a connection with a login ticket passed from some other GameSpy system or SDK.

C++

```
COMMON_API GPResult gpConnectLoginTicket(
    GPConnection * connection,
    const gsi_char loginticket[GP_LOGIN_TICKET_LEN],
    GPEnum firewall,
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
const gsi_char loginticket[GP_LOGIN_TICKET_LEN]	[in] The login ticket.
GPEnum firewall	[in] GP_FIREWALL or GP_NO_FIREWALL. This option may limit the user's ability to transfer files.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A user-supplied callback with an arg type of GPConnectResponseArg (see page 140).
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function establishes a connection with the server. If the local machine is behind a firewall, the firewall parameter should be set to GP_FIREWALL so that buddy messages are sent through the server.

gpDisconnect (see page 91) should be called when this connection is ready to be disconnected. When the connection is complete, the callback will be called.

Notes

gpConnectLoginTicketW and gpConnectLoginTicketA are UNICODE and ANSI mapped versions of gpConnectLoginTicket. The arguments of gpConnectLoginTicketA are ANSI strings; those of gpConnectLoginTicketW are wide-character strings.

See Also

GPConnectResponseArg (see page 140)

Buddies

Functions

	Name	Description
	gpAuthBuddyRequest (see page 96)	This function authorizes a buddy request. It is called in response to the gpRecvBuddyRequest callback getting called.
	gpDeleteBuddy (see page 97)	This function deletes a buddy from the local profile's buddy list.
	gpDenyBuddyRequest (see page 97)	This function denies a buddy request. It is called in response to the gpRecvBuddyRequest callback getting called.
	gpGetBuddyIndex (see page 98)	This function checks a remote profile to see if it is a buddy. If it is a buddy, the buddy's index is returned. If it is not a buddy, the index will be set to -1.
	gpGetBuddyStatus (see page 98)	This function gets the status for a particular buddy on the buddy list.
	gpGetNumBuddies (see page 99)	This function gets the number of buddies on the local profile's buddy list.
	gpGetProfileBuddyList (see page 99)	Get the buddies for a profile.
	gpGetReverseBuddies (see page 100)	Get profiles that have you on their buddy list.
	gpIsBuddy (see page 100)	Returns 1 if the given ProfileID is a buddy, 0 if not a buddy.
	gpIsBuddyConnectionOpen (see page 101)	Returns 1 if the given ProfileID is connected for direct peer-to-peer messaging. Returns 0 otherwise.
	gpRevokeBuddyAuthorization (see page 101)	Remove the local client from a remote users buddy list.
	gpSendBuddyMessage (see page 102)	This function sends a message to a buddy.
	gpSendBuddyRequest (see page 102)	This function sends a request to a remote profile to ask for permission to add the remote profile to the local profile's buddy list.
	gpSendBuddyUTM (see page 103)	Sends a UTM (under-the-table message) to a buddy.
	gpAddToBlockedList (see page 104)	Adds a remote profile to the local player's blocked list.
	gpGetBlockedProfile (see page 104)	This function gets the profileid for a particular player on the blocked list.
	gpGetNumBlocked (see page 105)	Gets the total number of blocked players in the local profile's blocked list.
	gpRemoveFromBlockedList (see page 105)	Removes a remote profile from the local player's blocked list.
	gpIsBlocked (see page 106)	Returns gsi_true if the given ProfileID is blocked, gsi_false if not blocked.
	gpInvitePlayer (see page 106)	This function invites a player to play a certain game.
	gpSetQuietMode (see page 107)	Turn on or off the flags that control what types of buddy messages the local profile will receive.
	gpGetReverseBuddiesList (see page 107)	Get a list of profiles that have the specified profiles as buddies.

gpAuthBuddyRequest Function

Summary

This function authorizes a buddy request. It is called in response to the gpRecvBuddyRequest callback getting called.

C++

```
COMMON_API GPResult gpAuthBuddyRequest(
    GPConnection * connection,
    GPProfile profile
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] The connection on which to authorize the request.
GPProfile profile	[in] The remote profile whose buddy request is being authorized.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to authorize a buddy request received with the gpRecvBuddyRequest callback. It is used only to authorize. This function does not need to be called immediately after a request has been received, however the request will be lost as soon as the local profile is disconnected. This function causes a status message to be sent to the remote profile.

gpDeleteBuddy Function

Summary

This function deletes a buddy from the local profile's buddy list.

C++

```
COMMON_API GPResult gpDeleteBuddy(
    GPConnection * connection,
    GPProfile profile
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection interface with an established connection.
GPProfile profile	[in] The profile ID of the buddy to delete.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function deletes the buddy indicated by profile from the local profile's buddy list.

gpDenyBuddyRequest Function

Summary

This function denies a buddy request. It is called in response to the gpRecvBuddyRequest callback getting called.

C++

```
COMMON_API GPResult gpDenyBuddyRequest(
    GPConnection * connection,
    GPProfile profile
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection interface with an established connection.
GPProfile profile	[in] The profile ID of the player who sent the AddBuddyRequest (i.e., the player you are denying).

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to deny a buddy request received with the gpRecvBuddyRequest callback. This function does not need to be called immediately after a request has been received. Nothing is sent to the remote profile letting them know the request was denied.

gpGetBuddyIndex Function

Summary

This function checks a remote profile to see if it is a buddy. If it is a buddy, the buddy's index is returned. If it is not a buddy, the index will be set to -1.

C++

```
COMMON_API GPResult gpGetBuddyIndex(
    GPConnection * connection,
    GPProfile profile,
    int * index
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profile ID of the buddy.
int * index	[out] The internal array index of the buddy.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to check if a remote profile is a buddy and to get its buddy index if it is a buddy. This buddy index can then be used in a call to gpGetBuddyStatus (see page 98). The buddy index may become invalid after a buddy is added to or deleted from the buddy list. If the profile is not a buddy, GP_NO_ERROR will be returned (as long as no other errors happen), and index will be set to -1.

gpGetBuddyStatus Function

Summary

This function gets the status for a particular buddy on the buddy list.

C++

```
COMMON_API GPResult gpGetBuddyStatus(
    GPConnection * connection,
    int index,
    GPBuddyStatus * status
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
int index	[in] The array index of the buddy.
GPBuddyStatus * status	[out] The status of this buddy.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to get the status of a particular buddy. index is a number greater than or equal to 0 and less than the total number of buddies. This function will typically be called in response to the gpRecvBuddyStatus callback being called.

gpGetNumBuddies Function

Summary

This function gets the number of buddies on the local profile's buddy list.

C++

```
COMMON_API GPResult gpGetNumBuddies(
    GPConnection * connection,
    int * numBuddies
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
int * numBuddies	[out] The number of buddies.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function gets the number of buddies on the local profile's buddy list. It may take some time to receive the total number of buddies from the server, so this function may report a number smaller than the actual total while the complete buddy list is being received. To see the status of each buddy, call gpGetBuddyStatus (see page 98). The number of buddies is only valid until a buddy is added to or deleted from the buddy list.

gpGetProfileBuddyList Function

Summary

Get the buddies for a profile.

C++

```
COMMON_API GPResult gpGetProfileBuddyList(
    GPConnection * connection,
    GPProfile profile,
    int maxBuddies,
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profile to get the buddy list for.
int maxBuddies	[in] The maximum number of buddies to return. If 0 is passed in, all buddies will be returned.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A GP (see page 87) callback that will be passed a GPGetProfileBuddyListArg (see page 143).
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

GPResult (see page 160) is described in the GP (see page 87) enums section.

Remarks

This function returns a valid GPResult (see page 160). Common return values are:

GP_NO_ERROR on success.

GP_PARAMETER_ERROR is returned if connection is NULL, profile is 0, callback is NULL, or the connection is not connected. GP_MEMORY_ERROR is returned when an allocation fails.

GP_NETWORK_ERROR is returned when there is a problem connecting to the Presence and Messaging backend.

See Also

GPGetProfileBuddyListArg (see page 143)

gpGetReverseBuddies Function

Summary

Get profiles that have you on their buddy list.

C++

```
COMMON_API GPResult gpGetReverseBuddies(
    GPConnection * connection,
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A GP (see page 87) callback that will be passed a GPGetReverseBuddiesResponseArg (see page 144).
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

Get profiles that have you on their buddy list.

See Also

GPGetReverseBuddiesResponseArg (see page 144)

gpIsBuddy Function

Summary

Returns 1 if the given ProfileID is a buddy, 0 if not a buddy.

C++

```
COMMON_API int gpIsBuddy(
    GPConnection * connection,
    GPProfile profile
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profile ID of the player to check.

Returns

Returns 1 if the given ProfileID is a buddy, 0 if not a buddy.

Remarks

Returns 1 if the given ProfileID is a buddy, 0 if not a buddy.

gpIsBuddyConnectionOpen Function

Summary

Returns 1 if the given ProfileID is connected for direct peer-to-peer messaging. Returns 0 otherwise.

C++

```
COMMON_API int gpIsBuddyConnectionOpen(
    GPConnection * connection,
    GPProfile profile
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profile ID of the player to check.

Returns

Returns 1 if the given ProfileID is directly connected, 0 if not.

Remarks

Returns 1 if the given ProfileID is connected for direct peer-to-peer messaging. Returns 0 otherwise.

gpRevokeBuddyAuthorization Function

Summary

Remove the local client from a remote users buddy list.

C++

```
COMMON_API GPResult gpRevokeBuddyAuthorization(
    GPConnection * connection,
    GPProfile profile
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profile ID of the remote player.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

Use this function when the local client no longer wants the remote player to be able to send buddy messages or view status info.

gpSendBuddyMessage Function

Summary

This function sends a message to a buddy.

C++

```
COMMON_API GPResult gpSendBuddyMessage(
    GPConnection * connection,
    GPProfile profile,
    const gsi_char * message
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profile object for the buddy to whom the message is going.
const gsi_char * message	[in] A user-readable text string containing the message to send to the buddy.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

If a direct connection to the buddy is possible (the buddy is not behind a firewall), the message can be any size. However, if the buddy is behind a firewall, then the message needs to be sent through the server. In this case, there is a limit of 1024 characters. Any message longer than 1024 characters that needs to be sent through the server will be truncated without warning or notice.

Notes

gpSendBuddyMessageW and gpSendBuddyMessageA are UNICODE and ANSI mapped versions of gpSendBuddyMessage. The arguments of gpSendBuddyMessageA are ANSI strings; those of gpSendBuddyMessageW are wide-character strings.

See Also

gpSendBuddyUTM (see page 103)

gpSendBuddyRequest Function

Summary

This function sends a request to a remote profile to ask for permission to add the remote profile to the local profile's buddy list.

C++

```
COMMON_API GPResult gpSendBuddyRequest(
    GPConnection * connection,
    GPProfile profile,
    const gsi_char reason[GP_REASON_LEN]
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).

GPProfile profile	[in] The remote profile to which the buddy request is being made.
const gsi_char reason[GP_REASON_LEN]	[in] A text string that (optionally) explains why the user is making the buddy request.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function sends a request to the given remote profile, asking if the local profile can make the remote profile a buddy. There is no immediate response to this message. If the remote profile authorizes the request, a buddy message and a status message will be received from the new buddy. However, this can take any amount of time. This message causes the gpRecvBuddyRequest callback to be called for the remote profile.

Notes

gpSendBuddyRequestW and gpSendBuddyRequestA are UNICODE and ANSI mapped versions of gpSendBuddyRequest. The arguments of gpSendBuddyRequestA are ANSI strings; those of gpSendBuddyRequestW are wide-character strings.

gpSendBuddyUTM Function

Summary

Sends a UTM (under-the-table message) to a buddy.

C++

```
COMMON_API GPResult gpSendBuddyUTM(
    GPConnection * connection,
    GPProfile profile,
    const gsi_char * message,
    int sendOption
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profile object for the buddy to whom the message is going.
const gsi_char * message	[in] A user-readable text string containing the message to send to the buddy.
int sendOption	[in] UTM sending options - defined in GPEnum (see page 151). Pass in 0 for no options.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

A UTM is a special type of message you can send that is usually not intended for direct display to users.

If a direct connection to the buddy is possible (i.e., the buddy is not behind a firewall), the message can be any size. However, if the buddy is behind a firewall, then the message needs to be sent through the server. In this case, there is a limit of 1024 characters. Any message longer than 1024 characters that needs to be sent through the server will be truncated without warning or notice.

If GP_DONT_ROUTE is listed as a sendOption, the SDK will only attempt to send this message directly to the player and not route it through the server.

Notes

gpSendBuddyUTMW and gpSendBuddyUTMA are UNICODE and ANSI mapped versions of gpSendBuddyUTM. The arguments of gpSendBuddyUTMA are ANSI strings; those of gpSendBuddyUTMW are wide-character strings.

See Also

gpSendBuddyMessage (see page 102)

gpAddToBlockedList Function

Summary

Adds a remote profile to the local player's blocked list.

C++

```
COMMON_API GPResult gpAddToBlockedList(
    GPConnection * connection,
    GPProfile profile
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profileid of the player to be blocked.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

A blocked player is essentially invisible to player who has him blocked. The local player will not receive any communication from the blocked player, nor will the local player be able to contact the blocked player in any way. This function will only work when GP (see page 87) is connected. This function will not return any callback on success, but the GP_ERROR callback will be called should an error occur during the add attempt.

See Also

gpRemoveFromBlockedList (see page 105), gpGetNumBlocked (see page 105), gpGetBlockedProfile (see page 104), gpIsBlocked (see page 106)

gpGetBlockedProfile Function

Summary

This function gets the profileid for a particular player on the blocked list.

C++

```
COMMON_API GPResult gpGetBlockedProfile(
    GPConnection * connection,
    int index,
    GPProfile * profile
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
int index	[in] The array index of the blocked player.
GPProfile * profile	[out] The profileid of the blocked player.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

The blocked list is fully obtained after the login process is complete. Index is a number greater than or equal to 0 and less than the total number of blocked players; this is generally called in conjunction with gpGetNumBlocked (see page 105) to enumerate through the list.

See Also

gpGetNumBlocked (see page 105), gpIsBlocked (see page 106)

gpGetNumBlocked Function

Summary

Gets the total number of blocked players in the local profile's blocked list.

C++

```
COMMON_API GPResult gpGetNumBlocked(
    GPConnection * connection,
    int * numBlocked
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
int * numBlocked	[out] The total number of blocked players in the local profile's blocked list.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function will return 0 when GP (see page 87) is not connected. The blocked list is fully obtained after the login process is complete.

See Also

gpGetBlockedProfile (see page 104), gpIsBlocked (see page 106)

gpRemoveFromBlockedList Function

Summary

Removes a remote profile from the local player's blocked list.

C++

```
COMMON_API GPResult gpRemoveFromBlockedList(
    GPConnection * connection,
    GPProfile profile
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profileid of the player to be removed from the blocked list.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

A blocked player is essentially invisible to player who has him blocked. The local player will not receive any communication from the blocked player, nor will the local player be able to contact the blocked player in any way. This function will only work when GP (see page 87) is connected. This function will not return any callback on success, but the GP_ERROR callback will be called should an error occur during the removal attempt.

See Also

gpAddToBlockedList (see page 104), gpGetNumBlocked (see page 105), gpGetBlockedProfile (see page 104), gpIsBlocked (see page 106)

gpIsBlocked Function

Summary

Returns gsi_true if the given ProfileID is blocked, gsi_false if not blocked.

C++

```
COMMON_API gsi_bool gpIsBlocked(
    GPConnection * connection,
    GPProfile profile
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profile ID of the player to check.

Returns

Returns gsi_true if the given ProfileID is blocked, gsi_false if not blocked.

gpInvitePlayer Function

Summary

This function invites a player to play a certain game.

C++

```
GPResult gpInvitePlayer(
    GPConnection * connection,
    GPProfile profile,
    int productID,
    const gsi_char location[GP_LOCATION_STRING_LEN]
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profile ID of the player to invite.
int productID	[in] The product ID of the game to which to invite the player.
const gsi_char location[GP_LOCATION_STRING_LEN]	[in] A message to send along with the invite. See remarks.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to invite another profile to join the local profile in a game's title room. The remote profile will receive a GP_RECV_GAME_INVITE callback.

gpInvitePlayer may now take an optional text message to be sent along with the invite. This usually contains the server IP and other connecting information. This parameter may be NULL. The max length for the location info is 255 characters. When compiling in Unicode mode, the location will be converted to ASCII.

Notes

gpInvitePlayerW and gpInvitePlayerA are UNICODE and ANSI mapped versions of gpInvitePlayer. The arguments of gpInvitePlayerA are ANSI strings; those of gpInvitePlayerW are wide-character strings.

gpSetQuietMode Function

Summary

Turn on or off the flags that control what types of buddy messages the local profile will receive.

C++

```
COMMON_API GPResult gpSetQuietMode(
    GPConnection * connection,
    GPEnum flags
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPEnum flags	[in] A bitwise OR of the GP_SILENCE_ flags of GPEnum (see page 151).

Returns

GPResult (see page 160) is described in the GP (see page 87) enums section.

Remarks

This function sets the types of buddy messages that the local profile will receive. This function returns a valid GPResult (see page 160). Common return values are:

GP_NO_ERROR on success.

GP_NETWORK_ERROR is returned when there is a problem connecting to the Presence and Messaging backend.

See Also

GPEnum (see page 151)

gpGetReverseBuddiesList Function

Summary

Get a list of profiles that have the specified profiles as buddies.

C++

```
COMMON_API GPResult gpGetReverseBuddiesList(
    GPConnection * connection,
    GPProfile * targets,
    int numOfTargets,
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile * targets	[out] A list of profiles for which to find reverse buddies.
int numOfTargets	[out] The length of the list of profiles in targets.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING.

GPCallback callback	[in] A GP (see page 87) callback that will be passed a GPGetReverseBuddiesListResponseArg (see page 144).
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

GPResult (see page 160) is described in the GP (see page 87) enums section.

Remarks

This function is used to find the reverse buddies of a list of profiles. A "reverse buddy" is a someone who has you on their buddy list. This function returns a valid GPResult (see page 160). Common return values are:

GP_NO_ERROR on success.

GP_PARAMETER_ERROR is returned if connection is NULL, profile is 0, callback is NULL, or the connection is not connected. GP_MEMORY_ERROR is returned when an allocation fails.

GP_NETWORK_ERROR is returned when there is a problem connecting to the Presence and Messaging backend.

See Also

GPGetProfileBuddyListArg (see page 143)

User and Profile Management

Functions

	Name	Description
	gpCheckUser (see page 108)	Validates a user's info without logging into the account.
	gpDeleteProfile (see page 109)	This function deletes the local profile. Note that this is a blocking call.
	gpNewProfile (see page 110)	This function creates a new profile for the local user.
	gpNewUser (see page 111)	This function creates a new user account and a new profile under that user account, and optionally a new uniquenick under that profile. The local user does not does not need to be signed in via gpConnect (see page 88) to use this function, although gpInitialize (see page 92) and the Available Services Check should first be successfully completed.
	gpProfilesReport (see page 112)	Debug function to dump information on known profiles to the console.
	gpRegisterUniqueNick (see page 112)	This function attempts to register a uniquenick and associate it with the local profile.
	gpSuggestUniqueNick (see page 113)	This function gets suggested uniquenicks from the backend.
	gpSetInfoCacheFilename (see page 114)	Sets the file name for the internal profile cache.
	gpSetInfod (see page 114)	These functions are used to set local info.
	gpSetInfoi (see page 115)	These functions are used to set local info.
	gpSetInfoMask (see page 115)	Sets a profile information mask with any combination of masks described in GPEnum (see page 151) enumeration.
	gpSetInfos (see page 116)	These functions are used to set local info.
	gpIsValidEmail (see page 116)	This function checks if there is an account with the given email address.
	gpSetStatus (see page 117)	This function sets the local profile's status.
	gpRegisterCdKey (see page 118)	This function attempts to register a cdkey and associate it with the local profile.

gpCheckUser Function

Summary

Validates a user's info without logging into the account.

C++

```
COMMON_API GPResult gpCheckUser(
    GPConnection * connection,
    const gsi_char nick[GP_NICK_LEN],
    const gsi_char email[GP_EMAIL_LEN],
    const gsi_char password[GP_PASSWORD_LEN],
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92). (Does not have to be connected).
const gsi_char nick[GP_NICK_LEN]	[in] The profile nickname.
const gsi_char email[GP_EMAIL_LEN]	[in] The profile email address.
const gsi_char password[GP_PASSWORD_LEN]	[in] The profile password.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A user supplied callback with an arg type of GPConnectResponseArg (see page 140).
void * param	[in] Pointer to user defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is rarely used but may be useful in certain situations. The main advantage is that a user's info may be verified without disrupting other external connections. gpConnect (see page 88) will usurp any previous connections.

Notes

gpCheckUserW and gpCheckUserA are UNICODE and ANSI mapped versions of gpCheckUser. The arguments of gpCheckUserA are ANSI strings; those of gpCheckUserW are wide-character strings.

See Also

GPCheckResponseArg (see page 140)

gpDeleteProfile Function

Summary

This function deletes the local profile. Note that this is a blocking call.

C++

```
COMMON_API GPResult gpDeleteProfile(
    GPConnection * connection,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection interface with an established connection.
GPCallback callback	[in] The callback used to confirm the deleted profile.
arg	[in] User data.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function deletes the local profile. Because the connection is between the local profile and the server, this automatically ends this connection (gpDisconnect (see page 91) does not need to be called). There is no way to delete any profile other than the current connected profile. The operation will fail if the connected profile is the user's only profile. A successful delete will result in the callback getting called. The callback will have the data about the delete profile and whether it was successful or not.

gpNewProfile Function

Summary

This function creates a new profile for the local user.

C++

```
COMMON_API GPResult gpNewProfile(
    GPConnection * connection,
    const gsi_char nick[GP_NICK_LEN],
    GPEnum replace,
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
const gsi_char nick[GP_NICK_LEN]	[in] The new profile nickname.
GPEnum replace	[in] Replacement option. (See remarks.)
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A user-supplied callback with an arg type of GPNewProfileResponseArg (see page 145).
void * param	[in] Pointer to user defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function creates a new profile for the local user. This function does not make the new profile the current profile. To switch to the newly created profile, the user must disconnect and then connect with the new nickname. If the nick for the new profile is the same as the nick for an existing profile, an error will be generated, unless replace is set to GP_REPLACE. An application should use GP_DONT_REPLACE by default. If an error with the error code of GP_NEWPROFILE_BAD_NICK is received, this means that a profile with the provided nickname already exists. The application should at this point ask the user if he wants to replace the old profile. If the user does want to replace the old profile, gpNewProfile should be called again with replace set to GP_REPLACE. When the new profile is created, the callback will be called.

Notes

gpNewProfileW and gpNewProfileA are UNICODE and ANSI mapped versions of gpNewProfile. The arguments of gpNewProfileA are ANSI strings; those of gpNewProfileW are wide-character strings.

See Also

GPNewProfileResponseArg (see page 145)

gpNewUser Function

Summary

This function creates a new user account and a new profile under that user account, and optionally a new uniquenick under that profile. The local user does not does not need to be signed in via gpConnect Function (see page 88) to use this function, although gpInitialize Function (see page 92) and the Available Services Check should first be successfully completed.

C++

```
COMMON_API GPResult gpNewUser(
    GPConnection * connection,
    const gsi_char nick[GP_NICK_LEN],
    const gsi_char uniquenick[GP_UNIQUENICK_LEN],
    const gsi_char email[GP_EMAIL_LEN],
    const gsi_char password[GP_PASSWORD_LEN],
    const gsi_char cdkey[GP_CDKEY_LEN],
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
const gsi_char nick[GP_NICK_LEN]	[in] The desired profile nickname for the initial profile of the new account.
const gsi_char uniquenick[GP_UNIQUENICK_LEN]	[in] The desired uniquenick for the initial profile of the new account.
const gsi_char email[GP_EMAIL_LEN]	[in] The desired email for the initial profile of the new account.
const gsi_char password[GP_PASSWORD_LEN]	[in] The desired password for the initial profile of the new account.
const gsi_char cdkey[GP_CDKEY_LEN]	[in] An optional CDKey to associate with the uniquenick.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] User supplied callback function with an arg type of GPNewUserResponseArg (see page 146).
void * param	[in] Pointer to user defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function attempts to create a new user account and a new profile under that user account, and optionally a new uniquenick under that profile. The local user does not does not need to be signed in via one of the gpConnect (see page 88) functions to use this function, although gpInitialize (see page 92) and the Available Services Check should be completed successfully first.

The nick, email, and password are required parameters; uniquenick and cdkey are optional. This function cannot be used to create a new profile under an existing user account (use gpNewProfile (see page 110) after successfully connecting to a profile for that). This function also cannot be used to create a uniquenick under an existing user account and profile (use gpRegisterUniqueNick (see page 112) after successfully connecting to a profile for that). Despite these limitations, gpNewUser can create a profile or a uniquenick when it is used to create a new user account.

The specified callback function will be called when the new user is created, or if an email address, nick, or password conflict is encountered.

If an email address and nick are specified and a uniquenick that's already in use is provided, then the specified callback function will receive a GP_NEWUSER_UNIQUENICK_INUSE error code regardless of whether or not the specified email address is available or whether or not the specified password is correct.

If the specified email address and password match an existing account and either the uniquenick isn't specified or it is specified and available, then the specified callback function will receive a GP_NEWUSER_BAD_NICK error code because gpNewUser cannot be used to create a new profile or uniquenick under an existing user account. See the gpNewProfile (see page 110) and gpRegisterUniqueNick (see page 112) functions for that.

If the specified email address and nick match an existing account, and the uniquenick is either not specified or available, and the password isn't correct for the specified email address, then the specified callback function will receive a GP_NEWUSER_BAD_PASSWORD error code.

Notes

gpNewUserW and gpNewUserA are UNICODE and ANSI mapped versions of gpNewUser. The arguments of gpNewUserA are ANSI strings; those of gpNewUserW are wide-character strings.

See Also

gpNewProfile (see page 110), gpRegisterUniqueNick (see page 112), GPNewUserResponseArg (see page 146)

gpProfilesReport Function

Summary

Debug function to dump information on known profiles to the console.

C++

```
COMMON_API void gpProfilesReport(
    GPConnection * connection,
    void (* report)(const char * output)
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
report	[in] A user-supplied function to be triggered with each line of info. See remarks.

Remarks

This is a debug-only function that will dump the contents of the internal profile map to the user-supplied function.

The user-supplied function is most commonly printf.

gpRegisterUniqueNick Function

Summary

This function attempts to register a uniquenick and associate it with the local profile.

C++

```
COMMON_API GPResult gpRegisterUniqueNick(
    GPConnection * connection,
    const gsi_char uniquenick[GP_UNIQUENICK_LEN],
    const gsi_char cdkey[GP_CDKEY_LEN],
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
const gsi_char uniquenick[GP_UNIQUENICK_LEN]	[in] The desired uniquenick; it can be up to GP_UNIQUENICK_LEN characters long, including the NUL.
const gsi_char cdkey[GP_CDKEY_LEN]	[in] An optional CDKey to associate with the uniquenick. If not using CDKeys this should be NULL.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A user supplied callback with an arg type of GPRegisterUniqueNickResponseArg (see page 149).
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function attempts to register a uniquenick and associate it with the local profile. It should only be used if the namespaceID passed to gpInitialize (see page 92) was greater than 0. The backend makes certain checks on a uniquenick before it is allowed to be registered. For details on what is checked, see the "Uniquenick Checks" section of the Presence and Messaging SDK Overview.

Notes

gpRegisterUniqueNickW and gpRegisterUniqueNickA are UNICODE and ANSI mapped versions of gpRegisterUniqueNick. The arguments of gpRegisterUniqueNickA are ANSI strings; those of gpRegisterUniqueNickW are wide-character strings.

See Also

GPRegisterUniqueNickResponseArg (see page 149)

gpSuggestUniqueNick Function

Summary

This function gets suggested uniquenicks from the backend.

C++

```
COMMON_API GPResult gpSuggestUniqueNick(
    GPConnection * connection,
    const gsi_char desirednick[GP_UNIQUENICK_LEN],
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
const gsi_char desirednick[GP_UNIQUENICK_LEN]	[in] The desired uniquenick, which can be up to GP_UNIQUENICK_LEN characters long, including the NUL.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A user supplied callback with an arg type of GPSuggestUniqueNickResponseArg (see page 150).
void * param	[in] Pointer to user defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function gets a set of suggested nicks based on the desirednick. A request is sent to the backend for suggestions based on the provided desirednick. After getting a response, the callback is called with a list of uniquenicks based on the desirednick. These suggested uniquenicks can then be used in a call to gpNewUser (see page 111), gpRegisterUniqueNick (see page 112), or gpSetInfos (see page 116).

Notes

gpSuggestUniqueNickW and gpSuggestUniqueNickA are UNICODE and ANSI mapped versions of gpSuggestUniqueNick. The arguments of gpSuggestUniqueNickA are ANSI strings; those of gpSuggestUniqueNickW are wide-character strings.

See Also

GPSuggestUniqueNickResponseArg (see page 150)

gpSetInfoCacheFilename Function

Summary

Sets the file name for the internal profile cache.

C++

```
COMMON_API void gpSetInfoCacheFilename(
    const gsi_char * filename
);
```

Parameters

Parameters	Description
const gsi_char * filename	[in] The filename to use for the profile cache.

Remarks

This function should be called before gpInitialize (see page 92).

Notes

gpSetInfoCacheFilenameW and gpSetInfoCacheFilenameA are UNICODE and ANSI mapped versions of gpSetInfoCacheFilename. The arguments of gpSetInfoCacheFilenameA are ANSI strings; those of gpSetInfoCacheFilenameW are wide-character strings.

gpSetInfod Function

Summary

These functions are used to set local info.

C++

```
COMMON_API GPResult gpSetInfod(
    GPConnection * connection,
    GPEnum info,
    int day,
    int month,
    int year
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPEnum info	[in] An enum indicating what info to update.

int day	[in] The day.
int month	[in] The month.
int year	[in] The year.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

These functions are used to set local info. The info does not actually get updated (sent to the server) until the next call to gpProcess (see page 94). If a string is longer than the allowable length for that info, it will be truncated without warning.

gpSetInfoi Function

Summary

These functions are used to set local info.

C++

```
COMMON_API GPResult gpSetInfoi(
    GPConnection * connection,
    GPEnum info,
    int value
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPEnum info	[in] An enum indicating what info to update.
int value	[in] The integer value.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

These functions are used to set local info. The info does not actually get updated (sent to the server) until the next call to gpProcess (see page 94). If a string is longer than the allowable length for that info, it will be truncated without warning.

gpSetInfoMask Function

Summary

Sets a profile information mask with any combination of masks described in GPEnum Enumeration (see page 151) enumeration.

C++

```
COMMON_API GPResult gpSetInfoMask(
    GPConnection * connection,
    GPEnum mask
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPEnum mask	[in] The info type. See remarks.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

The possible mask values are:

GP_MASK_NONE

GP_MASK_HOMEPAGE

GP_MASK_ZIPCODE

GP_MASK_COUNTRYCODE

GP_MASK_BIRTHDAY

GP_MASK_SEX

GP_MASK_EMAIL

GP_MASK_BUDDYLIST

GP_MASK_ALL

The mask can be any one or a combination of the above enumerations in GPEnum (see page 151).

See Also

GPEnum (see page 151)

gpSetInfos Function

Summary

These functions are used to set local info.

C++

```
COMMON_API GPResult gpSetInfos(
    GPConnection * connection,
    GPEnum info,
    const gsi_char * value
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPEnum info	[in] An enum indicating what info to update.
const gsi_char * value	[in] The string value.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

These functions are used to set local info. The info does not actually get updated (sent to the server) until the next call to gpProcess (see page 94). If a string is longer than the allowable length for that info, it will be truncated without warning.

Notes

gpSetInfosW and gpSetInfosA are UNICODE and ANSI mapped versions of gpSetInfos. The arguments of gpSetInfosA are ANSI strings; those of gpSetInfosW are wide-character strings.

gpIsValidEmail Function

Summary

This function checks if there is an account with the given email address.

C++

```
COMMON_API GPResult gpIsValidEmail(
    GPConnection * connection,
    const gsi_char email[GP_EMAIL_LEN],
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
const gsi_char email[GP_EMAIL_LEN]	[in] The email address to list accounts for.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A user supplied callback with an arg of the type GPIsValidEmailResponseArg (see page 145)
void * param	[in] Pointer to user defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function contacts the Search Manager and checks to see if there is a user with the given email address.

Notes

gpIsValidEmailW and gpIsValidEmailA are UNICODE and ANSI mapped versions of gpIsValidEmail. The arguments of gpIsValidEmailA are ANSI strings; those of gpIsValidEmailW are wide-character strings.

See Also

GPIsValidEmailResponseArg (see page 145)

gpSetStatus Function

Summary

This function sets the local profile's status.

C++

```
COMMON_API GPResult gpSetStatus(
    GPConnection * connection,
    GPEnum status,
    const gsi_char statusString[GP_STATUS_STRING_LEN],
    const gsi_char locationString[GP_LOCATION_STRING_LEN]
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPEnum status	[in] An enum indicating the status to set.
const gsi_char statusString[GP_STATUS_STRING_LEN]	[in] A text string with a user-readable explanation of the status.
const gsi_char locationString[GP_LOCATION_STRING_LEN]	[in] A URL indicating the local profile's game location in the form "gamename://IP.address:port/extra/info".

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function sets the local profile's status. The status consists of an enum specifying a mode (online, offline, playing, etc.), a text explanation of the status, and a URL specifying a game location and port in the form "quake://12.34.56.78:9999".

Notes

gpSetStatusW and gpSetStatusA are UNICODE and ANSI mapped versions of gpSetStatus. The arguments of gpSetStatusA are ANSI strings; those of gpSetStatusW are wide-character strings.

gpRegisterCdKey Function

Summary

This function attempts to register a cdkey and associate it with the local profile.

C++

```
COMMON_API GPResult gpRegisterCdKey(
    GPConnection * connection,
    const gsi_char cdkey[GP_CDKEY_LEN],
    int gameId,
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
const gsi_char cdkey[GP_CDKEY_LEN]	[in] A CDKey to associate with the currently signed-in profile.
int gameId	[in] Game ID
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A user supplied callback with an arg type of GPRegisterCdKeyResponseArg.
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

The gpRegisterCdKey() function attempts to register the specified CDKey value to the currently signed-in GP (see page 87) profile, and hence will only work when GP (see page 87) is connected. As a security measure, there is no way to retrieve the CDKey value once it is registered. It is assumed that the CDKey is available to the local game client. Once a CDKey is registered to a GP (see page 87) profile, this function can be called again as an anti-piracy measure, given that the callback function you assign in the gpRegisterCdKey() call will indicate success if the attempt to register a new CDKey to the current profile matches the CDKey already registered to that profile, or will be passed one of the following GPErrorCode (see page 157) values if not:

GP_REGISTERCDKEY = 4352, // 0x1100, There was an error registering the cdkey.

GP_REGISTERCDKEY_BAD_KEY = 4353, // 0x1101, The cdkey is invalid.

GP_REGISTERCDKEY_ALREADY_SET = 4354, // 0x1102, The profile has already been registered with a different cdkey.

GP_REGISTERCDKEY_ALREADY_TAKEN = 4355, // 0x1103, The cdkey has already been registered to another profile.

Note that only one CDKey can be associated with a single profile for single game. Once a CDKey has been associated, it cannot be associated with any other profiles.

Notes

gpRegisterCdKeyW and gpRegisterCdKeyA are UNICODE and ANSI mapped versions of gpRegisterCdKey. The arguments of gpRegisterCdKeyA are ANSI strings; those of gpRegisterCdKeyW are wide-character strings.

See Also

GPRegisterCdKeyResponseArg

Search

Functions

	Name	Description
	gpProfileSearch (see page 119)	This function searches for profiles based on certain criteria.
	gpUserIDFromProfile (see page 120)	This function gets a profile's user ID.
	gpProfileFromID (see page 120)	Translates a profile id into a GPProfile (see page 161).
	gpIDFromProfile (see page 121)	A GPProfile (see page 161) is now the same as a profileid.
	gpGetUserNicks (see page 121)	This function gets the nicknames for a given email/password (which identifies a user).
	gpGetInfo (see page 122)	This function gets info on a particular profile.

gpProfileSearch Function

Summary

This function searches for profiles based on certain criteria.

C++

```
COMMON_API GPResult gpProfileSearch(
    GPConnection * connection,
    const gsi_char nick[GP_NICK_LEN],
    const gsi_char uniquenick[GP_UNIQUENICK_LEN],
    const gsi_char email[GP_EMAIL_LEN],
    const gsi_char firstname[GP_FIRSTNAME_LEN],
    const gsi_char lastname[GP_LASTNAME_LEN],
    int icquin,
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
const gsi_char nick[GP_NICK_LEN]	[in] If not NULL or "", search for profiles with this nick.
const gsi_char uniquenick[GP_UNIQUENICK_LEN]	[in] If not NULL or "", search for profiles with this uniquenick.
const gsi_char email[GP_EMAIL_LEN]	[in] If not NULL or "", search for profiles with this email.
const gsi_char firstname[GP_FIRSTNAME_LEN]	[in] If not NULL or "", search for profiles with this firstname.
const gsi_char lastname[GP_LASTNAME_LEN]	[in] If not NULL or "", search for profiles with this lastname.
int icquin	[in] If not 0, search for profiles with this icquin.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A user supplied callback with an arg type of GPProfileSearchResponseArg (see page 147).
void * param	[in] Pointer to user defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function contacts the Search Manager and attempts to find all profiles that match the search criteria. A profile matches the provided search criteria only if its corresponding values are the same as those provided. Currently, there is no substring matching, and the criteria is case-sensitive.

When the search is complete, the callback will be called.

Notes

gpProfileSearchW and gpProfileSearchA are UNICODE and ANSI mapped versions of gpProfileSearch. The arguments of gpProfileSearchA are ANSI strings; those of gpProfileSearchW are wide-character strings.

See Also

GPProfileSearchResponseArg (see page 147)

gpUserIDFromProfile Function

Summary

This function gets a profile's user ID.

C++

```
COMMON_API GPResult gpUserIDFromProfile(
    GPConnection * connection,
    GPProfile profile,
    int * userid
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profile ID.
int * userid	[out] The user ID associated with the specified profile ID.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

Every profile is associated with a user account, and each user account has a user id associated with it. This functions gets the user id for a given profile's user account.

gpProfileFromID Function

Summary

Translates a profile id into a GPProfile Type (see page 161).

C++

```
COMMON_API GPResult gpProfileFromID(
    GPConnection * connection,
    GPProfile * profile,
    int id
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile * profile	[out] The GPProfile (see page 161) for the given profile ID.
int id	[in] The profile ID.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is deprecated as GPProfiles are now the same as profile ids. This function will be removed in a future version of the SDK.

gpIDFromProfile Function

Summary

A GPProfile Type (see page 161) is now the same as a profileid.

C++

```
COMMON_API GPResult gpIDFromProfile(
    GPConnection * connection,
    GPProfile profile,
    int * id
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The GPProfile (see page 161)
int * id	[out] The profile ID of the GPProfile (see page 161).

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is deprecated as GPProfiles are now the same as profile ids. This function will be removed in a future version of the SDK.

gpGetUserNicks Function

Summary

This function gets the nicknames for a given email/password (which identifies a user).

C++

```
COMMON_API GPResult gpGetUserNicks(
    GPConnection * connection,
    const gsi_char email[GP_EMAIL_LEN],
    const gsi_char password[GP_PASSWORD_LEN],
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92) (Does not have to be connected).
const gsi_char email[GP_EMAIL_LEN]	[in] The account email address.
const gsi_char password[GP_PASSWORD_LEN]	[in] The account password.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING.
GPCallback callback	[in] A user-supplied callback with an arg type of GPGetUserNicksResponseArg (see page 145)
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function contacts the Search Manager and gets a list of this user's nicks (profiles).

If you are unsure if the email address provided to this function is a valid email address, call gpIsValidEmail (see page 116) first.

Notes

gpGetUserNicksW and gpGetUserNicksA are UNICODE and ANSI mapped versions of gpGetUserNicks. The arguments of gpGetUserNicksA are ANSI strings; those of gpGetUserNicksW are wide-character strings.

See Also

GPGetUserNicksResponseArg (see page 145), gpIsValidEmail (see page 116)

gpGetInfo Function

Summary

This function gets info on a particular profile.

C++

```
COMMON_API GPResult gpGetInfo(
    GPConnection * connection,
    GPProfile profile,
    GPEnum checkCache,
    GPEnum blocking,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPProfile profile	[in] The profile ID of the user to get info on.
GPEnum checkCache	[in] When set to GP_CHECK_CACHE the SDK will use the currently known info.
GPEnum blocking	[in] GP_BLOCKING or GP_NON_BLOCKING
GPCallback callback	[in] A user-supplied callback with an argument type of GPGetInfoResponseArg (see page 141)
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function gets profile info for the profile object profile. When the info has been retrieved, the callback will be called.. If info-caching is enabled, the info may be available locally, in which case it will be returned immediately if checkCache is GP_CHECK_CACHE. Otherwise, the server will be contacted for the info. If the server needs to be contacted, then the function will return immediately in non-blocking mode. If info-caching is enabled, any info retrieved from the server will be cached.

See Also

GPGetInfoResponseArg (see page 141)

File Transfers

Functions

	Name	Description
	gpFreeTransfer (see page 124)	This function is used to free a file transfer.
	gpGetCurrentFile (see page 124)	This function is used to get the current file being transferred.
	gpGetFileModificationTime (see page 125)	This function is used to get a file's timestamp.
	gpGetFileName (see page 125)	This function is used to get the name of a file.
	gpGetFilePath (see page 126)	This function is used to get the local path to a file.
	gpGetFileProgress (see page 126)	This function is used to get the progress of a file being transferred.
	gpGetFileSize (see page 127)	This function is used to get the size of a file being transferred.
	gpGetNumFiles (see page 127)	This function is used to get the number of files (including directories) being transferred.
	gpGetNumTransfers (see page 128)	Returns the number of pending file transfers.
	gpGetTransfer (see page 128)	Returns the GPTransfer (see page 161) object at the specified index.
	gpGetTransferData (see page 129)	This function is used to retrieve arbitrary user-data stored with a transfer.
	gpGetTransferProfile (see page 129)	This function is used to get the remote profile for a transfer.
	gpGetTransferProgress (see page 129)	This function is used to get the total progress of the transfer, in bytes.
	gpGetTransferSide (see page 130)	This function is used to get which side of the transfer the local profile is on (sending or receiving).
	gpGetTransferSize (see page 130)	This function is used to get the total size of the transfer, in bytes.
	gpGetTransferThrottle (see page 131)	This function can be used to get a transfer's throttle setting.
	gpRejectTransfer (see page 131)	This function is used to reject a file transfer request.
	gpSendFiles (see page 132)	This function attempts to send one or more files (and/or sub-directory names) to another profile.
	gpSetTransferData (see page 133)	This function is used to store arbitrary user-data with a transfer.
	gpSetTransferDirectory (see page 133)	This function can be used to set the directory that files are received into.
	gpSkipFile (see page 134)	This function is used to skip transferring a certain file.
	gpAcceptTransfer (see page 134)	This function is used to accept a file transfer request.

gpFreeTransfer Function

Summary

This function is used to free a file transfer.

C++

```
COMMON_API GPResult gpFreeTransfer(
    GPConnection * connection,
    GPTransfer transfer
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to free a transfer object. If the transfer has completed, then this will simply free the object's resources. If the transfer has not yet completed, this will also cancel the transfer, causing the remote profile to get a GP_TRANSFER_CANCELLED callback.

See Also

gpSendFiles (see page 132), gpAcceptTransfer (see page 134), gpRejectTransfer (see page 131)

gpGetCurrentFile Function

Summary

This function is used to get the current file being transferred.

C++

```
COMMON_API GPResult gpGetCurrentFile(
    GPConnection * connection,
    GPTransfer transfer,
    int * index
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A GP (see page 87) transfer object
int * index	[out] Returns the index of the current transferring file.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to get the index of the current file being transferred. This will be 0 until the first file is finished, then 1 until the second file finishes, etc. When the transfer is complete, it will be set to the number of files in the transfer.

gpGetFileModificationTime Function

Summary

This function is used to get a file's timestamp.

C++

```
COMMON_API GPResult gpGetFileModificationTime(
    GPConnection * connection,
    GPTransfer transfer,
    int index,
    gsi_time * modTime
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
int index	[in] Index of the file within the GPTransfer (see page 161) object.
gsi_time * modTime	[out] The modification time.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to get the timestamp for a file being transferred. This is typically used by the receiver to set the file's timestamp correctly after a file has been received.

gpGetFileName Function

Summary

This function is used to get the name of a file.

C++

```
COMMON_API GPResult gpGetFileName(
    GPConnection * connection,
    GPTransfer transfer,
    int index,
    gsi_char ** name
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
int index	[in] The index of the file within the GPTransfer (see page 161) object.
gsi_char ** name	[out] The name of the file.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to get the name of a file in the transfer. The receiver should use this name to determine where to put the file after it is received. It may be a simple name ("file.ext"), or it may contain a directory path ("files/file.ext"). Any slashes in the name will be UNIX-style slashes ("files/file.ext") as opposed to Windows style slashes ("filesfile.ext").

Notes

gpGetFileNameW and gpGetFileNameA are UNICODE and ANSI mapped versions of gpGetFileName. The arguments of gpGetFileNameA are ANSI strings; those of gpGetFileNameW are wide-character strings.

gpGetFilePath Function

Summary

This function is used to get the local path to a file.

C++

```
COMMON_API GPResult gpGetFilePath(
    GPConnection * connection,
    GPTransfer transfer,
    int index,
    gsi_char ** path
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
int index	[in] The index of the file within the GPTransfer (see page 161) object.
gsi_char ** path	[in] The path of the file.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to get the local path to a file. For the sender, this will be the same path specified in the gpSendFilesCallback (see page 138). For the receiver, this will be NULL for directories and for files that haven't started transferring yet. For files that have are either transferring or have finished transferring, this is the local path where the file is being stored. It is the application's responsibility to move the file to an appropriate location (likely using the file's name) after the file has finished transferring.

Notes

gpGetFilePathW and gpGetFilePathA are UNICODE and ANSI mapped versions of gpGetFilePath. The arguments of gpGetFilePathA are ANSI strings; those of gpGetFilePathW are wide-character strings.

gpGetFileProgress Function

Summary

This function is used to get the progress of a file being transferred.

C++

```
COMMON_API GPResult gpGetFileProgress(
    GPConnection * connection,
    GPTransfer transfer,
    int index,
    int * progress
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).

GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
int index	[in] The index of the file within the GPTransfer (see page 161) object.
int * progress	[in] The transfer progress.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to get the progress of a file being transferred, or in other words, the number of bytes of the file either sent or received so far. If the file hasn't started transferring yet, the progress will be 0. The progress will be continually updated while the file is being transferred. If the file finishes transferring successfully, the progress should be the same as the file's size.

gpGetFileSize Function

Summary

This function is used to get the size of a file being transferred.

C++

```
COMMON_API GPResult gpGetFileSize(
    GPConnection * connection,
    GPTransfer transfer,
    int index,
    int * size
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
int index	[in] The index of the file within the GPTransfer (see page 161) object.
int * size	[in] The size of the file.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to get the size of a file being transferred. The size of each file is checked when the transfer is initialized, which is the size that will be reported before the file is actually transferred. The size of the file is checked again when the file actually begins transferring, which the size that will be reported from that moment on (the two sizes will only be different if the file has changed during that time).

gpGetNumFiles Function

Summary

This function is used to get the number of files (including directories) being transferred.

C++

```
COMMON_API GPResult gpGetNumFiles(
    GPConnection * connection,
    GPTransfer transfer,
    int * num
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A GPTransfer (see page 161) object.
int * num	[out] The number of files within the GPTransfer (see page 161) object.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to get the number of files being transferred. This total includes any directory names that are being sent.

gpGetNumTransfers Function

Summary

Returns the number of pending file transfers.

C++

```
COMMON_API GPResult gpGetNumTransfers(
    GPConnection * connection,
    int * num
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
int * num	[out] The number of pending transfers.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

Returns the number of pending file transfers.

gpGetTransfer Function

Summary

Returns the GPTransfer Type (see page 161) object at the specified index.

C++

```
COMMON_API GPResult gpGetTransfer(
    GPConnection * connection,
    int index,
    GPTransfer * transfer
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
int index	[in] Index of the GPTransfer (see page 161) object.
GPTransfer * transfer	[out] A pointer to a GPTransfer (see page 161) object.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

Returns the GPTransfer (see page 161) object at the specified index.

gpGetTransferData Function

Summary

This function is used to retrieve arbitrary user-data stored with a transfer.

C++

```
COMMON_API void * gpGetTransferData(
    GPConnection * connection,
    GPTransfer transfer
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.

Remarks

This function allows an application to retrieve arbitrary user-data stored with a transfer.

gpGetTransferProfile Function

Summary

This function is used to get the remote profile for a transfer.

C++

```
COMMON_API GPResult gpGetTransferProfile(
    GPConnection * connection,
    GPTransfer transfer,
    GPProfile * profile
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
GPProfile * profile	[out] The remote profile is stored here.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to get the remote profile for a transfer.

gpGetTransferProgress Function

Summary

This function is used to get the total progress of the transfer, in bytes.

C++

```
COMMON_API GPResult gpGetTransferProgress(
    GPConnection * connection,
    GPTransfer transfer,
    int * progress
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
int * progress	[out] The progress of the transfer, in bytes, is stored here.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to determine the total progress of a file transfer. This is the total number of bytes of file data that have been transferred so far.

gpGetTransferSide Function

Summary

This function is used to get which side of the transfer the local profile is on (sending or receiving).

C++

```
COMMON_API GPResult gpGetTransferSide(
    GPConnection * connection,
    GPTransfer transfer,
    GPEnum * side
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
GPEnum * side	[out] The side is stored here. This will be either GP_TRANSFER_SENDER or GP_TRANSFER_RECEIVER

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to determine if the local profile is the sender or receiver for this transfer. This is often useful inside of the gpTransferCallback when dealing with a message that both the sender and receiver may get, such as GP_FILE_END.

gpGetTransferSize Function

Summary

This function is used to get the total size of the transfer, in bytes.

C++

```
COMMON_API GPResult gpGetTransferSize(
    GPConnection * connection,
    GPTransfer transfer,
    int * size
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
int * size	[out] The size of the transfer, in bytes, will be stored here.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to determine the total size of a file transfer. This is the sum of the sizes of all the files being transferred. When a file is transferred, its size may be different than the size originally reported for the file. This can cause the total size of the transfer to change during the course of the transfer.

gpGetTransferThrottle Function

Summary

This function can be used to get a transfer's throttle setting.

C++

```
COMMON_API GPResult gpGetTransferThrottle(
    GPConnection * connection,
    GPTransfer transfer,
    int * throttle
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
int * throttle	[out] The throttle setting is stored here.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Notes

Throttling is not currently implemented.

This function is used to get the throttle setting for a transfer. If throttle is positive, it is the throttle setting in bytes-per-second. If zero, the transfer is paused. If -1, then there is no throttling.

gpRejectTransfer Function

Summary

This function is used to reject a file transfer request.

C++

```
COMMON_API GPResult gpRejectTransfer(
    GPConnection * connection,
    GPTransfer transfer,
    const gsi_char * message
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
const gsi_char * message	[in] An optional message to send along with the rejection.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to reject an incoming files request. This will also free the transfer, so it should not be referenced again once rejected.

Notes

gpRejectTransferW and gpRejectTransferA are UNICODE and ANSI mapped versions of gpRejectTransfer. The arguments of gpRejectTransferA are ANSI strings; those of gpRejectTransferW are wide-character strings.

See Also

gpSendFiles (see page 132), gpAcceptTransfer (see page 134)

gpSendFiles Function

Summary

This function attempts to send one or more files (and/or sub-directory names) to another profile.

C++

```
COMMON_API GPResult gpSendFiles(
    GPConnection * connection,
    GPTransfer * transfer,
    GPProfile profile,
    const gsi_char * message,
    gpSendFilesCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer * transfer	[out] A pointer to a GPTransfer (see page 161) object.
GPProfile profile	[in] The profile to send to. Must be a buddy, or we must be his buddy.
const gsi_char * message	[in] An optional message to send alone with the request.
gpSendFilesCallback callback	[in] This callback will get called repeatedly to get the list of files to send. See below.
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function attempts to send files to a remote profile. The profile must be either on the local profile's buddy list, or the local profile must be on the remote profile's buddy list. To send the files, a direct connection must be established between the two profiles. If both are behind firewalls, or a direct connection cannot be established for any other reason, the transfer will fail.

A successful call to this function will create a transfer object (which is identified by transfer). This object will not be freed until either the connection is destroyed with gpDestroy (see page 91)(), or the object is explicitly freed with gpFreeTransfer (see page 124)(). The object is not automatically freed when the transfer completes. Updates on this transfer will be passed back to the application through the GP_TRANSFER_CALLBACK callback.

Notes

gpSendFilesW and gpSendFilesA are UNICODE and ANSI mapped versions of gpSendFiles. The arguments of gpSendFilesA are ANSI strings; those of gpSendFilesW are wide-character strings.

See Also

gpFreeTranser, GPTransfer (see page 161), gpGetTransferProgress (see page 129), gpGetTransferData (see page 129)

gpSetTransferData Function

Summary

This function is used to store arbitrary user-data with a transfer.

C++

```
COMMON_API GPResult gpSetTransferData(
    GPConnection * connection,
    GPTransfer transfer,
    void * userData
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
void * userData	[in] Arbitrary user data to associate with the transfer.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function allows an application to associate arbitrary user-data with a transfer.

gpSetTransferDirectory Function

Summary

This function can be used to set the directory that files are received into.

C++

```
COMMON_API GPResult gpSetTransferDirectory(
    GPConnection * connection,
    GPTransfer transfer,
    const gsi_char * directory
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
const gsi_char * directory	[in] The directory to store received files in. This must be the path to a directory, and it must end in a slash or backslash.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This allows the application to set what directory in which received files are stored. They will all be stored in this directory, with names randomly generated by GP (see page 87). It is then up to the application to place the files in the appropriate directories with the appropriate names. If no directory is set, the application will pick one. On win32, the GetTempPath function is used to pick a directory. If the application wants to set a directory explicitly, it must call this function before accepting the transfer. The function will fail if it is called after the transfer has started. This function only sets the directory for the specified transfer.

Notes

gpSetTransferDirectoryW and gpSetTransferDirectoryA are UNICODE and ANSI mapped versions of gpSetTransferDirectory. The arguments of gpSetTransferDirectoryA are ANSI strings; those of gpSetTransferDirectoryW are wide-character strings.

gpSkipFile Function

Summary

This function is used to skip transferring a certain file.

C++

```
COMMON_API GPResult gpSkipFile(
    GPConnection * connection,
    GPTransfer transfer,
    int index
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] A pointer to a GPTransfer (see page 161) object.
int index	[in] Index of the file within the GPTransfer (see page 161) object.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to skip a file in the transfer. It can be called either before a file is transferred or while a file is being transferred. If it is called before the file starts transferring, then the a GP_FILE_SKIP callback will be received when the file becomes the current file. If it is called while a file is being transferred, then the GP_FILE_SKIP will be called as soon as possible, and the file will stop transferring.

gpAcceptTransfer Function

Summary

This function is used to accept a file transfer request.

C++

```
COMMON_API GPResult gpAcceptTransfer(
    GPConnection * connection,
    GPTransfer transfer,
    const gsi_char * message
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPTransfer transfer	[in] The transfer passed along with the GP_TRANSFER_SEND_REQUEST.
const gsi_char * message	[in] An optional message to send along with the accept.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function is used to accept an incoming files request. This will initiate the transfer from the remote profile to the local profile. When done with the transfer, the transfer should be freed with a call to gpFreeTransfer (see page 124).

Notes

gpAcceptTransferW and gpAcceptTransferA are UNICODE and ANSI mapped versions of gpAcceptTransfer. The arguments of gpAcceptTransferA are ANSI strings; those of gpAcceptTransferW are wide-character strings.

See Also

gpRejectTransfer (see page 131), gpSendFiles (see page 132)

Utilities

Functions

	Name	Description
◆	gpGetErrorCode (see page 135)	This function gets the current error code for a connection.
◆	gpGetErrorString (see page 136)	This function gets the current error string for a connection.
◆	gpSetCallback (see page 136)	This function is used to set callbacks. The callbacks that get set with this function are called as a result of data received from the server, such as messages or status updates.

gpGetErrorCode Function

Summary

This function gets the current error code for a connection.

C++

```
COMMON_API GPResult gpGetErrorCode(
    GPConnection * connection,
    GPErrorCode * errorCode
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPErrorCode * errorCode	[out] The current error code.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function gets the current error code for connection. It can be used to determine the specific cause of the most recent error. See the GP (see page 87) header, gp.h, for all of the possible error codes.

See Also

GPErrorCode (see page 157)

gpGetErrorString Function

Summary

This function gets the current error string for a connection.

C++

```
COMMON_API GPResult gpGetErrorString(
    GPConnection * connection,
    gsi_char errorString[GP_ERROR_STRING_LEN]
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
gsi_char errorString[GP_ERROR_STRING_LEN]	[in] A text description of the current error.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function gets the current error string for connection. The error string is a text description of the most recent error that occurred on this connection. If no errors have occurred on this connection, the error string will be empty ("").

Notes

gpGetErrorStringW and gpGetErrorStringA are UNICODE and ANSI mapped versions of gpGetErrorString. The arguments of gpGetErrorStringA are ANSI strings; those of gpGetErrorStringW are wide-character strings.

gpSetCallback Function

Summary

This function is used to set callbacks. The callbacks that get set with this function are called as a result of data received from the server, such as messages or status updates.

C++

```
COMMON_API GPResult gpSetCallback(
    GPConnection * connection,
    GPEnum func,
    GPCallback callback,
    void * param
);
```

Parameters

Parameters	Description
GPConnection * connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
GPEnum func	[in] An enum that indicates which callback is being set.
GPCallback callback	[in] The user-supplied function that will be called.
void * param	[in] Pointer to user-defined data. This value will be passed unmodified to the callback function.

Returns

This function returns GP_NO_ERROR on success. Otherwise a valid GPResult (see page 160) is returned.

Remarks

This function sets what callback to call when data is received from the server, such as messages or status updates, or an error is generated. If no callback is set for a certain situation, then no alert will be given when that situation occurs. For example, if no GP_RECV_BUDDY_REQUEST callback is set, then there will be no way of detecting when a remote profile wants to add the local profile as a buddy.

These callbacks can be generated during any function that checks for data received from the server, typically gpProcess (see page 94) or a blocking operation function.

The following can be used as parameters for callback type:

GP_ERROR,

GP_RECV_BUDDY_REQUEST,

GP_RECV_BUDDY_STATUS,

GP_RECV_BUDDY_MESSAGE,

GP_RECV_GAME_INVITE,

GP_TRANSFER_CALLBACK,

GP_RECV_BUDDY_AUTH,

GP_RECV_BUDDY_REVOKE.

Callbacks

Types

Name	Description
GPCallback (see page 137)	A generic callback function type used to specify the callback supplied to GP (see page 87) SDK functions often with gpSetCallback (see page 136).
gpSendFilesCallback (see page 138)	This is a callback used by gpSendFiles (see page 132)() to get the list of files to send.

GPCallback Type

Summary

A generic callback function type used to specify the callback supplied to Presence and Messaging (see page 87) SDK functions often with gpSetCallback Function (see page 136).

C++

```
typedef void (* GPCallback)(GPConnection * connection, void * arg, void * param);
```

Parameters

Parameters	Description
connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
arg	[in] A pointer to a response structure whose content depends on the task in progress
param	[in] The user-data, if any, that was passed into the function that triggered this callback event.

Remarks

This isn't an actual function, but a type of function which your callback function must adhere to. The arg parameter content varies depending on the task. For example, a callback that is specified when calling gpGetInfo (see page 122)() should cast its incoming arg pointer to type GPGetInfoResponseArg (see page 141).

See Also

gpSetCallback (see page 136)

gpSendFilesCallback Type

Summary

This is a callback used by gpSendFiles Function (see page 132)() to get the list of files to send.

C++

```
typedef void (* gpSendFilesCallback)(GPConnection * connection, int index, const gsi_char
** path, const gsi_char ** name, void * param);
```

Parameters

Parameters	Description
connection	[in] A GP (see page 87) connection object initialized with gpInitialize (see page 92).
index	[in] This starts at 0 and is incremented by 1 each time the callback gets called.
path	[in] Point this to the path to the file to send, or NULL for a directory.
name	[in] Point this to the name to send the file under, or NULL to use the file's local name.
param	[in] User-data that was passed into gpSendFiles (see page 132).

Remarks

This function will be called repeatedly until neither path nor name are set (or both set to NULL). The callback can be used to specify either a file or a directory. To specify a file, set the path to point to a string containing the path to the file. If the name is also set, then it contains the name to send the file under. The name can be a simple filename (i.e., "file.ext"), or it can contain a path ("files/file.ext"). If a path is specified, and name is not set (or set to NULL), then the filename part of the path will be used. For example, if path points to "c:filesfile.ext" and name is not set, then the name will be "file.ext". To specify a directory, don't set the path (or set it to NULL), then set a name. The name will be treated as a directory to create. For example, if path is not set, and name is "files", this instructs the receiver to create a directory named "files". The name for a file or folder cannot contain any directory-level references (e.g., "../file.exe"), cannot start with a slash or backslash, cannot contain any empty directory names in the path, and cannot contain any invalid characters (: * ? " < > | n).

Notes

gpSendFilesCallbackW and gpSendFilesCallbackA are UNICODE and ANSI mapped versions of gpSendFilesCallback. The arguments of gpSendFilesCallbackA are ANSI strings; those of gpSendFilesCallbackW are wide-character strings.

Structures

Structures

Name	Description
GPBuddyStatus (see page 140)	The availability level of a buddy.
GPCheckResponseArg (see page 140)	The arg parameter passed to a callback generated by a call to gpCheckUser (see page 108) is of this type.
GPConnectResponseArg (see page 140)	The arg parameter passed through to a GPCallback (see page 137) call after attempting to connect is of this type.
GPDeleteProfileResponseArg (see page 141)	This arg data type contains the data for a delete profile operation. It is generated by a call to the callback passed to gpDeleteProfile (see page 109).

GPErrorArg (see page 141)	Contains information about an error which has occurred.
GPGetInfoResponseArg (see page 141)	The arg parameter passed to a callback generated by a call to gpGetInfo (see page 122) is of this type. The structure provides information about the specified profile.
GPGetProfileBuddyListArg (see page 143)	The arg parameter passed to a callback generated by a call to gpGetProfileBuddyList (see page 99).
GPGetReverseBuddiesListResponseArg (see page 144)	The arg parameter result of a reverse buddy lookup ("Who has me as their buddy?") done with gpGetReverseBuddiesList (see page 107).
GPGetReverseBuddiesResponseArg (see page 144)	The arg parameter result of a reverse buddy lookup ("Who has me as their buddy?") done with gpGetReverseBuddies (see page 100).
GPGetUserNicksResponseArg (see page 145)	The arg parameter passed to a callback generated by a call to gpGetUserNicks (see page 121) is of this type.
GPIsValidEmailResponseArg (see page 145)	The arg parameter passed to a callback generated by a call to gpIsValidEmail (see page 116) is of this type.
GPNewProfileResponseArg (see page 145)	The arg parameter passed to a callback generated by a call to gpRegisterCdKey (see page 118) is of this type. GP_REGISTERCDKEY = 4352, // 0x1100, There was an error registering the cdkey. GP_REGISTERCDKEY_BAD_KEY = 4353, // 0x1101, The cdkey is invalid. GP_REGISTERCDKEY_ALREADY_SET = 4354, // 0x1102, The profile has already been registered with a different cdkey. GP_REGISTERCDKEY_ALREADY_TAKEN = 4355, // 0x1103, The cdkey has already been registered to another profile. ResponseArg The arg parameter passed to a callback generated by a call to gpNewProfile (see page 110) is of this type.
GPNewUserResponseArg (see page 146)	The arg parameter passed to a callback generated by a call to gpNewUser (see page 111) is of this type.
GPProfileSearchMatch (see page 146)	Information about a profile which is returned by a requested search. These structs are often collected in lists, such as those found in GPGetReverseBuddiesResponseArg (see page 144) or GPProfileSearchResponseArg (see page 147).
GPProfileSearchResponseArg (see page 147)	The arg parameter passed to a callback generated by a call to gpProfileSearch (see page 119) is of this type. Contains information about the profiles that matched the search criteria.
GPRecvBuddyAuthArg (see page 147)	An authorization add the profile as a buddy, received in the GP_RECV_BUDDY_AUTH callback.
GPRecvBuddyMessageArg (see page 147)	A buddy message received in the GP_RECV_BUDDY_MESSAGE callback.
GPRecvBuddyRequestArg (see page 148)	Information sent to the GP_RECV_BUDDY_REQUEST callback.
GPRecvBuddyRevokeArg (see page 148)	A revocation of buddy status, received in the GP_RECV_BUDDY_REVOKE callback.
GPRecvBuddyStatusArg (see page 148)	This structure indicates that a buddy's status has changed in the GP_RECV_BUDDY_STATUS callback without providing the new status. A separate call must be made to gpGetBuddyStatus (see page 98) from inside the callback to get the buddy's new status.
GPRecvBuddyUTMArg (see page 149)	A buddy UTM (Under-the-Table Message) received in the GP_RECV_BUDDY_UTM callback.
GPRecvGameInviteArg (see page 149)	An invitation to a game received in the GP_RECV_GAME_INVITE callback.
GPRegisterUniqueNickResponseArg (see page 149)	The arg parameter passed to a callback generated by a call to gpRegisterUniqueNick (see page 112) is of this type.
GPSuggestUniqueNickResponseArg (see page 150)	The arg parameter passed to a callback generated by a call to gpSuggestUniqueNick (see page 113) is of this type.
GPTransferCallbackArg (see page 150)	The arg parameter passed to a Transfer Callback.
GPUniqueMatch (see page 151)	A structure to hold both a profile and a corresponding unique nick for searches that return lists that include both for each search hit.

GPBuddyStatus Structure

Summary

The availability level of a buddy.

C++

```
typedef struct {
  GPProfile profile;
  GPEnum status;
  gsi_char statusString[GP_STATUS_STRING_LEN];
  gsi_char locationString[GP_LOCATION_STRING_LEN];
  unsigned int ip;
  int port;
  GPEnum quietModeFlags;
} GPBuddyStatus;
```

Members

Members	Description
GPProfile profile;	The profile of the buddy.
GPEnum status;	A value of GPEnum which represents the "Status" of the buddy.
gsi_char statusString[GP_STATUS_STRING_LEN];	The buddy "Status" in human-readable form.
gsi_char locationString[GP_LOCATION_STRING_LEN];	A URL indicating the game location of the buddy in the form "gamename://IP.address:port/extra/info".
unsigned int ip;	The buddy's IP address in network byte order (big-endian). This is used for buddy-to-buddy messaging.
int port;	The buddy's TCP listening port. If this is 0, the buddy is behind a firewall. This is used for buddy-to-buddy messaging.
GPEnum quietModeFlags;	A set of bit-flags indicating what message types are silenced by this buddy.

GPCheckResponseArg Structure

Summary

The arg parameter passed to a callback generated by a call to gpCheckUser Function (see page 108) is of this type.

C++

```
typedef struct {
  GPResult result;
  GPProfile profile;
} GPCheckResponseArg;
```

Members

Members	Description
GPResult result;	The result of the check; GP_NO_ERROR if successful.
GPProfile profile;	The profile of the user being checked.

GPConnectResponseArg Structure

Summary

The arg parameter passed through to a GPCallback Type (see page 137) call after attempting to connect is of this type.

C++

```
typedef struct {
  GPResult result;
  GPProfile profile;
  gsi_char uniquenick[GP_UNIQUENICK_LEN];
} GPConnectResponseArg;
```

Members

Members	Description
GPResult result;	The result of a call to a GP function; GP_NO_ERROR if successful.
GPProfile profile;	The profile of the user being connected.
gsi_char uniquenick[GP_UNIQUENICK_LEN];	The uniquenick for the newly connected user.

GPDeleteProfileResponseArg Structure

Summary

This arg data type contains the data for a delete profile operation. It is generated by a call to the callback passed to gpDeleteProfile Function (see page 109).

C++

```
typedef struct {
  GPResult result;
  GPProfile profile;
} GPDeleteProfileResponseArg;
```

Members

Members	Description
GPResult result;	The result of the delete profile operation; GP_NO_ERROR if successful.
GPProfile profile;	The deleted profile, if successful.

GPErrorArg Structure

Summary

Contains information about an error which has occurred.

C++

```
typedef struct {
  GPResult result;
  GPErrorCode errorCode;
  gsi_char * errorString;
  GPEnum fatal;
} GPErrorArg;
```

Members

Members	Description
GPResult result;	The result of a call to a GP function; GP_NO_ERROR if successful.
GPErrorCode errorCode;	The specific cause of the error.
gsi_char * errorString;	A readable text string representation of the errorCode.
GPEnum fatal;	Either GP_FATAL or GP_NON_FATAL to indicate whether error is fatal.

GPGetInfoResponseArg Structure

Summary

The arg parameter passed to a callback generated by a call to gpGetInfo Function (see page 122) is of this type. The structure provides information about the specified profile.

C++

```
typedef struct {
  GPResult result;
  GPProfile profile;
  gsi_char nick[GP_NICK_LEN];
  gsi_char uniquenick[GP_UNIQUENICK_LEN];
  gsi_char email[GP_EMAIL_LEN];
  gsi_char firstname[GP_FIRSTNAME_LEN];
  gsi_char lastname[GP_LASTNAME_LEN];
  gsi_char homepage[GP_HOMEPAGE_LEN];
  int icquin;
  gsi_char zipcode[GP_ZIPCODE_LEN];
  gsi_char countrycode[GP_COUNTRYCODE_LEN];
  float longitude;
  float latitude;
  gsi_char place[GP_PLACE_LEN];
  int birthday;
  int birthmonth;
  int birthyear;
  GPEnum sex;
  GPEnum publicmask;
  gsi_char aimname[GP_AIMNAME_LEN];
  int pic;
  int occupationid;
  int industryid;
  int incomeid;
  int marriedid;
  int childcount;
  int interests1;
  int ownership1;
  int conntypeid;
} GPGetInfoResponseArg;
```

Members

Members	Description
GPResult result;	The result of the inquiry; GP_NO_ERROR if successful.
GPProfile profile;	The profile for which further information was requested.
gsi_char nick[GP_NICK_LEN];	The nick associated with this profile.
gsi_char uniquenick[GP_UNIQUENICK_LEN];	The uniquenick associated with this profile.
gsi_char email[GP_EMAIL_LEN];	The email address associated with this profile.
gsi_char firstname[GP_FIRSTNAME_LEN];	The first name associated with this profile.
gsi_char lastname[GP_LASTNAME_LEN];	The last name associated with this profile.
gsi_char homepage[GP_HOMEPAGE_LEN];	The web page associated with this profile.
int icquin;	The ICQ UIN (User Identification Number) associated with this profile.
gsi_char zipcode[GP_ZIPCODE_LEN];	The ZIP (postal) code associated with this profile.
gsi_char countrycode[GP_COUNTRYCODE_LEN];	The country code associated with this profile.
float longitude;	The longitude associated with this profile. Negative is west; positive is east; 0,0 means unknown.
float latitude;	The latitude associated with this profile. Negative is south; positive is north; 0,0 means unknown.
gsi_char place[GP_PLACE_LEN];	A place name string associated with this profile (e.g., "USA|California|Irvine", "South Korea|Seoul", or "Turkey").
int birthday;	The birth day of month (1-31) associated with this profile.
int birthmonth;	The birth month (1-12) associated with this profile.
int birthyear;	The birth year associated with this profile.
GPEnum sex;	An enum indicating the sex associated with this profile info: GP_MALE (male), GP_FEMALE (female), or GP_PAT (unknown).

GPEnum publicmask;	A collection of bitwise-ORable flags indicating which fields of this profile's info are publicly visible. If the value of publicmask is GP_MASK_NONE then no fields are visible. If it is GP_MASK_ALL then all of the mask-able fields are visible. If any of the following bits are set, then the corresponding field is visible. If the bit is not set, then the field is hidden/masked: GP_MASK_HOMEPAGE: The web page associated with this profile. GP_MASK_ZIPCODE: The ZIP (postal) code associated with this profile. GP_MASK_COUNTRYCODE: The country code associated with this profile. GP_MASK_BIRTHDAY: The birthday fields associated with this profile. GP_MASK_SEX: The sex associated with this profile. If the flag for a field is not set, then its value in the structure should not be used. For example, if the GP_MASK_BIRTHDAY bit is not set, the birthday, birth month, and birth year fields will not be available.
gsi_char aimname[GP_AIMNAME_LEN];	The AOL IM screen name associated with this profile.
int pic;	The GameSpy Comrade/Arcade profile picture associated with this profile.
int occupationid;	The occupation id associated with this profile.
int industryid;	The industry id associated with this profile.
int incomeid;	The income associated with this profile.
int marriedid;	The marital status associated with this profile.
int childcount;	The number of children associated with this profile.
int interests1;	The bit-packed interest values associated with this profile.
int ownership1;	The bit-packed owned platform values associated with this profile.
int conntypeid;	The connection type associated with this profile.

GPGetProfileBuddyListArg Structure

Summary

The arg parameter passed to a callback generated by a call to gpGetProfileBuddyList Function (see page 99).

C++

```
typedef struct {
  GPResult result;
  GPProfile profileQueried;
  GPEnum hidden;
  int numProfiles;
  GPProfile * profiles;
} GPGetProfileBuddyListArg;
```

Members

Members	Description
GPResult result;	The result of the buddy query; GP_NO_ERROR if successful.
GPProfile profileQueried;	The profile that owns this buddy list.

GPEnum hidden;	GP_NOT_HIDDEN is returned when the queried profile allows others to view their buddy list. If others are not allowed to view their buddy list, GP_HIDDEN is returned, numProfiles is 0, and profiles is NULL.
int numProfiles;	The number of profiles returned. A profile can have no buddies so 0 can be returned.
GPProfile * profiles;	The list of profiles found.

GPGetReverseBuddiesListResponseArg Structure

Summary

The arg parameter result of a reverse buddy lookup ("Who has me as their buddy?") done with gpGetReverseBuddiesList Function (see page 107).

C++

```
typedef struct {
  GPResult result;
  int numOfUniqueMatches;
  GPUniqueMatch * matches;
} GPGetReverseBuddiesListResponseArg;
```

Members

Members	Description
GPResult result;	The result of the reverse buddy query; GP_NO_ERROR if successful.
int numOfUniqueMatches;	The number of profiles with uniquenicks that have the queried profile on their buddy list. 0 if none were found.
GPUniqueMatch * matches;	A list of profiles plus uniquenicks of length numOfUniqueMatches of those who had the queried profile on their buddy list.

GPGetReverseBuddiesResponseArg Structure

Summary

The arg parameter result of a reverse buddy lookup ("Who has me as their buddy?") done with gpGetReverseBuddies Function (see page 100).

C++

```
typedef struct {
  GPResult result;
  int numProfiles;
  GPProfileSearchMatch * profiles;
} GPGetReverseBuddiesResponseArg;
```

Members

Members	Description
GPResult result;	The result of the reverse buddy query; GP_NO_ERROR if successful.
int numProfiles;	The number of profiles that have the queried profile on their buddy list. Zero if none were found.
GPProfileSearchMatch * profiles;	A list of profiles of length numProfiles of those who had the queried profile on their buddy list.

GPGetUserNicksResponseArg Structure

Summary

The arg parameter passed to a callback generated by a call to gpGetUserNicks Function (see page 121) is of this type.

C++

```
typedef struct {
  GPResult result;
  gsi_char email[GP_EMAIL_LEN];
  int numNicks;
  gsi_char ** nicks;
  gsi_char ** uniquenicks;
} GPGetUserNicksResponseArg;
```

Members

Members	Description
GPResult result;	The result of the get nicks query; GP_NO_ERROR if successful.
gsi_char email[GP_EMAIL_LEN];	The email address that was queried.
int numNicks;	The number of profiles found to match the given email/password. If 0, then the email and password did not match. If you are unsure if the email address passed to gpGetUserNicks is valid, call gpIsValidEmail first. Then a value of 0 numNicks will always mean that the email address was valid but the password was incorrect.
gsi_char ** nicks;	The list of nicknames for the queried profile, numNicks in length.
gsi_char ** uniquenicks;	The list of profile uniquenicks, numNicks in length.

GPIsValidEmailResponseArg Structure

Summary

The arg parameter passed to a callback generated by a call to gpIsValidEmail Function (see page 116) is of this type.

C++

```
typedef struct {
  GPResult result;
  gsi_char email[GP_EMAIL_LEN];
  GPEnum isValid;
} GPIsValidEmailResponseArg;
```

GPNewProfileResponseArg Structure

Summary

The arg parameter passed to a callback generated by a call to gpRegisterCdKey Function (see page 118) is of this type.

GP_REGISTERCDKEY = 4352, // 0x1100, There was an error registering the cdkey.

GP_REGISTERCDKEY_BAD_KEY = 4353, // 0x1101, The cdkey is invalid.

GP_REGISTERCDKEY_ALREADY_SET = 4354, // 0x1102, The profile has already been registered with a different cdkey.

GP_REGISTERCDKEY_ALREADY_TAKEN = 4355, // 0x1103, The cdkey has already been registered to another profile. ResponseArg

The arg parameter passed to a callback generated by a call to gpNewProfile Function (see page 110) is of this type.

C++

```
typedef struct {
  GPResult result;
  GPProfile profile;
} GPNewProfileResponseArg;
```

Members

Members	Description
GPResult result;	The result of the create new profile operation; GP_NO_ERROR if successful.
GPProfile profile;	The newly created profile, if successful.

GPNewUserResponseArg Structure

Summary

The arg parameter passed to a callback generated by a call to gpNewUser Function (see page 111) is of this type.

C++

```
typedef struct {
  GPResult result;
  GPProfile profile;
} GPNewUserResponseArg;
```

Members

Members	Description
GPResult result;	The result of the creation attempt; GP_NO_ERROR if successful.
GPProfile profile;	The profile created for the new user, if successful.

GPProfileSearchMatch Structure

Summary

Information about a profile which is returned by a requested search. These structs are often collected in lists, such as those found in GPGetReverseBuddiesResponseArg Structure (see page 144) or GPProfileSearchResponseArg Structure (see page 147).

C++

```
typedef struct {
  GPProfile profile;
  gsi_char nick[GP_NICK_LEN];
  gsi_char uniquenick[GP_UNIQUENICK_LEN];
  int namespaceID;
  gsi_char firstname[GP_FIRSTNAME_LEN];
  gsi_char lastname[GP_LASTNAME_LEN];
  gsi_char email[GP_EMAIL_LEN];
} GPProfileSearchMatch;
```

Members

Members	Description
GPProfile profile;	The matching profile.
gsi_char nick[GP_NICK_LEN];	The profile's nickname.
gsi_char uniquenick[GP_UNIQUENICK_LEN];	The profile's uniquenick.
int namespaceID;	The namespace in which this profile lives.
gsi_char firstname[GP_FIRSTNAME_LEN];	The first name associated with the profile.
gsi_char lastname[GP_LASTNAME_LEN];	The last name associated with the profile.
gsi_char email[GP_EMAIL_LEN];	The email address associated with the profile.

GPProfileSearchResponseArg Structure

Summary

The arg parameter passed to a callback generated by a call to gpProfileSearch Function (see page 119) is of this type. Contains information about the profiles that matched the search criteria.

C++

```
typedef struct {
  GPResult result;
  int numMatches;
  GPEnum more;
  GPProfileSearchMatch * matches;
} GPProfileSearchResponseArg;
```

Members

Members	Description
GPResult result;	The result of the profile search; GP_NO_ERROR if successful.
int numMatches;	The number of profiles in this set of matches.
GPEnum more;	GP_MORE if there is another set of matches; GP_DONE if this is the last (or only) set of matches.
GPProfileSearchMatch * matches;	A set of profile search matches.

GPRecvBuddyAuthArg Structure

Summary

An authorization add the profile as a buddy, received in the GP_RECV_BUDDY_AUTH callback.

C++

```
typedef struct {
  GPProfile profile;
  unsigned int date;
} GPRecvBuddyAuthArg;
```

Members

Members	Description
GPProfile profile;	The profile that authorized the request.
unsigned int date;	The timestamp of when the auth was accepted, represented as seconds elapsed since 00:00:00 January 1st, 1970 UTC.

GPRecvBuddyMessageArg Structure

Summary

A buddy message received in the GP_RECV_BUDDY_MESSAGE callback.

C++

```
typedef struct {
  GPProfile profile;
  unsigned int date;
  gsi_char * message;
} GPRecvBuddyMessageArg;
```

Members

Members	Description
GPProfile profile;	The profile of the buddy who sent the message.

unsigned int date;	The timestamp of the message, represented as seconds elapsed since 00:00:00 January 1st, 1970 UTC.
gsi_char * message;	The text of the message.

GPRecvBuddyRequestArg Structure

Summary

Information sent to the GP_RECV_BUDDY_REQUEST callback.

C++

```
typedef struct {
  GPProfile profile;
  unsigned int date;
  gsi_char reason[GP_REASON_LEN];
} GPRecvBuddyRequestArg;
```

Members

Members	Description
GPProfile profile;	The profile of the buddy who has made the request.
unsigned int date;	The timestamp of the request, represented as seconds elapsed since 00:00:00 January 1st, 1970 UTC.
gsi_char reason[GP_REASON_LEN];	The reason for the request.

GPRecvBuddyRevokeArg Structure

Summary

A revocation of buddy status, received in the GP_RECV_BUDDY_REVOKE callback.

C++

```
typedef struct {
  GPProfile profile;
  unsigned int date;
} GPRecvBuddyRevokeArg;
```

Members

Members	Description
GPProfile profile;	The profile that revoked buddy status.
unsigned int date;	The timestamp of when the revocation occurred, represented as seconds elapsed since 00:00:00 January 1st, 1970 UTC.

GPRecvBuddyStatusArg Structure

Summary

This structure indicates that a buddy's status has changed in the GP_RECV_BUDDY_STATUS callback without providing the new status. A separate call must be made to gpGetBuddyStatus Function (see page 98) from inside the callback to get the buddy's new status.

C++

```
typedef struct {
  GPProfile profile;
  unsigned int date;
  int index;
} GPRecvBuddyStatusArg;
```

Members

Members	Description
GPProfile profile;	The profile of the buddy whose status has changed.
unsigned int date;	The timestamp of the status change, represented as seconds elapsed since 00:00:00 January 1st, 1970 UTC.
int index;	This buddy's index in the buddy list. This index can be used in a call to gpGetBuddyStatus from inside the callback to get the buddy's new status.

GPRecvBuddyUTMArg Structure

Summary

A buddy UTM (Under-the-Table Message) received in the GP_RECV_BUDDY_UTM callback.

C++

```
typedef struct {
  GPProfile profile;
  unsigned int date;
  gsi_char * message;
} GPRecvBuddyUTMArg;
```

Members

Members	Description
GPProfile profile;	The profile of the buddy who sent the UTM.
unsigned int date;	The timestamp of the UTM, represented as seconds elapsed since 00:00:00 January 1st, 1970 UTC.
gsi_char * message;	The text of the UTM.

GPRecvGameInviteArg Structure

Summary

An invitation to a game received in the GP_RECV_GAME_INVITE callback.

C++

```
typedef struct {
  GPProfile profile;
  int productID;
  gsi_char location[GP_LOCATION_STRING_LEN];
} GPRecvGameInviteArg;
```

Members

Members	Description
GPProfile profile;	The profile of the buddy who sent the invite.
int productID;	The product ID of the game to which the remote profile is inviting the local profile.
gsi_char location[GP_LOCATION_STRING_LEN];	The game location string for the game to which one is being invited in the form "gamename://ip.address:port/extra/info".

GPRegisterUniqueNickResponseArg Structure

Summary

The arg parameter passed to a callback generated by a call to gpRegisterUniqueNick Function (see page 112) is of this type.

C++

```
typedef struct {
  GPResult result;
} GPRegisterUniqueNickResponseArg;
```

Members

Members	Description
GPResult result;	The result of the register uniquenick operation; GP_NO_ERROR if successful.

GPSuggestUniqueNickResponseArg Structure

Summary

The arg parameter passed to a callback generated by a call to gpSuggestUniqueNick Function (see page 113) is of this type.

C++

```
typedef struct {
  GPResult result;
  int numSuggestedNicks;
  gsi_char ** suggestedNicks;
} GPSuggestUniqueNickResponseArg;
```

Members

Members	Description
GPResult result;	The result of the suggest uniquenick operation; GP_NO_ERROR if successful.
int numSuggestedNicks;	The number of suggested uniquenicks contained in this struct.
gsi_char ** suggestedNicks;	An array of suggested uniquenicks. The number of elements in the array is specified by numSuggestedNicks.

GPTransferCallbackArg Structure

Summary

The arg parameter passed to a Transfer Callback.

C++

```
typedef struct {
  GPTransfer transfer;
  GPEnum type;
  int index;
  int num;
  gsi_char * message;
} GPTransferCallbackArg;
```

Members

Members	Description
GPTransfer transfer;	The transfer object this callback is for.
GPEnum type;	The type of information being passed to the application. See the "Transfer callback type" section of GPEnum.
int index;	If this callback is related to a specific file being transferred, this is that file's index.

int num;	An integer used in conjunction with certain "type" values to pass supplementary information to the program.
gsi_char * message;	If the type is GP_TRANSFER_SEND_REQUEST, GP_TRANSFER_ACCEPTED, or GP_TRANSFER_REJECTED, then this may point to a user-readable text message sent with the request or reply. The message will be invalid once this callback returns.

GPUniqueMatch Structure

Summary

A structure to hold both a profile and a corresponding unique nick for searches that return lists that include both for each search hit.

C++

```
typedef struct {
  GPProfile profile;
  gsi_char uniqueNick[GP_UNIQUENICK_LEN];
} GPUniqueMatch;
```

Members

Members	Description
GPProfile profile;	A profile that matched the search hit.
gsi_char uniqueNick[GP_UNIQUENICK_LEN];	A unique nick belonging to that profile.

Enumerations

Enumerations

	Name	Description
	GPEnum (see page 151)	Presence and Messaging SDK's general enum list. These are arguments and return values for many GP (see page 87) functions.
	GPErrorCode (see page 157)	Error codes which can occur in Presence and Messaging.
	GPResult (see page 160)	Presence and Messaging SDK's possible Results which can be returned from GP (see page 87) functions. Check individual function definitions to see possible results.

GPEnum Enumeration

Summary

Presence and Messaging SDK's general enum list. These are arguments and return values for many Presence and Messaging (see page 87) functions.

C++

```
enum GPEnum {
  GP_ERROR = 0,
  GP_RECV_BUDDY_REQUEST = 1,
  GP_RECV_BUDDY_STATUS = 2,
  GP_RECV_BUDDY_MESSAGE = 3,
  GP_RECV_BUDDY_UTM = 4,
  GP_RECV_GAME_INVITE = 5,
  GP_TRANSFER_CALLBACK = 6,
  GP_RECV_BUDDY_AUTH = 7,
  GP_RECV_BUDDY_REVOKE = 8,
  GP_INFO_CACHING = 256,
  GP_SIMULATION = 257,
  GP_INFO_CACHING_BUDDY_AND_BLOCK_ONLY = 258,
  GP_NP_SYNC = 259,
  GP_BLOCKING = 1,
  GP_NON_BLOCKING = 0,
  GP_FIREWALL = 1,
  GP_NO_FIREWALL = 0,
  GP_CHECK_CACHE = 1,
  GP_DONT_CHECK_CACHE = 0,
  GP_VALID = 1,
  GP_INVALID = 0,
  GP_FATAL = 1,
  GP_NON_FATAL = 0,
  GP_MALE = 1280,
  GP_FEMALE = 1281,
  GP_PAT = 1282,
  GP_MORE = 1536,
  GP_DONE = 1537,
  GP_NICK = 1792,
  GP_UNIQUENICK = 1793,
  GP_EMAIL = 1794,
  GP_PASSWORD = 1795,
  GP_FIRSTNAME = 1796,
  GP_LASTNAME = 1797,
  GP_ICQUIN = 1798,
  GP_HOMEPAGE = 1799,
  GP_ZIPCODE = 1800,
  GP_COUNTRYCODE = 1801,
  GP_BIRTHDAY = 1802,
  GP_SEX = 1803,
  GP_CPUBRANDID = 1804,
  GP_CPUSPEED = 1805,
  GP_MEMORY = 1806,
  GP_VIDEOCARD1STRING = 1807,
  GP_VIDEOCARD1RAM = 1808,
  GP_VIDEOCARD2STRING = 1809,
  GP_VIDEOCARD2RAM = 1810,
  GP_CONNECTIONID = 1811,
  GP_CONNECTIONSPEED = 1812,
  GP_HASNETWORK = 1813,
  GP_OSSTRING = 1814,
  GP_AIMNAME = 1815,
  GP_PIC = 1816,
  GP_OCCUPATIONID = 1817,
  GP_INDUSTRYID = 1818,
  GP_INCOMEID = 1819,
  GP_MARRIEDID = 1820,
  GP_CHILDCOUNT = 1821,
  GP_INTERESTS1 = 1822,
  GP_REPLACE = 1,
  GP_DONT_REPLACE = 0,
  GP_CONNECTED = 1,
  GP_NOT_CONNECTED = 0,
  GP_MASK_NONE = 0,
  GP_MASK_HOMEPAGE = 1,
  GP_MASK_ZIPCODE = 2,
  GP_MASK_COUNTRYCODE = 4,
  GP_MASK_BIRTHDAY = 8,
  GP_MASK_SEX = 16,
  GP_MASK_EMAIL = 32,
  GP_MASK_BUDDYLIST = 64,
  GP_MASK_ALL = -1,
  GP_HIDDEN = 1,
  GP_NOT_HIDDEN = 0,
  GP_OFFLINE = 0,
  GP_ONLINE = 1,
  GP_PLAYING = 2,
  GP_STAGING = 3,
  GP_CHATTING = 4,
  GP_AWAY = 5,
  GP_INTEL = 1,
  GP_AMD = 2,
  GP_CYRIX = 3,
  GP_MOTOROLA = 4,
  GP_ALPHA = 5,
  GP_MODEM = 1,
  GP_ISDN = 2,
  GP_CABLEMODEM = 3,
  GP_DSL = 4,
  GP_SATELLITE = 5,
  GP_ETHERNET = 6,
  GP_WIRELESS = 7,
  GP_TRANSFER_SEND_REQUEST = 2048,
  GP_TRANSFER_ACCEPTED = 2049,
  GP_TRANSFER_REJECTED = 2050,
  GP_TRANSFER_NOT_ACCEPTING = 2051,
  GP_TRANSFER_NO_CONNECTION = 2052,
  GP_TRANSFER_DONE = 2053,
  GP_TRANSFER_CANCELLED = 2054,
  GP_TRANSFER_LOST_CONNECTION = 2055,
  GP_TRANSFER_ERROR = 2056,
  GP_TRANSFER_THROTTLE = 2057,
  GP_FILE_BEGIN = 2058,
  GP_FILE_PROGRESS = 2059,
  GP_FILE_END = 2060,
  GP_FILE_DIRECTORY = 2061,
  GP_FILE_SKIP = 2062,
  GP_FILE_FAILED = 2063,
  GP_FILE_READ_ERROR = 2304,
  GP_FILE_WRITE_ERROR = 2305,
  GP_FILE_DATA_ERROR = 2306,
  GP_TRANSFER_SENDER = 2560,
  GP_TRANSFER_RECEIVER = 2561,
  GP_DONT_ROUTE = 2816,
  GP_SILENCE_NONE = 0,
  GP_SILENCE_MESSAGES = 1,
  GP_SILENCE_UTMS = 2,
  GP_SILENCE_LIST = 4,
  GP_SILENCE_ALL = -1
};
```

Members

Members	Description
GP_ERROR = 0	Callback called whenever GP_NETWORK_ERROR or GP_SERVER_ERROR occur.
GP_RECV_BUDDY_REQUEST = 1	Callback called when another profile requests to add you to their buddy list.
GP_RECV_BUDDY_STATUS = 2	Callback called when one of your buddies changes status.
GP_RECV_BUDDY_MESSAGE = 3	Callback called when someone has sent you a buddy message.
GP_RECV_BUDDY_UTM = 4	Callback called when someone has sent you a UTM message.
GP_RECV_GAME_INVITE = 5	Callback called when someone invites you to a game.
GP_TRANSFER_CALLBACK = 6	Callback called for status updates on a file transfer.
GP_RECV_BUDDY_AUTH = 7	Callback called when someone authorizes your buddy request.
GP_RECV_BUDDY_REVOKE = 8	Callback called when another profile stops being your buddy.
GP_INFO_CACHING = 256	0x100, Turns on full caching of profiles with gpEnable().
GP_SIMULATION = 257	0x101, Turns on simulated GP function calls without network traffic with gpEnable().
GP_INFO_CACHING_BUDDY_AND_BLOCK_ONLY = 258	0x102, Recommended: Turns on caching of only buddy and blocked list profiles with gpEnable().
GP_NP_SYNC = 259	0x103,Turns on/off the NP to GP buddy sync.
GP_BLOCKING = 1	Tells the function call to stop and wait for a callback.

GP_NON_BLOCKING = 0	Recommended: Tells the function call to return and continue processing, but gpProcess() must be called periodically.
GP_FIREWALL = 1	Sets gpConnect() to send buddy messages through the GP backend.
GP_NO_FIREWALL = 0	Recommended: Sets gpConnect() to try to send buddy messages directly, then fall back to the GP backend.
GP_CHECK_CACHE = 1	Recommended: gpGetInfo() checks the local cache first for profile data, then the GP backend.
GP_DONT_CHECK_CACHE = 0	gpGetInfo() only queries the GP backend for profile data.
GP_VALID = 1	Indicates in GPIsValidEmailResponseArg.isValid that a gpIsValidEmail() call found the specified email address.
GP_INVALID = 0	Indicates in GPIsValidEmailResponseArg.isValid that a gpIsValidEmail() call did NOT find the specified email address.
GP_FATAL = 1	Indicates in GPErrorArg.fatal that a fatal GP_ERROR has occurred.
GP_NON_FATAL = 0	Indicates in GPErrorArg.fatal that a non-fatal GP_ERROR has occurred.
GP_MALE = 1280	0x500, Indicates in GPGetInfoResponseArg.sex that a gpGetInfo() call returned a male profile.
GP_FEMALE = 1281	0x501, Indicates in GPGetInfoResponseArg.sex that a gpGetInfo() call returned a female profile.
GP_PAT = 1282	0x502, Indicates in GPGetInfoResponseArg.sex that a gpGetInfo() call returned an sexless profile.
GP_MORE = 1536	0x600, Indicates in GPProfileSearchResponseArg.more that a gpProfileSearch() call has more matching records.
GP_DONE = 1537	0x601, Indicates in GPProfileSearchResponseArg.more that a gpProfileSearch() call has no more matching records.
GP_NICK = 1792	0x700, Profile info used in gpGetInfo() and gpSetInfo() calls, length limit: 30.
GP_UNIQUENICK = 1793	0x701, Profile info used in gpGetInfo() and gpSetInfo() calls, length limit: 20.
GP_EMAIL = 1794	0x702, Profile info used in gpGetInfo() and gpSetInfo() calls, length limit: 50.
GP_PASSWORD = 1795	0x703, Profile info used in gpGetInfo() and gpSetInfo() calls, length limit: 30.
GP_FIRSTNAME = 1796	0x704, Profile info used in gpGetInfo() and gpSetInfo() calls, length limit: 30.
GP_LASTNAME = 1797	0x705, Profile info used in gpGetInfo() and gpSetInfo() calls, length limit: 30.
GP_ICQUIN = 1798	0x706, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_HOMEPAGE = 1799	0x707, Profile info used in gpGetInfo() and gpSetInfo() calls, length limit: 75.
GP_ZIPCODE = 1800	0x708, Profile info used in gpGetInfo() and gpSetInfo() calls, length limit: 10.
GP_COUNTRYCODE = 1801	0x709, Profile info used in gpGetInfo() and gpSetInfo() calls, length limit: 2.
GP_BIRTHDAY = 1802	0x70A, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_SEX = 1803	0x70B, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_CPUBRANDID = 1804	0x70C, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_CPUSPEED = 1805	0x70D, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_MEMORY = 1806	0x70E, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_VIDEOCARD1STRING = 1807	0x70F, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_VIDEOCARD1RAM = 1808	0x710, Profile info used in gpGetInfo() and gpSetInfo() calls.

GP_VIDEOCARD2STRING = 1809	0x711, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_VIDEOCARD2RAM = 1810	0x712, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_CONNECTIONID = 1811	0x713, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_CONNECTIONSPEED = 1812	0x714, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_HASNETWORK = 1813	0x715, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_OSSTRING = 1814	0x716, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_AIMNAME = 1815	0x717, Profile info used in gpGetInfo() and gpSetInfo() calls, length limit: 50.
GP_PIC = 1816	0x718, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_OCCUPATIONID = 1817	0x719, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_INDUSTRYID = 1818	0x71A, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_INCOMEID = 1819	0x71B, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_MARRIEDID = 1820	0x71C, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_CHILDCOUNT = 1821	0x71D, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_INTERESTS1 = 1822	0x71E, Profile info used in gpGetInfo() and gpSetInfo() calls.
GP_REPLACE = 1	Tells gpNewProfile() to overwrite a matching user profile.
GP_DONT_REPLACE = 0	Recommended: Tells gpNewProfile() to notify you instead of overwriting a matching user profile.
GP_CONNECTED = 1	Output by gpIsConnected() when the GPConnection object has a connection to the server.
GP_NOT_CONNECTED = 0	Output by gpIsConnected() when the GPConnection object does NOT have a connection to the server.
GP_MASK_NONE = 0	0x00, Indicates that none of the profile's fields are visible.
GP_MASK_HOMEPAGE = 1	0x01, Indicates that the profile's homepage field is visible.
GP_MASK_ZIPCODE = 2	0x02, Indicates that the profile's zipcode field is visible.
GP_MASK_COUNTRYCODE = 4	0x04, Indicates that the profile's country code field is visible.
GP_MASK_BIRTHDAY = 8	0x08, Indicates that the profile's birthday field is visible.
GP_MASK_SEX = 16	0x10, Indicates that the profile's sex field is visible.
GP_MASK_EMAIL = 32	0x20, Indicates that the profile's email field is visible.
GP_MASK_BUDDYLIST = 64	0x40, Indicates that the profile's buddy list is visible.
GP_MASK_ALL = -1	0xFFFFFFFF, Indicates that all of a profile's fields are visible.
GP_HIDDEN = 1	Indicates in GPGetProfileBuddyListArg.hidden that a gpGetProfileBuddyList() call requested a profile that hides its buddies.
GP_NOT_HIDDEN = 0	Indicates in GPGetProfileBuddyListArg.hidden that a gpGetProfileBuddyList() call requested a profile that does NOT hide its buddies.
GP_OFFLINE = 0	Indicates in GPBuddyStatus.status that a gpGetBuddyStatus() call found a buddy that is not available.
GP_ONLINE = 1	Indicates in GPBuddyStatus.status that a gpGetBuddyStatus() call found a buddy that is available.
GP_PLAYING = 2	Indicates in GPBuddyStatus.status that a gpGetBuddyStatus() call found a buddy that is playing a game.
GP_STAGING = 3	Indicates in GPBuddyStatus.status that a gpGetBuddyStatus() call found a buddy that is getting ready to play a game.
GP_CHATTING = 4	Indicates in GPBuddyStatus.status that a gpGetBuddyStatus() call found a buddy that is communicating.
GP_AWAY = 5	Indicates in GPBuddyStatus.status that a gpGetBuddyStatus() call found a buddy that is not at his PC.
GP_INTEL = 1	Tells gpSetInfoi() the user's GP_CPUBRANDID is Intel.

GP_AMD = 2	Tells gpSetInfoi() the user's GP_CPUBRANDID is AMD.
GP_CYRIX = 3	Tells gpSetInfoi() the user's GP_CPUBRANDID is Cyrix.
GP_MOTOROLA = 4	Tells gpSetInfoi() the user's GP_CPUBRANDID is Motorola.
GP_ALPHA = 5	Tells gpSetInfoi() the user's GP_CPUBRANDID is Alpha.
GP_MODEM = 1	Tells gpSetInfoi() the user's GP_CONNECTIONID is a modem.
GP_ISDN = 2	Tells gpSetInfoi() the user's GP_CONNECTIONID is ISDN.
GP_CABLEMODEM = 3	Tells gpSetInfoi() the user's GP_CONNECTIONID is a cable modem.
GP_DSL = 4	Tells gpSetInfoi() the user's GP_CONNECTIONID is DSL.
GP_SATELLITE = 5	Tells gpSetInfoi() the user's GP_CONNECTIONID is a satellite.
GP_ETHERNET = 6	Tells gpSetInfoi() the user's GP_CONNECTIONID is ethernet.
GP_WIRELESS = 7	Tells gpSetInfoi() the user's GP_CONNECTIONID is wireless.
GP_TRANSFER_SEND_REQUEST = 2048	0x800, Indicates in GPTransferCallbackArg.type that a remote profile wants to send files to the local profile.
GP_TRANSFER_ACCEPTED = 2049	0x801, Indicates in GPTransferCallbackArg.type that a transfer request has been accepted.
GP_TRANSFER_REJECTED = 2050	0x802, Indicates in GPTransferCallbackArg.type that a transfer request has been rejected.
GP_TRANSFER_NOT_ACCEPTING = 2051	0x803, Indicates in GPTransferCallbackArg.type that the remote profile is not accepting file transfers.
GP_TRANSFER_NO_CONNECTION = 2052	0x804, Indicates in GPTransferCallbackArg.type that a direct connection with the remote profile could not be established.
GP_TRANSFER_DONE = 2053	0x805, Indicates in GPTransferCallbackArg.type that the file transfer has finished successfully.
GP_TRANSFER_CANCELLED = 2054	0x806, Indicates in GPTransferCallbackArg.type that the file transfer has been cancelled before completing.
GP_TRANSFER_LOST_CONNECTION = 2055	0x807, Indicates in GPTransferCallbackArg.type that the direct connection with the remote profile has been lost.
GP_TRANSFER_ERROR = 2056	0x808, Indicates in GPTransferCallbackArg.type that there was an error during the transfer.
GP_TRANSFER_THROTTLE = 2057	0x809, Reserved for future use.
GP_FILE_BEGIN = 2058	0x80A, Indicates in GPTransferCallbackArg.type that a file is about to be transferred.
GP_FILE_PROGRESS = 2059	0x80B, Indicates in GPTransferCallbackArg.type that file data has been either sent or received.
GP_FILE_END = 2060	0x80C, Indicates in GPTransferCallbackArg.type that a file has finished transferring successfully.
GP_FILE_DIRECTORY = 2061	0x80D, Indicates in GPTransferCallbackArg.type that the current "file" being transferred is a directory name.
GP_FILE_SKIP = 2062	0x80E, Indicates in GPTransferCallbackArg.type that the current file is being skipped.
GP_FILE_FAILED = 2063	0x80F, Indicates in GPTransferCallbackArg.type that the current file being transferred has failed.
GP_FILE_READ_ERROR = 2304	0x900, Indicates in GPTransferCallbackArg.num that the sender had an error reading the file.
GP_FILE_WRITE_ERROR = 2305	0x901, Indicates in GPTransferCallbackArg.num that the sender had an error writing the file.
GP_FILE_DATA_ERROR = 2306	0x902, Indicates in GPTransferCallbackArg.num that the MD5 check of the data being transferred failed.
GP_TRANSFER_SENDER = 2560	0xA00, Output by gpGetTransferSide() when the local profile is sending the file.

GP_TRANSFER_RECEIVER = 2561	0xA01, Output by gpGetTransferSide() when the local profile is receiving the file.
GP_DONT_ROUTE = 2816	0xB00, Tells gpSendBuddyUTM() to send this UTM message directly to the buddy, instead of routing it through the backend.
GP_SILENCE_NONE = 0	Indicates to gpSetQuietMode() that no message types should be silenced.
GP_SILENCE_MESSAGES = 1	Indicates to gpSetQuietMode() that messages should be silenced.
GP_SILENCE_UTMS = 2	Indicates to gpSetQuietMode() that UTM type messages should be silenced.
GP_SILENCE_LIST = 4	Indicates to gpSetQuietMode() that list type messages should be silenced.
GP_SILENCE_ALL = -1	0xFFFFFFFF, Indicates to gpSetQuietMode() that all message types should be silenced.

GPErrorCode Enumeration

Summary

Error codes which can occur in Presence and Messaging.

C++

```
enum GPErrorCode {
  GP_GENERAL = 0,
  GP_PARSE = 1,
  GP_NOT_LOGGED_IN = 2,
  GP_BAD_SESSKEY = 3,
  GP_DATABASE = 4,
  GP_NETWORK = 5,
  GP_FORCED_DISCONNECT = 6,
  GP_CONNECTION_CLOSED = 7,
  GP_UDP_LAYER = 8,
  GP_LOGIN = 256,
  GP_LOGIN_TIMEOUT = 257,
  GP_LOGIN_BAD_NICK = 258,
  GP_LOGIN_BAD_EMAIL = 259,
  GP_LOGIN_BAD_PASSWORD = 260,
  GP_LOGIN_BAD_PROFILE = 261,
  GP_LOGIN_PROFILE_DELETED = 262,
  GP_LOGIN_CONNECTION_FAILED = 263,
  GP_LOGIN_SERVER_AUTH_FAILED = 264,
  GP_LOGIN_BAD_UNIQUENICK = 265,
  GP_LOGIN_BAD_PREAUTH = 266,
  GP_LOGIN_BAD_LOGIN_TICKET = 267,
  GP_LOGIN_EXPIRED_LOGIN_TICKET = 268,
  GP_NEWUSER = 512,
  GP_NEWUSER_BAD_NICK = 513,
  GP_NEWUSER_BAD_PASSWORD = 514,
  GP_NEWUSER_UNIQUENICK_INVALID = 515,
  GP_NEWUSER_UNIQUENICK_INUSE = 516,
  GP_UPDATEUI = 768,
  GP_UPDATEUI_BAD_EMAIL = 769,
  GP_NEWPROFILE = 1024,
  GP_NEWPROFILE_BAD_NICK = 1025,
  GP_NEWPROFILE_BAD_OLD_NICK = 1026,
  GP_UPDATEPRO = 1280,
  GP_UPDATEPRO_BAD_NICK = 1281,
  GP_ADDBUDDY = 1536,
  GP_ADDBUDDY_BAD_FROM = 1537,
  GP_ADDBUDDY_BAD_NEW = 1538,
  GP_ADDBUDDY_ALREADY_BUDDY = 1539,
  GP_ADDBUDDY_IS_ON_BLOCKLIST = 1540,
  GP_AUTHADD = 1792,
  GP_AUTHADD_BAD_FROM = 1793,
  GP_AUTHADD_BAD_SIG = 1794,
  GP_AUTHADD_IS_ON_BLOCKLIST = 1795,
  GP_STATUS = 2048,
  GP_BM = 2304,
  GP_BM_NOT_BUDDY = 2305,
  GP_BM_EXT_INFO_NOT_SUPPORTED = 2306,
  GP_BM_BUDDY_OFFLINE = 2307,
  GP_GETPROFILE = 2560,
  GP_GETPROFILE_BAD_PROFILE = 2561,
  GP_DELBUDDY = 2816,
  GP_DELBUDDY_NOT_BUDDY = 2817,
  GP_DELPROFILE = 3072,
  GP_DELPROFILE_LAST_PROFILE = 3073,
  GP_SEARCH = 3328,
  GP_SEARCH_CONNECTION_FAILED = 3329,
  GP_SEARCH_TIMED_OUT = 3330,
  GP_CHECK = 3584,
  GP_CHECK_BAD_EMAIL = 3585,
  GP_CHECK_BAD_NICK = 3586,
  GP_CHECK_BAD_PASSWORD = 3587,
  GP_REVOKE = 3840,
  GP_REVOKE_NOT_BUDDY = 3841,
  GP_REGISTERUNIQUENICK = 4096,
  GP_REGISTERUNIQUENICK_TAKEN = 4097,
  GP_REGISTERUNIQUENICK_RESERVED = 4098,
  GP_REGISTERUNIQUENICK_BAD_NAMESPACE = 4099,
  GP_REGISTERCDKEY = 4352,
  GP_REGISTERCDKEY_BAD_KEY = 4353,
  GP_REGISTERCDKEY_ALREADY_SET = 4354,
  GP_REGISTERCDKEY_ALREADY_TAKEN = 4355,
  GP_ADDBLOCK = 4608,
  GP_ADDBLOCK_ALREADY_BLOCKED = 4609,
  GP_REMOVEBLOCK = 4864,
  GP_REMOVEBLOCK_NOT_BLOCKED = 4865
};
```

Members

Members	Description
GP_GENERAL = 0	There was an unknown error.
GP_PARSE = 1	Unexpected data was received from the server.
GP_NOT_LOGGED_IN = 2	The request cannot be processed because user has not logged in.
GP_BAD_SESSKEY = 3	The request cannot be processed because of an invalid session key.
GP_DATABASE = 4	There was a database error.
GP_NETWORK = 5	There was an error connecting a network socket.
GP_FORCED_DISCONNECT = 6	This profile has been disconnected by another login.
GP_CONNECTION_CLOSED = 7	The server has closed the connection.
GP_UDP_LAYER = 8	There was a problem with the UDP layer.
GP_LOGIN = 256	0x100, There was an error logging in to the GP backend.
GP_LOGIN_TIMEOUT = 257	0x101, The login attempt timed out.
GP_LOGIN_BAD_NICK = 258	0x102, The nickname provided was incorrect.
GP_LOGIN_BAD_EMAIL = 259	0x103, The email address provided was incorrect.
GP_LOGIN_BAD_PASSWORD = 260	0x104, The password provided was incorrect.
GP_LOGIN_BAD_PROFILE = 261	0x105, The profile provided was incorrect.
GP_LOGIN_PROFILE_DELETED = 262	0x106, The profile has been deleted.
GP_LOGIN_CONNECTION_FAILED = 263	0x107, The server has refused the connection.
GP_LOGIN_SERVER_AUTH_FAILED = 264	0x108, The server could not be authenticated.
GP_LOGIN_BAD_UNIQUENICK = 265	0x109, The uniquenick provided was incorrect.
GP_LOGIN_BAD_PREAUTH = 266	0x10A, There was an error validating the pre-authentication.
GP_LOGIN_BAD_LOGIN_TICKET = 267	0x10B, The login ticket was unable to be validated.
GP_LOGIN_EXPIRED_LOGIN_TICKET = 268	0x10C, The login ticket had expired and could not be used.

GP_NEWUSER = 512	0x200, There was an error creating a new user.
GP_NEWUSER_BAD_NICK = 513	0x201, A profile with that nick already exists.
GP_NEWUSER_BAD_PASSWORD = 514	0x202, The password does not match the email address.
GP_NEWUSER_UNIQUENICK_INVALID = 515	0x203, The uniquenick is invalid.
GP_NEWUSER_UNIQUENICK_INUSE = 516	0x204, The uniquenick is already in use.
GP_UPDATEUI = 768	0x300, There was an error updating the user information.
GP_UPDATEUI_BAD_EMAIL = 769	0x301, A user with the email address provided already exists.
GP_NEWPROFILE = 1024	0x400, There was an error creating a new profile.
GP_NEWPROFILE_BAD_NICK = 1025	0x401, The nickname to be replaced does not exist.
GP_NEWPROFILE_BAD_OLD_NICK = 1026	0x402, A profile with the nickname provided already exists.
GP_UPDATEPRO = 1280	0x500, There was an error updating the profile information.
GP_UPDATEPRO_BAD_NICK = 1281	0x501, A user with the nickname provided already exists.
GP_ADDBUDDY = 1536	0x600, There was an error adding a buddy.
GP_ADDBUDDY_BAD_FROM = 1537	0x601, The profile requesting to add a buddy is invalid.
GP_ADDBUDDY_BAD_NEW = 1538	0x602, The profile requested is invalid.
GP_ADDBUDDY_ALREADY_BUDDY = 1539	0x603, The profile requested is already a buddy.
GP_ADDBUDDY_IS_ON_BLOCKLIST = 1540	0x604, The profile requested is on the local profile's block list.
GP_AUTHADD = 1792	0x700, There was an error authorizing an add buddy request.
GP_AUTHADD_BAD_FROM = 1793	0x701, The profile being authorized is invalid.
GP_AUTHADD_BAD_SIG = 1794	0x702, The signature for the authorization is invalid.
GP_AUTHADD_IS_ON_BLOCKLIST = 1795	0x703, The profile requesting authorization is on a block list.
GP_STATUS = 2048	0x800, There was an error with the status string.
GP_BM = 2304	0x900, There was an error sending a buddy message.
GP_BM_NOT_BUDDY = 2305	0x901, The profile the message was to be sent to is not a buddy.
GP_BM_EXT_INFO_NOT_SUPPORTED = 2306	0x902, The profile does not support extended info keys.
GP_BM_BUDDY_OFFLINE = 2307	0x903, The buddy to send a message to is offline.
GP_GETPROFILE = 2560	0xA00, There was an error getting profile info.
GP_GETPROFILE_BAD_PROFILE = 2561	0xA01, The profile info was requested on is invalid.
GP_DELBUDDY = 2816	0xB00, There was an error deleting the buddy.
GP_DELBUDDY_NOT_BUDDY = 2817	0xB01, The buddy to be deleted is not a buddy.
GP_DELPROFILE = 3072	0xC00, There was an error deleting the profile.
GP_DELPROFILE_LAST_PROFILE = 3073	0xC01, The last profile cannot be deleted.
GP_SEARCH = 3328	0xD00, There was an error searching for a profile.
GP_SEARCH_CONNECTION_FAILED = 3329	0xD01, The search attempt failed to connect to the server.
GP_SEARCH_TIMED_OUT = 3330	0XD02, The search did not return in a timely fashion.
GP_CHECK = 3584	0xE00, There was an error checking the user account.
GP_CHECK_BAD_EMAIL = 3585	0xE01, No account exists with the provided email address.
GP_CHECK_BAD_NICK = 3586	0xE02, No such profile exists for the provided email address.
GP_CHECK_BAD_PASSWORD = 3587	0xE03, The password is incorrect.
GP_REVOKE = 3840	0xF00, There was an error revoking the buddy.
GP_REVOKE_NOT_BUDDY = 3841	0xF01, You are not a buddy of the profile.
GP_REGISTERUNIQUENICK = 4096	0x1000, There was an error registering the uniquenick.
GP_REGISTERUNIQUENICK_TAKEN = 4097	0x1001, The uniquenick is already taken.
GP_REGISTERUNIQUENICK_RESERVED = 4098	0x1002, The uniquenick is reserved.
GP_REGISTERUNIQUENICK_BAD_NAMESPACE = 4099	0x1003, Tried to register a nick with no namespace set.
GP_REGISTERCDKEY = 4352	0x1100, There was an error registering the cdkey.
GP_REGISTERCDKEY_BAD_KEY = 4353	0x1101, The cdkey is invalid.

GP_REGISTERCDKEY_ALREADY_SET = 4354	0x1102, The profile has already been registered with a different cdkey.
GP_REGISTERCDKEY_ALREADY_TAKEN = 4355	0x1103, The cdkey has already been registered to another profile.
GP_ADDBLOCK = 4608	0x1200, There was an error adding the player to the blocked list.
GP_ADDBLOCK_ALREADY_BLOCKED = 4609	0x1201, The profile specified is already blocked.
GP_REMOVEBLOCK = 4864	0x1300, There was an error removing the player from the blocked list.
GP_REMOVEBLOCK_NOT_BLOCKED = 4865	0x1301, The profile specified was not a member of the blocked list.

GPResult Enumeration

Summary

Presence and Messaging SDK's possible Results which can be returned from Presence and Messaging (see page 87) functions. Check individual function definitions to see possible results.

C++

```
enum GPResult {
  GP_NO_ERROR = 0,
  GP_MEMORY_ERROR = 1,
  GP_PARAMETER_ERROR = 2,
  GP_NETWORK_ERROR = 3,
  GP_SERVER_ERROR = 4,
  GP_MISC_ERROR = 5,
  GP_COUNT = 6
};
```

Members

Members	Description
GP_NO_ERROR = 0	Success.
GP_MEMORY_ERROR = 1	A call to allocate memory failed, probably due to insufficient memory.
GP_PARAMETER_ERROR = 2	A parameter passed to a function is either null or has an invalid value.
GP_NETWORK_ERROR = 3	An error occurred while reading or writing across the network.
GP_SERVER_ERROR = 4	One of the backend servers returned an error.
GP_MISC_ERROR = 5	An error occurred that was not covered by the other error conditions.
GP_COUNT = 6	The number of GPResults; reserved for internal use.

Types

Types

Name	Description
GPConnection (see page 161)	An instance of this type represents a GP (see page 87) connection. It is created at the beginning of a GP (see page 87) session with a call to gpInitialize (see page 92). A pointer to a GP (see page 87) object is passed as the first argument to every GP (see page 87) function.

GPProfile (see page 161)	An instance of this type represents a particular GP (see page 87) profile, such as the profile that the local user is logged into or queried information about another user's profile. A GPProfile object is passed to functions like gpSendBuddyMessage (see page 102) and gpGetInfo (see page 122), and it is returned in callbacks such as the GP_RECV_BUDDY_STATUS callback. A GPProfile object is equivalent to a profile ID. They are both int types, and can be used interchangeably.
GPTransfer (see page 161)	An instance of this type represents a particular buddy-to-buddy direct file transfer created with a call to gpSendFiles (see page 132) and managed by various gpGetTransfer (see page 128) and gpSetTransfer functions.

GPConnection Type

Summary

An instance of this type represents a Presence and Messaging (see page 87) connection. It is created at the beginning of a Presence and Messaging (see page 87) session with a call to gpInitialize Function (see page 92). A pointer to a Presence and Messaging (see page 87) object is passed as the first argument to every Presence and Messaging (see page 87) function.

C++

```
typedef void * GPConnection;
```

See Also

gpInitialize (see page 92)

GPProfile Type

Summary

An instance of this type represents a particular Presence and Messaging (see page 87) profile, such as the profile that the local user is logged into or queried information about another user's profile. A GPProfile object is passed to functions like gpSendBuddyMessage Function (see page 102) and gpGetInfo Function (see page 122), and it is returned in callbacks such as the GP_RECV_BUDDY_STATUS callback. A GPProfile object is equivalent to a profile ID. They are both int types, and can be used interchangeably.

C++

```
typedef int GPProfile;
```

GPTransfer Type

Summary

An instance of this type represents a particular buddy-to-buddy direct file transfer created with a call to gpSendFiles Function (see page 132) and managed by various gpGetTransfer Function (see page 128) and gpSetTransfer functions.

C++

```
typedef int GPTransfer;
```

See Also

gpSendFiles (see page 132)

Query and Reporting

API Documentation

Module

Query and Reporting (see page 161)

Functions

Functions

	Name	Description
	qr2_buffer_add (see page 162)	Add a string or integer to the qr2 buffer. This is used when responding to a qr2 query callback.
	qr2_buffer_add_int (see page 163)	Add a string or integer to the qr2 buffer. This is used when responding to a qr2 query callback.
	qr2_init (see page 163)	Initialize the Query and Reporting 2 SDK.
	qr2_init_socket (see page 164)	Initialize the Query and Reporting 2 SDK. Allows control over the qr2 socket object.
	qr2_keybuffer_add (see page 165)	Add a key identifier to the qr2_keybuffer_t (see page 174). This is used when enumerating the supported list of keys.
	qr2_send_statechanged (see page 166)	Notify the GameSpy Master Server of a change in gamestate.
	qr2_shutdown (see page 166)	Frees memory allocated by the qr2 SDK. This includes freeing user-registered keys.
	qr2_think (see page 166)	Allow the qr2 SDK to continue processing. Server queries can only be processed during this call.
	qr2_register_key (see page 167)	Register a key with the qr2 SDK. This tells the SDK that the application will report values for this key.

qr2_buffer_add Function

Summary

Add a string or integer to the qr2 buffer. This is used when responding to a qr2 query callback.

C++

```
COMMON_API gsi_bool qr2_buffer_add(
    qr2_buffer_t outbuf,
    const gsi_char * value
);
```

Parameters

Parameters	Description
qr2_buffer_t outbuf	[in] Buffer to which to add the value. This is obtained from the qr2callback.
const gsi_char * value	[in] String or integer value to append to the buffer.

Returns

gsi_bool

Remarks

The qr2_buffer_add function appends a string to the buffer. The qr2_buffer_add_int (see page 163) function appends an integer to the buffer. These buffers are used to construct responses to user queries and typically contain information pertaining to the game status.

See Also

qr2_buffer_add_int (see page 163)

qr2_buffer_add_int Function

Summary

Add a string or integer to the qr2 buffer. This is used when responding to a qr2 query callback.

C++

```
COMMON_API gsi_bool qr2_buffer_add_int(
    qr2_buffer_t outbuf,
    int value
);
```

Parameters

Parameters	Description
qr2_buffer_t outbuf	[in] Buffer to which to add the value. This is obtained from the qr2callback.
int value	[in] String or integer value to append to the buffer.

Returns

gsi_bool

Remarks

The qr2_buffer_add (see page 162) function appends a string to the buffer. The qr2_buffer_add_int function appends an integer to the buffer. These buffers are used to construct responses to user queries and typically contain information pertaining to the game status.

See Also

qr2_buffer_add (see page 162)

qr2_init Function

Summary

Initialize the Query and Reporting 2 SDK.

C++

```
COMMON_API qr2_error_t qr2_init(
    qr2_t * qrec,
    const gsi_char * ip,
    int baseport,
    const gsi_char * gamename,
    const gsi_char * secret_key,
    int ispublic,
    int natnegotiate,
    qr2_serverkeycallback_t server_key_callback,
    qr2_playerteamkeycallback_t player_key_callback,
    qr2_playerteamkeycallback_t team_key_callback,
    qr2_keylistcallback_t key_list_callback,
    qr2_countcallback_t playerteam_count_callback,
    qr2_adderrorcallback_t adderror_callback,
    void * userdata
);
```

Parameters

Parameters	Description
qr2_t * qrec	[out] The initialized QR2 SDK object.

const gsi_char * ip	[in] Optional IP address to which to bind; useful for multi-homed machines. Usually pass NULL.
int baseport	[in] Port to accept queries on. See remarks.
const gsi_char * gamename	[in] The gamename, assigned by GameSpy.
const gsi_char * secret_key	[in] The secret key for the specified gamename, also assigned by GameSpy.
int ispublic	[in] Set to 1 for an Internet listed server, 0 for a LAN only server.
int natnegotiate	[in] Set to 1 to allow server to be listed if it's behind a NAT (e.g., set to 1 if you support NAT Negotiation). Note: if you do not set this to 1 and you are behind a NAT you will receive the following error message from the GameSpy Master Server: "Unable to query the server. You may need to open port x for incoming traffic."
qr2_serverkeycallback_t server_key_callback	[in] Callback that is triggered when server keys are requested.
qr2_playerteamkeycallback_t player_key_callback	[in] Callback that is triggered when player keys are requested.
qr2_playerteamkeycallback_t team_key_callback	[in] Callback that is triggered when team keys are requested.
qr2_keylistcallback_t key_list_callback	[in] Callback that is triggered when the key list is requested.
qr2_countcallback_t playerteam_count_callback	[in] Callback that is triggered when the number of teams is requested.
qr2_adderrorcallback_t adderror_callback	[in] Callback that is triggered when there has been an error adding it to the list.
void * userdata	[in] Pointer to user data. This is optional and will be passed unmodified to the callback functions.

Returns

This function returns e_qrnoerrorfor a successful result. Otherwise a valid qr2_error_t (see page 174) is returned.

Remarks

The qr2_init function initializes the qr2 SDK. The baseport parameter specifies which local port should be used to accept queries on. If this port is in use, the next port value will be tried. The qr2 SDK will try up to NUM_PORTS_TO_TRYports. (Currently set at 100.).

qr2_init_socket Function

Summary

Initialize the Query and Reporting 2 SDK. Allows control over the qr2 socket object.

C++

```
COMMON_API qr2_error_t qr2_init_socket(
    qr2_t * qrec,
    SOCKET s,
    int boundport,
    const gsi_char * gamename,
    const gsi_char * secret_key,
    int ispublic,
    int natnegotiate,
    qr2_serverkeycallback_t server_key_callback,
    qr2_playerteamkeycallback_t player_key_callback,
    qr2_playerteamkeycallback_t team_key_callback,
    qr2_keylistcallback_t key_list_callback,
    qr2_countcallback_t playerteam_count_callback,
    qr2_adderrorcallback_t adderror_callback,
    void * userdata
);
```

Parameters

Parameters	Description
qr2_t * qrec	[out] The initialized QR2 SDK object.
SOCKET s	[in] Socket to be used for query traffic. This socket must have already been initialized.
int boundport	[in] The port that the socket was bound to. Chosen by the developer.
const gsi_char * gamename	[in] The gamename, assigned by GameSpy.
const gsi_char * secret_key	[in] The secret key for the specified gamename, also assigned by GameSpy.
int ispublic	[in] Set to 1 for an internet listed server, 0 for a LAN only server.
int natnegotiate	[in] Set to 1 to allow server to be listed if it's behind a NAT (e.g., set to 1 if you support NAT Negotiation). Note if you do not set this to 1 and are behind a NAT you will receive the following error message from the GameSpy Master Server: "Unable to query the server. You may need to open port x for incoming traffic."
qr2_serverkeycallback_t server_key_callback	[in] Callback that is triggered when server keys are requested.
qr2_playerteamkeycallback_t player_key_callback	[in] Callback that is triggered when player keys are requested.
qr2_playerteamkeycallback_t team_key_callback	[in] Callback that is triggered when team keys are requested.
qr2_keylistcallback_t key_list_callback	[in] Callback that is triggered when the key list is requested.
qr2_countcallback_t playerteam_count_callback	[in] Callback that is triggered when the number of teams is requested.
qr2_adderrorcallback_t adderror_callback	[in] Callback that is triggered when there has been an error adding it to the list.
void * userdata	[in] Pointer to user data. This is optional and will be passed unmodified to the callback functions.

Returns

This function returns e_qrnoerrorfor a successful result. Otherwise a valid qr2_error_t (see page 174) is returned.

Remarks

The qr2_init_socket function initializes the qr2 SDK. Instead of creating its own internal socket, the qr2 SDK will use the passed in socket for all traffic. The developer is responsible for receiving on this socket and passing received qr2 messages to qr2_parse_query.

This version of qr2_init (see page 163) allows the game to specify the UDP socket to use for sending heartbeats and query replies. This enables the game and the QR2 SDK to share a single UDP socket for all networking, which can make hosting games behind a NAT proxy possible (see the documentation for more information).

qr2_keybuffer_add Function

Summary

Add a key identifier to the qr2_keybuffer_t Type (see page 174). This is used when enumerating the supported list of keys.

C++

```
COMMON_API gsi_bool qr2_keybuffer_add(
    qr2_keybuffer_t keybuffer,
    int keyid
);
```

Parameters

Parameters	Description
qr2_keybuffer_t keybuffer	[in] Buffer to which to append the key ID.
int keyid	[in] The ID of the supported key. Add one ID for each key supported.

Remarks

The qr2_keybuffer_add function is used to when enumerating the locally supported list of keys. Add the appropriate id number for each key supported.

qr2_send_statechanged Function

Summary

Notify the GameSpy Master Server of a change in gamestate.

C++

```
COMMON_API void qr2_send_statechanged(
    qr2_t qrec
);
```

Parameters

Parameters	Description
qr2_t qrec	[in] Initialized QR2 SDK object.

Remarks

The qr2_send_statechanged function notifies the GameSpy backend of a change in game state. This call is typically reserved for major changes such as mapname or gametype. Only one statechange message may be sent per 10-second interval. If a statechange is requested within this timeframe, it will be automatically delayed until the 10-second interval has elapsed. Under no circumstances should you call this function on a regular timer.

qr2_shutdown Function

Summary

Frees memory allocated by the qr2 SDK. This includes freeing user-registered keys.

C++

```
COMMON_API void qr2_shutdown(
    qr2_t qrec
);
```

Parameters

Parameters	Description
qr2_t qrec	[in] QR2 SDK initialized with qr2_init (see page 163).

Remarks

The qr2_shutdown function may be used to free memory allocated by the qr2 SDK. The qr2 SDK should not be used after this call. This call will cease server reporting and remove the server from the backend list.

If you pass in a qrec that was returned from qr_init, all resources associated with that qrec will be freed. If you passed NULL into qr_int, you can pass NULL in here as well.

qr2_think Function

Summary

Allow the qr2 SDK to continue processing. Server queries can only be processed during this call.

C++

```
COMMON_API void qr2_think(
    qr2_t qrec
);
```

Parameters

Parameters	Description
qr2_t qrec	[in] The initialized QR2 SDK.

Remarks

The qr2_think function allows the qr2 SDK to continue processing. This processing includes responding to user queries and triggering local callbacks. If q2_think is not called often, server responses may be delayed thereby increasing perceived latency. We recommend that you call qr2_think as frequently as possible. (10-15ms is not unusual.).

qr2_register_key Function

Summary

Register a key with the qr2 SDK. This tells the SDK that the application will report values for this key.

C++

```
COMMON_API void qr2_register_key(
    int keyid,
    const gsi_char * key
);
```

Parameters

Parameters	Description
int keyid	[in] Id of the key. See remarks.
const gsi_char * key	[in] Name of the key. Player keys should end in "_" (such as "score_") and team keys should end in "_t".

Remarks

The qr2_register_key function tell the qr2 SDK that it should report values for the specified key. Key IDs 0 through NUM_RESERVED_KEYS are reserved for common key names. Keys upward to MAX_REGISTERED_KEYS are available for custom use.

All custom keys should be registered prior to calling qr2_init (see page 163). Reserved keys are already registered and should not be passed to this function.

The names of player keys reported with this function must end in an "_" character and team keys always end in a "_t".

Callbacks

Functions

	Name	Description
●	qr2_register_clientmessage_callback (see page 168)	Sets the function that will be triggered when a client message is received.
●	qr2_register_hostregistered_callback (see page 168)	Sets the function that will be called when the master server has registered the game host as available for connections.
●	qr2_register_natneg_callback (see page 169)	Sets the function that will be triggered when a NAT negotiation request is received.
●	qr2_register_publicaddress_callback (see page 169)	Sets the function that will be triggered when the local client's public address is received.

Types

Name	Description
qr2_adderrorcallback_t (see page 169)	This callback is provided to qr2_init (see page 163); called in response to a message from the master server indicating a problem listing the server.
qr2_clientmessagecallback_t (see page 170)	This callback is set via qr2_register_clientmessage_callback (see page 168); called when a client message is received.
qr2_countcallback_t (see page 170)	One of the callbacks provided to qr2_init (see page 163); called when the SDK needs to get a count of player or teams on the server.
qr2_hostregisteredcallback_t (see page 171)	This callback is set via qr2_register_hostregistered_callback (see page 168); it is called when the master server has registered the game host as available.
qr2_keylistcallback_t (see page 171)	One of the callbacks provided to qr2_init (see page 163); called when the SDK needs to determine all of the keys you game has values for.
qr2_natnegcallback_t (see page 171)	This callback is set via qr2_register_natneg_callback (see page 169); it is called when a NAT negotiation request is received.
qr2_playerteamkeycallback_t (see page 172)	One of the callbacks provided to qr2_init (see page 163); called when a client requests information about a player key or a team key.
qr2_publicaddresscallback_t (see page 172)	This callback is set via qr2_register_publicaddress_callback (see page 169); called when the local client's public address is received.
qr2_serverkeycallback_t (see page 173)	One of the callbacks provided to qr2_init (see page 163), called when a client requests information about a specific server key.

qr2_register_clientmessage_callback Function

Summary

Sets the function that will be triggered when a client message is received.

C++

```
COMMON_API void qr2_register_clientmessage_callback(
    qr2_t qrec,
    qr2_clientmessagecallback_t cmcallback
);
```

Parameters

Parameters	Description
qr2_t qrec	[in] QR2 SDK initialized with qr2_init (see page 163).
qr2_clientmessagecallback_t cmcallback	[in] Function to be called when a client message is received.

Remarks

The qr2_register_clientmessage_callback function is used to set a function that will be triggered when a client message is received.

See Also

qr2_init (see page 163), qr2_clientmessagecallback_t (see page 170)

qr2_register_hostregistered_callback Function

Summary

Sets the function that will be called when the master server has registered the game host as available for connections.

C++

```
COMMON_API void qr2_register_hostregistered_callback(
    qr2_t qrec,
    qr2_hostregisteredcallback_t hrcallback
);
```

Parameters

Parameters	Description
qr2_t qrec	[in] QR2 SDK initialized with qr2_init (see page 163).
qr2_hostregisteredcallback_t hrcallback	[in] Function to be called when the game host has been registered.

See Also

qr2_init (see page 163), qr2_clientconnectedcallback_t

qr2_register_natneg_callback Function

Summary

Sets the function that will be triggered when a NAT negotiation request is received.

C++

```
COMMON_API void qr2_register_natneg_callback(
    qr2_t qrec,
    qr2_natnegcallback_t nncallback
);
```

Parameters

Parameters	Description
qr2_t qrec	[in] QR2 SDK initialized with qr2_init (see page 163).
qr2_natnegcallback_t nncallback	[in] Function to be called when a nat negotiation request is received.

See Also

qr2_init (see page 163), qr2_natnegcallback_t (see page 171)

qr2_register_publicaddress_callback Function

Summary

Sets the function that will be triggered when the local client's public address is received.

C++

```
COMMON_API void qr2_register_publicaddress_callback(
    qr2_t qrec,
    qr2_publicaddresscallback_t pacallback
);
```

Parameters

Parameters	Description
qr2_t qrec	[in] QR2 SDK initialized with qr2_init (see page 163).
qr2_publicaddresscallback_t pacallback	[in] Function to be called when the local client's public address is received.

See Also

qr2_init (see page 163), qr2_publicaddresscallback_t (see page 172)

qr2_adderrorcallback_t Type

Summary

This callback is provided to qr2_init Function (see page 163); called in response to a message from the master server indicating a problem listing the server.

C++

```
typedef void (* qr2_adderrorcallback_t)(qr2_error_t error, gsi_char *errmsg, void
*userdata);
```

Parameters

Parameters	Description
error	[in] The code that can be used to determine the specific listing error.
errmsg	[in] A human-readable error string returned from the master server.
userdata	[in] The userdata that was passed into qr2_init (see page 163).

Remarks

The most common error that will be reported is if the master is unable to list the server due to a firewall or proxy

These types of errors must be appropriate handled and relayed to the user.

See Also

qr2_init (see page 163)

qr2_clientmessagecallback_t Type

Summary

This callback is set via qr2_register_clientmessage_callback Function (see page 168); called when a client message is received.

C++

```
typedef void (* qr2_clientmessagecallback_t)(gsi_char *data, int len, void *userdata);
```

Parameters

Parameters	Description
data	[in] The buffer containing the message
len	[in] The length of the data buffer
userdata	[in] The userdata that was passed into qr2_init (see page 163).

See Also

qr2_init (see page 163), qr2_register_clientmessage_callback (see page 168)

qr2_countcallback_t Type

Summary

One of the callbacks provided to qr2_init Function (see page 163); called when the SDK needs to get a count of player or teams on the server.

C++

```
typedef int (* qr2_countcallback_t)(qr2_key_type keytype, void *userdata);
```

Parameters

Parameters	Description
keytype	[in] Indicates whether the player or team count is being requested (key_player or key_team)
userdata	[in] The same userdata that was passed into qr2_init (see page 163).

Returns

The callback should return the count for either the player or team, as indicated.

Remarks

If your game does not support teams, return 0 for the count of teams.

See Also

qr2_init (see page 163)

qr2_hostregisteredcallback_t Type

Summary

This callback is set via qr2_register_hostregistered_callback Function (see page 168); it is called when the master server has registered the game host as available.

C++

```
typedef void (* qr2_hostregisteredcallback_t)(void *userdata);
```

Parameters

Parameters	Description
userdata	[in] The userdata that was passed into qr2_init (see page 163).

See Also

qr2_register_hostregistered_callback (see page 168)

qr2_keylistcallback_t Type

Summary

One of the callbacks provided to qr2_init Function (see page 163); called when the SDK needs to determine all of the keys you game has values for.

C++

```
typedef void (* qr2_keylistcallback_t)(qr2_key_type keytype, qr2_keybuffer_t keybuffer,
void *userdata);
```

Parameters

Parameters	Description
keytype	[in] The type of keys being requested (server, player, team). You should only add keys of this type to the keybuffer.
keybuffer	[in] The structure that holds the list of keys. Use qr2_keybuffer_add (see page 165) to add a key to the buffer.
userdata	[in] The same userdata that was passed into qr2_init (see page 163).

See Also

qr2_init (see page 163), qr2_keybuffer_add (see page 165)

qr2_natnegcallback_t Type

Summary

This callback is set via qr2_register_natneg_callback Function (see page 169); it is called when a NAT negotiation request is received.

C++

```
typedef void (* qr2_natnegcallback_t)(int cookie, void *userdata);
```

Parameters

Parameters	Description
cookie	[in] The cookie associated with the NAT Negotiation request.
userdata	[in] The userdata that was passed into qr2_init (see page 163).

See Also

qr2_init (see page 163), qr2_register_natneg_callback (see page 169)

qr2_playerteamkeycallback_t Type

Summary

One of the callbacks provided to qr2_init Function (see page 163); called when a client requests information about a player key or a team key.

C++

```
typedef void (* qr2_playerteamkeycallback_t)(int keyid, int index, qr2_buffer_t outbuf,
void *userdata);
```

Parameters

Parameters	Description
keyid	[in] The key being requested.
index	[in] The zero-based index of the player or team being requested.
outbuf	[in] The destination buffer for the value information. Use qr2_buffer_add (see page 162) to report the value.
userdata	[in] The same userdata that was passed into qr2_init (see page 163). You can use this for an object or structure pointer if needed.

Remarks

As a player key callback, this is called when a client requests information about a specific key for a specific player.

As a team key callback, this is called when a client requests the value for a team key.

If you don't have a value for the provided keyid, you should add an empty ("") string to the buffer.

See Also

qr2_init (see page 163), qr2_buffer_add (see page 162)

qr2_publicaddresscallback_t Type

Summary

This callback is set via qr2_register_publicaddress_callback Function (see page 169); called when the local client's public address is received.

C++

```
typedef void (* qr2_publicaddresscallback_t)(unsigned int ip, unsigned short port, void
*userdata);
```

Parameters

Parameters	Description
ip	[in] IP address in string form: xxx.xxx.xxx.xxx

port	[in] Port number
userdata	[in] The userdata that was passed into qr2_init (see page 163).

Remarks

The address is that of the external most NAT or firewall device, and is determined by the GameSpy master server during the qr2_init (see page 163) process.

See Also

qr2_init (see page 163), qr2_register_publicaddress_callback (see page 169)

qr2_serverkeycallback_t Type

Summary

One of the callbacks provided to qr2_init Function (see page 163), called when a client requests information about a specific server key.

C++

```
typedef void (* qr2_serverkeycallback_t)(int keyid, qr2_buffer_t outbuf, void *userdata);
```

Parameters

Parameters	Description
keyid	[in] The key being requested.
index	[in] The 0-based index of the player or team being requested.
outbuf	[in] The destination buffer for the value information. Use qr2_buffer_add (see page 162) to report the value.
userdata	[in] The same userdata that was passed into qr2_init (see page 163).

Remarks

If you don't have a value for the provided keyid, you should add an empty ("") string to the buffer.

See Also

qr2_init (see page 163), qr2_buffer_add (see page 162)

Structures

Types

Name	Description
qr2_buffer_t (see page 174)	This structure stores data that will be sent back to a client in response to a query. Use the qr2_buffer_add (see page 162) functions to add data to the buffer in your callbacks.
qr2_keybuffer_t (see page 174)	This structure is used to store a list of keys when enumerating available keys. Use the qr2_keybuffer_add (see page 165) function to add keys to the list.
qr2_t (see page 174)	This abstract type is used to instantiate multiple instances of the Query & Reporting SDK (for example, if you are running multiple servers in the same process). For most games, you can ignore this value and pass NULL in to all functions that require it. A single global instance will be used in this case.

qr2_buffer_t Type

Summary

This structure stores data that will be sent back to a client in response to a query. Use the qr2_buffer_add Function (see page 162) functions to add data to the buffer in your callbacks.

C++

```
typedef struct qr2_buffer_s * qr2_buffer_t;
```

qr2_keybuffer_t Type

Summary

This structure is used to store a list of keys when enumerating available keys. Use the qr2_keybuffer_add Function (see page 165) function to add keys to the list.

C++

```
typedef struct qr2_keybuffer_s * qr2_keybuffer_t;
```

qr2_t Type

Summary

This abstract type is used to instantiate multiple instances of the Query & Reporting SDK (for example, if you are running multiple servers in the same process). For most games, you can ignore this value and pass NULL in to all functions that require it. A single global instance will be used in this case.

C++

```
typedef struct qr2_implementation_s * qr2_t;
```

Enumerations

Enumerations

Name	Description
qr2_error_t (see page 174)	Constants returned from qr2_init (see page 163) and the error callback to signal an error condition.
qr2_key_type (see page 175)	Keytype indicates the type of keys being referenced: server, player, or team.

qr2_error_t Enumeration

Summary

Constants returned from qr2_init Function (see page 163) and the error callback to signal an error condition.

C++

```
typedef enum {
  e_qrnoerror,
  e_qrwsockerror,
  e_qrbinderror,
  e_qrdnserror,
  e_qrconnerror,
  e_qrnochallengeerror,
  e_qrnotauthenticated,
  qr2_error_t_count
} qr2_error_t;
```

Members

Members	Description
e_qrnoerror	No error occurred.
e_qrwsockerror	A standard socket call failed (e.g., exhausted resources).
e_qrbinderror	The SDK was unable to find an available port on which to bind.
e_qrdnserror	A DNS lookup (for the master server) failed.
e_qrconnerror	The server is behind a NAT and does not support negotiation.
e_qrnochallengeerror	No challenge was received from the master. The common reasons for this error are: 1. Not enabling the NatNegotiate flag in the peerSetTitle or qr2_init calls: this should be PEERTrue (or 1 for QR2) for games that support NATs. Otherwise, the master server will assume this server is not behind a NAT and can directly connect to it. 2. Calling qr2_buffer_add more than once on a particular key-value. 3. Firewall or NAT configuration is blocking incoming traffic. You may need to open ports to allow communication (see related). 4. Using the same port for socket communications: shared socket implementations with qr2/peer together. 5. The heartbeat packet has exceeded the max buffer size of 1400 bytes. Try abbreviating some of the custom keys to fit within the 1400 byte buffer. The reason for this restriction is to support as many routers as possible, as UDP packets beyond this data range are more inclined to be dropped. 6. "numplayers" or "maxplayers" being set to negative values. 7. Having 2 network adapters connected to an internal and external network, and the internal one set as primary.
e_qrnotauthenticated	Did not use AuthService, deny service.

qr2_key_type Enumeration

Summary

Keytype indicates the type of keys being referenced: server, player, or team.

C++

```
typedef enum {
  key_server,
  key_player,
  key_team,
  key_type_count
} qr2_key_type;
```

Members

Members	Description
key_server	General information about the game in progress.
key_player	Information about a specific player.
key_team	Information about a specific team.

Sake

API Documentation

Module

Sake (see page 176)

Functions

Functions

	Name	Description
◆	sakeCreateRecord (see page 177)	Creates a new Record in a Sake (see page 176) table.
◆	sakeGetFieldByName (see page 177)	This utility function retrieves the specified field from the record, via the name that identifies it.
◆	sakeGetMyRecords (see page 178)	Gets all of the records owned by the local player from a table.
◆	sakeGetRandomRecord (see page 178)	Retrieves a random record from the provided search criteria.
◆	sakeGetRecordCount (see page 179)	Gets a count for the number of records in a table based on the given filter criteria.
◆	sakeGetRecordLimit (see page 179)	Checks the maximum number of records that a profile can own for a particular table.
◆	sakeGetSpecificRecords (see page 180)	Gets a list of specific records from a table.
◆	sakeGetStartRequestResult (see page 181)	Called to retrieve the result of a request. Normally used to determine the reason for a failed request.
◆	sakeRateRecord (see page 181)	Rates a specified record.
◆	sakeSetGame (see page 182)	Authenticates the game to use Sake (see page 176).
◆	sakeShutdown (see page 182)	Shuts down the SDK and frees any memory that was allocated for the Sake (see page 176) object.
◆	sakeStartup (see page 183)	Initializes the Sake (see page 176) SDK for use.
◆	sakeDeleteRecord (see page 183)	Deletes the specified record.
◆	sakeUpdateRecord (see page 184)	Updates the values stored in an existing record.
◆	sakeGetFileDownloadURL (see page 184)	Used to get a download URL for a particular file id.
◆	sakeGetFileIdFromHeaders (see page 185)	If the file was uploaded successfully, this function obtains the file id that references the file.
◆	sakeGetFileResultFromHeaders (see page 185)	Checks the headers from the uploaded file to see the result.
◆	sakeGetFileUploadURL (see page 186)	Retrieves the URL which can be used to upload files.
◆	sakeSearchForRecords (see page 186)	Searches a table for records that match certain specified criteria.

◆	sakeCancelRequest (see page 187)	Used for cancelling any outstanding requests made to the Sake (see page 176) Storage Server.
◆	sakeDownloadContent (see page 187)	This API function allows downloading content from the backend service either to a memory location or to a local file.
◆	sakeSetProfile (see page 188)	Provides Sake (see page 176) authentication information for the current player.
◆	sakeThink (see page 188)	This function is used for processing the ghttp request and responses. It should be called in the main loop.
◆	sakeUploadContent (see page 188)	This API function allows uploading content either from a memory location or from a file to the server.

sakeCreateRecord Function

Summary

Creates a new Record in a Sake (see page 176) table.

C++

```
COMMON_API SAKERequest SAKE_CALL sakeCreateRecord(
    SAKE sake,
    SAKECreateRecordInput * input,
    SAKERequestCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
SAKERequestCallback callback	[in] The request callback function.
userdata	[in] Pointer to user-specified data sent to the request callback.

Returns

Reference to internal object that tracks the request. If this is NULL, then the request has failed to initialize. You can call sakeGetStartRequestResult (see page 181) to obtain the reason for the failure.

Remarks

If the request completed successfully, then the output object contains the recordid of the newly created record.

See Also

SAKECreateRecordInput, SAKECreateRecordOutput (see page 193), SAKERequestCallback (see page 190)

sakeGetFieldByName Function

Summary

This utility function retrieves the specified field from the record, via the name that identifies it.

C++

```
COMMON_API SAKEField * SAKE_CALL sakeGetFieldByName(
    const char * name,
    SAKEField * fields,
    int numFields
);
```

Parameters

Parameters	Description
const char * name	[in] The name of the field to retrieve.
SAKEField * fields	[in] An array of fields, representing a record.

int numFields	[in] The number of fields in the array.

Returns

Pointer to a SAKEField (see page 195) which represents the field that was identified by the given name.

See Also

SAKEField (see page 195)

sakeGetMyRecords Function

Summary

Gets all of the records owned by the local player from a table.

C++

```
COMMON_API SAKERequest SAKE_CALL sakeGetMyRecords(
    SAKE sake,
    SAKEGetMyRecordsInput * input,
    SAKERequestCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
SAKERequestCallback callback	[in] The request callback function.
void * userData	[in] Pointer to user-specified data sent to the request callback.

Returns

Reference to internal object that tracks the request. If this is NULL, then the request has failed to initialize. You can call sakeGetStartRequestResult (see page 181) to obtain the reason for the failure.

Remarks

If the request completed successfully, then the output object contains all of the records which the local player owns in the table. See the definitions of the Input & Output structs for more information about how to limit what is retrieved in the request and certain metadeta fields that can be retrieved.

See Also

SAKEGetMyRecordsInput (see page 195), SAKEGetMyRecordsOutput (see page 196), SAKERequestCallback (see page 190), sakeGetStartRequestResult (see page 181)

sakeGetRandomRecord Function

Summary

Retrieves a random record from the provided search criteria.

C++

```
COMMON_API SAKERequest SAKE_CALL sakeGetRandomRecord(
    SAKE sake,
    SAKEGetRandomRecordInput * input,
    SAKERequestCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.

SAKERequestCallback callback	[in] The request callback function.
void * userData	[in] Pointer to user-specified data sent to the request callback.

Returns

Reference to internal object that tracks the request. If this is NULL, then the request has failed to initialize. You can call sakeGetStartRequestResult (see page 181) to obtain the reason for the failure.

Remarks

The output will always be a single record (unless no records pass the filter, in which case the output will contain NULL data for the returned record). Note that this function works best in a table in which records are not deleted or are deleted in order of oldest first (in other words, in tables in which recordids are contiguous).

See Also

SAKEGetRandomRecordInput (see page 196), SAKEGetRandomRecordOutput (see page 197), SAKERequestCallback (see page 190), sakeGetStartRequestResult (see page 181)

sakeGetRecordCount Function

Summary

Gets a count for the number of records in a table based on the given filter criteria.

C++

```
COMMON_API SAKERequest SAKE_CALL sakeGetRecordCount(
    SAKE sake,
    SAKEGetRecordCountInput * input,
    SAKERequestCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
SAKERequestCallback callback	[in] The request callback function.
void * userData	[in] Pointer to user-specified data sent to the request callback.

Returns

Reference to internal object that tracks the request. If this is NULL, then the request has failed to initialize. You can call sakeGetStartRequestResult (see page 181) to obtain the reason for the failure.

Remarks

If the request completed successfully, then the Output object contains info about the count for the specified table.

Only call this function very occasionally. Example: after the first call for a specific leaderboard, the information this function returns will be the same, so it should be cached for subsequent page calls.

See Also

SAKEGetRecordCountInput (see page 197), SAKEGetRecordCountOutput (see page 197), SAKERequestCallback (see page 190)

sakeGetRecordLimit Function

Summary

Checks the maximum number of records that a profile can own for a particular table.

C++

```
COMMON_API SAKERequest SAKE_CALL sakeGetRecordLimit(
    SAKE sake,
    SAKEGetRecordLimitInput * input,
    SAKERequestCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
SAKERequestCallback callback	[in] The request callback function.
void * userData	[in] Pointer to user-specified data sent to the request callback.

Returns

Reference to internal object that tracks the request. If this is NULL, then the request has failed to initialize. You can call sakeGetStartRequestResult (see page 181) to obtain the reason for the failure.

Remarks

If the request completed successfully, then the Output object contains info about the record limit for the specified table.

See Also

SAKEGetRecordLimitInput (see page 198), SAKEGetRecordLimitOutput (see page 198), SAKERequestCallback (see page 190), sakeGetStartRequestResult (see page 181)

sakeGetSpecificRecords Function

Summary

Gets a list of specific records from a table.

C++

```
COMMON_API SAKERequest SAKE_CALL sakeGetSpecificRecords(
    SAKE sake,
    SAKEGetSpecificRecordsInput * input,
    SAKERequestCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
SAKERequestCallback callback	[in] The request callback function.
void * userData	[in] Pointer to user-specified data sent to the request callback.

Returns

Reference to internal object that tracks the request. If this is NULL, then the request has failed to initialize. You can call sakeGetStartRequestResult (see page 181) to obtain the reason for the failure.

Remarks

If the request completed successfully, then the output object contains all of the records which were specified in the request. See the definitions of the Input & Output structs for more information about how to limit what is retrieved in the request and certain metadeta fields that can be retrieved.

See Also

SAKEGetSpecificRecordsInput (see page 198), SAKEGetSpecificRecordsOutput (see page 199), SAKERequestCallback (see page 190)

sakeGetStartRequestResult Function

Summary

Called to retrieve the result of a request. Normally used to determine the reason for a failed request.

C++

```
COMMON_API SAKEStartRequestResult SAKE_CALL sakeGetStartRequestResult(
    SAKE sake
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object

Returns

Enum value used to indicate the specific result of the request.

Remarks

This function will always return the most recent request that was attempted, so it must be called immediately after a failure to get the reason for that failure.

See Also

SAKEStartRequestResult (see page 206)

sakeRateRecord Function

Summary

Rates a specified record.

C++

```
COMMON_API SAKERequest SAKE_CALL sakeRateRecord(
    SAKE sake,
    SAKERateRecordInput * input,
    SAKERequestCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
SAKERequestCallback callback	[in] The request callback function.
userdata	[in] Pointer to user-specified data sent to the request callback.

Returns

Reference to internal object that tracks the request. If this is NULL, then the request has failed to initialize. You can call sakeGetStartRequestResult (see page 181) to obtain the reason for the failure.

Remarks

The range of ratings which Sake (see page 176) supports is 0 to 255. However, a game can restrict itself to a subset of that range. For example, a game may want to use a rating of 1 to 5 (a star rating), or it may want to use a range of 0 to 100.

Sake (see page 176) allows users to rate records which they own, however no profile can rate the same record more than once.

See Also

SAKERateRecordInput (see page 199), SAKERequestCallback (see page 190)

sakeSetGame Function

Summary

Authenticates the game to use Sake (see page 176).

C++

```
COMMON_API void SAKE_CALL sakeSetGame(
    SAKE sake,
    const gsi_char * gameName,
    int gameId,
    const gsi_char * secretKey
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
const gsi_char * gameName	[in] Your title's gamename, as assigned by GameSpy.
int gameId	[in] Your title's gameid, as assigned by GameSpy.
const gsi_char * secretKey	[in] Your title's secret key, as assigned by GameSpy.

Remarks

The function provides no indication of whether or not the gamename and gameid are correct. If they are not correct, your subsequent Sake (see page 176) requests may fail.

Your game will also need to call sakeSetProfile (see page 188) to authenticate the current player before making Sake (see page 176) requests.

See Also

sakeSetProfile (see page 188)

sakeShutdown Function

Summary

Shuts down the SDK and frees any memory that was allocated for the Sake (see page 176) object.

C++

```
COMMON_API void SAKE_CALL sakeShutdown(
    SAKE sake
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.

Remarks

After this function returns, the reference to the Sake (see page 176) object is no longer valid and should not be used. The game should also shutdown the GameSpy Core object by calling gsCoreShutdown. Sample code for this is available in the Sake (see page 176) test app.

sakeStartup Function

Summary

Initializes the Sake (see page 176) SDK for use.

C++

```
COMMON_API SAKEStartupResult SAKE_CALL sakeStartup(
    SAKE * sakePtr
);
```

Parameters

Parameters	Description
SAKE * sakePtr	[in] Pointer to a Sake (see page 176) object, which is initialized by startup. This will be used in nearly all subsequent Sake (see page 176) calls.

Returns

An enumeration of possible results. If the result is SAKEStartupResult_SUCCESS, then the startup has succeeded. Any other value indicates a failure, and the game should not continue calling other Sake (see page 176) functions.

Remarks

Before using Sake (see page 176), the GameSpy Availability Check must have been performed and indicated that the GameSpy services are available for this game title and the Core object must have been initialized by calling gsCoreInitialize. Sample code for this is available in the Sake (see page 176) test app.

The Sake (see page 176) object initialized by this startup call is valid until the game shutdowns the Sake (see page 176) SDK with sakeShutdown (see page 182).

See Also

SAKEStartupResult (see page 207)

sakeDeleteRecord Function

Summary

Deletes the specified record.

C++

```
COMMON_API SAKERequest SAKE_CALL sakeDeleteRecord(
    SAKE sake,
    SAKEDeleteRecordInput * input,
    SAKERequestCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
SAKERequestCallback callback	[in] The request callback function.
void * userData	[in] Pointer to user-specified data sent to the request callback.

Returns

Reference to internal object that tracks the request. If this is NULL, then the request has failed to initialize. You can call sakeGetStartRequestResult (see page 181) to obtain the reason for the failure.

Remarks

DeleteRecord does not have an output object, because the Sake (see page 176) service response only indicates success or failure in the registered callback function. When the registered callback is invoked, the outputData parameter will always be set to NULL.

See Also

SAKEDeleteRecordInput (see page 194), SAKERequestCallback (see page 190)

sakeUpdateRecord Function

Summary

Updates the values stored in an existing record.

C++

```
COMMON_API SAKERequest SAKE_CALL sakeUpdateRecord(
    SAKE sake,
    SAKEUpdateRecordInput * input,
    SAKERequestCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
SAKERequestCallback callback	[in] The request callback function.
void * userData	[in] Pointer to user-specified data sent to the request callback.

Returns

Reference to internal object that tracks the request. If this is NULL, then the request has failed to initialize. You can call sakeGetStartRequestResult (see page 181) to obtain the reason for the failure.

Remarks

UpdateRecord does not have an output object, because the Sake (see page 176) service response only indicates success or failure in the registered callback function. When the registered callback is invoked, the outputData parameter will always be set to NULL.

See Also

SAKEUpdateRecordInput (see page 203), SAKERequestCallback (see page 190), sakeGetStartRequestResult (see page 181)

sakeGetFileDownloadURL Function

Summary

Used to get a download URL for a particular file id.

C++

```
COMMON_API gsi_bool SAKE_CALL sakeGetFileDownloadURL(
    SAKE sake,
    int fileId,
    gsi_char url[SAKE_MAX_URL_LENGTH]
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
gsi_char url[SAKE_MAX_URL_LENGTH]	[out] The download url for the specified file.

fileid	[in] The file id returned by the headers of the uploaded file. Call sakeGetFileIdFromHeaders (see page 185) to obtain this.

Returns

gsi_true if download url was retrieved successfully, gsi_false otherwise.

See Also

sakeGetFileIdFromHeaders (see page 185)

sakeGetFileIdFromHeaders Function

Summary

If the file was uploaded successfully, this function obtains the file id that references the file.

C++

```
COMMON_API gsi_bool SAKE_CALL sakeGetFileIdFromHeaders(
    const char * headers,
    int * fileId
);
```

Parameters

Parameters	Description
const char * headers	[in] The headers to parse for the file id. You can get these by calling ghttpGetHeaders (see page 69).
int * fileId	[ref] The file id returned by SakeFileServer when the file was uploaded.

Returns

gsi_true if able to parse the file id successfully, gsi_false otherwise.

Remarks

To get the file id from the headers manually, look for the Sake (see page 176)-File-Id Eheader. Once obtained, the file id can now be stored in a file id field in the database.

Notes

If you upload a file, but do not store the file id in the database, then the Sake (see page 176) service may automatically delete the file after approximately 24 hours.

See Also

ghttpGetHeaders (see page 69)

sakeGetFileResultFromHeaders Function

Summary

Checks the headers from the uploaded file to see the result.

C++

```
COMMON_API gsi_bool SAKE_CALL sakeGetFileResultFromHeaders(
    const char * headers,
    SAKEFileResult * result
);
```

Parameters

Parameters	Description
const char * headers	[in] The headers to parse for the file id. You can get these from ghttpGetHeaders (see page 69).

SAKEFileResult * result	[ref] Reference to the result as obtained in the headers.

Returns

gsi_true if it was able to parse the result successfully, gsi_false otherwise.

Remarks

You can also check the headers manually for the Sake (see page 176)-File-Result•E Eheader. The value stored in the header is an integer, the possible values of which are enumerated in SAKEFileResult (see page 204). SAKEFileResult_SUCCESS means that the file was uploaded successfully, while any other value indicates that there was an error uploading the file.

See Also

SAKEFileResult (see page 204)

sakeGetFileUploadURL Function

Summary

Retrieves the URL which can be used to upload files.

C++

```
COMMON_API gsi_bool SAKE_CALL sakeGetFileUploadURL(
    SAKE sake,
    gsi_char url[SAKE_MAX_URL_LENGTH]
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
gsi_char url[SAKE_MAX_URL_LENGTH]	[out] The URL where the file can be uploaded.

Returns

gsi_true if upload url was retrieved successfully, gsi_false otherwise.

sakeSearchForRecords Function

Summary

Searches a table for records that match certain specified criteria.

C++

```
COMMON_API SAKERequest SAKE_CALL sakeSearchForRecords(
    SAKE sake,
    SAKESearchForRecordsInput * input,
    SAKERequestCallback callback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
SAKERequestCallback callback	[in] The request callback function.
void * userData	[in] Pointer to user-specified data sent to the request callback.

Returns

Reference to internal object that tracks the request. If this is NULL, then the request has failed to initialize. You can call sakeGetStartRequestResult (see page 181) to obtain the reason for the failure.

Remarks

If the request completed successfully, then the output object contains contains records founds by the search. See the definitions of the Input & Output structs for more information about how to limit what is retrieved in the request and certain metadeta fields that can be retrieved.

See Also

SAKESearchForRecordsInput (see page 200), SAKESearchForRecordsOutput (see page 202), SAKERequestCallback (see page 190), sakeGetStartRequestResult (see page 181)

sakeCancelRequest Function

Summary

Used for cancelling any outstanding requests made to the Sake (see page 176) Storage Server.

C++

```
COMMON_API SAKEStartRequestResult SAKE_CALL sakeCancelRequest(
    SAKE sake,
    SAKERequest request
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
SAKERequest request	[in] The returned value from the original request to be cancelled.

Returns

SAKEStartRequestResult_SUCCESS if successful.

Remarks

To be used for cancelling requests which taking too long or unresponsive. No actions are cancelled at the backend if the request was processed and response was not received yet. The cancel request only to prevent the SDK waiting indefinitely for a response from the backend.

sakeDownloadContent Function

Summary

This API function allows downloading content from the backend service either to a memory location or to a local file.

C++

```
COMMON_API SAKEStartRequestResult SAKE_CALL sakeDownloadContent(
    SAKE sake,
    SAKEDownloadContentInput * input,
    SAKEDownloadContentCompletedCallback completedCallback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The instance of the SDK.
SAKEDownloadContentCompletedCallback completedCallback	[in] Developer's callback function when the download completes.
void * userData	[in/out] Optional. The developer specified void pointer data shared between the API and its callback. Set to NULL if unused.

progressCallback	[in] Optional. Developer's callback function to keep track of the downloading. Set to NULL if unused.

Returns

SAKEStartRequestResult_SUCCESS if successful.

Remarks

When downloading to memory, there is no need to preallocate memory for the buffer prior to calling this function. The buffer pointer and the buffer length, are updated with the memory location and the size by the SDK. For downloading to a file, the file path must exist

sakeSetProfile Function

Summary

Provides Sake (see page 176) authentication information for the current player.

C++

```
COMMON_API void SAKE_CALL sakeSetProfile(
    SAKE sake,
    int profileId,
    const GSLoginCertificate * certificate,
    const GSLoginPrivateData * privateData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The Sake (see page 176) object.
int profileId	[in] Current player's profile ID.
const GSLoginCertificate * certificate	[in] A valid certificate obtained from the GameSpy AuthService.
const GSLoginPrivateData * privateData	[in] Valid private data obtained from the GameSpy AuthService.

Remarks

The profile id, cert, and proof are obtained from the AuthService callback (e.g., after calling wsLoginUnique (see page 48)).

As with sakeSetGame (see page 182), sakeSetProfile provides no indication of whether or not the information provided is correct. The Sake (see page 176) service checks these values and uses them to authenticate the player and, for certain requests, to identify which player's data is being accessed or updated.

See Also

AuthserviceWSLoginProfile (or WSLoginUnique, or WSLoginRemote), sakeSetGame (see page 182)

sakeThink Function

Summary

This function is used for processing the ghttp request and responses. It should be called in the main loop.

C++

```
COMMON_API void sakeThink();
```

sakeUploadContent Function

Summary

This API function allows uploading content either from a memory location or from a file to the server.

C++

```
COMMON_API SAKEStartRequestResult SAKE_CALL sakeUploadContent(
    SAKE sake,
    SAKEUploadContentInput * input,
    SAKEUploadContentCompletedCallback completedCallback,
    void * userData
);
```

Parameters

Parameters	Description
SAKE sake	[in] The instance of the SDK.
SAKEUploadContentCompletedCallback completedCallback	[in] Developer's callback function when the upload completes.
void * userData	[in/out] Optional. The developer specified void pointer data shared between the API and its callback. Set to NULL if unused.

Returns

SAKEStartRequestResult_SUCCESS if successful.

Remarks

This API call should be invoked only when a new file for the content is being created at the back end.

Callbacks

Types

Name	Description
SAKEDownloadContentCompletedCallback (see page 189)	This typedef defines the callback function called when the Download Content request completes. After this is called if successful, the content is downloaded to the specified memory location or to a file.
SAKEDownloadContentProgressCallback (see page 190)	This typedef defines the progress callback function for the download content API. It is invoked periodically to indicate data transfer progress.
SAKERequestCallback (see page 190)	This typedef defines the general callback function.
SAKEUploadContentCompletedCallback (see page 191)	This typedef defines the callback function called when the Upload Content request completes. After this is called if successful, the data is uploaded to the back end and a new file id is assigned.
SAKEUploadContentProgressCallback (see page 191)	This typedef defines the progress callback function for the upload content API. It is invoked periodically to indicate data transfer progress.

SAKEDownloadContentCompletedCallback Type

Summary

This typedef defines the callback function called when the Download Content request completes. After this is called if successful, the content is downloaded to the specified memory location or to a file.

C++

```
typedef void (* SAKEDownloadContentCompletedCallback)(SAKE sake, SAKERequestResult result,
SAKEFileResult fileResult, char *buffer, gsi_i32 bufferLength, void *userData);
```

Parameters

Parameters	Description
sake	[in] The Sake (see page 176) instance the callback is running on.
result	[in] The download file result as it is returned by the service call.

buffer	[in] The buffer which contains the downloaded content, if downloading to memory. Otherwise, it is NULL.
bufferLength	[in] The buffer length for the downloaded content, if downloading to memory. Otherwise, it is 0.
userData	[in/out] The developer specified void pointer data shared between the API and its callback.

Remarks

IMPORTANT: If downloading to memory, the developer must delete the allocated memory by the callback when the buffer is no longer needed.

See Also

SAKEFileResult (see page 204), SAKEDownloadContentProgressCallback (see page 190)

SAKEDownloadContentProgressCallback Type

Summary

This typedef defines the progress callback function for the download content API. It is invoked periodically to indicate data transfer progress.

C++

```
typedef void (* SAKEDownloadContentProgressCallback)(SAKE sake, gsi_u32 bytesTransfered,
gsi_u32 totalSize, void *userData);
```

Parameters

Parameters	Description
sake	[in] The Sake (see page 176) instance the callback is running on.
bytesTransfered	[in] The number of bytes currently download.
totalSize	[in] The total number of bytes to be download.
userData	[in/out] The developer specified void pointer data shared between the API and its callback.

See Also

SAKEDownloadContentCompletedCallback (see page 189)

SAKERequestCallback Type

Summary

This typedef defines the general callback function.

C++

```
typedef void (* SAKERequestCallback)(SAKE sake, SAKERequest request, SAKERequestResult
result, void *inputData, void *outputData, void *userData);
```

Parameters

Parameters	Description
sake	[in] The Sake (see page 176) instance the callback is running on.
request	[in] The request data containing everything related.
inputData	[in] Pointer for the input data to API call.
outputData	[in] Pointer for the input data from API callback function.
userData	[in/out] The developer specified void pointer data shared between the API and its callback.

Remarks

Not all request types have output objects. If a request type does not have an output object, then outputData will be always be NULL when the callback is called.

See Also

SAKERequestResult (see page 205)

SAKEUploadContentCompletedCallback Type

Summary

This typedef defines the callback function called when the Upload Content request completes. After this is called if successful, the data is uploaded to the back end and a new file id is assigned.

C++

```
typedef void (* SAKEUploadContentCompletedCallback)(SAKE sake, SAKERequestResult result,
gsi_i32 fileId, SAKEFileResult fileResult, void *userData);
```

Parameters

Parameters	Description
sake	[in] The Sake (see page 176) instance the callback is running on.
fileId	[in] A unique new file id to point to the content at the server.
result	[in] The upload file result as it is returned by the service call.
userData	[in/out] The developer specified void pointer data shared between the API and its callback.
IMPORTANT	If uploading from memory developer must delete the allocated memory when the buffer is no longer needed.

Remarks

IMPORTANT: If uploading from memory developer must delete the allocated memory when the buffer is no longer needed.

See Also

SAKEFileResult (see page 204), SAKEUpdateContentProgressCallback

SAKEUploadContentProgressCallback Type

Summary

This typedef defines the progress callback function for the upload content API. It is invoked periodically to indicate data transfer progress.

C++

```
typedef void (* SAKEUploadContentProgressCallback)(SAKE sake, gsi_u32 bytesTransfered,
gsi_u32 totalSize, void *userData);
```

Parameters

Parameters	Description
sake	[in] The Sake (see page 176) instance the callback is running on.
bytesTransfered	[in] The number of bytes currently uploaded.
totalSize	[in] The total number of bytes to be uploaded.
userData	[in/out] The developer specified void pointer data shared between the API and its callback.

See Also

SAKEUploadContentCompletedCallback (see page 191)

Structures

Structures

	Name	Description
	SAKEBinaryData (see page 193)	Data struct used to store arbitrary binary data in a Sake (see page 176) field.
	SAKECreateRecordOutput (see page 193)	Returned output object that specifies the recordid for the newly created record.
◆	SAKEContentBuffer (see page 193)	Memory location and the length for the locally stored content. Members/Constants mBuffer : [in/out] Pointer to the memory location. mLength : [in/out] Size of buffer.
	SAKEDeleteRecordInput (see page 194)	Input object passed to sakeDeleteRecord (see page 183).
◆	SAKEContentInfo (see page 194)	Structure to pass all the related information regarding content to the API calls.
◆	SAKEField (see page 195)	Object used to represent the field of a record.
	SAKEGetMyRecordsInput (see page 195)	Input object passed to sakeGetMyRecords (see page 178).
	SAKEGetMyRecordsOutput (see page 196)	Returned output object that specifies all of the records which the local player owns in the table.
	SAKEGetRandomRecordInput (see page 196)	Input object passed to sakeGetRandomRecord (see page 178).
	SAKEGetRandomRecordOutput (see page 197)	Returned output object that contains a random record.
	SAKEGetRecordCountInput (see page 197)	Input object passed to sakeGetRecordCount (see page 179).
	SAKEGetRecordCountOutput (see page 197)	Returned record count based on the specified table and search filter used.
	SAKEGetRecordLimitInput (see page 198)	Input object passed to sakeGetRecordLimit (see page 179).
	SAKEGetRecordLimitOutput (see page 198)	Returned output object that specifies the maximum number of records that a profile can own in the table.
	SAKEGetSpecificRecordsInput (see page 198)	Input object passed to sakeGetSpecificRecords (see page 180).
	SAKEGetSpecificRecordsOutput (see page 199)	Returned output object that contains all of the records which were specified in the request.
	SAKERateRecordInput (see page 199)	Input object passed to sakeRateRecord (see page 181).
	SAKERateRecordOutput (see page 200)	Returned output object that lists the new number of ratings and the new average rating for the specified record.
	SAKESearchForRecordsInput (see page 200)	Input object passed to sakeSearchForRecords (see page 186). Members/Constants mTableId mFieldNames mNumFields mFilter mSort mOffset mMaxRecords mTargetRecordFilter mSurroundingRecordsCount mOwnerIds mNumOwnerIds mCacheFlag
	SAKESearchForRecordsOutput (see page 202)	Returned output object that contains the records founds by the search.
	SAKEUpdateRecordInput (see page 203)	Input object passed to sakeUpdateRecord (see page 184).
	SAKEReportRecordInput (see page 203)	Input object passed to sakeReportRecord.

SAKEBinaryData Structure

Summary

Data struct used to store arbitrary binary data in a Sake (see page 176) field.

C++

```
typedef struct {
  gsi_u8 * mValue;
  int mLength;
} SAKEBinaryData;
```

Members

Members	Description
gsi_u8 * mValue;	Pointer to the data.
int mLength;	The number of bytes of data.

Remarks

mValue may be NULL if mLength is 0.

See Also

SAKEFieldType

SAKECreateRecordOutput Structure

Summary

Returned output object that specifies the recordid for the newly created record.

C++

```
typedef struct {
  int mRecordId;
} SAKECreateRecordOutput;
```

Members

Members	Description
int mRecordId;	The recordid for the newly created record.

See Also

sakeCreateRecord (see page 177), SAKERequestCallback (see page 190)

SAKEContentBuffer Structure

Summary

Memory location and the length for the locally stored content. Members/Constants mBuffer : [in/out] Pointer to the memory location. mLength : [in/out] Size of buffer.

C++

```
struct SAKEContentBuffer {
  char * mBuffer;
  gsi_i32 mLength;
};
```

Members

Members	Description
char * mBuffer;	Pointer to the memory location.
gsi_i32 mLength;	Size of buffer.

Remarks

This used when the content is saved in a memory location.

See Also

SAKEContentStorage (see page 208)

SAKEDeleteRecordInput Structure

Summary

Input object passed to sakeDeleteRecord Function (see page 183).

C++

```
typedef struct {
  char * mTableId;
  int mRecordId;
} SAKEDeleteRecordInput;
```

Members

Members	Description
char * mTableId;	Points to the tableid of the table in which the record to be deleted exists.
int mRecordId;	Identifies the record to be deleted.

Remarks

DeleteRecord does not have an output object, because the Sake (see page 176) service response only indicates success or failure in the registered callback function. When the registered callback is invoked, the outputData parameter will always be set to NULL.

See Also

sakeDeleteRecord (see page 183)

SAKEContentInfo Structure

Summary

Structure to pass all the related information regarding content to the API calls.

C++

```
struct SAKEContentInfo {
  gsi_i32 mFileid;
  SAKEContentStorageType mType;
  SAKEContentStorage mStorage;
};
```

Members

Members	Description
gsi_i32 mFileid;	File id for the content stored at the backend.
SAKEContentStorageType mType;	The location (disk or memory) of the content locally.
SAKEContentStorage mStorage;	Either memory and length or a file name.

Remarks

The mFileid refers to the file id stored by the service.

See Also

SAKEContentStorageType (see page 207), SAKEContentStorage (see page 208), sakeUploadContent (see page 188), sakeDownloadContent (see page 187), sakeUpdateContent

SAKEField Structure

Summary

Object used to represent the field of a record.

C++

```
struct SAKEField {
  char * mName;
  SAKEFieldType mType;
  SAKEValue mValue;
};
```

Members

Members	Description
char * mName;	The name used to identify the field.
SAKEFieldType mType;	The type of data stored in the field.
SAKEValue mValue;	The value that will be stored in the field.

See Also

SAKEFieldType, SAKEBinaryData (see page 193)

SAKEGetMyRecordsInput Structure

Summary

Input object passed to sakeGetMyRecords Function (see page 178).

C++

```
typedef struct {
  char * mTableId;
  char ** mFieldNames;
  int mNumFields;
} SAKEGetMyRecordsInput;
```

Members

Members	Description
char * mTableId;	Points to the tableid of the table from which to return records.
char ** mFieldNames;	Points to an array of strings, each of which contains the name of a field for which to return values.
int mNumFields;	Stores the number of strings in the mFieldNames array. This list controls the values which will be returned as part of the response. The array can contain just one field name, the names of all the fields in the table, or any subset of the field names.

Remarks

In addition to the fields which the developer defines, you can also request values for the "recordid" field, "ownerid" field (if the table has an owner-type of profile), and "num_ratings" and "average_rating" fields (if the table has its ratings option set to true).

See Also

sakeGetMyRecords (see page 178)

SAKEGetMyRecordsOutput Structure

Summary

Returned output object that specifies all of the records which the local player owns in the table.

C++

```
typedef struct {
  int mNumRecords;
  SAKEField ** mRecords;
} SAKEGetMyRecordsOutput;
```

Members

Members	Description
int mNumRecords;	The number of records found.
SAKEField ** mRecords;	Points an array of records, each of which is represented as an array of fields.

See Also

sakeGetMyRecords (see page 178), SAKERequestCallback (see page 190), SAKEField (see page 195)

SAKEGetRandomRecordInput Structure

Summary

Input object passed to sakeGetRandomRecord Function (see page 178).

C++

```
typedef struct {
  char * mTableId;
  char ** mFieldNames;
  int mNumFields;
  gsi_char * mFilter;
} SAKEGetRandomRecordInput;
```

Members

Members	Description
char * mTableId;	Points to the tableid of the table to be searched.
char ** mFieldNames;	Points to an array of strings, each of which contains the name of a field for which to return values. This list controls the values which will be returned as part of the response. The array can contain just one field name, the names of all the fields in the table, or any subset of the field names.
int mNumFields;	Stores the number of strings in the mFieldNames array.
gsi_char * mFilter;	SQL-like filter string which is used to filter which records are to be looked at when choosing a random record. Note that if the search criteria is too specific and no records are found, then the output will return no random record. Note that a field can be used in the filter string even if it is not listed in the mFieldNames array, and that file metadata fields can be used in a filter string.

Remarks

In addition to the fields which the developer defines, you can also request values for the "recordid" field, "ownerid" field (if the table has an owner type of profile), and "num_ratings" and "average_rating" fields (if the table has its ratings option set to true).

See Also

sakeGetRandomRecord (see page 178)

SAKEGetRandomRecordOutput Structure

Summary

Returned output object that contains a random record.

C++

```
typedef struct {
  SAKEField * mRecord;
} SAKEGetRandomRecordOutput;
```

Members

Members	Description
SAKEField * mRecord;	An array of fields representing the random record.

Remarks

If no record was found due to constrained search criteria, the returned record will be set to NULL.

SAKEGetRecordCountInput Structure

Summary

Input object passed to sakeGetRecordCount Function (see page 179).

C++

```
typedef struct {
  char * mTableId;
  gsi_char * mFilter;
  gsi_bool mCacheFlag;
} SAKEGetRecordCountInput;
```

Members

Members	Description
char * mTableId;	Points to the tableid of the table to be searched.
gsi_char * mFilter;	SQL-like filter string which is used to filter which records are to be looked at when getting the record count.
gsi_bool mCacheFlag;	Enables caching if set to gsi_true. Defaults to no caching if none is specified.

See Also

sakeGetRecordCount (see page 179)

SAKEGetRecordCountOutput Structure

Summary

Returned record count based on the specified table and search filter used.

C++

```
typedef struct {
  int mCount;
} SAKEGetRecordCountOutput;
```

Members

Members	Description
int mCount;	Contains the value of the record count. If no records exist or the search criteria was too specific so that no records were found, this value will be 0.

See Also

sakeGetRecordCount (see page 179), SAKERequestCallback (see page 190)

SAKEGetRecordLimitInput Structure

Summary

Input object passed to sakeGetRecordLimit Function (see page 179).

C++

```
typedef struct {
  char * mTableId;
} SAKEGetRecordLimitInput;
```

Members

Members	Description
char * mTableId;	Points to the tableid of the table for which to check the limit.

See Also

sakeGetRecordLimit (see page 179)

SAKEGetRecordLimitOutput Structure

Summary

Returned output object that specifies the maximum number of records that a profile can own in the table.

C++

```
typedef struct {
  int mLimitPerOwner;
  int mNumOwned;
} SAKEGetRecordLimitOutput;
```

Members

Members	Description
int mLimitPerOwner;	Contains the maximum number of records that a profile can own in the table; corresponds to the limit per owner option that can be set using the Sake Web Admin Panel.
int mNumOwned;	Contains the number of records that the local profile currently owns in the table.

See Also

sakeGetRecordLimit (see page 179), SAKERequestCallback (see page 190)

SAKEGetSpecificRecordsInput Structure

Summary

Input object passed to sakeGetSpecificRecords Function (see page 180).

C++

```
typedef struct {
    char * mTableId;
    int * mRecordIds;
    int mNumRecordIds;
    char ** mFieldNames;
    int mNumFields;
  } SAKEGetSpecificRecordsInput;
```

Members

Members	Description
char * mTableId;	Points to the tableid of the table from which to get the records.
int * mRecordIds;	An array of recordids, each one identifying a record which is to be returned.
int mNumRecordIds;	The number of recordids in the mRecordIds array.
char ** mFieldNames;	Points to an array of strings, each of which contains the name of a field for which to return values.
int mNumFields;	Stores the number of strings in the mFieldNames array. This list controls the values which will be returned as part of the response. The array can contain just one field name, the names of all the fields in the table, or any subset of the field names.

Remarks

In addition to the fields which the developer defines, you can also request values for the "recordid" field, "ownerid" field (if the table has an owner type of profile), and "num_ratings" and "average_rating" fields (if the table has its ratings option set to true).

See Also

sakeGetSpecificRecords (see page 180)

SAKEGetSpecificRecordsOutput Structure

Summary

Returned output object that contains all of the records which were specified in the request.

C++

```
  typedef struct {
    int mNumRecords;
    SAKEField ** mRecords;
  } SAKEGetSpecificRecordsOutput;
```

Members

Members	Description
int mNumRecords;	The number of records found.
SAKEField ** mRecords;	Points an array of records, each of which is represented as an array of fields.

See Also

sakeGetSpecificRecords (see page 180), SAKERequestCallback (see page 190), SAKEField (see page 195)

SAKERateRecordInput Structure

Summary

Input object passed to sakeRateRecord Function (see page 181).

C++

```
typedef struct {
  char * mTableId;
  int mRecordId;
  gsi_u8 mRating;
} SAKERateRecordInput;
```

Members

Members	Description
char * mTableId;	Points to the tableid of the table in which the record to be rated exists.
int mRecordId;	The recordid of the record to rate.
gsi_u8 mRating;	The rating the user wants to give the record.

Remarks

The range of ratings which Sake (see page 176) supports is 0 to 255. However, a game can restrict itself to a subset of that range if it wishes.

For example, a game may want to use a rating scale of 1 to 5 (e.g., stars), or it may want to use a range of 0 to 100. Sake (see page 176) allows users to rate records which they own, however no player profile can rate a single record more than once.

Internally, Sake (see page 176) stores every rating given by every user, which allows it to compute accurate averages and to prevent repeat ratings. The field name "my_rating" can be used to obtain the current profile's rating for a given record or when searching for records. By default, my_rating = -1 for records the current user has not yet rated. When browsing for records, you can use the special search tags @rated or @unrated in the filter string to limit the searches to either rated or unrated records.

See Also

sakeRateRecord (see page 181)

SAKERateRecordOutput Structure

Summary

Returned output object that lists the new number of ratings and the new average rating for the specified record.

C++

```
typedef struct {
  int mNumRatings;
  float mAverageRating;
} SAKERateRecordOutput;
```

Members

Members	Description
int mNumRatings;	The number of ratings associated with this record.
float mAverageRating;	The average rating of this record.

See Also

sakeRateRecord (see page 181), SAKERequestCallback (see page 190)

SAKESearchForRecordsInput Structure

Summary

Input object passed to sakeSearchForRecords Function (see page 186). Members/Constants mTableId mFieldNames mNumFields mFilter mSort mOffset mMaxRecords mTargetRecordFilter mSurroundingRecordsCount mOwnerIds mNumOwnerIds mCacheFlag

C++

```
typedef struct {
  char * mTableId;
  char ** mFieldNames;
  int mNumFields;
  gsi_char * mFilter;
  char * mSort;
  int mOffset;
  int mMaxRecords;
  gsi_char * mTargetRecordFilter;
  int mSurroundingRecordsCount;
  int * mOwnerIds;
  int mNumOwnerIds;
  gsi_bool mCacheFlag;
} SAKESearchForRecordsInput;
```

Members

Members	Description
char * mTableId;	Points to the tableid of the table to be searched.
char ** mFieldNames;	Points to an array of strings, each of which contains the name of a field for which to return values. This list controls the values that will be returned as part of the response. The array can contain just one field name, the names of all the fields in the table, or any subset of the field names.
int mNumFields;	Stores the number of strings in the mFieldNames array.
gsi_char * mFilter;	SQL-like filter string which is used to search for records based on the values in their fields. For example, to find everyone who has a score of more than 50 use "score > 50", or to find everyone who has a name that starts with an A use "name like A%". Note that a field can be used in the filter string even if it is not listed in the mFieldNames array, and that file metadata fields can be used in a filter string. The length of this string has been tested to 12,000 characters without issue.
char * mSort;	SQL-like sort string which is used to sort the records which are found by the search. To sort the results on a particular field, just pass in the name of that field, and the results will be sorted from lowest to highest based on that field. To make the sort descending instead of ascending add " desc" after the name of the field. Note that a field can be used in the sort string even if it is not listed in the mFieldNames array, and that file metadata fields can be used in a sort.
int mOffset;	If not set to 0, then the Sake service will return records starting from the given offset into the result set.
int mMaxRecords;	Used to specify the maximum number of records to return for a particular search.

gsi_char * mTargetRecordFilter;	Used to specify a single record to return - when done in conjunction with mSurroundingRecordsCount, this will return the "target" record plus the surrounding records above and below this target record. Can also be used to specify a "set" of target records to return, but when used in this context the surrounding records count does not apply.
int mSurroundingRecordsCount;	Used in conjunction with mTargetRecordFilter - specifies the number of records to return above and below the target record. (e.g., if = 5, you will receive a maximum of 11 possible records, the target record + 5 above and 5 below).
int * mOwnerIds;	Specifies an array of ownerIds (profileid of record owner) to return from the search.
int mNumOwnerIds;	Specifies the number of ids contained in the mOwnerIds array.
gsi_bool mCacheFlag;	Enables caching if set to gsi_true. Defaults to no caching if none is specified. Please turn on caching for big, leaderboard-style queries. Please turn off caching for individual player queries.

Remarks

In addition to the fields which the developer defines, you can also request values for the "recordid" field, "ownerid" field (if the table has an owner type of profile), and "num_ratings" and "average_rating" fields (if the table has its ratings option set to true).

See Also

sakeSearchForRecords (see page 186)

SAKESearchForRecordsOutput Structure

Summary

Returned output object that contains the records founds by the search.

C++

```
typedef struct {
  int mNumRecords;
  SAKEField ** mRecords;
} SAKESearchForRecordsOutput;
```

Members

Members	Description
int mNumRecords;	The number of records found.
SAKEField ** mRecords;	Points to an array of records, each of which is represented as an array of fields.

See Also

sakeSearchForRecords (see page 186), SAKERequestCallback (see page 190), SAKEField (see page 195)

SAKEUpdateRecordInput Structure

Summary

Input object passed to sakeUpdateRecord Function (see page 184).

C++

```
typedef struct {
  char * mTableId;
  int mRecordId;
  SAKEField * mFields;
  int mNumFields;
} SAKEUpdateRecordInput;
```

Members

Members	Description
char * mTableId;	Points to the tableid of the table in which the record to be updated exists.
int mRecordId;	Identifies the record to be updated.
SAKEField * mFields;	Points to an array of fields which has the new values for the record's fields.
int mNumFields;	Stores the number of fields in the mFields array.

Remarks

Unlike with a CreateRecord request, mNumFields cannot be 0; at least one field must be updated.

UpdateRecord does not have an output object, because the Sake (see page 176) service does not send any response other than the success or failure indicated by the result parameter passed to the callback. When the callback is called, the outputData parameter will always be set to NULL.

See Also

sakeUpdateRecord (see page 184)

SAKEReportRecordInput Structure

Summary

Input object passed to sakeReportRecord.

C++

```
typedef struct {
  char * mTableId;
  int mRecordId;
  int mReasonCode;
  char * mReason;
} SAKEReportRecordInput;
```

Members

Members	Description
char * mTableId;	Points to the tableid of the table in which the record to be reported exists.
int mRecordId;	The recordid of the record to report.
int mReasonCode;	A code for the report reason, for if the developer has a few predefined options
char * mReason;	Description of reasons why the record is reported. Maximum 256 characters.

See Also

sakeReportRecord

Enumerations

Enumerations

	Name	Description
	SAKEFileResult (see page 204)	Used to determine the status of a file uploaded to Sake (see page 176).
	SAKERequestResult (see page 205)	The result of Sake (see page 176) calls used to modify or read from records (returned to the SAKERequestCallback (see page 190)).
	SAKEStartRequestResult (see page 206)	The status result of the most recent request.
	SAKEStartupResult (see page 207)	Value returned from the call to sakeStartup (see page 183).
	SAKEContentStorageType (see page 207)	This enum is used to determine the type of local storage that the SDK will use to upload/update from or download to from the backend.

Unions

	Name	Description
	SAKEContentStorage (see page 208)	Union for the content stored locally.

SAKEFileResult Enumeration

Summary

Used to determine the status of a file uploaded to Sake (see page 176).

C++

```
typedef enum {
  SAKEFileResult_SUCCESS = 0,
  SAKEFileResult_BAD_HTTP_METHOD = 1,
  SAKEFileResult_BAD_FILE_COUNT = 2,
  SAKEFileResult_MISSING_PARAMETER = 3,
  SAKEFileResult_FILE_NOT_FOUND = 4,
  SAKEFileResult_FILE_TOO_LARGE = 5,
  SAKEFileResult_SERVER_ERROR = 6,
  SAKEFileResult_UNKNOWN_ERROR
} SAKEFileResult;
```

Members

Members	Description
SAKEFileResult_SUCCESS = 0	Upload succeeded.
SAKEFileResult_BAD_HTTP_METHOD = 1	Incorrect ghttp call used to upload file.
SAKEFileResult_BAD_FILE_COUNT = 2	Number of files uploaded is incorrect.
SAKEFileResult_MISSING_PARAMETER = 3	Missing parameter in the ghttp upload call.
SAKEFileResult_FILE_NOT_FOUND = 4	No file was found.
SAKEFileResult_FILE_TOO_LARGE = 5	File uploaded larger than the specified size.
SAKEFileResult_SERVER_ERROR = 6	Unknown error occurred on the server when processing this request.
SAKEFileResult_UNKNOWN_ERROR	Error is unknown (used if none of the above).

See Also

sakeGetFileResultFromHeaders (see page 185)

SAKERequestResult Enumeration

Summary

The result of Sake (see page 176) calls used to modify or read from records (returned to the SAKERequestCallback Type (see page 190)).

C++

```
typedef enum {
  SAKERequestResult_SUCCESS,
  SAKERequestResult_SECRET_KEY_INVALID,
  SAKERequestResult_SERVICE_DISABLED,
  SAKERequestResult_CONNECTION_TIMEOUT,
  SAKERequestResult_CONNECTION_ERROR,
  SAKERequestResult_MALFORMED_RESPONSE,
  SAKERequestResult_OUT_OF_MEMORY,
  SAKERequestResult_DATABASE_UNAVAILABLE,
  SAKERequestResult_LOGIN_TICKET_INVALID,
  SAKERequestResult_LOGIN_TICKET_EXPIRED,
  SAKERequestResult_TABLE_NOT_FOUND,
  SAKERequestResult_RECORD_NOT_FOUND,
  SAKERequestResult_FIELD_NOT_FOUND,
  SAKERequestResult_FIELD_TYPE_INVALID,
  SAKERequestResult_NO_PERMISSION,
  SAKERequestResult_RECORD_LIMIT_REACHED,
  SAKERequestResult_ALREADY_RATED,
  SAKERequestResult_NOT_RATEABLE,
  SAKERequestResult_NOT_OWNED,
  SAKERequestResult_FILTER_INVALID,
  SAKERequestResult_SORT_INVALID,
  SAKERequestResult_TARGET_FILTER_INVALID,
  SAKERequestResult_CERTIFICATE_INVALID,
  SAKERequestResult_UNKNOWN_ERROR,
  SAKERequestResult_REQUEST_CANCELLED,
  SAKERequestResult_CONTENTSERVER_FAILURE,
  SAKERequestResult_ALREADY_REPORTED,
  SAKERequestResult_INVALID_GAMEID,
  SAKERequestResult_INVALID_SESSIONTOKEN,
  SAKERequestResult_SESSIONTOKEN_EXPIRED
} SAKERequestResult;
```

Members

Members	Description
SAKERequestResult_SUCCESS	Sake request completed successfully.
SAKERequestResult_SECRET_KEY_INVALID	The secretKey passed to sakeSetGame is invalid (sakeSetGame does not actually authenticate the game info).
SAKERequestResult_SERVICE_DISABLED	GameSpy services have been disabled for this title.
SAKERequestResult_CONNECTION_TIMEOUT	A connection to the Sake service could not be established before the timeout was reached.
SAKERequestResult_CONNECTION_ERROR	An error occurred while connecting to the Sake service.
SAKERequestResult_MALFORMED_RESPONSE	The SOAP response sent from the Sake service was corrupt.
SAKERequestResult_OUT_OF_MEMORY	A memory allocation failed.
SAKERequestResult_DATABASE_UNAVAILABLE	The Sake database is temporarily down.
SAKERequestResult_LOGIN_TICKET_INVALID	The profile's loginTicket, obtained via gpGetLoginTicket and passed to sakeSetProfile, is invalid (sakeSetProfile does not actually authenticate the player info).

SAKERequestResult_LOGIN_TICKET_EXPIRED	The profile's loginTicket, obtained via gpGetLoginTicket and passed to sakeSetProfile, has expired. The profile must obtain an updated login ticket (via gpGetLoginTicket) and call sakeSetProfile once again with the updated ticket.
SAKERequestResult_TABLE_NOT_FOUND	No Sake table matches the mTableId passed in the Sake query.
SAKERequestResult_RECORD_NOT_FOUND	No records in the specified table match the mRecordId passed in the Sake query.
SAKERequestResult_FIELD_NOT_FOUND	mFieldNames or mFields contains a field that is not in the specified table.
SAKERequestResult_FIELD_TYPE_INVALID	The mType set in a SAKEField struct is not a valid SAKEFieldType.
SAKERequestResult_NO_PERMISSION	The profile is not the owner of the record and the record does not have public access (permissions are set on the Sake Web Admin Panel).
SAKERequestResult_RECORD_LIMIT_REACHED	The profile cannot create any more records in the table specified (Limit per owner is set on the Sake Web Admin Panel).
SAKERequestResult_ALREADY_RATED	No profile can rate a single record more than once.
SAKERequestResult_NOT_RATEABLE	The record is in a table that is not rateable (The 'Rateable' flag for a table is set on the Sake Web Admin Panel).
SAKERequestResult_NOT_OWNED	Only the record owner profile can perform the attempted function on the specified record.
SAKERequestResult_FILTER_INVALID	The mFilter string has invalid SQL grammar.
SAKERequestResult_SORT_INVALID	The mSort string has invalid SQL grammar.
SAKERequestResult_TARGET_FILTER_INVALID	Error is unknown used if none of the above.
SAKERequestResult_CERTIFICATE_INVALID	AuthService certificate/proof cannot be authenticated.
SAKERequestResult_ALREADY_REPORTED	No profile can report a single record more than once (while it's in queue for moderation).
SAKERequestResult_INVALID_GAMEID	Make sure GameID is properly set with wsSetGameCredentials.
SAKERequestResult_INVALID_SESSIONTOKEN	Make sure wsSetGameCredentials was called with valid credentials and you have logged in via AuthService.
SAKERequestResult_SESSIONTOKEN_EXPIRED	Re-login via AuthService to refresh your 'session'.

See Also

SAKERequestCallback (see page 190)

SAKEStartRequestResult Enumeration

Summary

The status result of the most recent request.

C++

```
typedef enum {
  SAKEStartRequestResult_SUCCESS,
  SAKEStartRequestResult_NOT_AUTHENTICATED,
  SAKEStartRequestResult_OUT_OF_MEMORY,
  SAKEStartRequestResult_BAD_INPUT,
  SAKEStartRequestResult_BAD_TABLEID,
  SAKEStartRequestResult_BAD_FIELDS,
  SAKEStartRequestResult_BAD_NUM_FIELDS,
  SAKEStartRequestResult_BAD_FIELD_NAME,
  SAKEStartRequestResult_BAD_FIELD_TYPE,
  SAKEStartRequestResult_BAD_FIELD_VALUE,
  SAKEStartRequestResult_BAD_OFFSET,
  SAKEStartRequestResult_BAD_MAX,
  SAKEStartRequestResult_BAD_RECORDIDS,
  SAKEStartRequestResult_BAD_NUM_RECORDIDS,
  SAKEStartRequestResult_BAD_REASON_CODE,
  SAKEStartRequestResult_BAD_REASON,
  SAKEStartRequestResult_INVALID_DATA,
  SAKEStartRequestResult_HTTP_ERROR,
  SAKEStartRequestResult_UNKNOWN_ERROR,
  SAKEStartRequestResult_FILE_NOT_FOUND,
  SAKEStartRequestResult_HTTP_INVALID_POST,
  SAKEStartRequestResult_HTTP_INVALID_BUFFERSIZE,
  SAKEStartRequestResult_HTTP_INVALID_URL
} SAKEStartRequestResult;
```

See Also

sakeGetStartRequestResult (see page 181)

SAKEStartupResult Enumeration

Summary

Value returned from the call to sakeStartup Function (see page 183).

C++

```
typedef enum {
  SAKEStartupResult_SUCCESS,
  SAKEStartupResult_NOT_AVAILABLE,
  SAKEStartupResult_CORE_SHUTDOWN,
  SAKEStartupResult_OUT_OF_MEMORY
} SAKEStartupResult;
```

Members

Members	Description
SAKEStartupResult_SUCCESS	Startup succeeded.
SAKEStartupResult_NOT_AVAILABLE	The Sake service is unavailable.
SAKEStartupResult_CORE_SHUTDOWN	Error in the gsCore.
SAKEStartupResult_OUT_OF_MEMORY	Not enough memory to initialize Sake.

SAKEContentStorageType Enumeration

Summary

This enum is used to determine the type of local storage that the SDK will use to upload/update from or download to from the backend.

C++

```
enum SAKEContentStorageType {
  SAKEContentStorageType_DISK,
  SAKEContentStorageType_MEMORY
};
```

Members

Members	Description
SAKEContentStorageType_DISK	Content is stored as a local file.
SAKEContentStorageType_MEMORY	Content is stored in memory.

Remarks

See SAKEContentInfo (see page 194) structure.

SAKEContentStorage Union

Summary

Union for the content stored locally.

C++

```
union SAKEContentStorage {
  SAKEContentBuffer * mMemory;
  gsi_char * mFile;
};
```

Members

Members	Description
SAKEContentBuffer * mMemory;	Use when content is in memory.
gsi_char * mFile;	File name (or path name) when content is on Disk.

Remarks

If local content is stored in memory, use mMemory. If local content is stored in a file, use mFile.

Server Browsing

API Documentation

Module

Server Browsing (see page 208)

Functions

Functions

	Name	Description
◆	SBServerDirectConnect (see page 210)	Indicates whether the game host supports direct UDP connections.
◆	SBServerEnumKeys (see page 211)	Enumerates the key-value pairs for a given game host by calling KeyEnumFn with each key-value. The user-defined instance data will be passed to the KeyFn callback.
◆	SBServerGetBoolValue (see page 211)	Returns the value associated with the specified key. This value is returned as the appropriate type: SBBool (see page 238), float, int, or string.
◆	SBServerGetConnectionInfo (see page 212)	Checks to see if NAT Negotiation is required, based on whether the match is a LAN game, a public IP is present and several other factors. This function fills a supplied pointer with an IP string to use for NAT Negotiation, or a direct connection if possible.

	SBServerGetFloatValue (see page 212)	Returns the value associated with the specified key. This value is returned as the appropriate type: SBBool (see page 238), float, int, or string.
	SBServerGetIntValue (see page 213)	Returns the value associated with the specified key. This value is returned as the appropriate type: SBBool (see page 238), float, int, or string.
	SBServerGetPing (see page 213)	Returns the stored ping time for the specified game host.
	SBServerGetPlayerFloatValue (see page 214)	Returns the value associated with the specified player's key. This value is returned as the appropriate type, float, int, or string.
	SBServerGetPlayerIntValue (see page 214)	Returns the value associated with the specified player's key. This value is returned as the appropriate type: float, int, or string.
	SBServerGetPlayerStringValue (see page 215)	Returns the value associated with the specified player's key. This value is returned as the appropriate type: float, int, or string.
	SBServerGetPrivateAddress (see page 216)	Returns the internal address of the SBServer (see page 240), if any. For users behind a NAT or firewall, this is the local DHCP or assigned IP address of the machine.
	SBServerGetPrivateInetAddress (see page 216)	Returns the internal address of the SBServer (see page 240), if any. For users behind a NAT or firewall, this is the local DHCP or assigned IP address of the machine.
	SBServerGetPrivateQueryPort (see page 217)	Returns the private query port of the specified game host. This is the internal port on which the game host communicates to the GameSpy matchmaking service.
	SBServerGetPublicAddress (see page 217)	Returns the external address of the SBServer (see page 240), if any. For users behind a NAT or firewall, this is the address of the outermost NAT or firewall layer.
	SBServerGetPublicInetAddress (see page 218)	Returns the external address of the SBServer (see page 240), if any. For users behind a NAT or firewall, this is the address of the outermost NAT or firewall layer.
	SBServerGetPublicQueryPort (see page 218)	Returns the public query port of the specified game host. This is the external port on which the GameSpy matchmaking service communicates with the game host.
	SBServerGetStringValue (see page 219)	Returns the value associated with the specified key. This value is returned as the appropriate type. SBBool (see page 238), float, int, or string.
	SBServerGetTeamFloatValue (see page 219)	Returns the value associated with the specified team's key. This value is returned as the appropriate type: float, int, or string.
	SBServerGetTeamIntValue (see page 220)	Returns the value associated with the specified team's key. This value is returned as the appropriate type: float, int, or string.
	SBServerGetTeamStringValue (see page 221)	Returns the value associated with the specified team's key. This value is returned as the appropriate type: float, int, or string.
	SBServerHasBasicKeys (see page 221)	Determine if basic information is available for the specified game host.
	SBServerHasFullKeys (see page 222)	Determine if full information is available for the specified game host.
	SBServerHasPrivateAddress (see page 222)	Tests to see if a private address is available for the game host.
	SBServerHasValidPing (see page 223)	Determines if a game host has a valid ping value (otherwise the ping will be 0).
	ServerBrowserAuxUpdateIP (see page 223)	Queries key-values from a single game host.
	ServerBrowserAuxUpdateServer (see page 224)	Query key-values from a single game host that has already been added to the internal list.
	ServerBrowserClear (see page 224)	Clear the current server list.
	ServerBrowserCount (see page 225)	Retrieves the current list of games from the GameSpy matchmaking service.

	ServerBrowserDisconnect (see page 225)	Disconnect from the GameSpy matchmaking service.
	ServerBrowserErrorDesc (see page 226)	Returns a human readable string for the specified SBError (see page 239).
	ServerBrowserFree (see page 226)	Frees memory allocated by the ServerBrowser (see page 240) SDK. Terminates any pending queries.
	ServerBrowserGetMyPublicIP (see page 227)	Returns the local client's external (firewall) address.
	ServerBrowserGetMyPublicIPAddr (see page 227)	Returns the local client's external (firewall) address.
	ServerBrowserGetServer (see page 228)	Returns the SBServer (see page 240) object at the specified index.
	ServerBrowserHalt (see page 228)	Stop an update in progress.
	ServerBrowserLANUpdate (see page 229)	Retrieves the current list of games broadcasting on the local network.
	ServerBrowserLimitUpdate (see page 229)	Retrieves the current limited list of games from the GameSpy matchmaking service. Useful for low-memory systems.
	ServerBrowserListQueryError (see page 230)	Returns the ServerList error string, if any.
	ServerBrowserNew (see page 231)	Initialize the ServerBrowser (see page 240) SDK.
	ServerBrowserPendingQueryCount (see page 232)	Retrieves the number of game hosts with outstanding queries. Use this to check progress while asynchronously updating the game host list.
	ServerBrowserRemoveIP (see page 232)	Removes a game host from the local list.
	ServerBrowserRemoveServer (see page 233)	Removes a game host from the local list.
	ServerBrowserSendMessageToServer (see page 233)	Sends a game specific message to the specified IP/port. This message is routed through the matchmaking service.
	ServerBrowserSort (see page 234)	Sort the current list of game hosts.
	ServerBrowserState (see page 234)	Gets current state of the Server Browser object.
	ServerBrowserThink (see page 235)	Allows ServerBrowsingSDK to continue internal processing, including processing query replies.
	ServerBrowserUpdate (see page 235)	Retrieves the current list of games from the GameSpy matchmaking service.

SBServerDirectConnect Function

Summary

Indicates whether the game host supports direct UDP connections.

C++

```
COMMON_API SBBool SBServerDirectConnect(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

Returns SBTrue if a direct connection is possible, otherwise SBFalse.

Remarks

A return of SBFalse usually means that NAT negotiation is required.

Notes

This function should only be used to check public game hosts (where SBServerHasPrivateAddress (see page 222) returns SBFalse).

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerEnumKeys Function

Summary

Enumerates the key-value pairs for a given game host by calling KeyEnumFn with each key-value. The user-defined instance data will be passed to the KeyFn callback.

C++

```
COMMON_API void SBServerEnumKeys(
    SBServer server,
    SBServerKeyEnumFn KeyFn,
    void * instance
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.
SBServerKeyEnumFn KeyFn	[in] A callback that is called once for each key.
void * instance	[in] A user-defined data value that will be passed into each call to KeyFn.

Remarks

The SBServerEnumKeys function is used to list the available keys for a particular SBServer (see page 240) object. This is often useful when the number of keys or custom keys is unknown or variable. Most often, the number of keys is predefined and constant, making this function call unnecessary. No query is sent when enumerating keys, instead the keys are stored from the previous game host update.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetBoolValue Function

Summary

Returns the value associated with the specified key. This value is returned as the appropriate type: SBBool Enumeration (see page 238), float, int, or string.

C++

```
COMMON_API SBBool SBServerGetBoolValue(
    SBServer server,
    const gsi_char * key,
    SBBool bdefault
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.
const gsi_char * key	[in] The value associated with this key will be returned.
SBBool bdefault	[in] The value to return if the key is not found.

Returns

If the key is invalid or missing, the specified default is returned. For an existing key, the value is converted from string form to the appropriate data type. These functions do not perform any type checking.

Remarks

These functions are useful for converting custom keys to a native data type. No type checking is performed, the string value is simply cast to the appropriate data type. If a key is not found,the supplied default is returned.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetConnectionInfo Function

Summary

Checks to see if NAT Negotiation is required, based on whether the match is a LAN game, a public IP is present and several other factors. This function fills a supplied pointer with an IP string to use for NAT Negotiation, or a direct connection if possible.

C++

```
COMMON_API SBBool SBServerGetConnectionInfo(
    ServerBrowser gSB,
    SBServer server,
    gsi_u16 PortToConnectTo,
    char * ipstring_out
);
```

Parameters

Parameters	Description
ServerBrowser gSB	[in] ServerBrowser (see page 240) object returned from ServerBrowserNew (see page 231).
SBServer server	[in] A valid SBServer (see page 240) object.
char * ipstring_out	[out] An IP String you can use for a direct connection, or with which to attempt NAT negotiation.
portToConnectTo	[in] The game port to connect to.

Returns

Returns SBTrue if NAT negotiation is required, SBFalse if not.

Remarks

The connection test will result in one of three scenarios, based on the return value of the function.

Returns SBFalse: 1) LAN game: connect using the IP string. 2) Internet game with a direct connection available: connect using the IP string.

Returns SBTrue: 3) NAT traversal required, perform NAT negotiation with the IP string before connecting.

SBServerGetFloatValue Function

Summary

Returns the value associated with the specified key. This value is returned as the appropriate type: SBBool Enumeration (see page 238), float, int, or string.

C++

```
COMMON_API double SBServerGetFloatValue(
    SBServer server,
    const gsi_char * key,
    double fdefault
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.
const gsi_char * key	[in] The value associated with this key will be returned.
double fdefault	[in] The value to return if the key is not found.

Returns

If the key is invalid or missing, the specified default is returned. For an existing key, the value is converted from string form to the appropriate data type. These functions do not perform any type checking.

Remarks

These functions are useful for converting custom keys to a native data type. No type checking is performed, the string value is simply cast to the appropriate data type. If a key is not found,the supplied default is returned.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetIntValue Function

Summary

Returns the value associated with the specified key. This value is returned as the appropriate type: SBBool Enumeration (see page 238), float, int, or string.

C++

```
COMMON_API int SBServerGetIntValue(
    SBServer server,
    const gsi_char * key,
    int idefault
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.
const gsi_char * key	[in] The value associated with this key will be returned.
int idefault	[in] The value to return if the key is not found.

Returns

If the key is invalid or missing, the specified default is returned. For an existing key, the value is converted from string form to the appropriate data type. These functions do not perform any type checking.

Remarks

These functions are usefull for converting custom keys to a native data type. No type checking is performed, the string value is simply cast to the appropriate data type. If a key is not found, the supplied default is returned.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetPing Function

Summary

Returns the stored ping time for the specified game host.

C++

```
COMMON_API int SBServerGetPing(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

The stored game host response time.

Remarks

The SBServerGetPing function will return the stored response time of the game host. This response time is caculated from the last game host update.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetPlayerFloatValue Function

Summary

Returns the value associated with the specified player's key. This value is returned as the appropriate type, float, int, or string.

C++

```
COMMON_API double SBServerGetPlayerFloatValue(
    SBServer server,
    int playernum,
    const gsi_char * key,
    double fdefault
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.
int playernum	[in] The zero based index for the desired player.
const gsi_char * key	[in] The value associated with this key will be returned.
double fdefault	[in] The value to return if the key is not found.

Returns

If the player or key is invalid or missing, the specified default is returned. For an existing key, the value is converted from string form to the appropriate data type. These functions do not perform any type checking.

Remarks

These functions are useful for converting custom player keys to a native data type. No type checking is performed, the string value is simply cast to the appropriate data type. If a key is not found, the supplied default is returned.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetPlayerIntValue Function

Summary

Returns the value associated with the specified player's key. This value is returned as the appropriate type: float, int, or string.

C++

```
COMMON_API int SBServerGetPlayerIntValue(
    SBServer server,
    int playernum,
    const gsi_char * key,
    int idefault
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.
int playernum	[in] The zero based index for the desired player.
const gsi_char * key	[in] The value associated with this key will be returned.
int idefault	[in] The value to return if the key is not found.

Returns

If the player or key is invalid or missing, the specified default is returned. For an existing key, the value is converted from string form to the appropriate data type. These functions do not perform any type checking.

Remarks

These functions are useful for converting custom player keys to a native data type. No type checking is performed, the string value is simply cast to the appropriate data type. If a key is not found, the supplied default is returned.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetPlayerStringValue Function

Summary

Returns the value associated with the specified player's key. This value is returned as the appropriate type: float, int, or string.

C++

```
COMMON_API const gsi_char * SBServerGetPlayerStringValue(
    SBServer server,
    int playernum,
    const gsi_char * key,
    const gsi_char * sdefault
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.
int playernum	[in] The zero based index for the desired player.
const gsi_char * key	[in] The value associated with this key will be returned.
const gsi_char * sdefault	[in] The value to return if the key is not found. Note: This default string will be returned if the key has been reported as an empty string.

Returns

If the player or key is invalid or missing, the specified default is returned. For an existing key, the value is converted from string form to the appropriate data type. These functions do not perform any type checking.

Remarks

These functions are useful for converting custom player keys to a native data type. No type checking is performed, the string value is simply cast to the appropriate data type. If a key is not found or is reported as an empty string, the supplied default is returned.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetPrivateAddress Function

Summary

Returns the internal address of the SBServer Type (see page 240), if any. For users behind a NAT or firewall, this is the local DHCP or assigned IP address of the machine.

C++

```
COMMON_API char * SBServerGetPrivateAddress(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

The private address of the SBServer (see page 240), in string or integer form.

Remarks

When a client machine is behind a NAT or Firewall device, communication must go through the public address. The private address may be used by clients behind the same NAT or Firewall, and may be used to specifically identify two clients with the same public address. Often the private address is of the form "192.168.###.###" and is not usable for communication outside the local network.

The SBServer (see page 240) object may be obtained during the SBCallback from ServerBrowserUpdate (see page 235), or by calling ServerBrowserGetServer (see page 228). An SBServer (see page 240) object will only be accessible for game hosts in the list.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetPrivateInetAddress Function

Summary

Returns the internal address of the SBServer Type (see page 240), if any. For users behind a NAT or firewall, this is the local DHCP or assigned IP address of the machine.

C++

```
COMMON_API unsigned int SBServerGetPrivateInetAddress(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

The private address of the SBServer (see page 240), in string or integer form.

Remarks

When a client machine is behind a NAT or Firewall device, communication must go through the public address. The private address may be used by clients behind the same NAT or Firewall, and may be used to specifically identify two clients with the same public address. Often the private address is of the form "192.168.###.###" and is not usable for communication outside the local network.

The SBServer (see page 240) object may be obtained during the SBCallback from ServerBrowserUpdate (see page 235), or by calling ServerBrowserGetServer (see page 228). An SBServer (see page 240) object will only be accessible for game hosts in the list.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetPrivateQueryPort Function

Summary

Returns the private query port of the specified game host. This is the internal port on which the game host communicates to the GameSpy matchmaking service.

C++

```
COMMON_API unsigned short SBServerGetPrivateQueryPort(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

The private query port.

Remarks

The SBServerGetPrivateQueryPort function will return the private query port of the game host.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetPublicAddress Function

Summary

Returns the external address of the SBServer Type (see page 240), if any. For users behind a NAT or firewall, this is the address of the outermost NAT or firewall layer.

C++

```
COMMON_API char * SBServerGetPublicAddress(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

The public address of the SBServer (see page 240), in string or integer form.

Remarks

When a client machine is behind a NAT or Firewall device, communication must go through the public address. The public address of the SBServer (see page 240) is the address of the outermost Firewall or NAT device.

The SBServer (see page 240) object may be obtained during the SBCallback from ServerBrowserUpdate (see page 235), or by calling ServerBrowserGetServer (see page 228). An SBServer (see page 240) object will only be accessible for game hosts in the list.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetPublicInetAddress Function

Summary

Returns the external address of the SBServer Type (see page 240), if any. For users behind a NAT or firewall, this is the address of the outermost NAT or firewall layer.

C++

```
COMMON_API unsigned int SBServerGetPublicInetAddress(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

The public address of the SBServer (see page 240), in string or integer form.

Remarks

When a client machine is behind a NAT or Firewall device, communication must go through the public address. The public address of the SBServer (see page 240) is the address of the outermost Firewall or NAT device.

The SBServer (see page 240) object may be obtained during the SBCallback from ServerBrowserUpdate (see page 235), or by calling ServerBrowserGetServer (see page 228). An SBServer (see page 240) object will only be accessible for game hosts in the list.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetPublicQueryPort Function

Summary

Returns the public query port of the specified game host. This is the external port on which the GameSpy matchmaking service communicates with the game host.

C++

```
COMMON_API unsigned short SBServerGetPublicQueryPort(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

The public query port.

Remarks

The SBServerGetPublicQueryPort function will return the public query port of the game host.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetStringValue Function

Summary

Returns the value associated with the specified key. This value is returned as the appropriate type. SBBool Enumeration (see page 238), float, int, or string.

C++

```
COMMON_API const gsi_char * SBServerGetStringValue(
    SBServer server,
    const gsi_char * keyname,
    const gsi_char * def
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.
const gsi_char * keyname	[in] The value associated with this key will be returned.
const gsi_char * def	[in] The value to return if the key is not found. Note: this default string will be returned if the key has been reported as an empty string.

Returns

If the key is invalid or missing, the specified default is returned. For an existing key, the value is converted from string form to the appropriate data type. These functions do not perform any type checking.

Remarks

These functions are useful for converting custom keys to a native data type. No type checking is performed, the string value is simply cast to the appropriate data type. If a key is not found or is reported as an empty string, the supplied default is returned.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetTeamFloatValue Function

Summary

Returns the value associated with the specified team's key. This value is returned as the appropriate type: float, int, or string.

C++

```
COMMON_API double SBServerGetTeamFloatValue(
    SBServer server,
    int teamnum,
    const gsi_char * key,
    double fdefault
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.
int teamnum	[in] The integer index of the team.
const gsi_char * key	[in] The value associated with this key will be returned.
double fdefault	[in] The value to return if the key is not found.

Returns

If the key is invalid or missing, the specified default is returned. For an existing key, the value is converted from string form to the appropriate data type. These functions do not perform any type checking.

Remarks

These functions are useful for converting custom keys to a native data type. No type checking is performed, the string value is simply cast to the appropriate data type. If a key is not found,the supplied default is returned.

The SBServer (see page 240) object may be obtained during the SBCallback from ServerBrowserUpdate (see page 235), or by calling ServerBrowserGetServer (see page 228). An SBServer (see page 240) object will only exist for game hosts in the list. IP addresses removed from the game host list will not have an associated SBServer (see page 240) object.

Team indexes are determined on a per-game basis. The only requirement is that they match the game host's reporting indexes.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetTeamIntValue Function

Summary

Returns the value associated with the specified team's key. This value is returned as the appropriate type: float, int, or string.

C++

```
COMMON_API int SBServerGetTeamIntValue(
    SBServer server,
    int teamnum,
    const gsi_char * key,
    int idefault
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.
int teamnum	[in] The integer index of the team.
const gsi_char * key	[in] The value associated with this key will be returned.
int idefault	[in] The value to return if the key is not found.

Returns

If the key is invalid or missing, the specified default is returned. For an existing key, the value is converted from string form to the appropriate data type. These functions do not perform any type checking.

Remarks

These functions are useful for converting custom keys to a native data type. No type checking is performed, the string value is simply cast to the appropriate data type. If a key is not found, the supplied default is returned.

The SBServer (see page 240) object may be obtained during the SBCallback from ServerBrowserUpdate (see page 235), or by calling ServerBrowserGetServer (see page 228). An SBServer (see page 240) object will only exist for game hosts in the list. IP addresses removed from the game host list will not have an associated SBServer (see page 240) object.

Team indexes are determined on a per-game basis. The only requirement is that they match the game host's reporting indexes.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerGetTeamStringValue Function

Summary

Returns the value associated with the specified team's key. This value is returned as the appropriate type: float, int, or string.

C++

```
COMMON_API const gsi_char * SBServerGetTeamStringValue(
    SBServer server,
    int teamnum,
    const gsi_char * key,
    const gsi_char * sdefault
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.
int teamnum	[in] The integer index of the team.
const gsi_char * key	[in] The value associated with this key will be returned.
const gsi_char * sdefault	[in] The value to return if the key is not found.

Returns

If the key is invalid or missing, the specified default is returned. For an existing key, the value is converted from string form to the appropriate data type. These functions do not perform any type checking.

Remarks

These functions are useful for converting custom keys to a native data type. No type checking is performed, the string value is simply cast to the appropriate data type. If a key is not found or is reported as an empty string, the supplied default is returned.

The SBServer (see page 240) object may be obtained during the SBCallback from ServerBrowserUpdate (see page 235), or by calling ServerBrowserGetServer (see page 228). An SBServer (see page 240) object will only exist for game hosts in the list. IP addresses removed from the game host list will not have an associated SBServer (see page 240) object.

Team indexes are determined on a per-game basis. The only requirement is that they match the server's reporting indexes.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerHasBasicKeys Function

Summary

Determine if basic information is available for the specified game host.

C++

```
COMMON_API SBBool SBServerHasBasicKeys(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

SBTrue if available; otherwise SBFalse.

Remarks

The SBServerHasBasicKeys function is used to determine if the game host object has been populated with the 'basicFields' keys as passed to the ServerBrowserUpdate (see page 235). Information may not be available if a game host query is still pending.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerHasFullKeys Function

Summary

Determine if full information is available for the specified game host.

C++

```
COMMON_API SBBool SBServerHasFullKeys(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

SBTrue if available; otherwise SBFalse.

Remarks

The SBServerHasFullKeys function is used to determine if the game host object has been populated with all keys reported by the game host. 'Full' game host information is retrieved after a ServerBrowserAuxUpdate call. Information may not be available if a game host query is still pending.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerHasPrivateAddress Function

Summary

Tests to see if a private address is available for the game host.

C++

```
COMMON_API SBBool SBServerHasPrivateAddress(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

Returns SBTrue if the game host has a private address; otherwise it returns SBFalse.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

SBServerHasValidPing Function

Summary

Determines if a game host has a valid ping value (otherwise the ping will be 0).

C++

```
COMMON_API SBBool SBServerHasValidPing(
    SBServer server
);
```

Parameters

Parameters	Description
SBServer server	[in] A valid SBServer (see page 240) object.

Returns

SBTrue if the game host has a valid ping value, otherwise SBFalse.

ServerBrowserAuxUpdateIP Function

Summary

Queries key-values from a single game host.

C++

```
COMMON_API SBError ServerBrowserAuxUpdateIP(
    ServerBrowser sb,
    const gsi_char * ip,
    unsigned short port,
    SBBool viaMaster,
    SBBool async,
    SBBool fullUpdate
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object returned from ServerBrowserNew (see page 231).
const gsi_char * ip	[in] Address string of the game host.
unsigned short port	[in] Query port of the game host.
SBBool viaMaster	[in] Set to SBTrue to retrieve cached values from the matchmaking service.
SBBool async	[in] Set to SBTrue to run in non-blocking mode.
SBBool fullUpdate	[in] Set to SBTrue to retrieve the full set of key-values from the game host.

Returns

This function returns sbe_noerror for success. On an error condition, this function will return an SBError (see page 239) code. If the async option is SBTrue, the status condition will be reported to the SBCallback.

Remarks

The ServerBrowserAuxUpdateIP function is used to retrieve information about a specific game host. Information returned is in the form of key-value pairs and may be accessed through the standard SBServer (see page 240) object accessors.

See Also

ServerBrowserUpdate (see page 235), ServerBrowserLANUpdate (see page 229), ServerBrowserAuxUpdateServer (see page 224)

ServerBrowserAuxUpdateServer Function

Summary

Query key-values from a single game host that has already been added to the internal list.

C++

```
COMMON_API SBError ServerBrowserAuxUpdateServer(
    ServerBrowser sb,
    SBServer server,
    SBBool async,
    SBBool fullUpdate
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object returned from ServerBrowserNew (see page 231).
SBServer server	[in] SBServer (see page 240) object for the matchmaking service to update (usually obtained from SBCallback).
SBBool async	[in] Set to SBTrue to run in non-blocking mode.
SBBool fullUpdate	[in] Set to SBTrue to retrieve the full set of key-values from the game host.

Returns

This function returns sbe_noerror for success. On an error condition, this function will return an SBError (see page 239) code. If the async option is SBTrue, the status condition will be reported to the SBCallback.

Remarks

The ServerBrowserAuxUpdateServer function is used to retrieve information about a specific game host. Information returned is in the form of key-value pairs and may be accessed through the standard SBServer (see page 240) object accessors.

This function is generally used to get additional information about a game host (for example, to get full rules and player information from a game host that only has basic information so far), but can also be used to "refresh" the information about a given game host. Data will automatically be retrieved from the matchmaking service directly or from the game host as appropriate. When called asynchronously, multiple game host update requests can be queued and will be executed by the query engine in turn.

A game host update is only performed when a user selects a specific game host. Full updates for all game hosts on the list are not requested automatically (and ServerBrowserAuxUpdateServer() is not called from within the ServerBrowserCallback (see page 236)).

See Also

ServerBrowserUpdate (see page 235), ServerBrowserLANUpdate (see page 229), ServerBrowserAuxUpdateIP (see page 223)

ServerBrowserClear Function

Summary

Clear the current server list.

C++

```
COMMON_API void ServerBrowserClear(
    ServerBrowser sb
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).

Remarks

The ServerBrowserClear function empties the current list of game hosts in preparation for a ServerBrowserUpdate (see page 235) or other list populating call.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserFree (see page 226)

ServerBrowserCount Function

Summary

Retrieves the current list of games from the GameSpy matchmaking service.

C++

```
COMMON_API int ServerBrowserCount(
    ServerBrowser sb
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).

Returns

Returns the number of game hosts in the current list. The index is zero-based when referencing.

Remarks

The ServerBrowserCount function returns the number of game hosts in the current list. This may be a combination of game hosts returned by ServerBrowserUpdate (see page 235) and game hosts added manually by ServerBrowserAuxUpdateIP (see page 223). Please note that index functions such as ServerBrowserGetServer (see page 228) use a zero-based index. The actual valid indexes are 0 to ServerBrowserCount()-1.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserGetServer (see page 228)

ServerBrowserDisconnect Function

Summary

Disconnect from the GameSpy matchmaking service.

C++

```
COMMON_API void ServerBrowserDisconnect(
    ServerBrowser sb
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).

Remarks

The ServerBrowserDisconnect function disconnects a maintained connection to the GameSpy matchmaking service.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235)

ServerBrowserErrorDesc Function

Summary

Returns a human readable string for the specified SBError Enumeration (see page 239).

C++

```
COMMON_API const gsi_char * ServerBrowserErrorDesc(
    ServerBrowser sb,
    SBError error
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).
SBError error	[in] A valid SBError (see page 239) code.

Returns

For a valid SBError (see page 239), this function will return a human readable description. Otherwise this function returns an empty string.

Remarks

The ServerBrowserErrorDesc function is useful for displaying error information to a user that might not understand SBError (see page 239) codes. These descriptions are in English. For localization purposes, you will need to provide your own translation functions.

See Also

ServerBrowserNew (see page 231), ServerBrowserListQueryError (see page 230)

ServerBrowserFree Function

Summary

Frees memory allocated by the ServerBrowser Type (see page 240) SDK. Terminates any pending queries.

C++

```
COMMON_API void ServerBrowserFree(
    ServerBrowser sb
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] A ServerBrowser (see page 240) interface previously allocated with ServerBrowserNew (see page 231).

Remarks

The ServerBrowserFree function frees any allocated memory associated with the SDK and terminates any pending queries. This function must be called once for every call to ServerBrowserNew (see page 231) to ensure proper cleanup of the ServerBrowsing SDK.

See Also

ServerBrowserNew (see page 231)

Example

```
#include <sb_serverbrowsing.h>

void main( void )
{
   ServerBrowser aServerBrowser = SBServerBrowserNew("gmtest",
 "gmtest", "HA6zkS", 0, 20, QVERSION_QR2, SBFalse, SBCallback, NULL);

    ServerBrowserFree(aServerBrowser);
}
```

ServerBrowserGetMyPublicIP Function

Summary

Returns the local client's external (firewall) address.

C++

```
COMMON_API char * ServerBrowserGetMyPublicIP(
    ServerBrowser sb
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).

Returns

The local game client's external (firewall) address. This may be returned as a string or integer address.

Remarks

The ServerBrowserGetMyPublicIP and ServerBrowserGetMyPublicIPAddr (see page 227) functions return the external address of the local client, as report by the GameSpy matchmaking service. Because of this, the return value is only valid after a successful call to ServerBrowserUpdate (see page 235). The reason for this is that a client cannot determine their external address without first sending an outgoing packet. It is up to that packet's receiver to report the public address back to the local client.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235)

ServerBrowserGetMyPublicIPAddr Function

Summary

Returns the local client's external (firewall) address.

C++

```
COMMON_API unsigned int ServerBrowserGetMyPublicIPAddr(
    ServerBrowser sb
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).

Returns

The local clients external (firewall) address. This may be returned as a string or integer address.

Remarks

The ServerBrowserGetMyPublicIP (see page 227) and ServerBrowserGetMyPublicIPAddr functions return the external address of the local client, as reported by the GameSpy matchmaking service. Because of this, the return value is only valid after a successful call to ServerBrowserUpdate (see page 235). The reason for this is that a client cannot determine their external address without first sending an outgoing packet. It is up to that packet's receiver to report the public address back to the local client.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235)

ServerBrowserGetServer Function

Summary

Returns the SBServer Type (see page 240) object at the specified index.

C++

```
COMMON_API SBServer ServerBrowserGetServer(
    ServerBrowser sb,
    int index
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).
int index	[in] The array index.

Returns

Returns the SBServer (see page 240) at the specified array index. If index is greater than the bounds of the array, NULL is returned.

Remarks

Use ServerBrowserCount (see page 225) to retrieve the current number of game hosts in the array. This index is zero-based, so a list containing 5 game hosts will have the valid indexes 0 through 4. This list is usually populated using one of the list retrieval methods such as ServerBrowserUpdate (see page 235).

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235)

ServerBrowserHalt Function

Summary

Stop an update in progress.

C++

```
COMMON_API void ServerBrowserHalt(
    ServerBrowser sb
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).

Remarks

The ServerBrowserHalt function will stop an update in progress. This is often tied to a "cancel" button presented to the user on the game host list screen. This function clears any game hosts queued to be queried, and disconnects from the matchmaking service.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235)

ServerBrowserLANUpdate Function

Summary

Retrieves the current list of games broadcasting on the local network.

C++

```
COMMON_API SBError ServerBrowserLANUpdate(
    ServerBrowser sb,
    SBBool async,
    unsigned short startSearchPort,
    unsigned short endSearchPort
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).
SBBool async	[in] When set to SBTrue this function will run in non-blocking mode.
unsigned short startSearchPort	[in] The lowest port the SDK will listen to broadcasts from, in network byte order.
unsigned short endSearchPort	[in] The highest port the SDK will listen to broadcasts from, in network byte order.

Returns

If an error occurs, a valid SBError (see page 239) error code is returned. Otherwise, sbe_noerror is returned.

Remarks

The ServerBrowserLANUpdate function listens for broadcast packets on the local network. game hosts that are broadcasting within the specified port range will be detected. As each game host broadcast is received, one corresponding call to the SBCallbackfunction will be made with the status sbc_serveradded.

Generally this should start with your standard query port, and range above it, since the QR and QR2 SDKs will automatically allocate higher port numbers when running multiple game hosts on the same machine. You should limit your search to 100 ports or less in most cases to prevent flooding of the LAN with broadcast packets.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235)

ServerBrowserLimitUpdate Function

Summary

Retrieves the current limited list of games from the GameSpy matchmaking service. Useful for low-memory systems.

C++

```
COMMON_API SBError ServerBrowserLimitUpdate(
    ServerBrowser sb,
    SBBool async,
    SBBool disconnectOnComplete,
    const unsigned char * basicFields,
    int numBasicFields,
    const gsi_char * serverFilter,
    int maxServers
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object intialized with ServerBrowserNew (see page 231).
SBBool async	[in] When set to SBTrue, this function will run in non-blocking mode.
SBBool disconnectOnComplete	[in] When set to SBTrue this function will terminate the connection with the GameSpy matchmaking service after the download is complete.
const unsigned char * basicFields	[in] A byte array of basic field identifiers to retrieve for each game host. See remarks.
int numBasicFields	[in] The number of valid fields in the basicFields array.
const gsi_char * serverFilter	[in] SQL like string used to remove unwanted game hosts from the downloaded list.
int maxServers	[in] Maximum number of game hosts to be returned.

Returns

If an error occurs, a valid SBError (see page 239) error code is returned. Otherwise, sbe_noerror is returned.

Remarks

The ServerBrowserLimitUpdate function retrieves a limited set of the game hosts registered with the GameSpy matchmaking service. This is mostly useful for low memory systems such as the DS which may not be capable of loading an entire game host list.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235)

ServerBrowserListQueryError Function

Summary

Returns the ServerList error string, if any.

C++

```
COMMON_API const gsi_char * ServerBrowserListQueryError(
    ServerBrowser sb
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).

Returns

If a list error has occurred, a string description of the error is returned. Otherwise, an empty string "" is returned.

Remarks

The ServerBrowserListQueryError function returns the last string error reported by the matchmaking service. For localization purposes, you may safely assume that this string will not change, and you can test for it as you would a numeric error code.

See Also

ServerBrowserNew (see page 231)

ServerBrowserNew Function

Summary

Initialize the ServerBrowser Type (see page 240) SDK.

C++

```
COMMON_API ServerBrowser ServerBrowserNew(
    const gsi_char * queryForGamename,
    const gsi_char * queryFromGamename,
    const gsi_char * queryFromKey,
    int queryFromVersion,
    int maxConcUpdates,
    int queryVersion,
    SBBool lanBrowse,
    ServerBrowserCallback callback,
    void * instance
);
```

Parameters

Parameters	Description
const gsi_char * queryForGamename	[in] Game hosts returned will be for this Gamename.
const gsi_char * queryFromGamename	[in] Your assigned GameName.
const gsi_char * queryFromKey	[in] Secret key that corresponds to the queryFromGamename.
int queryFromVersion	[in] Set to zero unless directed otherwise by GameSpy.
int maxConcUpdates	[in] The maximum number of queries the ServerBrowsing SDK will send out at one time. Must be set to 20 or greater per GameSpy Certification Process/GameSpy Open Usage Guidelines.
int queryVersion	[in] The QueryReporting protocol used by the game host. Should be QVERSION_GOA or QVERSION_QR2. See remarks.
SBBool lanBrowse	[in] The switch to turn on only LAN browsing.
ServerBrowserCallback callback	[in] Function to be called when the operation completes.
void * instance	[in] Pointer to user data. This is optional and will be passed unmodified to the callback function.

Returns

This function returns the initialized ServerBrowser (see page 240) interface. No return value is reserved to indicate an error.

Remarks

The ServerBrowserNew function initializes the ServerBrowsing SDK. Developers should then use ServerBrowserUpdate (see page 235) or ServerBrowserLANUpdate (see page 229) to begin retrieving the list of registered game hosts.

See Also

ServerBrowserFree (see page 226), ServerBrowserUpdate (see page 235), ServerBrowserLANUpdate (see page 229)

Example

```
#include <sb_serverbrowsing.h>
void main( void )
{
    ServerBrowser aServerBrowser = SBServerBrowserNew("gmtest",
 "gmtest", "HA6zkS", 0, 20, QVERSION_QR2, SBFalse, SBCallback, NULL);

  ServerBrowserFree(aServerBrowser);
}
```

ServerBrowserPendingQueryCount Function

Summary

Retrieves the number of game hosts with outstanding queries. Use this to check progress while asynchronously updating the game host list.

C++

```
COMMON_API int ServerBrowserPendingQueryCount(
    ServerBrowser sb
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).

Returns

Returns the number of game hosts that have not yet been queried.

Remarks

The ServerBrowserPendingQueryCount function is most useful when updating a large list of game hosts. Use this function to display progress information to the user (e.g., "1048/2063 game hosts updated", or as a progress bar graphic).

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235)

ServerBrowserRemoveIP Function

Summary

Removes a game host from the local list.

C++

```
COMMON_API void ServerBrowserRemoveIP(
    ServerBrowser sb,
    const gsi_char * ip,
    unsigned short port
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).
const gsi_char * ip	[in] The address of the game host to remove.
unsigned short port	[in] The port of the game host to remove, in network byte order.

Remarks

The ServerBrowserRemoveIP function removes a single SBServer (see page 240) from the local list. This does not affect the matchmaking service or remote users. Please refer to the QR2 SDK for removing a registered game host from the matchmaking service list.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserRemoveServer (see page 233)

ServerBrowserRemoveServer Function

Summary

Removes a game host from the local list.

C++

```
COMMON_API void ServerBrowserRemoveServer(
    ServerBrowser sb,
    SBServer server
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).
SBServer server	[in] The game host to remove.

Remarks

The ServerBrowserRemoveServer function removes a single SBServer (see page 240) from the local list. This does not affect the matchmaking service or remote users. Please refer to the QR2 SDK for removing a registered game host from the matchmaking service list.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserRemoveIP (see page 232)

ServerBrowserSendMessageToServer Function

Summary

Sends a game specific message to the specified IP/port. This message is routed through the matchmaking service.

C++

```
COMMON_API SBError ServerBrowserSendMessageToServer(
    ServerBrowser sb,
    const gsi_char * ip,
    unsigned short port,
    const char * data,
    int len
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).
const gsi_char * ip	[in] Address of the game host in string form (e.g., "xxx.xxx.xxx.xxx").
unsigned short port	[in] The query port of the game host to which to send the message, in network byte order.
const char * data	[in] The raw data buffer.
int len	[in] The length of the data buffer.

Returns

If an error occurs, a valid SBError (see page 239) error code is returned. Otherwise, sbe_noerror is returned.

Remarks

The ServerBrowserSendMessageToServer function can be used to relay a raw data buffer to a game host behind a firewall. The raw buffer is sent through the matchmaking service since direct communication with the game host is not always possible. The buffer is sent in raw form to the game host's query port and does not contain any header information. This message is mostly useful in a shared-socket QR2 implementation.

See Also

ServerBrowserNew (see page 231), ServerBrowserUpdate (see page 235), ServerBrowserSendNatNegotiateCookieToServer

ServerBrowserSort Function

Summary

Sort the current list of game hosts.

C++

```
COMMON_API void ServerBrowserSort(
    ServerBrowser sb,
    SBBool ascending,
    const gsi_char * sortkey,
    SBCompareMode comparemode
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).
SBBool ascending	[in] When set to SBTrue this function will sort in ascending order (i.e., "a-b-c" order, not "c-b-a").
const gsi_char * sortkey	[in] The "key" of the key-value pair to sort by.
SBCompareMode comparemode	[in] Specifies the data type of the sortkey. See remarks.

Remarks

The ServerBrowserSort function will order the game host list, sorting by the specified sortkey. Sorting may be in ascending or descending order and various data-types are supported. SBCompareMode (see page 238) may be one of the following values:

sbcm_int: Uses integer comparison ("1,2,3,12,15,20").

sbcm_float: Similar to above, but considers decimal values ("1.1,1.2,2.1,3.0").

sbcm_strcase: Uses case-sensitive string comparison. Uses strcmp.

sbcm_stricase: Non-case-sensitive string comparision. Uses _stricmp or equivalent.

Please note that calling this function repeatedly for a large game host list may impact performance due to the standard qsort algorithm being ineffecient when sorting an already ordered list. This performance issue is rarely a cause for concern, but certain optimizations may be made if performance is noticeably impacted.

See Also

ServerBrowserUpdate (see page 235), ServerBrowserThink (see page 235)

ServerBrowserState Function

Summary

Gets current state of the Server Browser object.

C++

```
COMMON_API SBState ServerBrowserState(
    ServerBrowser sb
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).

Returns

Returns the current state.

Remarks

Descriptions of the possible state values can be found in the main header file.

ServerBrowserThink Function

Summary

Allows ServerBrowsingSDK to continue internal processing, including processing query replies.

C++

```
COMMON_API SBError ServerBrowserThink(
    ServerBrowser sb
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).

Returns

If an error occurs, a valid SBError (see page 239) error code is returned. Otherwise, sbe_noerror is returned.

Remarks

The ServerBrowserThink function is required for the SDK to process incoming data. Because of the single-threaded design of the GameSpy SDKs, all data is processed during this call, and processing is paused when this call is complete. When updating game host lists, this function should be called as frequently as possible to reduce the latency associated with game host response times. If this function is not called often enough, game host pings may be inflated due to processing delays; ~10ms is ideal.

See Also

ServerBrowserNew (see page 231)

ServerBrowserUpdate Function

Summary

Retrieves the current list of games from the GameSpy matchmaking service.

C++

```
COMMON_API SBError ServerBrowserUpdate(
    ServerBrowser sb,
    SBBool async,
    SBBool disconnectOnComplete,
    const unsigned char * basicFields,
    int numBasicFields,
    const gsi_char * serverFilter
);
```

Parameters

Parameters	Description
ServerBrowser sb	[in] ServerBrowser (see page 240) object initialized with ServerBrowserNew (see page 231).
SBBool async	[in] When set to SBTrue this function will run in non-blocking mode.
SBBool disconnectOnComplete	[in] When set to SBTrue this function will terminate the connection with the GameSpy matchmaking service after the download is complete.
const unsigned char * basicFields	[in] A byte array of basic field identifiers to retrieve for each server. See remarks.
int numBasicFields	[in] The number of valid fields in the basicFields array.
const gsi_char * serverFilter	[in] SQL-like string used to remove unwanted game hosts from the downloaded list.

Returns

If an error occurs, a valid SBError (see page 239) error code is returned. Otherwise, sbe_noerror is returned.

Remarks

The ServerBrowserUpdate function retrieves the current list of game hosts registered with the GameSpy matchmaking service. As each game host entry is received, one corresponding call to the SBCallback function will be made with the status sbc_serveradded. If basic keys are not yet available (check with SBServerHasBasickeys), another call will be made for this game host with status sbc_serverupdated.

This should only be called based on a user input trigger (e.g., a 'Refresh' button). It should never be called on a timer.

See Also

ServerBrowserNew (see page 231), ServerBrowserLANUpdate (see page 229)

Callbacks

Types

Name	Description
ServerBrowserCallback (see page 236)	The callback provided to ServerBrowserNew (see page 231). This gets called as the Server Browser updates the game host list.
SBServerKeyEnumFn (see page 237)	Callback function used for enumerating the key-value pairs for a game host.

ServerBrowserCallback Type

Summary

The callback provided to ServerBrowserNew Function (see page 231). This gets called as the Server Browser updates the game host list.

C++

```
typedef void (* ServerBrowserCallback)(ServerBrowser sb, SBCallbackReason reason, SBServer
server, void * instance);
```

Remarks

"instance" is any game-specific data you want passed to the callback function. For example, you can pass a structure pointer or object pointer for use within the callback. If you can access any needed data within the callback already, then you can just pass NULL for "instance".

See Also

ServerBrowserNew (see page 231)

Example

Your callback function should look something like:

```
void SBCallback(ServerBrowser sb, SBCallbackReason reason, SBServer server, void *instance)
{
    CMyGame *g = (CMyGame *)instance;

    switch (reason)
    {
        case sbc_serveradded :
            g->ServerView->AddServerToList(server);
            break;
        case sbc_serverupdated :
            g->ServerView->UpdateServerInList(server);
            break;
        case sbc_updatecomplete:
            g->ServerView->SetStatus("Update Complete");
            break;
        case sbc_queryerror:
            g->ServerView->SetStatus("Query Error Occurred:",
            ServerBrowserListQueryError(sb));
        break;
    }
}

Example use of the Callback:
int CMyGame::OnMultiplayerButtonClicked(...)
{
    m_ServerBrowser = ServerBrowserNew("mygame", "mygame", "123456",
                                        0, 10, QVERSION_QR2,
                                        SBCallBack, this);
}
```

SBServerKeyEnumFn Type

Summary

Callback function used for enumerating the key-value pairs for a game host.

C++

```
typedef void (* SBServerKeyEnumFn)(gsi_char * key, gsi_char * value, void * instance);
```

Parameters

Parameters	Description
key	[in] The enumerated key.
value	[in] The enumerated value.
instance	[in] User-provided data.

See Also

SBServerEnumKeys (see page 211)

Enumerations

Enumerations

Name	Description
SBBool (see page 238)	Standard Boolean.
SBCallbackReason (see page 238)	Callbacks that can occur during server browsing operations.
SBCompareMode (see page 238)	Comparison types for the ServerBrowserSort (see page 234) function.
SBError (see page 239)	Error codes that can be returned from Server Browsing functions.
SBState (see page 240)	States the ServerBrowser (see page 240) object can be in.

SBBool Enumeration

Summary

Standard Boolean.

C++

```
typedef enum {
  SBFalse,
  SBTrue
} SBBool;
```

Members

Members	Description
SBFalse	False.
SBTrue	True.

SBCallbackReason Enumeration

Summary

Callbacks that can occur during server browsing operations.

C++

```
typedef enum {
  sbc_serveradded,
  sbc_serverupdated,
  sbc_serverupdatefailed,
  sbc_serverdeleted,
  sbc_updatecomplete,
  sbc_queryerror,
  sbc_serverchallengereceived
} SBCallbackReason;
```

Members

Members	Description
sbc_serveradded	A game host was added to the list; it may just have an IP and port at this point.
sbc_serverupdated	Game host information has been updated; either basic or full information is now available about this game host.
sbc_serverupdatefailed	Either a direct or matchmaking service query to retrieve information about this game host failed.
sbc_serverdeleted	A game host was removed from the list.
sbc_updatecomplete	The server query engine is now idle.
sbc_queryerror	The matchmaking service returned an error string for the provided query.
sbc_serverchallengereceived	Prequery IP verification challenge was received (this is informational and no action is required).

SBCompareMode Enumeration

Summary

Comparison types for the ServerBrowserSort Function (see page 234) function.

C++

```
typedef enum {
  sbcm_int,
  sbcm_float,
  sbcm_strcase,
  sbcm_stricase
} SBCompareMode;
```

Members

Members	Description
sbcm_int	Assume the values are int and do an integer comparison.
sbcm_float	Assume the values are float and do a float comparison.
sbcm_strcase	Assume the values are strings and do a case-sensitive string comparison.
sbcm_stricase	Assume the values are strings and do a case-insensitive string comparison.

See Also

ServerBrowserSort (see page 234)

SBError Enumeration

Summary

Error codes that can be returned from Server Browsing functions.

C++

```
typedef enum {
  sbe_noerror,
  sbe_socketerror,
  sbe_dnserror,
  sbe_connecterror,
  sbe_dataerror,
  sbe_allocerror,
  sbe_paramerror,
  sbe_duplicateupdateerror
} SBError;
```

Members

Members	Description
sbe_noerror	No error has occurred.
sbe_socketerror	A socket function has returned an unexpected error.
sbe_dnserror	DNS lookup of the master address failed.
sbe_connecterror	Connection to the matchmaking service failed.
sbe_dataerror	Invalid data was returned from the matchmaking service.
sbe_allocerror	Memory allocation failed.
sbe_paramerror	An invalid parameter was passed to a function.
sbe_duplicateupdateerror	Matchmaking service update was requested on a game host that was already being updated.

SBState Enumeration

Summary

States the ServerBrowser Type (see page 240) object can be in.

C++

```
typedef enum {
  sb_disconnected,
  sb_listxfer,
  sb_querying,
  sb_connected
} SBState;
```

Members

Members	Description
sb_disconnected	Idle and not connected to the matchmaking service.
sb_listxfer	Downloading list of game hosts from the matchmaking service.
sb_querying	Querying game hosts.
sb_connected	Idle, but still connected to the matchmaking service.

Types

Types

Name	Description
SBServer (see page 240)	SBServer is an abstract data type representing a single game host.
ServerBrowser (see page 240)	ServerBrowser is an abstract data type used to represent the game host (server) list and query engine objects.

SBServer Type

C++

```
typedef struct _SBServer * SBServer;
```

ServerBrowser Type

C++

```
typedef struct _ServerBrowser * ServerBrowser;
```

Transport

API Documentation

Module

Transport (see page 240)

Functions

Functions

	Name	Description
◆	gt2Accept (↗ see page 242)	Accepts an incoming connection attempt.
◆	gt2AddReceiveFilter (↗ see page 243)	Adds a filter to the connection's incoming data filter list.
◆	gt2AddressToString (↗ see page 244)	Converts an IP and a port into a text string.
◆	gt2AddSendFilter (↗ see page 244)	Adds a filter to the connection's outgoing data filter list.
◆	gt2CloseAllConnections (↗ see page 245)	Closes all of a socket's connections.
◆	gt2CloseAllConnectionsHard (↗ see page 245)	Does a hard close on all of a socket's connections.
◆	gt2CloseConnection (↗ see page 245)	Starts closing a connection.
◆	gt2CloseConnectionHard (↗ see page 246)	Closes a connection immediately.
◆	gt2CloseSocket (↗ see page 246)	Closes a socket.
◆	gt2Connect (↗ see page 247)	Initiates a connection between a local socket and a remote socket.
◆	gt2CreateAdHocSocket (↗ see page 247)	Creates a new socket, which can be used for making outgoing connections or accepting incoming connections. See gt2CreateSocket (↗ see page 248) for details.
◆	gt2CreateSocket (↗ see page 248)	Creates a new socket, which can be used for making outgoing connections or accepting incoming connections.
◆	gt2FilteredReceive (↗ see page 249)	Called in response to a gt2ReceiveFilterCallback (↗ see page 272) being called. It can be called from within the callback, or at any later time.
◆	gt2FilteredSend (↗ see page 249)	Called in response to a gt2SendFilterCallback (↗ see page 273) being called. It can be called from within the callback, or at any later time.
◆	gt2GetConnectionData (↗ see page 250)	Returns the user data pointer stored with this connection.
◆	gt2GetConnectionSocket (↗ see page 250)	Returns the socket which this connection exists on.
◆	gt2GetConnectionState (↗ see page 251)	Gets the connection's state.
◆	gt2GetIncomingBufferFreeSpace (↗ see page 251)	Gets the amount of available space in the connection's incoming buffer.
◆	gt2GetIncomingBufferSize (↗ see page 252)	Gets the total size of the connection's incoming buffer.
◆	gt2GetLastSentMessageID (↗ see page 252)	Gets the message id for the last reliably sent message. Unreliable messages do not have an id.
◆	gt2GetLocalIP (↗ see page 253)	Gets a socket's local IP.
◆	gt2GetLocalPort (↗ see page 253)	Get's a socket's local port.
◆	gt2GetOutgoingBufferFreeSpace (↗ see page 253)	Gets the amount of available space in the connection's outgoing buffer.
◆	gt2GetOutgoingBufferSize (↗ see page 254)	Gets the total size of the connection's outgoing buffer.
◆	gt2GetRemoteIP (↗ see page 254)	Gets the connection's remote IP.
◆	gt2GetRemotePort (↗ see page 254)	Get's the connection's remote port.
◆	gt2GetSocketData (↗ see page 255)	Returns the user data pointer stored with this socket.

◆	gt2GetSocketSOCKET (see page 255)	This function returns the actual underlying socket for a GT2Socket (see page 277).
◆	gt2HostToNetworkInt (see page 256)	Convert an int from host to network byte order.
◆	gt2HostToNetworkShort (see page 256)	Convert a short from host to network byte order.
◆	gt2IPToAliases (see page 257)	Get the aliases associated with an IP address.
◆	gt2IPToHostInfo (see page 257)	Looks up DNS host information based on an IP.
◆	gt2IPToHostname (see page 258)	Get the hostname associated with an IP address.
◆	gt2IPToIPs (see page 258)	Get the IPs associated with an IP address.
◆	gt2Listen (see page 259)	Start (or stop) listening for incoming connections on a socket.
◆	gt2NetworkToHostInt (see page 259)	Convert an int from network to host byte order.
◆	gt2NetworkToHostShort (see page 260)	Convert a short from network to host byte order.
◆	gt2Ping (see page 260)	Sends a ping on a connection in an attempt to determine latency.
◆	gt2Reject (see page 261)	Rejects a connection attempt.
◆	gt2RemoveReceiveFilter (see page 261)	Removes a filter from the connection's incoming data filter list.
◆	gt2RemoveSendFilter (see page 262)	Removes a filter from the connection's outgoing data filter list.
◆	gt2Send (see page 262)	Sends data over a connection, reliably or unreliably.
◆	gt2SendRawUDP (see page 263)	Sends a raw UDP datagram through the socket. This function bypasses the normal connection logic. Note that all messages sent this way will be unreliable. To broadcast a datagram, omit the IP from the remoteAddress (e.g., ":12345").
◆	gt2SetConnectionData (see page 263)	Stores a user data pointer with this connection.
◆	gt2SetReceiveDump (see page 263)	Sets a callback to which all incoming UDP packets are passed. This is at a lower level than the filters, can only be used for monitoring, and is designed for debugging purposes.
◆	gt2SetSendDump (see page 264)	Sets a callback to which all outgoing UDP packets are passed. This is at a lower level than the filters, can only be used for monitoring, and is designed for debugging purposes.
◆	gt2SetSocketData (see page 264)	Stores a user data pointer with this socket.
◆	gt2SetUnrecognizedMessageCallback (see page 265)	Used to handle unrecognized messages, usually used for sharing a socket with another SDK.
◆	gt2StringToAddress (see page 265)	Converts a string address, which is either a hostname ("www.gamespy.net") or a dotted IP ("1.2.3.4") into an IP and a port.
◆	gt2StringToAliases (see page 266)	Get the aliases associated with a hostname or dotted IP.
◆	gt2StringToHostInfo (see page 267)	Looks up DNS host information based on a hostname or dotted IP.
◆	gt2StringToHostname (see page 267)	Get the hostname associated with a hostname or dotted IP.
◆	gt2StringToIPs (see page 268)	Get the IPs associated with a hostname or dotted IP.
◆	gt2Think (see page 268)	Does any thinking for this socket and its connections.
◆	gt2WasMessageIDConfirmed (see page 269)	Checks if confirmation has been received that the remote end received a particular reliable message.

gt2Accept Function

Summary

Accepts an incoming connection attempt.

C++

```
COMMON_API GT2Bool gt2Accept(
    GT2Connection connection,
    GT2ConnectionCallbacks * callbacks
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.
GT2ConnectionCallbacks * callbacks	[in] The set of callbacks associated with the connection.

Returns

GT2False means the connection was closed between when the gt2ConnectAttemptCallback (see page 270) was called and this function was called. The connection cannot be used.

Remarks

After a socket's gt2ConnectAttemptCallback (see page 270) has been called, this function can be used to accept the incoming connection attempt. It can be called from either within the callback or some later time. As soon as it is called the connection is active, and messages can be sent and received. The remote side of the connection will have it's connected callback called with the result set to GT2Success. The callbacks that are passed in to this function are the same callbacks that get passed to gt2Connect (see page 247), with the exception that the connected callback can be ignored, as the connection is already established. If this function returns GT2True, then the connection has been successfully accepted. If it returns GT2False, then the remote side has already closed the connection attempt. In that case, the connection is considered closed, and it cannot be referenced again.

See Also

gt2Listen (see page 259), gt2ConnectAttemptCallback (see page 270), gt2Reject (see page 261)

gt2AddReceiveFilter Function

Summary

Adds a filter to the connection's incoming data filter list.

C++

```
COMMON_API GT2Bool gt2AddReceiveFilter(
    GT2Connection connection,
    gt2ReceiveFilterCallback callback
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.
gt2ReceiveFilterCallback callback	[in] The filtering callback.

Returns

Returns GT2False if there was an error adding the filter (due to no free memory).

Remarks

The callback will get called when a message is being received. Callbacks will be called in the order they were added to the connection's filter list.

See Also

gt2ReceiveFilterCallback (see page 272), gt2RemoveReceiveFilter (see page 261), gt2FilteredReceive (see page 249)

gt2AddressToString Function

Summary

Converts an IP and a port into a text string.

C++

```
COMMON_API const char * gt2AddressToString(
    unsigned int ip,
    unsigned short port,
    char string[22]
);
```

Parameters

Parameters	Description
unsigned int ip	[in] IP in network byte order. Can be 0.
unsigned short port	[in] Port in host byte order. Can be 0.
char string[22]	[out] String will be placed in here. Can be NULL.

Returns

The string is returned. If the string paramater is NULL, then an internal static string will be used. There are two internal strings that are alternated between.

Remarks

The IP must be in network byte order, and the port in host byte order. The string must be able to hold at least 22 characters (including the NUL).

"XXX.XXX.XXX.XXX:XXXXX" If both the IP and port are non-zero, the string will be of the form "1.2.3.4:5" (":"). If the port is zero, and the IP is non-zero, the string will be of the form "1.2.3.4" (""). If the IP is zero, and the port is non-zero, the string will be of the form ":5" (":"). If both the IP and port are zero, the string will be an empty string ("") The string is returned. If the string parameter is NULL, then an internal static string will be used. There are two internal strings that are alternated between.

See Also

gt2StringToAddress (see page 265)

gt2AddSendFilter Function

Summary

Adds a filter to the connection's outgoing data filter list.

C++

```
COMMON_API GT2Bool gt2AddSendFilter(
    GT2Connection connection,
    gt2SendFilterCallback callback
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.
gt2SendFilterCallback callback	[in] The filtering callback.

Returns

Returns GT2False if there was an error adding the filter (due to no free memory).

Remarks

The callback will get called when a message is being sent. Callbacks will be called in the order they were added to the connection's filter list.

See Also

gt2SendFilterCallback (see page 273), gt2RemoveSendFilter (see page 262), gt2FilteredSend (see page 249)

gt2CloseAllConnections Function

Summary

Closes all of a socket's connections.

C++

```
COMMON_API void gt2CloseAllConnections(
    GT2Socket socket
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.

Remarks

Same effect as calling gt2CloseConnection (see page 245) on all of the socket's connections.

See Also

gt2CloseConnection (see page 245), gt2CloseConnectionHard (see page 246), gt2CloseAllConnectionsHard (see page 245)

gt2CloseAllConnectionsHard Function

Summary

Does a hard close on all of a socket's connections.

C++

```
COMMON_API void gt2CloseAllConnectionsHard(
    GT2Socket socket
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.

Remarks

Has the same effect as calling gt2CloseConnectionHard (see page 246) on all of the socket's connection.

See Also

gt2CloseConnection (see page 245), gt2CloseConnectionHard (see page 246), gt2CloseAllConnections (see page 245)

gt2CloseConnection Function

Summary

Starts closing a connection.

C++

```
COMMON_API void gt2CloseConnection(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Remarks

This function attempts to synchronize the close with the remote side of the connection. This means that the connection does not close immediately, and messages may be received while attempting the close. When the close is completed, the connection's closed callback will be called. Use gt2CloseConnectionHard (see page 246) to immediately close a connection.

See Also

gt2CloseConnectionHard (see page 246), gt2CloseAllConnections (see page 245), gt2CloseAllConnectionsHard (see page 245)

gt2CloseConnectionHard Function

Summary

Closes a connection immediately.

C++

```
COMMON_API void gt2CloseConnectionHard(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Remarks

This function closes a connection without waiting for confirmation from the remote side of the connection. Messages in transit may be lost. The connection's closed callback will be called from within this function.

See Also

gt2CloseConnection (see page 245), gt2CloseAllConnections (see page 245), gt2CloseAllConnectionsHard (see page 245)

gt2CloseSocket Function

Summary

Closes a socket.

C++

```
COMMON_API void gt2CloseSocket(
    GT2Socket socket
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.

Remarks

All existing connections will be hard closed, as if gt2CloseAllConnectionsHard (see page 245) was called for this socket. All connections send a close message to the remote side, and any closed callbacks will be called from within this function.

See Also

gt2CreateSocket (see page 248)

gt2Connect Function

Summary

Initiates a connection between a local socket and a remote socket.

C++

```
COMMON_API GT2Result gt2Connect(
    GT2Socket socket,
    GT2Connection * connection,
    const char * remoteAddress,
    const GT2Byte * message,
    int len,
    int timeout,
    GT2ConnectionCallbacks * callbacks,
    GT2Bool blocking
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.
GT2Connection * connection	[out] A pointer to where the connection handle will be stored.
const char * remoteAddress	[in] The address to connect to.
const GT2Byte * message	[in] An optional initial message (may be NULL).
int len	[in] Length of the initial message (may be 0, or -1 for strlen)
int timeout	[in] Timeout in milliseconds (may be 0 for infinite retries)
GT2ConnectionCallbacks * callbacks	[in] GT2Connection (see page 277) related callbacks.
GT2Bool blocking	[in] If GT2True, don't return until the attempt has completed (successfully or unsuccessfuly).

Returns

If blocking is true, GT2Success means the connect attempt succeeded, and anything else means it failed.

Remarks

The gt2Connect function is used to initiate a connection attempt to a remote socket on the Internet. After the remote socket is contacted, both it and the local connector will authenticate the other during a negotation phase. Once the remote socket accepts the connection attempt, the connection will be established. The connection lasts until the closed callback gets called, which can happen because one side closed the connection with gt2CloseConnection (see page 245) (or gt2CloseConnectionHard (see page 246)), there was some sort of error on the connection, or the socket either connection uses is closed.

See Also

gt2ConnectedCallback (see page 271), gt2ClosedCallback (see page 269), gt2CloseConnection (see page 245), gt2AddressToString (see page 244)

gt2CreateAdHocSocket Function

Summary

Creates a new socket, which can be used for making outgoing connections or accepting incoming connections. See gt2CreateSocket Function (see page 248) for details.

C++

```
COMMON_API GT2Result gt2CreateAdHocSocket(
    GT2Socket * socket,
    const char * localAddress,
    int outgoingBufferSize,
    int incomingBufferSize,
    gt2SocketErrorCallback callback
);
```

Parameters

Parameters	Description
GT2Socket * socket	If the result is GT2Success, the socket object handle will be stored at this address.
const char * localAddress	The local address to bind to.
int outgoingBufferSize	Size of per-connection buffer where sent messages waiting to be confirmed are held, use 0 for default.
int incomingBufferSize	Size of per-connection buffer where out-of-order received messages are held, use 0 for default.
gt2SocketErrorCallback callback	A callback that is called if there is an error with the socket.

Remarks

AdHoc Sockets use MAC address instead of IP address.

See Also

gt2CreateSocket (see page 248)

gt2CreateSocket Function

Summary

Creates a new socket, which can be used for making outgoing connections or accepting incoming connections.

C++

```
COMMON_API GT2Result gt2CreateSocket(
    GT2Socket * socket,
    const char * localAddress,
    int outgoingBufferSize,
    int incomingBufferSize,
    gt2SocketErrorCallback callback
);
```

Parameters

Parameters	Description
GT2Socket * socket	[out] Pointer to the socket handle.
const char * localAddress	[in] The address to bind to locally. Typically of the form ":<port>", e.g., ":7777". Can be NULL or "".
int outgoingBufferSize	[in] The byte size of the per-connection buffer for reliable outgoing messages. Can be 0 to use the internal default.
int incomingBufferSize	[in] The byte size of the per-connection buffer for out-of-order reliable incoming messages. Can be 0 to use the internal default.
gt2SocketErrorCallback callback	[in] The callback to be called if there is a fatal error with the socket.

Returns

If the function returns GT2Success then the socket was successfully created. Otherwise, GT2 was unable to create the socket.

Remarks

A socket is an endpoint on the local machine that allows an application to communicate with other applications (through their own sockets) that are typically on remote machines, although they can also be on the local machine (the other application will often be referred to as the "remote machine", even though technically it may be the same machine). A single socket allows an application to both accept connections from remote machines and make connections to remote machines. For most applications, only one socket needs to be created. All incoming connections can be accepted on the socket, and all outgoing connections can be made using the socket. A socket is created with the gt2CreateSocket function. If the function returns GT2Success then the socket was successfully created and bound to the local address (if one was provided). The socket that the "socket" parameter points to is valid until it is closed with gt2CloseSocket (see page 246), or an error is reported to the gt2SocketErrorCallback (see page 274) callback parameter. It is now ready to be used for making outgoing connections, and can be readied for allowing incoming connections by calling gt2Listen (see page 259). If the return result is anything other than GT2Success, GT2 was unable to create the socket.

See Also

gt2SocketErrorCallback (see page 274), gt2CloseSocket (see page 246), gt2Listen (see page 259), gt2Connect (see page 247)

gt2FilteredReceive Function

Summary

Called in response to a gt2ReceiveFilterCallback Type (see page 272) being called. It can be called from within the callback, or at any later time.

C++

```
COMMON_API void gt2FilteredReceive(
    GT2Connection connection,
    int filterID,
    GT2Byte * message,
    int len,
    GT2Bool reliable
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.
int filterID	[in] The ID passed to the gt2ReceiveFilterCallback (see page 272).
GT2Byte * message	[in] The message that was received. May be NULL.
int len	[in] The length of the message in bytes. May be 0.
GT2Bool reliable	[in] True if this is a reliable message.

Remarks

Used to pass on a message after a filter callback has been called. This will cause the message to either be passed to the next filter or, if this was the last filter, to be received. If this is called from the filter callback, the message passed in can be the same message that was passed into the callback.

See Also

gt2ReceiveFilterCallback (see page 272), gt2AddReceiveFilter (see page 243), gt2RemoveReceiveFilter (see page 261)

gt2FilteredSend Function

Summary

Called in response to a gt2SendFilterCallback Type (see page 273) being called. It can be called from within the callback, or at any later time.

C++

```
COMMON_API void gt2FilteredSend(
    GT2Connection connection,
    int filterID,
    const GT2Byte * message,
    int len,
    GT2Bool reliable
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.
int filterID	[in] The ID passed to the gt2SendFilterCallback (see page 273).
const GT2Byte * message	[in] The message that was sent. May be NULL.
int len	[in] The length of the message in bytes. May be 0.
GT2Bool reliable	[in] True if this is a reliable message.

Remarks

Used to pass on a message after a filter callback has been called. This will cause the message to either be passed to the next filter or, if this was the last filter, to be sent. If this is called from the filter callback, the message passed in can be the same message that was passed into the callback. Note that the 7 byte header must be accounted for in the message if the function sends the message reliably.

See Also

gt2SendFilterCallback (see page 273), gt2AddSendFilter (see page 244), gt2RemoveSendFilter (see page 262)

gt2GetConnectionData Function

Summary

Returns the user data pointer stored with this connection.

C++

```
COMMON_API void * gt2GetConnectionData(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Returns

A pointer to this connection's user data.

Remarks

Each connection has a user data pointer associated with it that can be used by the application for any purpose.

See Also

gt2SetSocketData (see page 264), gt2GetSocketData (see page 255), gt2SetConnectionData (see page 263)

gt2GetConnectionSocket Function

Summary

Returns the socket which this connection exists on.

C++

```
COMMON_API GT2Socket gt2GetConnectionSocket(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Returns

The socket on which the connection was created or accepted.

Remarks

All connections are created through either gt2Connect (see page 247) or gt2ConnectAttemptCallback (see page 270). This function will return the socket associated with the connection.

See Also

gt2Connect (see page 247), gt2ConnectAttemptCallback (see page 270)

gt2GetConnectionState Function

Summary

Gets the connection's state.

C++

```
COMMON_API GT2ConnectionState gt2GetConnectionState(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Returns

GT2Connecting, GT2Connected, GT2Closing, or GT2Closed

Remarks

A connection is either connecting, connected, closing, or closed.

GT2Connecting - the connection is still being negotiated GT2Connected - the connection is active (has successfully connected, and not yet closing) GT2Closing - the connection is in the process of closing (sent a close message and waiting for confirmation) GT2Closed - the connection has already been closed and will soon be freed.

gt2GetIncomingBufferFreeSpace Function

Summary

Gets the amount of available space in the connection's incoming buffer.

C++

```
COMMON_API int gt2GetIncomingBufferFreeSpace(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Returns

The size in bytes of the free space in the connection's incoming buffer.

See Also

gt2CreateSocket (see page 248), gt2GetIncomingBufferSize (see page 252), gt2GetOutgoingBufferSize (see page 254), gt2GetOutgoingBufferFreeSpace (see page 253)

gt2GetIncomingBufferSize Function

Summary

Gets the total size of the connection's incoming buffer.

C++

```
COMMON_API int gt2GetIncomingBufferSize(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Returns

The size in bytes of the connection's incoming buffer.

See Also

gt2CreateSocket (see page 248), gt2GetIncomingBufferFreeSpace (see page 251), gt2GetOutgoingBufferSize (see page 254), gt2GetOutgoingBufferFreeSpace (see page 253)

gt2GetLastSentMessageID Function

Summary

Gets the message id for the last reliably sent message. Unreliable messages do not have an id.

C++

```
COMMON_API GT2MessageID gt2GetLastSentMessageID(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Returns

The message ID of the last reliably sent message.

Remarks

This should be called immediately after gt2Send (see page 262). Waiting until after a call to gt2Think (see page 268) can result in an invalid message id being returned. Note that the use of filters that can either drop or delay messages can complicate the process, because in those cases a call to gt2Send (see page 262) does not guarantee that a message will actually be sent. In those cases, gt2GetLastSentMessageID should be called after gt2FilteredSend (see page 249), because the actual message will be sent from within that function.

See Also

gt2WasMessageIDConfirmed (see page 269)

gt2GetLocalIP Function

Summary

Gets a socket's local IP.

C++

```
COMMON_API unsigned int gt2GetLocalIP(
    GT2Socket socket
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.

Returns

The local IP in network byte order.

See Also

gt2GetRemoteIP (see page 254), gt2GetRemotePort (see page 254), gt2GetLocalPort (see page 253)

gt2GetLocalPort Function

Summary

Get's a socket's local port.

C++

```
COMMON_API unsigned short gt2GetLocalPort(
    GT2Socket socket
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.

Returns

The local port in host byte order.

See Also

gt2GetRemoteIP (see page 254), gt2GetRemotePort (see page 254), gt2GetLocalIP (see page 253)

gt2GetOutgoingBufferFreeSpace Function

Summary

Gets the amount of available space in the connection's outgoing buffer.

C++

```
COMMON_API int gt2GetOutgoingBufferFreeSpace(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Returns

The size in bytes of the free space in the connection's outgoing buffer.

See Also

gt2CreateSocket (see page 248), gt2GetIncomingBufferSize (see page 252), gt2GetIncomingBufferFreeSpace (see page 251), gt2GetOutgoingBufferSize (see page 254)

gt2GetOutgoingBufferSize Function

Summary

Gets the total size of the connection's outgoing buffer.

C++

```
COMMON_API int gt2GetOutgoingBufferSize(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Returns

The size in bytes of the connection's outgoing buffer.

See Also

gt2CreateSocket (see page 248), gt2GetIncomingBufferSize (see page 252), gt2GetIncomingBufferFreeSpace (see page 251), gt2GetOutgoingBufferFreeSpace (see page 253)

gt2GetRemoteIP Function

Summary

Gets the connection's remote IP.

C++

```
COMMON_API unsigned int gt2GetRemoteIP(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Returns

The remote IP in network byte order.

Remarks

Gets the IP of the computer on the remote side of the connection.

See Also

gt2GetRemotePort (see page 254), gt2GetLocalIP (see page 253), gt2GetLocalPort (see page 253)

gt2GetRemotePort Function

Summary

Get's the connection's remote port.

C++

```
COMMON_API unsigned short gt2GetRemotePort(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Returns

The remote port in host byte order.

Remarks

Gets the port of the computer on the remote side of the connection.

See Also

gt2GetRemoteIP (see page 254), gt2GetLocalIP (see page 253), gt2GetLocalPort (see page 253)

gt2GetSocketData Function

Summary

Returns the user data pointer stored with this socket.

C++

```
COMMON_API void * gt2GetSocketData(
    GT2Socket socket
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.

Returns

A pointer to this socket's user data.

Remarks

Each socket has a user data pointer associated with it that can be used by the application for any purpose.

See Also

gt2SetSocketData (see page 264), gt2SetConnectionData (see page 263), gt2GetConnectionData (see page 250)

gt2GetSocketSOCKET Function

Summary

This function returns the actual underlying socket for a GT2Socket Type (see page 277).

C++

```
COMMON_API SOCKET gt2GetSocketSOCKET(
    GT2Socket socket
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.

Returns

The underlying socket associated with the GT2Socket (see page 277).

Remarks

This can be used for socket sharing purposes, along with the gt2UnrecognizedMessageCallback (see page 274).

See Also

gt2SetUnrecognizedMessageCallback (see page 265)

gt2HostToNetworkInt Function

Summary

Convert an int from host to network byte order.

C++

```
COMMON_API unsigned int gt2HostToNetworkInt(
    unsigned int i
);
```

Parameters

Parameters	Description
unsigned int i	[in] Int to convert.

Returns

The int in network byte order.

Remarks

This is a utility function to help deal with byte order differences for multi-platform applications. Convert from host to network byte order before sending over the network, then convert back to host byte order when receiving.

See Also

gt2NetworkToHostInt (see page 259), gt2NetworkToHostShort (see page 260), gt2HostToNetworkShort (see page 256)

gt2HostToNetworkShort Function

Summary

Convert a short from host to network byte order.

C++

```
COMMON_API unsigned short gt2HostToNetworkShort(
    unsigned short s
);
```

Parameters

Parameters	Description
unsigned short s	[in] Short to convert.

Returns

The short in network byte order.

Remarks

This is a utility function to help deal with byte order differences for multi-platform applications. Convert from host to network byte order before sending over the network, then convert back to host byte order when receiving.

See Also

gt2NetworkToHostInt (see page 259), gt2HostToNetworkInt (see page 256), gt2NetworkToHostShort (see page 260)

gt2IPToAliases Function

Summary

Get the aliases associated with an IP address.

C++

```
COMMON_API char ** gt2IPToAliases(
    unsigned int ip
);
```

Parameters

Parameters	Description
unsigned int ip	[in] IP to lookup, in network byte order.

Returns

Aliases associated with the IP address.

Remarks

This is a utility function which provides a subset of the functionality of gt2IPToHostInfo (see page 257). See the gt2IPToHostInfo (see page 257) documentation for important details.

See Also

gt2IPToHostInfo (see page 257)

gt2IPToHostInfo Function

Summary

Looks up DNS host information based on an IP.

C++

```
COMMON_API const char * gt2IPToHostInfo(
    unsigned int ip,
    char *** aliases,
    unsigned int *** ips
);
```

Parameters

Parameters	Description
unsigned int ip	[in] IP to look up, in network byte order.
char *** aliases	[out] On success, the variable passed in will point to a NULL-terminated list of alternate names for the host. Can be NULL.
unsigned int *** ips	[out] On success, the variable passed in will point to a NULL-terminated list of alternate IPs for the host. Can be NULL.

Returns

The hostname associated with the IP, or NULL if no information was available for the host.

Description

Gets the host information for a machine on the Internet. The first version takes an IP in network byte order, and the second version takes a string that is either a dotted ip ("1.2.3.4"), or a hostname ("www.gamespy.com"). If the function can successfully lookup the host's info, the host's main hostname will be returned. If it cannot find the host's info, it returns NULL. For the aliases parameter, pass in a pointer to a variable of type (char **). If this parameter is not NULL, and the function succeeds, the variable will point to a NULL-terminated list of alternate names for the host. For the ips parameter, pass in a pointer to a variable of type (int **). If this parameter is not NULL, and the function succeeds, the variable will point to a NULL-terminated list of altername IPs for the host. Each element in the list is actually a pointer to an unsigned int, which is an IP address in network byte order. The return value, aliases, and IPs all point to an internal data structure, and none of these values should be modified directly. Also, the data is only valid until another call needs to use the same data structure (virtually ever internet address function will use this data structure). If the data will be needed in the future, it should be copied off. If this function needs to resolve a hostname ("host.com") it may need to contact a DNS server, which can cause the function to block for an indefinite period of time. Usually it is < 2 seconds, but on certain systems, and under certain circumstances, it can take 30 seconds or longer.

Remarks

If the function can successfully lookup the host's info, the host's main hostname will be returned. If it cannot find the host's info, it returns NULL.

For the aliases parameter, pass in a pointer to a variable of type (char **). If this parameter is not NULL, and the function succeeds, the variable will point to a NULL-terminated list of alternate names for the host. For the ips parameter, pass in a pointer to a variable of type (int **). If this parameter is not NULL, and the function succeeds, the variable will point to a NULL-terminated list of altername IPs for the host. Each element in the list is actually a pointer to an unsigned int, which is an IP address in network byte order. The return value, aliases, and IPs all point to an internal data structure, and none of these values should be modified directly. Also, the data is only valid until another call needs to use the same data structure (virtually ever internet address function will use this data structure). If the data will be needed in the future, it should be copied off. This function may need to contact a DNS server, which can cause the function to block for an indefinite period of time. Usually it is < 2 seconds, but on certain systems, and under certain circumstances, it can take 30 seconds or longer.

See Also

gt2StringToHostInfo (see page 267), gt2IPToHostname (see page 258), gt2IPToAliases (see page 257), gt2IPToIPs (see page 258)

gt2IPToHostname Function

Summary

Get the hostname associated with an IP address.

C++

```
COMMON_API const char * gt2IPToHostname(
    unsigned int ip
);
```

Parameters

Parameters	Description
unsigned int ip	[in] IP to lookup, in network byte order.

Returns

Hostname associated with the IP address.

Remarks

This is a utility function which provides a subset of the functionality of gt2IPToHostInfo (see page 257). See the gt2IPToHostInfo (see page 257) documentation for important details.

See Also

gt2IPToHostInfo (see page 257)

gt2IPToIPs Function

Summary

Get the IPs associated with an IP address.

C++

```
COMMON_API unsigned int ** gt2IPToIPs(
    unsigned int ip
);
```

Parameters

Parameters	Description
unsigned int ip	[in] IP to lookup, in network byte order.

Returns

IPs associated with the IP address.

Remarks

This is a utility function which provides a subset of the functionality of gt2IPToHostInfo (see page 257). See the gt2IPToHostInfo (see page 257) documentation for important details.

See Also

gt2IPToHostInfo (see page 257)

gt2Listen Function

Summary

Start (or stop) listening for incoming connections on a socket.

C++

```
COMMON_API void gt2Listen(
    GT2Socket socket,
    gt2ConnectAttemptCallback callback
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.
gt2ConnectAttemptCallback callback	[in] Function to be called when the operation completes

Remarks

Once a socket starts listening, any connections attempts will cause the callback to be called.

If the socket is already listening, this callback will replace the existing callback being used If the callback is NULL, this will cause the connection to stop listening.

See Also

gt2CreateSocket (see page 248), gt2ConnectAttemptCallback (see page 270)

gt2NetworkToHostInt Function

Summary

Convert an int from network to host byte order.

C++

```
COMMON_API unsigned int gt2NetworkToHostInt(
    unsigned int i
);
```

Parameters

Parameters	Description
unsigned int i	[in] Int to convert.

Returns

The int in host byte order.

Remarks

This is a utility function to help deal with byte order differences for multi-platform applications. Convert from host to network byte order before sending over the network, then convert back to host byte order when receiving.

See Also

gt2HostToNetworkInt (see page 256), gt2NetworkToHostShort (see page 260), gt2HostToNetworkShort (see page 256)

gt2NetworkToHostShort Function

Summary

Convert a short from network to host byte order.

C++

```
COMMON_API unsigned short gt2NetworkToHostShort(
    unsigned short s
);
```

Parameters

Parameters	Description
unsigned short s	[in] Short to convert.

Returns

The short in host byte order.

Remarks

This is a utility function to help deal with byte order differences for multi-platform applications. Convert from host to network byte order before sending over the network, then convert back to host byte order when receiving.

See Also

gt2NetworkToHostInt (see page 259), gt2HostToNetworkInt (see page 256), gt2HostToNetworkShort (see page 256)

gt2Ping Function

Summary

Sends a ping on a connection in an attempt to determine latency.

C++

```
COMMON_API void gt2Ping(
    GT2Connection connection
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.

Remarks

The ping callback will, which is set as part of the GT2ConnectionCallbacks (see page 275) in either gt2Connect (see page 247) or gt2Accept (see page 242), will be called if and when a ping finishes making a round-trip between the local end of the connection and the remote end. The ping is unreliable, and either it or the pong sent in reply could be dropped, resulting in the callback never being called. Or it could even arrive multiple times, resulting in multiple calls to the callback (this case is very rare).

See Also

gt2PingCallback (see page 272)

gt2Reject Function

Summary

Rejects a connection attempt.

C++

```
COMMON_API void gt2Reject(
    GT2Connection connection,
    const GT2Byte * message,
    int len
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.
const GT2Byte * message	[in] Rejection message. May be NULL.
int len	[in] Length of the rejection message. May be 0. A len of -1 is equivalent to (strlen(message) + 1).

Remarks

After a socket's gt2ConnectAttemptCallback (see page 270) has been called, this function can be used to reject the incoming connection attempt. It can be called from either within the callback or some later time. Once the function is called the connection is considered closed and cannot be referenced again. The remote side attempting the connection will have its connected callback called with the result set to gt2Rejected. If the message is not NULL and the len is not 0, the message will be sent with the rejection, and passed into the remote side's connected callback. Note that the 7 byte header must be accounted for in the message since this function will send the rejection message reliably.

See Also

gt2Listen (see page 259), gt2ConnectAttemptCallback (see page 270), gt2Accept (see page 242)

gt2RemoveReceiveFilter Function

Summary

Removes a filter from the connection's incoming data filter list.

C++

```
COMMON_API void gt2RemoveReceiveFilter(
    GT2Connection connection,
    gt2ReceiveFilterCallback callback
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.
gt2ReceiveFilterCallback callback	[in] The filtering callback to remove. NULL removes all filters.

Remarks

Filters should not be removed while a message is being filtered. If the callback is NULL, all of the filters will be removed.

See Also

gt2ReceiveFilterCallback (see page 272), gt2AddReceiveFilter (see page 243), gt2FilteredReceive (see page 249)

gt2RemoveSendFilter Function

Summary

Removes a filter from the connection's outgoing data filter list.

C++

```
COMMON_API void gt2RemoveSendFilter(
    GT2Connection connection,
    gt2SendFilterCallback callback
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.
gt2SendFilterCallback callback	[in] The filtering callback to remove. NULL removes all filters.

Remarks

Filters should not be removed while a message is being filtered. If the callback is NULL, all of the filters will be removed.

See Also

gt2SendFilterCallback (see page 273), gt2AddSendFilter (see page 244), gt2FilteredSend (see page 249)

gt2Send Function

Summary

Sends data over a connection, reliably or unreliably.

C++

```
COMMON_API GT2Result gt2Send(
    GT2Connection connection,
    const GT2Byte * message,
    int len,
    GT2Bool reliable
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.
const GT2Byte * message	[in] The message to send. Can be NULL.
int len	[in] The length of the message. Can be 0. A len of -1 is equivalent to (strlen(message) + 1).
GT2Bool reliable	[in] if GT2True, send the message reliably, otherwise send it unreliably.

Remarks

Once a connection has been established, messages can be sent back and forth on it. To send a message, use the gt2Send function. If message is NULL or len is 0, then an empty message will be sent. When an empty message is received, message will be NULL and len will be 0. If the message is sent reliably, it is guaranteed to arrive, arrive only once, and arrive in order (relative to other reliable messages). If the message is sent unreliably, then it is not guaranteed to arrive, and if it does arrive, it is not guaranteed to arrive in order, or only once. Note that the 7 byte header must be accounted for in the message if the function sends the message reliably.

See Also

gt2ReceivedCallback (see page 272)

gt2SendRawUDP Function

Summary

Sends a raw UDP datagram through the socket. This function bypasses the normal connection logic. Note that all messages sent this way will be unreliable. To broadcast a datagram, omit the IP from the remoteAddress (e.g., ":12345").

C++

```
COMMON_API GT2Result gt2SendRawUDP(
    GT2Socket socket,
    const char * remoteAddress,
    const GT2Byte * message,
    int len
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The socket through which to send the raw UDP datagram.
const char * remoteAddress	[in] The address to which to send the datagram.
const GT2Byte * message	[in] The message to send, or NULL for an empty datagram.
int len	[in] The len of the message (0 for an empty message, ignored if message==NULL).

gt2SetConnectionData Function

Summary

Stores a user data pointer with this connection.

C++

```
COMMON_API void gt2SetConnectionData(
    GT2Connection connection,
    void * data
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.
void * data	[in] A pointer to this connection's user data.

Remarks

Each connection has a user data pointer associated with it that can be used by the application for any purpose.

See Also

gt2SetSocketData (see page 264), gt2GetSocketData (see page 255), gt2GetConnectionData (see page 250)

gt2SetReceiveDump Function

Summary

Sets a callback to which all incoming UDP packets are passed. This is at a lower level than the filters, can only be used for monitoring, and is designed for debugging purposes.

C++

```
COMMON_API void gt2SetReceiveDump(
    GT2Socket socket,
    gt2DumpCallback callback
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.
gt2DumpCallback callback	[in] The dump callback to set.

Remarks

Sets a callback to be called whenever a UDP datagram or a connection reset is received. Pass in a callback of NULL to remove the callback. The dump sit at a lower level than the filters, and allow an app to keep an eye on exactly what datagrams are being received, allowing for close monitoring. The dump cannot be used to modify data, only monitor it. The dump is useful for debugging purposes, and to keep track of data receive rates (e.g., the Quake 3 engine's netgraph). Note that these are the actual UDP datagrams being received - datagrams may be dropped, repeated, or out-of-order. Control datagrams (those used internally by the protocol) will be passed to the dump callback, and certain application messages will have a header at the beginning.

See Also

gt2DumpCallback (see page 271), gt2SetSendDump (see page 264)

gt2SetSendDump Function

Summary

Sets a callback to which all outgoing UDP packets are passed. This is at a lower level than the filters, can only be used for monitoring, and is designed for debugging purposes.

C++

```
COMMON_API void gt2SetSendDump(
    GT2Socket socket,
    gt2DumpCallback callback
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.
gt2DumpCallback callback	[in] The dump callback to set.

Remarks

Sets a callback to be called whenever a UDP datagram is sent. Pass in a callback of NULL to remove the callback. The dump sit at a lower level than the filters, and allow an app to keep an eye on exactly what datagrams are being sent, allowing for close monitoring. The dump cannot be used to modify data, only monitor it. The dump is useful for debugging purposes, and to keep track of data send rates (e.g., the Quake 3 engine's netgraph). Note that these are the actual UDP datagrams being sent - datagrams may be dropped, repeated, or out-of-order. Control datagrams (those used internally by the protocol) will be passed to the dump callback, and certain application messages will have a header at the beginning.

See Also

gt2DumpCallback (see page 271), gt2SetReceiveDump (see page 263)

gt2SetSocketData Function

Summary

Stores a user data pointer with this socket.

C++

```
COMMON_API void gt2SetSocketData(
    GT2Socket socket,
    void * data
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.
void * data	[in] A pointer to this socket's user data.

Remarks

Each socket has a user data pointer associated with it that can be used by the application for any purpose.

See Also

gt2GetSocketData (see page 255), gt2SetConnectionData (see page 263), gt2GetConnectionData (see page 250)

gt2SetUnrecognizedMessageCallback Function

Summary

Used to handle unrecognized messages, usually used for sharing a socket with another SDK.

C++

```
COMMON_API void gt2SetUnrecognizedMessageCallback(
    GT2Socket socket,
    gt2UnrecognizedMessageCallback callback
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.
gt2UnrecognizedMessageCallback callback	[in] Function to be called when an unrecognized message is received.

Remarks

This is used to set a callback to be called every time a socket receives a message that it cannot match up to an existing connection. If a GT2Socket (see page 277) object's underlying socket is being shared, this allows an application to check for data that was not meant for GT2. If the callback parameter is NULL, then any previously set callback will be removed.

This is typically used when you are sharing a GT2Socket (see page 277) with another SDK, such as QR2 or NAT Negotiation. Setting an unrecognized callback allows you to pass messages meant for another SDK to the appropriate place.

See Also

gt2UnrecognizedMessageCallback (see page 274), gt2GetSocketSOCKET (see page 255)

gt2StringToAddress Function

Summary

Converts a string address, which is either a hostname ("www.gamespy.net") or a dotted IP ("1.2.3.4") into an IP and a port.

C++

```
COMMON_API GT2Bool gt2StringToAddress(
    const char * string,
    unsigned int * ip,
    unsigned short * port
);
```

Parameters

Parameters	Description
const char * string	[in] String to convert.
unsigned int * ip	[out] IP in network byte order. Can be NULL.
unsigned short * port	[out] Port in host byte order. Can be NULL.

Returns

Returns GT2False if there was an error parsing the string, or if a supplied hostname can't be resolved.

Remarks

The IP is stored in network byte order, and the port is stored in host byte order.

Possible string forms:

NULL => all IPs, any port (localAddress only). "" => all IPs, any port (localAddress only).

"1.2.3.4" => 1.2.3.4 IP, any port (localAddress only).

"host.com" => host.com's IP, any port (localAddress only).

":2786" => all IPs, 2786 port (localAddress only).

"1.2.3.4:0" => 1.2.3.4 IP, any port (localAddress only).

"host.com:0" => host.com's IP, any port (localAddress only).

"0.0.0.0:2786" => all IPs, 2786 port (localAddress only).

"1.2.3.4:2786" => 1.2.3.4 IP, 2786 port (localAddress or remoteAddress).

"host.com:2786" => host.com's IP, 2786 port (localAddress or remoteAddress).

If this function needs to resolve a hostname ("host.com") it may need to contact a DNS server, which can cause the function to block for an indefinite period of time. Usually it is < 2 seconds, but on certain systems, and under certain circumstances, it can take 30 seconds or longer.

See Also

gt2Connect (see page 247), gt2CreateSocket (see page 248), gt2StringToHostInfo (see page 267)

gt2StringToAliases Function

Summary

Get the aliases associated with a hostname or dotted IP.

C++

```
COMMON_API char ** gt2StringToAliases(
    const char * string
);
```

Parameters

Parameters	Description
const char * string	[in] Hostname or IP to lookup.

Returns

Aliases associated with a hostname or dotted IP.

Remarks

This is a utility function which provides a subset of the functionality of gt2StringToHostInfo (see page 267). See the gt2StringToHostInfo (see page 267) documentation for important details.

See Also

gt2StringToHostInfo (see page 267)

gt2StringToHostInfo Function

Summary

Looks up DNS host information based on a hostname or dotted IP.

C++

```
COMMON_API const char * gt2StringToHostInfo(
    const char * string,
    char *** aliases,
    unsigned int *** ips
);
```

Parameters

Parameters	Description
const char * string	[in] Hostname ("www.gamespy.net") or dotted IP ("1.2.3.4") to lookup.
char *** aliases	[in] On success, the variable passed in will point to a NULL-terminated list of alternate names for the host. Can be NULL.
unsigned int *** ips	[in] On success, the variable passed in will point to a NULL-terminated list of alternate IPs for the host. Can be NULL.

Returns

The hostname associated with the string, or NULL if no information was available for the host.

Remarks

If the function can successfully lookup the host's info, the host's main hostname will be returned. If it cannot find the host's info, it returns NULL.

For the aliases parameter, pass in a pointer to a variable of type (char **). If this parameter is not NULL, and the function succeeds, the variable will point to a NULL-terminated list of alternate names for the host. For the ips parameter, pass in a pointer to a variable of type (int **). If this parameter is not NULL, and the function succeeds, the variable will point to a NULL-terminated list of altername IPs for the host. Each element in the list is actually a pointer to an unsigned int, which is an IP address in network byte order. The return value, aliases, and IPs all point to an internal data structure, and none of these values should be modified directly. Also, the data is only valid until another call needs to use the same data structure (virtually ever internet address function will use this data structure). If the data will be needed in the future, it should be copied off. This function may need to contact a DNS server, which can cause the function to block for an indefinite period of time. Usually it is < 2 seconds, but on certain systems, and under certain circumstances, it can take 30 seconds or longer.

See Also

gt2IPToHostInfo (see page 257), gt2StringToHostname (see page 267), gt2StringToAliases (see page 266), gt2StringToIPs (see page 268)

gt2StringToHostname Function

Summary

Get the hostname associated with a hostname or dotted IP.

C++

```
COMMON_API const char * gt2StringToHostname(
    const char * string
);
```

Parameters

Parameters	Description
const char * string	[in] Hostname or IP to lookup.

Returns

Hostname associated with a hostname or dotted IP.

Remarks

This is a utility function which provides a subset of the functionality of gt2StringToHostInfo (see page 267). See the gt2StringToHostInfo (see page 267) documentation for important details.

See Also

gt2StringToHostInfo (see page 267)

gt2StringToIPs Function

Summary

Get the IPs associated with a hostname or dotted IP.

C++

```
COMMON_API unsigned int ** gt2StringToIPs(
    const char * string
);
```

Parameters

Parameters	Description
const char * string	[in] Hostname or IP to lookup.

Returns

IPs associated with a hostname or dotted IP.

Remarks

This is a utility function which provides a subset of the functionality of gt2StringToHostInfo (see page 267). See the gt2StringToHostInfo (see page 267) documentation for important details.

See Also

gt2StringToHostInfo (see page 267)

gt2Think Function

Summary

Does any thinking for this socket and its connections.

C++

```
COMMON_API void gt2Think(
    GT2Socket socket
);
```

Parameters

Parameters	Description
GT2Socket socket	[in] The handle to the socket.

Remarks

Callbacks are typically called from within this function (although they can also be called from other places). It is possible that during this think the socket or any of its connections may be closed, so you must take care if calling other GT2 functions immediately after thinking. The more frequently this function is called, the faster GT2 will be able to respond (and reply to) messages. The general rule is to call it at frequently as you can, although calling it faster than every 10-20 milliseconds is probably unnecessary. If you are using gt2Ping (see page 260) to measure ping times, then the accuracy of the latency measurement will increase with the frequency at which this function is called.

gt2WasMessageIDConfirmed Function

Summary

Checks if confirmation has been received that the remote end received a particular reliable message.

C++

```
COMMON_API GT2Bool gt2WasMessageIDConfirmed(
    GT2Connection connection,
    GT2MessageID messageID
);
```

Parameters

Parameters	Description
GT2Connection connection	[in] The handle to the connection.
GT2MessageID messageID	[in] The ID of the message to check for confirmation.

Returns

GT2True if confirmation was received locally that the reliable message represented by messageID was received by the remote end of the connection, GT2False if confirmation was not yet received.

Remarks

This should only be called on message ids that were returned by gt2GetLastSendMessageID, and should be used relatively soon after the message was sent, due to message ids wrapping around after a period of time.

See Also

gt2GetLastSentMessageID (see page 252)

Callbacks

Types

Name	Description
gt2ClosedCallback (see page 269)	This callback is called when the connection has been closed.
gt2ConnectAttemptCallback (see page 270)	This notifies the socket that a remote system is attempting a connection.
gt2ConnectedCallback (see page 271)	This callback is called when a connection attempt with gt2Connect (see page 247) finishes.
gt2DumpCallback (see page 271)	Called whenever data is sent or received over a socket.
gt2PingCallback (see page 272)	This callback is called when a response to a ping sent on this connection is received.
gt2ReceivedCallback (see page 272)	This callback is called when a message is sent from the remote system with a gt2Send (see page 262).
gt2ReceiveFilterCallback (see page 272)	Callback for filtering incoming data.
gt2SendFilterCallback (see page 273)	Callback for filtering outgoing data.
gt2SocketErrorCallback (see page 274)	This callback is used to notify the application of a closed socket or fatal socket error condition.
gt2UnrecognizedMessageCallback (see page 274)	This callback gets called when the sock receives a message that it cannot match to an existing connection.

gt2ClosedCallback Type

Summary

This callback is called when the connection has been closed.

C++

```
typedef void (* gt2ClosedCallback)(GT2Connection connection, GT2CloseReason reason);
```

Parameters

Parameters	Description
connection	[in] The handle to the connection.
reason	[in] The reason the connection closed.

Remarks

A connection close can be caused by either side calling gt2CloseConnection (see page 245) (or gt2CloseConnectionHard (see page 246)), either side closing the socket, or some sort of error. The connection cannot be used again once this callback returns.

See Also

gt2CloseConnection (see page 245), gt2CloseConnectionHard (see page 246), gt2Connect (see page 247), gt2Accept (see page 242)

gt2ConnectAttemptCallback Type

Summary

This notifies the socket that a remote system is attempting a connection.

C++

```
typedef void (* gt2ConnectAttemptCallback)(GT2Socket socket, GT2Connection connection,
unsigned int ip, unsigned short port, int latency, GT2Byte * message, int len);
```

Parameters

Parameters	Description
socket	[in] The handle to the socket.
connection	[in] The handle to the connection.
ip	[in] The IP (network byte order) from which the connect attempt is coming.
port	[in] The port (host byte order) from which the connect attempt is coming
latency	[in] estimate of the round-trip time between the two machines (in milliseconds).
message	[in] Optional initial data sent with the connect attempt. May be NULL.
len	[in] Length of the initial data. May be 0.

Remarks

The IP and port of the remote system is provided, along with an optional initial message, and a latency estimate. These can be used to validate/authenticate the connecting system. This connection must either be accepted with gt2Accept (see page 242), or rejected with gt2Reject (see page 261). These can be called from within this callback, however they do not need to be. They can be called at any time after this callback is received. This is very useful for systems that need to check with another machine to authenticate the user (such as for a CDKey system). The latency is only an estimate, however it can be used for things such as only allowing low-ping or high-ping users onto a server.

See Also

gt2Listen (see page 259), gt2Connect (see page 247), gt2Accept (see page 242), gt2Reject (see page 261)

gt2ConnectedCallback Type

Summary

This callback is called when a connection attempt with gt2Connect Function (see page 247) finishes.

C++

```
typedef void (* gt2ConnectedCallback)(GT2Connection connection, GT2Result result, GT2Byte *
message, int len);
```

Parameters

Parameters	Description
connection	[in] The handle to the connection.
result	[in] The result of the connection attempt. Anything aside from GT2Success indicates failure.
message	[in] If result is GT2Rejected, this is the rejection message. May be NULL.
len	[in] If result is GT2Rejected, the length of message. May be 0.

Remarks

If result is GT2Success, then this connection attempt succeeded. The connection object can now be used for sending/receiving messages. Any other result indicates connection failure, and the connection object cannot be used again after this callback returns. If the result is GT2Rejected, then message contains an optional rejection message sent by the listener. If result is not GT2Rejected, then message will be NULL and len will be 0.

See Also

gt2Connect (see page 247)

gt2DumpCallback Type

Summary

Called whenever data is sent or received over a socket.

C++

```
typedef void (* gt2DumpCallback)(GT2Socket socket, GT2Connection connection, unsigned int
ip, unsigned short port, GT2Bool reset, const GT2Byte * message, int len);
```

Parameters

Parameters	Description
socket	[in] The handle to the socket.
connection	[in] The handle to the connection associated with this message, or NULL if there is no connection for this message.
ip	[in] The remote IP address, in network byte order.
port	[in] The remote host, in host byte order.
reset	[in] If true, the connection has been reset (only used by the receive callback).
message	[in] The message (should not be modified).
len	[in] The length of the message.

Remarks

Trying to send a message from within the send dump callback, or letting the socket think from within the receive dump callback can cause serious problems, and should not be done.

See Also

gt2SetSendDump (see page 264), gt2SetReceiveDump (see page 263)

gt2PingCallback Type

Summary

This callback is called when a response to a ping sent on this connection is received.

C++

```
typedef void (* gt2PingCallback)(GT2Connection connection, int latency);
```

Parameters

Parameters	Description
connection	[in] The handle to the connection.
latency	[in] The round trip time of the ping, in milliseconds.

Remarks

This callback gives a measure of the time it takes for a datagram to make a round-trip from one connection to the other. The latency reported in this callback will typically be larger than that reported by using ICMP pings between the two machines (the "ping" program uses ICMP pings), because ICMP pings happen at a lower level in the operating system. However, the ping reported in this callback will much more accurately reflect the latency of the application, as the application's messages must go through the same path as these pings, as opposed to ICMP.

See Also

gt2Ping (see page 260)

gt2ReceivedCallback Type

Summary

This callback is called when a message is sent from the remote system with a gt2Send Function (see page 262).

C++

```
typedef void (* gt2ReceivedCallback)(GT2Connection connection, GT2Byte * message, int len,
GT2Bool reliable);
```

Parameters

Parameters	Description
connection	[in] The handle to the connection.
message	[in] The message that was sent. May be NULL.
len	[in] The length of the message. May be 0.
reliable	[in] GT2True if the message was sent reliably.

Remarks

If an message is sent from the remote end of the connection reliably, then it will always be received with this callback. If it is not sent reliably, then the message might not be received, it might be received out of order, or it might be received more than once (very rare).

See Also

gt2Send (see page 262)

gt2ReceiveFilterCallback Type

Summary

Callback for filtering incoming data.

C++

```
typedef void (* gt2ReceiveFilterCallback)(GT2Connection connection, int filterID, GT2Byte *
message, int len, GT2Bool reliable);
```

Parameters

Parameters	Description
connection	[in] The handle to the connection.
filterID	[in] Pass this ID to gtFilteredReceive.
message	[in] The message that was received. Will be NULL if an empty message.
len	[in] The length of the message in bytes. Will be 0 if an empty message.
reliable	[in] True if this is a reliable message.

Description

Callback for filtering incoming data. Call gt2FilteredRecieve with the filtered data, either from within the callback or later. the message may point to a memory location supplied to gt2FilteredReceive (see page 249) by a previous filter. so if this filter's call to gt2FilteredReceive (see page 249) is delayed, it is the filter's responsibility to make sure the data is still around when and if it is needed.

Remarks

Call gt2FilteredRecieve with the filtered data, either from within the callback or later.

The message may point to a memory location supplied to gt2FilteredReceive (see page 249) by a previous filter. So if this filter's call to gt2FilteredReceive (see page 249) is delayed, it is the filter's responsibility to make sure the data is still around when and if it is needed.

See Also

gt2AddReceiveFilter (see page 243), gt2RemoveReceiveFilter (see page 261), gt2FilteredReceive (see page 249)

gt2SendFilterCallback Type

Summary

Callback for filtering outgoing data.

C++

```
typedef void (* gt2SendFilterCallback)(GT2Connection connection, int filterID, const
GT2Byte * message, int len, GT2Bool reliable);
```

Parameters

Parameters	Description
connection	[in] The handle to the connection.
filterID	[in] Pass this ID to gt2FilteredSend (see page 249).
message	[in] The message being sent. Will be NULL if an empty message.
len	[in] The length of the message being sent, in bytes. Will be 0 if an empty message.
reliable	[in] If the message is being sent reliably.

Remarks

Call gt2FilteredSend (see page 249) with the filtered data, either from within the callback or later.

The message points to the same memory location as the message passed to gt2Send (see page 262) (or gt2FilteredSend (see page 249)). So if the call to gt2FilteredSend (see page 249) is delayed, it is the filter's responsibility to make sure the data is still around when and if it is needed.

See Also

gt2AddSendFilter (see page 244), gt2RemoveSendFilter (see page 262), gt2FilteredSend (see page 249)

gt2SocketErrorCallback Type

Summary

This callback is used to notify the application of a closed socket or fatal socket error condition.

C++

```
typedef void (* gt2SocketErrorCallback)(GT2Socket socket);
```

Parameters

Parameters	Description
socket	[in] The handle to the socket.

Remarks

Once this callback returns, the socket and all of its connections are invalid and can no longer be used. All connections that use this socket are terminated, and their gt2CloseCallback callbacks will be called before this callback is called (with the reason set to GT2SocketError).

See Also

gt2CreateSocket (see page 248)

gt2UnrecognizedMessageCallback Type

Summary

This callback gets called when the sock receives a message that it cannot match to an existing connection.

C++

```
typedef GT2Bool (* gt2UnrecognizedMessageCallback)(GT2Socket socket, unsigned int ip,
unsigned short port, GT2Byte * message, int len);
```

Parameters

Parameters	Description
socket	[in] The handle to the socket.
ip	[in] The IP address of the remote machine the message came from (in network byte order).
port	[in] The port on the remote machine (in host byte order).
message	[in] The message (may be NULL for an empty message).
len	[in] The length of the message (may be 0).

Returns

GT2True if the callback recognizes the message and handles it. GT2False if GT2 should handle the message.

Remarks

If the callback recognizes the message and handles it, it should return GT2True, which will tell the socket to ignore the message. If the callback does not recognize the message, it should return GT2False, which tells the socket to let the other side know there is no connection.

See Also

gt2SetUnrecognizedMessageCallback (see page 265), gt2GetSocketSOCKET (see page 255)

Structures

Structures

Name	Description
GT2ConnectionCallbacks (see page 275)	Callbacks set for each connection.

GT2ConnectionCallbacks Structure

Summary

Callbacks set for each connection.

C++

```
typedef struct {
  gt2ConnectedCallback connected;
  gt2ReceivedCallback received;
  gt2ClosedCallback closed;
  gt2PingCallback ping;
} GT2ConnectionCallbacks;
```

Members

Members	Description
gt2ConnectedCallback connected;	Called when gt2Connect is complete.
gt2ReceivedCallback received;	Called when a message is received.
gt2ClosedCallback closed;	Called when the connection is closed (remotely or locally).
gt2PingCallback ping;	Called when a ping reply is received.

Enumerations

Enumerations

Name	Description
GT2CloseReason (see page 275)	Reason the connection was closed.
GT2ConnectionState (see page 276)	Possible states for any GT2Connection (see page 277).
GT2Result (see page 276)	Result of a GT2 operation. Check individual function definitions to see possible results.

GT2CloseReason Enumeration

Summary

Reason the connection was closed.

C++

```
typedef enum {
  GT2LocalClose,
  GT2RemoteClose,
  GT2CommunicationError,
  GT2SocketError,
  GT2NotEnoughMemory
} GT2CloseReason;
```

Members

Members	Description
GT2LocalClose	The connection was closed with gt2CloseConnection.
GT2RemoteClose	The connection was closed remotely.

GT2CommunicationError	An invalid message was received (it was either unexpected or incorrectly formatted).
GT2SocketError	An error with the socket forced the connection to close.
GT2NotEnoughMemory	There wasn't enough memory to store an incoming or outgoing message.

GT2ConnectionState Enumeration

Summary

Possible states for any GT2Connection Type (see page 277).

C++

```
typedef enum {
  GT2Connecting,
  GT2Connected,
  GT2Closing,
  GT2Closed
} GT2ConnectionState;
```

Members

Members	Description
GT2Connecting	Negotiating the connection.
GT2Connected	The connection is active.
GT2Closing	The connection is being closed.
GT2Closed	The connection has been closed and can no longer be used.

GT2Result Enumeration

Summary

Result of a GT2 operation. Check individual function definitions to see possible results.

C++

```
typedef enum {
  GT2Success,
  GT2OutOfMemory,
  GT2Rejected,
  GT2NetworkError,
  GT2AddressError,
  GT2DuplicateAddress,
  GT2TimedOut,
  GT2NegotiationError,
  GT2InvalidConnection,
  GT2InvalidMessage,
  GT2SendFailed
} GT2Result;
```

Members

Members	Description
GT2Success	Success.
GT2OutOfMemory	Ran out of memory.
GT2Rejected	Attempt rejected.
GT2NetworkError	Networking error (could be local or remote).
GT2AddressError	Invalid or unreachable address.
GT2DuplicateAddress	A connection was attempted to an address that already has a connection on the socket.
GT2TimedOut	Time out reached.
GT2NegotiationError	There was an error negotiating with the remote side.

GT2InvalidConnection	The connection didn't exist.
GT2InvalidMessage	Used for vdp reliable messages containing voice data, no voice data in reliable messages.
GT2SendFailed	The send failed.

Types

Types

Name	Description
GT2Bool (see page 277)	Boolean
GT2Byte (see page 277)	A byte
GT2Connection (see page 277)	A handle to an object representing a connection to a specific IP and port the local endpoint is a GT2Socket.
GT2MessageID (see page 277)	The id of a reliably sent message; unreliable messages don't have ids.
GT2Socket (see page 277)	A handle to a socket object (can be used to accept connections and initiate connections).

GT2Bool Type

C++

```
typedef int GT2Bool;
```

GT2Byte Type

C++

```
typedef unsigned char GT2Byte;
```

GT2Connection Type

C++

```
typedef struct GTI2Connection * GT2Connection;
```

GT2MessageID Type

C++

```
typedef unsigned short GT2MessageID;
```

GT2Socket Type

C++

```
typedef struct GTI2Socket * GT2Socket;
```

Index