TES3MP Documentation

Release 0.7.0-alpha

TES3MP Team

Jan 23, 2019

Contents

Contents:

CHAPTER 1

TES3MP's Lua API reference

1.1 Actor functions

class ActorFunctions

Public Static Functions

void **ReadReceivedActorList**()
Use the last actor list received by the server as the one being read.

Return void

void **ReadCellActorList**(**const** char **cellDescription*)
Use the temporary actor list stored for a cell as the one being read.

This type of actor list is used to store actor positions and dynamic stats and is deleted when the cell is unloaded.

Return void

Parameters

- `cellDescription`: The description of the cell whose actor list should be read.

void **ClearActorList**()
Clear the data from the actor list stored on the server.

Return void

void **SetActorListPid**(unsigned short *pid*)
Set the pid attached to the ActorList.

Return void

Parameters

- `pid`: The player ID to whom the actor list should be attached.

void **CopyReceivedActorListToStore**()
Take the contents of the read-only actor list last received by the server from a player and move its contents to the stored object list that can be sent by the server.

Return void

unsigned int **GetActorListSize**()
Get the number of indexes in the read actor list.

Return The number of indexes.

unsigned char **GetActorListAction**()
Get the action type used in the read actor list.

Return The action type (0 for SET, 1 for ADD, 2 for REMOVE, 3 for REQUEST).

const char ***GetActorCell**(unsigned int *index*)
Get the cell description of the actor at a certain index in the read actor list.

Return The cell description.

Parameters

- `index`: The index of the actor.

const char ***GetActorRefId**(unsigned int *index*)
Get the refId of the actor at a certain index in the read actor list.

Return The refId.

Parameters

- `index`: The index of the actor.

unsigned int **GetActorRefNum**(unsigned int *index*)
Get the refNum of the actor at a certain index in the read actor list.

Return The refNum.

Parameters

- `index`: The index of the actor.

unsigned int **GetActorMpNum**(unsigned int *index*)
Get the mpNum of the actor at a certain index in the read actor list.

Return The mpNum.

Parameters

- `index`: The index of the actor.

double **GetActorPosX**(unsigned int *index*)
Get the X position of the actor at a certain index in the read actor list.

Return The X position.

Parameters

- `index`: The index of the actor.

double **GetActorPosY**(unsigned int *index*)
Get the Y position of the actor at a certain index in the read actor list.

Return The Y position.

Parameters

- `index`: The index of the actor.

double **GetActorPosZ**(unsigned int *index*)
Get the Z position of the actor at a certain index in the read actor list.

Return The Z position.

Parameters

- `index`: The index of the actor.

double **GetActorRotX**(unsigned int *index*)
Get the X rotation of the actor at a certain index in the read actor list.

Return The X rotation.

Parameters

- `index`: The index of the actor.

double **GetActorRotY**(unsigned int *index*)
Get the Y rotation of the actor at a certain index in the read actor list.

Return The Y rotation.

Parameters

- `index`: The index of the actor.

double **GetActorRotZ**(unsigned int *index*)
Get the Z rotation of the actor at a certain index in the read actor list.

Return The Z rotation.

Parameters

- `index`: The index of the actor.

double **GetActorHealthBase**(unsigned int *index*)
Get the base health of the actor at a certain index in the read actor list.

Return The base health.

Parameters

- `index`: The index of the actor.

double **GetActorHealthCurrent**(unsigned int *index*)
Get the current health of the actor at a certain index in the read actor list.

Return The current health.

Parameters

- `index`: The index of the actor.

double **GetActorHealthModified**(unsigned int *index*)
Get the modified health of the actor at a certain index in the read actor list.

Return The modified health.

Parameters

- `index`: The index of the actor.

double **GetActorMagickaBase**(unsigned int *index*)
Get the base magicka of the actor at a certain index in the read actor list.

Return The base magicka.

Parameters

- `index`: The index of the actor.

double **GetActorMagickaCurrent**(unsigned int *index*)
Get the current magicka of the actor at a certain index in the read actor list.

Return The current magicka.

Parameters

- `index`: The index of the actor.

double **GetActorMagickaModified**(unsigned int *index*)
Get the modified magicka of the actor at a certain index in the read actor list.

Return The modified magicka.

Parameters

- `index`: The index of the actor.

double **GetActorFatigueBase**(unsigned int *index*)
Get the base fatigue of the actor at a certain index in the read actor list.

Return The base fatigue.

Parameters

- `index`: The index of the actor.

double **GetActorFatigueCurrent**(unsigned int *index*)
Get the current fatigue of the actor at a certain index in the read actor list.

Return The current fatigue.

Parameters

- `index`: The index of the actor.

double **GetActorFatigueModified**(unsigned int *index*)
Get the modified fatigue of the actor at a certain index in the read actor list.

Return The modified fatigue.

Parameters

- `index`: The index of the actor.

const char ***GetActorEquipmentItemRefId**(unsigned int *index*, unsigned short *slot*)
Get the refId of the item in a certain slot of the equipment of the actor at a certain index in the read actor list.

Return The refId.

Parameters

- `index`: The index of the actor.
- `slot`: The slot of the equipment item.

int **GetActorEquipmentItemCount**(unsigned int *index*, unsigned short *slot*)
Get the count of the item in a certain slot of the equipment of the actor at a certain index in the read actor list.

Return The item count.

Parameters

- `index`: The index of the actor.
- `slot`: The slot of the equipment item.

int **GetActorEquipmentItemCharge**(unsigned int *index*, unsigned short *slot*)
Get the charge of the item in a certain slot of the equipment of the actor at a certain index in the read actor list.

Return The charge.

Parameters

- `index`: The index of the actor.
- `slot`: The slot of the equipment item.

double **GetActorEquipmentItemEnchantmentCharge**(unsigned int *index*, unsigned short *slot*)
Get the enchantment charge of the item in a certain slot of the equipment of the actor at a certain index in the read actor list.

Return The enchantment charge.

Parameters

- `index`: The index of the actor.
- `slot`: The slot of the equipment item.

bool **DoesActorHavePlayerKiller**(unsigned int *index*)
Check whether the killer of the actor at a certain index in the read actor list is a player.

Return Whether the actor was killed by a player.

Parameters

- `index`: The index of the actor.

int **GetActorKillerPid**(unsigned int *index*)
Get the player ID of the killer of the actor at a certain index in the read actor list.

Return The player ID of the killer.

Parameters

- `index`: The index of the actor.

const char ***GetActorKillerRefId**(unsigned int *index*)
Get the refId of the actor killer of the actor at a certain index in the read actor list.

Return The refId of the killer.

Parameters

- `index`: The index of the actor.

unsigned int **GetActorKillerRefNum**(unsigned int *index*)
Get the refNum of the actor killer of the actor at a certain index in the read actor list.

Return The refNum of the killer.

Parameters

- `index`: The index of the actor.

unsigned int **GetActorKillerMpNum**(unsigned int *index*)
Get the mpNum of the actor killer of the actor at a certain index in the read actor list.

Return The mpNum of the killer.

Parameters

- `index`: The index of the actor.

const char ***GetActorKillerName**(unsigned int *index*)
Get the name of the actor killer of the actor at a certain index in the read actor list.

Return The name of the killer.

Parameters

- `index`: The index of the actor.

bool **DoesActorHavePosition**(unsigned int *index*)
Check whether there is any positional data for the actor at a certain index in the read actor list.

This is only useful when reading the actor list data recorded for a particular cell.

Return Whether the read actor list contains positional data.

Parameters

- `index`: The index of the actor.

bool **DoesActorHaveStatsDynamic**(unsigned int *index*)
Check whether there is any dynamic stats data for the actor at a certain index in the read actor list.

This is only useful when reading the actor list data recorded for a particular cell.

Return Whether the read actor list contains dynamic stats data.

Parameters

- `index`: The index of the actor.

void **SetActorListCell**(**const** char **cellDescription*)
Set the cell of the temporary actor list stored on the server.

The cell is determined to be an exterior cell if it fits the pattern of a number followed by a comma followed by another number.

Return void

Parameters

- `cellDescription`: The description of the cell.

void **SetActorListAction**(unsigned char *action*)
Set the action type of the temporary actor list stored on the server.

Return void

Parameters

- `action`: The action type (0 for SET, 1 for ADD, 2 for REMOVE, 3 for REQUEST).

void **SetActorCell**(**const** char **cellDescription*)
Set the cell of the temporary actor stored on the server.

Used for ActorCellChange packets, where a specific actor's cell now differs from that of the actor list.

The cell is determined to be an exterior cell if it fits the pattern of a number followed by a comma followed by another number.

Return void

Parameters

- `cellDescription`: The description of the cell.

void **SetActorRefId**(**const** char **refId*)
Set the refId of the temporary actor stored on the server.

Return void

Parameters

- `refId`: The refId.

void **SetActorRefNum**(int *refNum*)
Set the refNum of the temporary actor stored on the server.

Return void

Parameters

- `refNum`: The refNum.

void **SetActorMpNum**(int *mpNum*)
Set the mpNum of the temporary actor stored on the server.

Return void

Parameters

- `mpNum`: The mpNum.

void **SetActorPosition**(double *x*, double *y*, double *z*)
Set the position of the temporary actor stored on the server.

Return void

Parameters

- `x`: The X position.
- `y`: The Y position.
- `z`: The Z position.

void **SetActorRotation**(double *x*, double *y*, double *z*)
Set the rotation of the temporary actor stored on the server.

Return void

Parameters

- `x`: The X rotation.
- `y`: The Y rotation.
- `z`: The Z rotation.

void **SetActorHealthBase**(double *value*)
Set the base health of the temporary actor stored on the server.

Return void

Parameters

- `value`: The new value.

void **SetActorHealthCurrent**(double *value*)
Set the current health of the temporary actor stored on the server.

Return void

Parameters

- `value`: The new value.

void **SetActorHealthModified**(double *value*)
Set the modified health of the temporary actor stored on the server.

Return void

Parameters

- `value`: The new value.

void **SetActorMagickaBase**(double *value*)
Set the base magicka of the temporary actor stored on the server.

Return void

Parameters

- `value`: The new value.

void **SetActorMagickaCurrent**(double *value*)
Set the current magicka of the temporary actor stored on the server.

Return void

Parameters

- `value`: The new value.

void **SetActorMagickaModified**(double *value*)
Set the modified magicka of the temporary actor stored on the server.

Return void

Parameters

- `value`: The new value.

void **SetActorFatigueBase**(double *value*)
Set the base fatigue of the temporary actor stored on the server.

Return void

Parameters

- `value`: The new value.

void **SetActorFatigueCurrent**(double *value*)
Set the current fatigue of the temporary actor stored on the server.

Return void

Parameters

- `value`: The new value.

void **SetActorFatigueModified**(double *value*)
Set the modified fatigue of the temporary actor stored on the server.

Return void

Parameters

- `value`: The new value.

void **SetActorSound**(**const** char **sound*)
Set the sound of the temporary actor stored on the server.

Return void

Parameters

- `sound`: The sound.

void **SetActorAIAction**(unsigned int *action*)
Set the AI action of the temporary actor stored on the server.

Return void

Parameters

- `action`: The new action.

void **SetActorAITargetToPlayer**(unsigned short *pid*)
Set a player as the AI target of the temporary actor stored on the server.

Return void

Parameters

- `pid`: The player ID.

void **SetActorAITargetToObject**(int *refNum*, int *mpNum*)
Set another object as the AI target of the temporary actor stored on the server.

Return void

Parameters

- `refNum`: The refNum of the target object.
- `mpNum`: The mpNum of the target object.

void **SetActorAICoordinates**(double *x*, double *y*, double *z*)
Set the coordinates for the AI package associated with the current AI action.

Return void

Parameters

- `x`: The X coordinate.
- `y`: The Y coordinate.
- `z`: The Z coordinate.

void **SetActorAIDistance**(unsigned int *distance*)
Set the distance of the AI package associated with the current AI action.

Return void

Parameters

- `duration`: The distance of the package.

void **SetActorAIDuration**(unsigned int *duration*)
Set the duration of the AI package associated with the current AI action.

Return void

Parameters

- `duration`: The duration of the package.

void **SetActorAIRepetition**(bool *shouldRepeat*)
Set whether the current AI package should be repeated.

Note: This only has an effect on the WANDER package.

Return void

Parameters

- `shouldRepeat`: Whether the package should be repeated.

void **EquipActorItem**(unsigned short *slot*, **const** char **refId*, unsigned int *count*, int *charge*, double *enchantmentCharge* = -1)
Equip an item in a certain slot of the equipment of the temporary actor stored on the server.

Return void

Parameters

- `slot`: The equipment slot.
- `refId`: The refId of the item.
- `count`: The count of the item.
- `charge`: The charge of the item.
- `enchantmentCharge`: The enchantment charge of the item.

void **UnequipActorItem**(unsigned short *slot*)
Unequip the item in a certain slot of the equipment of the temporary actor stored on the server.

Return void

Parameters

- `slot`: The equipment slot.

void **AddActor**()
Add a copy of the server's temporary actor to the server's temporary actor list.

In the process, the server's temporary actor will automatically be cleared so a new one can be set up.

Return void

void **SendActorList**()
Send an ActorList packet.

It is sent only to the player for whom the current actor list was initialized.

Return void

void **SendActorAuthority**()
Send an ActorAuthority packet.

The player for whom the current actor list was initialized is recorded in the server memory as the new actor authority for the actor list's cell.

The packet is sent to that player as well as all other players who have the cell loaded.

Return void

void **SendActorPosition**(bool *sendToOtherVisitors*, bool *skipAttachedPlayer*)
Send an ActorPosition packet.

Return void

Parameters

- `sendToOtherVisitors`: Whether this packet should be sent to cell visitors other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **SendActorStatsDynamic**(bool *sendToOtherVisitors*, bool *skipAttachedPlayer*)
Send an ActorStatsDynamic packet.

Return void

Parameters

- `sendToOtherVisitors`: Whether this packet should be sent to cell visitors other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **SendActorEquipment**(bool *sendToOtherVisitors*, bool *skipAttachedPlayer*)
Send an ActorEquipment packet.

Return void

Parameters

- `sendToOtherVisitors`: Whether this packet should be sent to cell visitors other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **SendActorSpeech** (bool *sendToOtherVisitors*, bool *skipAttachedPlayer*)
Send an ActorSpeech packet.

Return void

Parameters

- `sendToOtherVisitors`: Whether this packet should be sent to cell visitors other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **SendActorAI** (bool *sendToOtherVisitors*, bool *skipAttachedPlayer*)
Send an ActorAI packet.

Return void

Parameters

- `sendToOtherVisitors`: Whether this packet should be sent to cell visitors other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **SendActorCellChange** (bool *sendToOtherVisitors*, bool *skipAttachedPlayer*)
Send an ActorCellChange packet.

Return void

Parameters

- `sendToOtherVisitors`: Whether this packet should be sent to cell visitors other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

1.2 Book functions

class BookFunctions

Public Static Functions

void **ClearBookChanges** (unsigned short *pid*)
Clear the last recorded book changes for a player.

This is used to initialize the sending of new PlayerBook packets.

Return void

Parameters

- `pid`: The player ID whose book changes should be used.

unsigned int **GetBookChangesSize**(unsigned short *pid*)
Get the number of indexes in a player's latest book changes.

Return The number of indexes.

Parameters

- `pid`: The player ID whose book changes should be used.

void **AddBook**(unsigned short *pid*, **const** char **bookId*)
Add a new book to the book changes for a player.

Return void

Parameters

- `pid`: The player ID whose book changes should be used.
- `bookId`: The bookId of the book.

const char ***GetBookId**(unsigned short *pid*, unsigned int *index*)
Get the bookId at a certain index in a player's latest book changes.

Return The bookId.

Parameters

- `pid`: The player ID whose book changes should be used.
- `index`: The index of the book.

void **SendBookChanges**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a PlayerBook packet with a player's recorded book changes.

Return void

Parameters

- `pid`: The player ID whose book changes should be used.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

1.3 Cell functions

class CellFunctions

Public Static Functions

unsigned int **GetCellStateChangesSize**(unsigned short *pid*)
Get the number of indexes in a player's latest cell state changes.

Return The number of indexes.

Parameters

- `pid`: The player ID whose cell state changes should be used.

unsigned int **`GetCellStateType`**(unsigned short *pid*, unsigned int *index*)
Get the cell state type at a certain index in a player's latest cell state changes.

Return The cell state type (0 for LOAD, 1 for UNLOAD).

Parameters

- `pid`: The player ID whose cell state changes should be used.
- `index`: The index of the cell state.

`const` char ***`GetCellStateDescription`**(unsigned short *pid*, unsigned int *index*)
Get the cell description at a certain index in a player's latest cell state changes.

Return The cell description.

Parameters

- `pid`: The player ID whose cell state changes should be used.
- `index`: The index of the cell state.

`const` char ***`GetCell`**(unsigned short *pid*)
Get the cell description of a player's cell.

Return The cell description.

Parameters

- `pid`: The player ID.

int **`GetExteriorX`**(unsigned short *pid*)
Get the X coordinate of the player's exterior cell.

Return The X coordinate of the cell.

Parameters

- `pid`: The player ID.

int **`GetExteriorY`**(unsigned short *pid*)
Get the Y coordinate of the player's exterior cell.

Return The Y coordinate of the cell.

Parameters

- `pid`: The player ID.

bool **`IsInExterior`**(unsigned short *pid*)
Check whether the player is in an exterior cell or not.

Return Whether the player is in an exterior cell.

Parameters

- `pid`: The player ID.

const char ***GetRegion**(unsigned short *pid*)
Get the region of the player's exterior cell.

A blank value will be returned if the player is in an interior.

Return The region.

Parameters

- `pid`: The player ID.

bool **IsChangingRegion**(unsigned short *pid*)
Check whether the player's last cell change has involved a region change.

Return Whether the player has changed their region.

Parameters

- `pid`: The player ID.

void **SetCell**(unsigned short *pid*, **const** char **cellDescription*)
Set the cell of a player.

This changes the cell recorded for that player in the server memory, but does not by itself send a packet.

The cell is determined to be an exterior cell if it fits the pattern of a number followed by a comma followed by another number.

Return void

Parameters

- `pid`: The player ID.
- `cellDescription`: The cell description.

void **SetExteriorCell**(unsigned short *pid*, int *x*, int *y*)
Set the cell of a player to an exterior cell.

This changes the cell recorded for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `x`: The X coordinate of the cell.
- `y`: The Y coordinate of the cell.

void **SendCell**(unsigned short *pid*)
Send a PlayerCellChange packet about a player.

It is only sent to the affected player.

Return void

Parameters

- `pid`: The player ID.

1.4 Char class functions

class CharClassFunctions

Public Static Functions

const char ***GetDefaultClass**(unsigned short *pid*)
Get the default class used by a player.

Return The ID of the default class.

Parameters

- `pid`: The player ID.

const char ***GetClassName**(unsigned short *pid*)
Get the name of the custom class used by a player.

Return The name of the custom class.

Parameters

- `pid`: The player ID.

const char ***GetClassDesc**(unsigned short *pid*)
Get the description of the custom class used by a player.

Return The description of the custom class.

Parameters

- `pid`: The player ID.

int **GetClassMajorAttribute**(unsigned short *pid*, unsigned char *slot*)
Get the ID of one of the two major attributes of a custom class used by a player.

Return The ID of the major attribute.

Parameters

- `pid`: The player ID.
- `slot`: The slot of the major attribute (0 or 1).

int **GetClassSpecialization**(unsigned short *pid*)
Get the specialization ID of the custom class used by a player.

Return The specialization ID of the custom class (0 for Combat, 1 for Magic, 2 for Stealth).

Parameters

- `pid`: The player ID.

int **GetClassMajorSkill** (unsigned short *pid*, unsigned char *slot*)
Get the ID of one of the five major skills of a custom class used by a player.

Return The ID of the major skill.

Parameters

- `pid`: The player ID.
- `slot`: The slot of the major skill (0 to 4).

int **GetClassMinorSkill** (unsigned short *pid*, unsigned char *slot*)
Get the ID of one of the five minor skills of a custom class used by a player.

Return The ID of the minor skill.

Parameters

- `pid`: The player ID.
- `slot`: The slot of the minor skill (0 to 4).

int **IsClassDefault** (unsigned short *pid*)
Check whether the player is using a default class instead of a custom one.

Return Whether the player is using a default class.

Parameters

- `pid`: The player ID.

void **SetDefaultClass** (unsigned short *pid*, **const** char **id*)
Set the default class used by a player.

If this is left blank, the custom class data set for the player will be used instead.

Return void

Parameters

- `pid`: The player ID.
- `id`: The ID of the default class.

void **SetClassName** (unsigned short *pid*, **const** char **name*)
Set the name of the custom class used by a player.

Return void

Parameters

- `pid`: The player ID.
- `name`: The name of the custom class.

void **SetClassDesc** (unsigned short *pid*, **const** char **desc*)
Set the description of the custom class used by a player.

Return void

Parameters

- `pid`: The player ID.
- `desc`: The description of the custom class.

void **`SetClassMajorAttribute`**(unsigned short *pid*, unsigned char *slot*, int *attrId*)
Set the ID of one of the two major attributes of the custom class used by a player.

Return void

Parameters

- `pid`: The player ID.
- `slot`: The slot of the major attribute (0 or 1).
- `attrId`: The ID to use for the attribute.

void **`SetClassSpecialization`**(unsigned short *pid*, int *spec*)
Set the specialization of the custom class used by a player.

Return void

Parameters

- `pid`: The player ID.
- `spec`: The specialization ID to use (0 for Combat, 1 for Magic, 2 for Stealth).

void **`SetClassMajorSkill`**(unsigned short *pid*, unsigned char *slot*, int *skillId*)
Set the ID of one of the five major skills of the custom class used by a player.

Return void

Parameters

- `pid`: The player ID.
- `slot`: The slot of the major skill (0 to 4).
- `skillId`: The ID to use for the skill.

void **`SetClassMinorSkill`**(unsigned short *pid*, unsigned char *slot*, int *skillId*)
Set the ID of one of the five minor skills of the custom class used by a player.

Return void

Parameters

- `pid`: The player ID.
- `slot`: The slot of the minor skill (0 to 4).
- `skillId`: The ID to use for the skill.

void **`SendClass`**(unsigned short *pid*)
Send a PlayerCharClass packet about a player.

It is only sent to the affected player.

Return void

Parameters

- `pid`: The player ID.

1.5 Chat functions

class ChatFunctions

Public Static Functions

void **SendMessage**(unsigned short *pid*, **const** char **message*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a message to a certain player.

Return void

Parameters

- `pid`: The player ID.
- `message`: The contents of the message.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **CleanChatForPid**(unsigned short *pid*)
Remove all messages from chat for a certain player.

Return void

Parameters

- `pid`: The player ID.

void **CleanChat**()
Remove all messages from chat for everyone on the server.

Return void

1.6 Dialogue functions

class DialogueFunctions

Public Static Functions

void **ClearTopicChanges**(unsigned short *pid*)
Clear the last recorded topic changes for a player.

This is used to initialize the sending of new PlayerTopic packets.

Return void

Parameters

- `pid`: The player ID whose topic changes should be used.

unsigned int **`GetTopicChangesSize`**(unsigned short *pid*)
Get the number of indexes in a player's latest topic changes.

Return The number of indexes.

Parameters

- `pid`: The player ID whose topic changes should be used.

void **`AddTopic`**(unsigned short *pid*, **`const`** char **topicId*)
Add a new topic to the topic changes for a player.

Return void

Parameters

- `pid`: The player ID whose topic changes should be used.
- `topicId`: The topicId of the topic.

`const` char ***`GetTopicId`**(unsigned short *pid*, unsigned int *index*)
Get the topicId at a certain index in a player's latest topic changes.

Return The topicId.

Parameters

- `pid`: The player ID whose topic changes should be used.
- `index`: The index of the topic.

void **`SendTopicChanges`**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a PlayerTopic packet with a player's recorded topic changes.

Return void

Parameters

- `pid`: The player ID whose topic changes should be used.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **`PlayAnimation`**(unsigned short *pid*, **`const`** char **groupname*, int *mode*, int *count*, bool *persist*)
Play a certain animation on a player's character by sending a PlayerAnimation packet.

Return void

Parameters

- `pid`: The player ID of the character playing the animation.
- `groupname`: The groupname of the animation.

- `mode`: The mode of the animation.
- `count`: The number of times the animation should be played.
- `bool`: Whether the animation should persist or not.

void **PlaySpeech**(unsigned short *pid*, **const** char **sound*)
Play a certain sound for a player as spoken by their character by sending a PlayerSpeech packet.

Return void

Parameters

- `pid`: The player ID of the character playing the sound.
- `sound`: The path of the sound file.

1.7 Faction functions

class FactionFunctions

Public Static Functions

void **ClearFactionChanges**(unsigned short *pid*)
Clear the last recorded faction changes for a player.

This is used to initialize the sending of new PlayerFaction packets.

Return void

Parameters

- `pid`: The player ID whose faction changes should be used.

unsigned int **GetFactionChangesSize**(unsigned short *pid*)
Get the number of indexes in a player's latest faction changes.

Return The number of indexes.

Parameters

- `pid`: The player ID whose faction changes should be used.

unsigned char **GetFactionChangesAction**(unsigned short *pid*)
Get the action type used in a player's latest faction changes.

Return The action type (0 for RANK, 1 for EXPULSION, 2 for REPUTATION).

Parameters

- `pid`: The player ID whose faction changes should be used.

const char ***GetFactionId**(unsigned short *pid*, unsigned int *index*)
Get the factionId at a certain index in a player's latest faction changes.

Return The factionId.

Parameters

- `pid`: The player ID whose faction changes should be used.
- `index`: The index of the faction.

int **GetFactionRank**(unsigned short *pid*, unsigned int *index*)
Get the rank at a certain index in a player's latest faction changes.

Return The rank.

Parameters

- `pid`: The player ID whose faction changes should be used.
- `index`: The index of the faction.

bool **GetFactionExpulsionState**(unsigned short *pid*, unsigned int *index*)
Get the expulsion state at a certain index in a player's latest faction changes.

Return The expulsion state.

Parameters

- `pid`: The player ID whose faction changes should be used.
- `index`: The index of the faction.

int **GetFactionReputation**(unsigned short *pid*, unsigned int *index*)
Get the reputation at a certain index in a player's latest faction changes.

Return The reputation.

Parameters

- `pid`: The player ID whose faction changes should be used.
- `index`: The index of the faction.

void **SetFactionChangesAction**(unsigned short *pid*, unsigned char *action*)
Set the action type in a player's faction changes.

Return void

Parameters

- `pid`: The player ID whose faction changes should be used.
- `action`: The action (0 for RANK, 1 for EXPULSION, 2 for REPUTATION).

void **SetFactionId**(**const** char **factionId*)
Set the factionId of the temporary faction stored on the server.

Return void

Parameters

- `factionId`: The factionId.

void **SetFactionRank**(unsigned int *rank*)
Set the rank of the temporary faction stored on the server.

Return void

Parameters

- `rank`: The rank.

void **`SetFactionExpulsionState`**(bool *expulsionState*)
Set the expulsion state of the temporary faction stored on the server.

Return void

Parameters

- `expulsionState`: The expulsion state.

void **`SetFactionReputation`**(int *reputation*)
Set the reputation of the temporary faction stored on the server.

Return void

Parameters

- `reputation`: The reputation.

void **`AddFaction`**(unsigned short *pid*)
Add the server's temporary faction to the faction changes for a player.

In the process, the server's temporary faction will automatically be cleared so a new one can be set up.

Return void

Parameters

- `pid`: The player ID whose faction changes should be used.

void **`SendFactionChanges`**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a PlayerFaction packet with a player's recorded faction changes.

Return void

Parameters

- `pid`: The player ID whose faction changes should be used.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

1.8 GUI functions

`class GUIFunctions`

Public Static Functions

void **_MessageBox**(unsigned short *pid*, int *id*, **const** char **label*)
Display a simple messagebox at the bottom of the screen that vanishes after a few seconds.

Note for C++ programmers: do not rename into MessageBox so as to not conflict with WINAPI's MessageBox.

Return void

Parameters

- `pid`: The player ID for whom the messagebox should appear.
- `id`: The numerical ID of the messagebox.
- `label`: The text in the messagebox.

void **CustomMessageBox**(unsigned short *pid*, int *id*, **const** char **label*, **const** char **buttons*)
Display an interactive messagebox at the center of the screen that vanishes only when one of its buttons is clicked.

Return void

Parameters

- `pid`: The player ID for whom the messagebox should appear.
- `id`: The numerical ID of the messagebox.
- `label`: The text in the messagebox. buttons The captions of the buttons, separated by semicolons (e.g. "Yes;No;Maybe").

void **InputDialog**(unsigned short *pid*, int *id*, **const** char **label*, **const** char **note*)
Display an input dialog at the center of the screen.

Return void

Parameters

- `pid`: The player ID for whom the input dialog should appear.
- `id`: The numerical ID of the input dialog.
- `label`: The text at the top of the input dialog. note The text at the bottom of the input dialog.

void **PasswordDialog**(unsigned short *pid*, int *id*, **const** char **label*, **const** char **note*)
Display a password dialog at the center of the screen.

Although similar to an input dialog, the password dialog replaces all input characters with asterisks.

Return void

Parameters

- `pid`: The player ID for whom the password dialog should appear.
- `id`: The numerical ID of the password dialog.
- `label`: The text at the top of the password dialog. note The text at the bottom of the password dialog.

void **ListBox**(unsigned short *pid*, int *id*, **const** char **label*, **const** char **items*)
Display a listbox at the center of the screen where each item takes up a row and is selectable, with the listbox only vanishing once the Ok button is pressed.

Return void

Parameters

- `pid`: The player ID for whom the listbox should appear.
- `id`: The numerical ID of the listbox.
- `label`: The text at the top of the listbox. items The items in the listbox, separated by newlines (e.g. "Item 1\nItem 2").

void **ClearQuickKeyChanges**(unsigned short *pid*)
Clear the last recorded quick key changes for a player.

This is used to initialize the sending of new PlayerQuickKeys packets.

Return void

Parameters

- `pid`: The player ID whose quick key changes should be used.

unsigned int **GetQuickKeyChangesSize**(unsigned short *pid*)
Get the number of indexes in a player's latest quick key changes.

Return The number of indexes.

Parameters

- `pid`: The player ID whose quick key changes should be used.

void **AddQuickKey**(unsigned short *pid*, unsigned short *slot*, int *type*, **const** char **itemId* = "")
Add a new quick key to the quick key changes for a player.

Return void

Parameters

- `pid`: The player ID whose quick key changes should be used.
- `slot`: The slot to be used.
- `slot`: The type of the quick key (0 for ITEM, 1 for ITEM_MAGIC, 2 for MAGIC, 3 for UNASSIGNED).
- `itemId`: The itemId of the item.

int **GetQuickKeySlot**(unsigned short *pid*, unsigned int *index*)
Get the slot of the quick key at a certain index in a player's latest quick key changes.

Return The slot.

Parameters

- `pid`: The player ID whose quick key changes should be used.
- `index`: The index of the quick key in the quick key changes vector.

int **GetQuickKeyType**(unsigned short *pid*, unsigned int *index*)
Get the type of the quick key at a certain index in a player's latest quick key changes.

Return The quick key type.

Parameters

- `pid`: The player ID whose quick key changes should be used.
- `index`: The index of the quick key in the quick key changes vector.

const char ***GetQuickKeyItemId**(unsigned short *pid*, unsigned int *index*)
Get the itemId at a certain index in a player's latest quick key changes.

Return The itemId.

Parameters

- `pid`: The player ID whose quick key changes should be used.
- `index`: The index of the quick key in the quick key changes vector.

void **SendQuickKeyChanges**(unsigned short *pid*)
Send a PlayerQuickKeys packet with a player's recorded quick key changes.

Return void

Parameters

- `pid`: The player ID whose quick key changes should be used.

void **SetMapVisibility**(unsigned short *targetPid*, unsigned short *affectedPid*, unsigned short *state*)
Determine whether a player can see the map marker of another player.

Note: This currently has no effect, and is just an unimplemented stub.

Return void

Parameters

- `targetPid`: The player ID whose map marker should be hidden or revealed.
- `affectedPid`: The player ID for whom the map marker will be hidden or revealed.
- `state`: The state of the map marker (false to hide, true to reveal).

void **SetMapVisibilityAll**(unsigned short *targetPid*, unsigned short *state*)
Determine whether a player's map marker can be seen by all other players.

Note: This currently has no effect, and is just an unimplemented stub.

Return void

Parameters

- `targetPid`: The player ID whose map marker should be hidden or revealed.
- `state`: The state of the map marker (false to hide, true to reveal).

1.9 Item functions

class ItemFunctions

Public Static Functions

void **ClearInventoryChanges**(unsigned short *pid*)
Clear the last recorded inventory changes for a player.

This is used to initialize the sending of new PlayerInventory packets.

Return void

Parameters

- `pid`: The player ID whose inventory changes should be used.

int **GetEquipmentSize**()
Get the number of slots used for equipment.

The number is 19 before any dehardcoding is done in OpenMW.

Return The number of slots.

unsigned int **GetInventoryChangesSize**(unsigned short *pid*)
Get the number of indexes in a player's latest inventory changes.

Return The number of indexes.

Parameters

- `pid`: The player ID whose inventory changes should be used.

unsigned int **GetInventoryChangesAction**(unsigned short *pid*)
Get the action type used in a player's latest inventory changes.

Return The action type (0 for SET, 1 for ADD, 2 for REMOVE).

Parameters

- `pid`: The player ID whose inventory changes should be used.

void **SetInventoryChangesAction**(unsigned short *pid*, unsigned char *action*)
Set the action type in a player's inventory changes.

Return void

Parameters

- `pid`: The player ID whose inventory changes should be used.
- `action`: The action (0 for SET, 1 for ADD, 2 for REMOVE).

void **EquipItem**(unsigned short *pid*, unsigned short *slot*, **const** char **refId*, unsigned int *count*, int *charge*, double *enchantmentCharge* = -1)
Equip an item in a certain slot of the equipment of a player.

Return void

Parameters

- `pid`: The player ID.
- `slot`: The equipment slot.
- `refId`: The refId of the item.
- `count`: The count of the item.
- `charge`: The charge of the item.
- `enchantmentCharge`: The enchantment charge of the item.

void **UnequipItem**(unsigned short *pid*, unsigned short *slot*)
Unequip the item in a certain slot of the equipment of a player.

Return void

Parameters

- `pid`: The player ID.
- `slot`: The equipment slot.

void **AddItemChange**(unsigned short *pid*, **const** char **refId*, unsigned int *count*, int *charge*, double *enchantmentCharge*, **const** char **soul*)
Add an item change to a player's inventory changes.

Return void

Parameters

- `pid`: The player ID.
- `refId`: The refId of the item.
- `count`: The count of the item.
- `charge`: The charge of the item.
- `enchantmentCharge`: The enchantment charge of the item.
- `soul`: The soul of the item.

bool **HasItemEquipped**(unsigned short *pid*, **const** char **refId*)
Check whether a player has equipped an item with a certain refId in any slot.

Return Whether the player has the item equipped.

Parameters

- `pid`: The player ID.
- `refId`: The refId of the item.

const char ***GetEquipmentItemRefId**(unsigned short *pid*, unsigned short *slot*)
Get the refId of the item in a certain slot of the equipment of a player.

Return The refId.

Parameters

- `pid`: The player ID.
- `slot`: The slot of the equipment item.

int **GetEquipmentItemCount**(unsigned short *pid*, unsigned short *slot*)
Get the count of the item in a certain slot of the equipment of a player.

Return The item count.

Parameters

- `pid`: The player ID.
- `slot`: The slot of the equipment item.

int **GetEquipmentItemCharge**(unsigned short *pid*, unsigned short *slot*)
Get the charge of the item in a certain slot of the equipment of a player.

Return The charge.

Parameters

- `pid`: The player ID.
- `slot`: The slot of the equipment item.

double **GetEquipmentItemEnchantmentCharge**(unsigned short *pid*, unsigned short *slot*)
Get the enchantment charge of the item in a certain slot of the equipment of a player.

Return The enchantment charge.

Parameters

- `pid`: The player ID.
- `slot`: The slot of the equipment item.

const char ***GetInventoryItemRefId**(unsigned short *pid*, unsigned int *index*)
Get the refId of the item at a certain index in a player's latest inventory changes.

Return The refId.

Parameters

- `pid`: The player ID whose inventory changes should be used.
- `index`: The index of the inventory item.

int **GetInventoryItemCount**(unsigned short *pid*, unsigned int *index*)
Get the count of the item at a certain index in a player's latest inventory changes.

Return The item count.

Parameters

- `pid`: The player ID whose inventory changes should be used.
- `index`: The index of the inventory item.

int **GetInventoryItemCharge**(unsigned short *pid*, unsigned int *index*)
Get the charge of the item at a certain index in a player's latest inventory changes.

Return The charge.

Parameters

- `pid`: The player ID whose inventory changes should be used.
- `index`: The index of the inventory item.

double **GetInventoryItemEnchantmentCharge**(unsigned short *pid*, unsigned int *index*)
Get the enchantment charge of the item at a certain index in a player's latest inventory changes.

Return The enchantment charge.

Parameters

- `pid`: The player ID whose inventory changes should be used.
- `index`: The index of the inventory item.

const char ***GetInventoryItemSoul**(unsigned short *pid*, unsigned int *index*)
Get the soul of the item at a certain index in a player's latest inventory changes.

Return The soul.

Parameters

- `pid`: The player ID whose inventory changes should be used.
- `index`: The index of the inventory item.

const char ***GetUsedItemRefId**(unsigned short *pid*)
Get the refId of the item last used by a player.

Return The refId.

Parameters

- `pid`: The player ID.

int **GetUsedItemCount**(unsigned short *pid*)
Get the count of the item last used by a player.

Return The item count.

Parameters

- `pid`: The player ID.

int **GetUsedItemCharge**(unsigned short *pid*)
Get the charge of the item last used by a player.

Return The charge.

Parameters

- `pid`: The player ID.

double **GetUsedItemEnchantmentCharge**(unsigned short *pid*)
Get the enchantment charge of the item last used by a player.

Return The enchantment charge.

Parameters

- `pid`: The player ID.

const char ***GetUsedItemSoul**(unsigned short *pid*)
Get the soul of the item last used by a player.

Return The soul.

Parameters

- `pid`: The player ID.

void **SendEquipment**(unsigned short *pid*)
Send a PlayerEquipment packet with a player's equipment.

It is always sent to all players.

Return void

Parameters

- `pid`: The player ID whose equipment should be sent.

void **SendInventoryChanges**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a PlayerInventory packet with a player's recorded inventory changes.

Return void

Parameters

- `pid`: The player ID whose inventory changes should be used.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **SendItemUse**(unsigned short *pid*)
Send a PlayerItemUse causing a player to use their recorded usedItem.

Return void

Parameters

- `pid`: The player ID affected.

1.10 Mechanics functions

class MechanicsFunctions

Public Static Functions

unsigned char **GetMiscellaneousChangeType**(unsigned short *pid*)
Get the type of a PlayerMiscellaneous packet.

Return The type.

Parameters

- `pid`: The player ID.

const char ***GetMarkCell**(unsigned short *pid*)
Get the cell description of a player's Mark cell.

Return The cell description.

Parameters

- `pid`: The player ID.

double **GetMarkPosX**(unsigned short *pid*)
Get the X position of a player's Mark.

Return The X position.

Parameters

- `pid`: The player ID.

double **GetMarkPosY**(unsigned short *pid*)
Get the Y position of a player's Mark.

Return The Y position.

Parameters

- `pid`: The player ID.

double **GetMarkPosZ**(unsigned short *pid*)
Get the Z position of a player's Mark.

Return The Z position.

Parameters

- `pid`: The player ID.

double **GetMarkRotX**(unsigned short *pid*)
Get the X rotation of a player's Mark.

Return The X rotation.

Parameters

- `pid`: The player ID.

double **GetMarkRotZ**(unsigned short *pid*)
Get the Z rotation of a player's Mark.

Return The X rotation.

Parameters

- `pid`: The player ID.

`const` char *`GetSelectedSpellId`(unsigned short *pid*)
Get the ID of a player's selected spell.

Return The spell ID.

Parameters

- `pid`: The player ID.

bool `DoesPlayerHavePlayerKiller`(unsigned short *pid*)
Check whether the killer of a certain player is also a player.

Return Whether the player was killed by another player.

Parameters

- `pid`: The player ID of the killed player.

int `GetPlayerKillerPid`(unsigned short *pid*)
Get the player ID of the killer of a certain player.

Return The player ID of the killer.

Parameters

- `pid`: The player ID of the killed player.

`const` char *`GetPlayerKillerRefId`(unsigned short *pid*)
Get the refId of the actor killer of a certain player.

Return The refId of the killer.

Parameters

- `pid`: The player ID of the killed player.

unsigned int `GetPlayerKillerRefNum`(unsigned short *pid*)
Get the refNum of the actor killer of a certain player.

Return The refNum of the killer.

Parameters

- `pid`: The player ID of the killed player.

unsigned int `GetPlayerKillerMpNum`(unsigned short *pid*)
Get the mpNum of the actor killer of a certain player.

Return The mpNum of the killer.

Parameters

- `pid`: The player ID of the killed player.

const char ***GetPlayerKillerName**(unsigned short *pid*)
Get the name of the actor killer of a certain player.

Return The name of the killer.

Parameters

- `pid`: The player ID of the killed player.

unsigned int **GetDrawState**(unsigned short *pid*)
Get the draw state of a player (0 for nothing, 1 for drawn weapon, 2 for drawn spell).

Return The draw state.

Parameters

- `pid`: The player ID.

bool **GetSneakState**(unsigned short *pid*)
Get the sneak state of a player.

Return Whether the player is sneaking.

Parameters

- `pid`: The player ID.

void **SetMarkCell**(unsigned short *pid*, **const** char **cellDescription*)
Set the Mark cell of a player.

This changes the Mark cell recorded for that player in the server memory, but does not by itself send a packet.

The cell is determined to be an exterior cell if it fits the pattern of a number followed by a comma followed by another number.

Return void

Parameters

- `pid`: The player ID.
- `cellDescription`: The cell description.

void **SetMarkPos**(unsigned short *pid*, double *x*, double *y*, double *z*)
Set the Mark position of a player.

This changes the Mark positional coordinates recorded for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `x`: The X position.
- `y`: The Y position.
- `z`: The Z position.

void **SetMarkRot**(unsigned short *pid*, double *x*, double *z*)
Set the Mark rotation of a player.

This changes the Mark positional coordinates recorded for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `x`: The X rotation.
- `z`: The Z rotation.

void **SetSelectedSpellId**(unsigned short *pid*, **const** char **spellId*)
Set the ID of a player's selected spell.

This changes the spell ID recorded for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `spellId`: The spell ID.

void **SendMarkLocation**(unsigned short *pid*)
Send a PlayerMiscellaneous packet with a Mark location to a player.

Return void

Parameters

- `pid`: The player ID.

void **SendSelectedSpell**(unsigned short *pid*)
Send a PlayerMiscellaneous packet with a selected spell ID to a player.

Return void

Parameters

- `pid`: The player ID.

void **Jail**(unsigned short *pid*, int *jailDays*, bool *ignoreJailTeleportation*, bool *ignoreJailSkillIncreases*, **const** char **jailProgressText*, **const** char **jailEndText*)
Send a PlayerJail packet about a player.

This is similar to the player being jailed by a guard, but provides extra parameters for increased flexibility.

It is only sent to the player being jailed, as the other players will be informed of the jailing's actual consequences via other packets sent by the affected client.

Return void

Parameters

- `pid`: The player ID.

- `jailDays`: The number of days to spend jailed, where each day affects one skill point.
- `ignoreJailTeleportation`: Whether the player being teleported to the nearest jail marker should be overridden.
- `ignoreJailSkillIncrease`: Whether the player's Sneak and Security skills should be prevented from increasing as a result of the jailing, overriding default behavior.
- `jailProgressText`: The text that should be displayed while jailed.
- `jailEndText`: The text that should be displayed once the jailing period is over.

void **`Resurrect`**(unsigned short *pid*, unsigned int *type*)
Send a PlayerResurrect packet about a player.

This sends the packet to all players connected to the server.

Return void

Parameters

- `pid`: The player ID.
- `type`: The type of resurrection (0 for REGULAR, 1 for IMPERIAL_SHRINE, 2 for TRIBUNAL_TEMPLE).

1.11 Miscellaneous functions

`class MiscellaneousFunctions`

Public Static Functions

bool **`DoesFileExist`**(**`const`** char ***filePath*)
Check whether a certain file exists.

This will be a case sensitive check on case sensitive filesystems.

Whenever you want to enforce case insensitivity, use *GetCaseInsensitiveFilename()* instead.

Return Whether the file exists or not.

`const` char ***`GetCaseInsensitiveFilename`**(**`const`** char **folderPath*, **`const`** char **filename*)
Get the first filename in a folder that has a case insensitive match with the filename argument.

This is used to retain case insensitivity when opening data files on Linux.

Return The filename that matches.

unsigned int **`GetLastPlayerId`**()
Get the last player ID currently connected to the server.

Every player receives a unique numerical index known as their player ID upon joining the server.

Return The player ID.

int **GetCurrentMpNum**()
Get the current (latest) mpNum generated by the server.

Every object that did not exist in an .ESM or .ESP data file and has instead been placed or spawned through a server-sent packet has a numerical index known as its mpNum.

When ObjectPlace and ObjectSpawn packets are received from players, their objects lack mpNums, so the server assigns them some based on incrementing the server's current mpNum, with the operation's final mpNum becoming the server's new current mpNum.

Return The mpNum.

void **SetCurrentMpNum**(int *mpNum*)
Set the current (latest) mpNum generated by the server.

When restarting a server, it is important to revert to the previous current (latest) mpNum as stored in the server's data, so as to avoid starting over from 0 and ending up assigning duplicate mpNums to objects.

Return void

Parameters

- `mpNum`: The number that should be used as the new current mpNum.

void **LogMessage**(unsigned short *level*, **const** char **message*)
Write a log message with its own timestamp.

It will have "[Script]:" prepended to it so as to mark it as a script-generated log message.

Return void

Parameters

- `level`: The logging level used (0 for LOG_VERBOSE, 1 for LOG_INFO, 2 for LOG_WARN, 3 for LOG_ERROR, 4 for LOG_FATAL).
- `message`: The message logged.

void **LogAppend**(unsigned short *level*, **const** char **message*)
Write a log message without its own timestamp.

It will have "[Script]:" prepended to it so as to mark it as a script-generated log message.

Return void

Parameters

- `level`: The logging level used (0 for LOG_VERBOSE, 1 for LOG_INFO, 2 for LOG_WARN, 3 for LOG_ERROR, 4 for LOG_FATAL).
- `message`: The message logged.

1.12 Object functions

class ObjectFunctions

Public Static Functions

void **ReadReceivedObjectList**()
Use the last object list received by the server as the one being read.

Return void

void **ClearObjectList**()
Clear the data from the object list stored on the server.

Return void

void **SetObjectListPid**(unsigned short *pid*)
Set the pid attached to the ObjectList.

Return void

Parameters

- `pid`: The player ID to whom the object list should be attached.

void **CopyReceivedObjectListToStore**()
Take the contents of the read-only object list last received by the server from a player and move its contents to the stored object list that can be sent by the server.

Return void

unsigned int **GetObjectListSize**()
Get the number of indexes in the read object list.

Return The number of indexes.

unsigned char **GetObjectListOrigin**()
Get the origin of the read object list.

Return The origin (0 for CLIENT_GAMEPLAY, 1 for CLIENT_CONSOLE, 2 for CLIENT_DIALOGUE, 3 for CLIENT_SCRIPT_LOCAL, 4 for CLIENT_SCRIPT_GLOBAL, 5 for SERVER_SCRIPT).

const char ***GetObjectListClientScript**()
Get the client script that the read object list originated from.

Note: This is not yet implemented.

Return The ID of the client script.

unsigned char **GetObjectListAction**()
Get the action type used in the read object list.

Return The action type (0 for SET, 1 for ADD, 2 for REMOVE, 3 for REQUEST).

unsigned char **GetObjectListContainerSubAction**()
Get the container subaction type used in the read object list.

Return The action type (0 for NONE, 1 for DRAG, 2 for DROP, 3 for TAKE_ALL).

bool **IsObjectPlayer**(unsigned int *index*)
Check whether the object at a certain index in the read object list is a player.

Note: Although most player data and events are dealt with in Player packets, object activation is general enough for players themselves to be included as objects in ObjectActivate packets.

Return Whether the object is a player.

Parameters

- `index`: The index of the object.

int **GetObjectPid**(unsigned int *index*)
Get the player ID of the object at a certain index in the read object list, only valid if the object is a player.

Note: Currently, players can only be objects in ObjectActivate and ConsoleCommand packets.

Return The player ID of the object.

Parameters

- `index`: The index of the object.

const char ***GetObjectRefId**(unsigned int *index*)
Get the refId of the object at a certain index in the read object list.

Return The refId.

Parameters

- `index`: The index of the object.

unsigned int **GetObjectRefNum**(unsigned int *index*)
Get the refNum of the object at a certain index in the read object list.

Return The refNum.

Parameters

- `index`: The index of the object.

unsigned int **GetObjectMpNum**(unsigned int *index*)
Get the mpNum of the object at a certain index in the read object list.

Return The mpNum.

Parameters

- `index`: The index of the object.

int **GetObjectCount**(unsigned int *index*)
Get the count of the object at a certain index in the read object list.

Return The object count.

Parameters

- `index`: The index of the object.

int **GetObjectCharge**(unsigned int *index*)
Get the charge of the object at a certain index in the read object list.

Return The charge.

Parameters

- `index`: The index of the object.

double **GetObjectEnchantmentCharge**(unsigned int *index*)
Get the enchantment charge of the object at a certain index in the read object list.

Return The enchantment charge.

Parameters

- `index`: The index of the object.

const char ***GetObjectSoul**(unsigned int *index*)
Get the soul of the object at a certain index in the read object list.

Return The soul.

Parameters

- `index`: The index of the object.

int **GetObjectGoldValue**(unsigned int *index*)
Get the gold value of the object at a certain index in the read object list.

This is used solely to get the gold value of gold. It is not used for other objects.

Return The gold value.

Parameters

- `index`: The index of the object.

double **GetObjectScale**(unsigned int *index*)
Get the object scale of the object at a certain index in the read object list.

Return The object scale.

Parameters

- `index`: The index of the object.

bool **GetObjectState**(unsigned int *index*)
Get the object state of the object at a certain index in the read object list.

Return The object state.

Parameters

- `index`: The index of the object.

int **GetObjectDoorState**(unsigned int *index*)
Get the door state of the object at a certain index in the read object list.

Return The door state.

Parameters

- `index`: The index of the object.

int **GetObjectLockLevel**(unsigned int *index*)
Get the lock level of the object at a certain index in the read object list.

Return The lock level.

Parameters

- `index`: The index of the object.

bool **DoesObjectHavePlayerActivating**(unsigned int *index*)
Check whether the object at a certain index in the read object list has been activated by a player.

Return Whether the object has been activated by a player.

Parameters

- `index`: The index of the object.

int **GetObjectActivatingPid**(unsigned int *index*)
Get the player ID of the player activating the object at a certain index in the read object list.

Return The player ID of the activating player.

Parameters

- `index`: The index of the object.

const char ***GetObjectActivatingRefId**(unsigned int *index*)
Get the refId of the actor activating the object at a certain index in the read object list.

Return The refId of the activating actor.

Parameters

- `index`: The index of the object.

unsigned int **GetObjectActivatingRefNum**(unsigned int *index*)
Get the refNum of the actor activating the object at a certain index in the read object list.

Return The refNum of the activating actor.

Parameters

- `index`: The index of the object.

unsigned int **GetObjectActivatingMpNum**(unsigned int *index*)
Get the mpNum of the actor activating the object at a certain index in the read object list.

Return The mpNum of the activating actor.

Parameters

- `index`: The index of the object.

const char ***GetObjectActivatingName**(unsigned int *index*)
Get the name of the actor activating the object at a certain index in the read object list.

Return The name of the activating actor.

Parameters

- `index`: The index of the object.

bool **GetObjectSummonState**(unsigned int *index*)
Check whether the object at a certain index in the read object list is a summon.

Only living actors can be summoned.

Return The summon state.

double **GetObjectSummonDuration**(unsigned int *index*)
Get the summon duration of the object at a certain index in the read object list.

Note: Returns -1 if indefinite.

Return The summon duration.

Parameters

- `index`: The index of the object.

bool **DoesObjectHavePlayerSummoner**(unsigned int *index*)
Check whether the object at a certain index in the read object list has a player as its summoner.

Only living actors can be summoned.

Return Whether a player is the summoner of the object.

Parameters

- `index`: The index of the object.

int **GetObjectSummonerPid**(unsigned int *index*)
Get the player ID of the summoner of the object at a certain index in the read object list.

Return The player ID of the summoner.

Parameters

- `index`: The index of the object.

const char ***GetObjectSummonerRefId**(unsigned int *index*)
Get the refId of the actor summoner of the object at a certain index in the read object list.

Return The refId of the summoner.

Parameters

- `index`: The index of the object.

unsigned int **GetObjectSummonerRefNum**(unsigned int *index*)
Get the refNum of the actor summoner of the object at a certain index in the read object list.

Return The refNum of the summoner.

Parameters

- `index`: The index of the object.

unsigned int **GetObjectSummonerMpNum**(unsigned int *index*)
Get the mpNum of the actor summoner of the object at a certain index in the read object list.

Return The mpNum of the summoner.

Parameters

- `index`: The index of the object.

double **GetObjectPosX**(unsigned int *index*)
Get the X position of the object at a certain index in the read object list.

Return The X position.

Parameters

- `index`: The index of the object.

double **GetObjectPosY**(unsigned int *index*)
Get the Y position of the object at a certain index in the read object list.

Return The Y position.

Parameters

- `index`: The index of the object.

double **GetObjectPosZ**(unsigned int *index*)
Get the Z position at a certain index in the read object list.

Return The Z position.

Parameters

- `index`: The index of the object.

double **GetObjectRotX**(unsigned int *index*)
Get the X rotation of the object at a certain index in the read object list.

Return The X rotation.

Parameters

- `index`: The index of the object.

double **GetObjectRotY**(unsigned int *index*)
Get the Y rotation of the object at a certain index in the read object list.

Return The Y rotation.

Parameters

- `index`: The index of the object.

double **GetObjectRotZ**(unsigned int *index*)
Get the Z rotation of the object at a certain index in the read object list.

Return The Z rotation.

Parameters

- `index`: The index of the object.

const char ***GetVideoFilename**(unsigned int *index*)
Get the videoFilename of the object at a certain index in the read object list.

Return The videoFilename.

unsigned int **GetContainerChangesSize**(unsigned int *objectIndex*)
Get the number of container item indexes of the object at a certain index in the read object list.

Return The number of container item indexes.

Parameters

- `index`: The index of the object.

const char ***GetContainerItemRefId**(unsigned int *objectIndex*, unsigned int *itemIndex*)
Get the refId of the container item at a certain itemIndex in the container changes of the object at a certain objectIndex in the read object list.

Return The refId.

Parameters

- `objectIndex`: The index of the object.
- `itemIndex`: The index of the container item.

int **GetContainerItemCount**(unsigned int *objectIndex*, unsigned int *itemIndex*)
Get the item count of the container item at a certain itemIndex in the container changes of the object at a certain objectIndex in the read object list.

Return The item count.

Parameters

- `objectIndex`: The index of the object.
- `itemIndex`: The index of the container item.

int **GetContainerItemCharge**(unsigned int *objectIndex*, unsigned int *itemIndex*)
Get the charge of the container item at a certain itemIndex in the container changes of the object at a certain objectIndex in the read object list.

Return The charge.

Parameters

- `objectIndex`: The index of the object.
- `itemIndex`: The index of the container item.

double **GetContainerItemEnchantmentCharge**(unsigned int *objectIndex*, unsigned int *itemIndex*)

Get the enchantment charge of the container item at a certain itemIndex in the container changes of the object at a certain objectIndex in the read object list.

Return The enchantment charge.

Parameters

- `objectIndex`: The index of the object.
- `itemIndex`: The index of the container item.

const char ***GetContainerItemSoul**(unsigned int *objectIndex*, unsigned int *itemIndex*)

Get the soul of the container item at a certain itemIndex in the container changes of the object at a certain objectIndex in the read object list.

Return The soul.

Parameters

- `objectIndex`: The index of the object.
- `itemIndex`: The index of the container item.

int **GetContainerItemActionCount**(unsigned int *objectIndex*, unsigned int *itemIndex*)

Get the action count of the container item at a certain itemIndex in the container changes of the object at a certain objectIndex in the read object list.

Return The action count.

Parameters

- `objectIndex`: The index of the object.
- `itemIndex`: The index of the container item.

bool **DoesObjectHaveContainer**(unsigned int *index*)

Check whether the object at a certain index in the read object list has a container.

Note: Only ObjectLists from ObjectPlace packets contain this information. Objects from received ObjectSpawn packets can always be assumed to have a container.

Return Whether the object has a container.

Parameters

- `index`: The index of the object.

void **SetObjectListCell**(**const** char **cellDescription*)

Set the cell of the temporary object list stored on the server.

The cell is determined to be an exterior cell if it fits the pattern of a number followed by a comma followed by another number.

Return void

Parameters

- `cellDescription`: The description of the cell.

void **SetObjectListAction**(unsigned char *action*)
Set the action type of the temporary object list stored on the server.

Return void

Parameters

- `action`: The action type (0 for SET, 1 for ADD, 2 for REMOVE, 3 for REQUEST).

void **SetObjectListConsoleCommand**(**const** char **consoleCommand*)
Set the console command of the temporary object list stored on the server.

When sent, the command will run once on every object added to the object list. If no objects have been added, it will run once without any object reference.

Return void

Parameters

- `consoleCommand`: The console command.

void **SetObjectRefId**(**const** char **refId*)
Set the refId of the temporary object stored on the server.

Return void

Parameters

- `refId`: The refId.

void **SetObjectRefNum**(int *refNum*)
Set the refNum of the temporary object stored on the server.

Every object loaded from .ESM and .ESP data files has a unique refNum which needs to be retained to refer to it in packets.

On the other hand, objects placed or spawned via the server should always have a refNum of 0.

Return void

Parameters

- `refNum`: The refNum.

void **SetObjectMpNum**(int *mpNum*)
Set the mpNum of the temporary object stored on the server.

Every object placed or spawned via the server is assigned an mpNum by incrementing the last mpNum stored on the server. Scripts should take care to ensure that mpNums are kept unique for these objects.

Objects loaded from .ESM and .ESP data files should always have an mpNum of 0, because they have unique refNumes instead.

Return void

Parameters

- `mpNum`: The mpNum.

void **`SetObjectCount`**(int *count*)
Set the object count of the temporary object stored on the server.

This determines the quantity of an object, with the exception of gold.

Return void

Parameters

- `count`: The object count.

void **`SetObjectCharge`**(int *charge*)
Set the charge of the temporary object stored on the server.

Object durabilities are set through this value.

Return void

Parameters

- `charge`: The charge.

void **`SetObjectEnchantmentCharge`**(double *enchantmentCharge*)
Set the enchantment charge of the temporary object stored on the server.

Object durabilities are set through this value.

Return void

Parameters

- `charge`: The enchantment charge.

void **`SetObjectSoul`**(**`const`** char **soul*)
Set the soul of the temporary object stored on the server.

Return void

Parameters

- `refId`: The soul.

void **`SetObjectGoldValue`**(int *goldValue*)
Set the gold value of the temporary object stored on the server.

This is used solely to set the gold value for gold. It has no effect on other objects.

Return void

Parameters

- `goldValue`: The gold value.

void **`SetObjectScale`**(double *scale*)
Set the scale of the temporary object stored on the server.

Objects are smaller or larger than their default size based on their scale.

Return void

Parameters

- `scale`: The scale.

void **SetObjectState**(bool *objectState*)
Set the object state of the temporary object stored on the server.

Objects are enabled or disabled based on their object state.

Return void

Parameters

- `objectState`: The object state.

void **SetObjectLockLevel**(int *lockLevel*)
Set the lock level of the temporary object stored on the server.

Return void

Parameters

- `lockLevel`: The lock level.

void **SetObjectSummonDuration**(float *summonDuration*)
Set the summon duration of the temporary object stored on the server.

Return void

Parameters

- `summonDuration`: The summon duration.

void **SetObjectDisarmState**(bool *disarmState*)
Set the disarm state of the temporary object stored on the server.

Return void

Parameters

- `disarmState`: The disarmState.

void **SetObjectSummonState**(bool *summonState*)
Set the summon state of the temporary object stored on the server.

This only affects living actors and determines whether they are summons of another living actor.

Return void

Parameters

- `summonState`: The summon state.

void **SetObjectPosition**(double *x*, double *y*, double *z*)
Set the position of the temporary object stored on the server.

Return void

Parameters

- `x`: The X position.
- `y`: The Y position.
- `z`: The Z position.

void **`SetObjectRotation`**(double *x*, double *y*, double *z*)
Set the rotation of the temporary object stored on the server.

Return void

Parameters

- `x`: The X rotation.
- `y`: The Y rotation.
- `z`: The Z rotation.

void **`SetObjectActivatingPid`**(unsigned short *pid*)
Set the player ID of the player activating the temporary object stored on the server. Currently only used for ObjectActivate packets.

Return void

Parameters

- `pid`: The pid of the player.

void **`SetObjectDoorState`**(int *doorState*)
Set the door state of the temporary object stored on the server.

Doors are open or closed based on their door state.

Return void

Parameters

- `doorState`: The door state.

void **`SetObjectDoorTeleportState`**(bool *teleportState*)
Set the teleport state of the temporary object stored on the server.

If a door's teleport state is true, interacting with the door teleports a player to its destination. If it's false, it opens and closes like a regular door.

Return void

Parameters

- `teleportState`: The teleport state.

void **`SetObjectDoorDestinationCell`**(**`const`** char **cellDescription*)
Set the door destination cell of the temporary object stored on the server.

The cell is determined to be an exterior cell if it fits the pattern of a number followed by a comma followed by another number.

Return void

Parameters

- `cellDescription`: The description of the cell.

void **SetObjectDoorDestinationPosition**(double *x*, double *y*, double *z*)
Set the door destination position of the temporary object stored on the server.

Return void

Parameters

- `x`: The X position.
- `y`: The Y position.
- `z`: The Z position.

void **SetObjectDoorDestinationRotation**(double *x*, double *z*)
Set the door destination rotation of the temporary object stored on the server.

Note: Because this sets the rotation a player will have upon using the door, and rotation on the Y axis has no effect on players, the Y value has been omitted as an argument.

Return void

Parameters

- `x`: The X rotation.
- `z`: The Z rotation.

void **SetPlayerAsObject**(unsigned short *pid*)
Set a player as the object in the temporary object stored on the server. Currently only used for ConsoleCommand packets.

Return void

Parameters

- `pid`: The pid of the player.

void **SetContainerItemRefId**(**const** char **refId*)
Set the refId of the temporary container item stored on the server.

Return void

Parameters

- `refId`: The refId.

void **SetContainerItemCount**(int *count*)
Set the item count of the temporary container item stored on the server.

Return void

Parameters

- `count`: The item count.

void **SetContainerItemCharge**(int *charge*)
Set the charge of the temporary container item stored on the server.

Return void

Parameters

- `charge`: The charge.

void **SetContainerItemEnchantmentCharge**(double *enchantmentCharge*)
Set the enchantment charge of the temporary container item stored on the server.

Return void

Parameters

- `charge`: The enchantment charge.

void **SetContainerItemSoul**(**const** char **soul*)
Set the soul of the temporary container item stored on the server.

Return void

Parameters

- `refId`: The soul.

void **SetContainerItemActionCountByIndex**(unsigned int *objectIndex*, unsigned int *itemIndex*, int *actionCount*)
Set the action count of the container item at a certain itemIndex in the container changes of the object at a certain objectIndex in the object list stored on the server.

When resending a received Container packet, this allows you to correct the amount of items removed from a container by a player when it conflicts with what other players have already taken.

Return void

Parameters

- `objectIndex`: The index of the object.
- `itemIndex`: The index of the container item.
- `actionCount`: The action count.

void **AddObject**()
Add a copy of the server's temporary object to the server's currently stored object list.

In the process, the server's temporary object will automatically be cleared so a new one can be set up.

Return void

void **AddContainerItem**()
Add a copy of the server's temporary container item to the container changes of the server's temporary object.

In the process, the server's temporary container item will automatically be cleared so a new one can be set up.

Return void

void **SendObjectActivate**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send an ObjectActivate packet.

Return void

Parameters

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **SendObjectPlace**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send an ObjectPlace packet.

Return void

Parameters

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **SendObjectSpawn**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send an ObjectSpawn packet.

Return void

Parameters

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **SendObjectDelete**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send an ObjectDelete packet.

Return void

Parameters

- `broadcast`: Whether this packet should be sent only to the player for whom the current object list was initialized or to everyone on the server.

void **SendObjectLock**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send an ObjectLock packet.

Return void

Parameters

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **`SendObjectTrap`**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send an ObjectTrap packet.

Return void

Parameters

- `broadcast`: Whether this packet should be sent only to the player for whom the current object list was initialized or to everyone on the server.

void **`SendObjectScale`**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send an ObjectScale packet.

Return void

Parameters

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **`SendObjectState`**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send an ObjectState packet.

Return void

Parameters

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **`SendDoorState`**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a DoorState packet.

Return void

Parameters

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **`SendDoorDestination`**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a DoorDestination packet.

Return void

Parameters

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **`SendContainer`**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a Container packet.

Return void

Parameters

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **`SendVideoPlay`**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a VideoPlay packet.

Return void

Parameters

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **`SendConsoleCommand`**(bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a ConsoleCommand packet.

Return void

Parameters

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

1.13 Position functions

`class PositionFunctions`

Public Static Functions

double **`GetPosX`**(unsigned short *pid*)
Get the X position of a player.

Return The X position.

Parameters

- `pid`: The player ID.

double **GetPosY**(unsigned short *pid*)
Get the Y position of a player.

Return The Y position.

Parameters

- `pid`: The player ID.

double **GetPosZ**(unsigned short *pid*)
Get the Z position of a player.

Return The Z position.

Parameters

- `pid`: The player ID.

double **GetPreviousCellPosX**(unsigned short *pid*)
Get the X position of a player from before their latest cell change.

Return The X position.

Parameters

- `pid`: The player ID.

double **GetPreviousCellPosY**(unsigned short *pid*)
Get the Y position of a player from before their latest cell change.

Return The Y position.

Parameters

- `pid`: The player ID.

double **GetPreviousCellPosZ**(unsigned short *pid*)
Get the Z position of a player from before their latest cell change.

Return The Z position.

Parameters

- `pid`: The player ID.

double **GetRotX**(unsigned short *pid*)
Get the X rotation of a player.

Return The X rotation.

Parameters

- `pid`: The player ID.

double **GetRotZ**(unsigned short *pid*)
Get the Z rotation of a player.

Return The Z rotation.

Parameters

- `pid`: The player ID.

void **SetPos**(unsigned short *pid*, double *x*, double *y*, double *z*)
Set the position of a player.

This changes the positional coordinates recorded for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `x`: The X position.
- `y`: The Y position.
- `z`: The Z position.

void **SetRot**(unsigned short *pid*, double *x*, double *z*)
Set the rotation of a player.

This changes the rotational coordinates recorded for that player in the server memory, but does not by itself send a packet.

A player's Y rotation is always 0, which is why there is no Y rotation parameter.

Return void

Parameters

- `pid`: The player ID.
- `x`: The X position.
- `z`: The Z position.

void **SetMomentum**(unsigned short *pid*, double *x*, double *y*, double *z*)
Set the momentum of a player.

This changes the coordinates recorded for that player's momentum in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `x`: The X momentum.
- `y`: The Y momentum.
- `z`: The Z momentum.

void **SendPos**(unsigned short *pid*)
Send a PlayerPosition packet about a player.

It is only sent to the affected player.

Return void

Parameters

- `pid`: The player ID.

void **SendMomentum**(unsigned short *pid*)
Send a PlayerMomentum packet about a player.

It is only sent to the affected player.

Return void

Parameters

- `pid`: The player ID.

1.14 Quest functions

class QuestFunctions

Public Static Functions

void **ClearJournalChanges**(unsigned short *pid*)
Clear the last recorded journal changes for a player.

This is used to initialize the sending of new PlayerJournal packets.

Return void

Parameters

- `pid`: The player ID whose journal changes should be used.

void **ClearKillChanges**(unsigned short *pid*)
Clear the last recorded kill count changes for a player.

This is used to initialize the sending of new WorldKillCount packets.

Return void

Parameters

- `pid`: The player ID whose kill count changes should be used.

unsigned int **GetJournalChangesSize**(unsigned short *pid*)
Get the number of indexes in a player's latest journal changes.

Return The number of indexes.

Parameters

- `pid`: The player ID whose journal changes should be used.

unsigned int **`GetKillChangesSize`**(unsigned short *pid*)
Get the number of indexes in a player's latest kill count changes.

Return The number of indexes.

Parameters

- `pid`: The player ID whose kill count changes should be used.

void **`AddJournalEntry`**(unsigned short *pid*, **`const`** char **quest*, unsigned int *index*, **`const`** char **actorRefId*)
Add a new journal item of type ENTRY to the journal changes for a player, with a specific timestamp.

Return void

Parameters

- `pid`: The player ID whose journal changes should be used.
- `quest`: The quest of the journal item.
- `index`: The quest index of the journal item.
- `actorRefId`: The actor refId of the journal item.

void **`AddJournalEntryWithTimestamp`**(unsigned short *pid*, **`const`** char **quest*, unsigned int *index*, **`const`** char **actorRefId*, unsigned int *daysPassed*, unsigned int *month*, unsigned int *day*)
Add a new journal item of type ENTRY to the journal changes for a player, with a specific timestamp.

Return void

Parameters

- `pid`: The player ID whose journal changes should be used.
- `quest`: The quest of the journal item.
- `index`: The quest index of the journal item.
- `actorRefId`: The actor refId of the journal item.
- `The`: daysPassed for the journal item.
- `The`: month for the journal item.
- `The`: day of the month for the journal item.

void **`AddJournalIndex`**(unsigned short *pid*, **`const`** char **quest*, unsigned int *index*)
Add a new journal item of type INDEX to the journal changes for a player.

Return void

Parameters

- `pid`: The player ID whose journal changes should be used.
- `quest`: The quest of the journal item.
- `index`: The quest index of the journal item.

void **AddKill**(unsigned short *pid*, **const** char **refId*, int *number*)
Add a new kill count to the kill count changes for a player.

Return void

Parameters

- `pid`: The player ID whose kill count changes should be used.
- `refId`: The refId of the kill count.
- `number`: The number of kills in the kill count.

void **SetReputation**(unsigned short *pid*, int *value*)
Set the reputation of a certain player.

Return void

Parameters

- `pid`: The player ID.
- `value`: The reputation.

const char ***GetJournalItemQuest**(unsigned short *pid*, unsigned int *index*)
Get the quest at a certain index in a player's latest journal changes.

Return The quest.

Parameters

- `pid`: The player ID whose journal changes should be used.
- `index`: The index of the journalItem.

int **GetJournalItemIndex**(unsigned short *pid*, unsigned int *index*)
Get the quest index at a certain index in a player's latest journal changes.

Return The quest index.

Parameters

- `pid`: The player ID whose journal changes should be used.
- `index`: The index of the journalItem.

int **GetJournalItemType**(unsigned short *pid*, unsigned int *index*)
Get the journal item type at a certain index in a player's latest journal changes.

Return The type (0 for ENTRY, 1 for INDEX).

Parameters

- `pid`: The player ID whose journal changes should be used.
- `index`: The index of the journalItem.

const char ***GetJournalItemActorRefId**(unsigned short *pid*, unsigned int *index*)
Get the actor refId at a certain index in a player's latest journal changes.

Every journal change has an associated actor, which is usually the quest giver.

Return The actor refId.

Parameters

- `pid`: The player ID whose journal changes should be used.
- `index`: The index of the journalItem.

const char ***GetKillRefId**(unsigned short *pid*, unsigned int *index*)
Get the refId at a certain index in a player's latest kill count changes.

Return The refId.

Parameters

- `pid`: The player ID whose kill count changes should be used.
- `index`: The index of the kill count.

int **GetKillNumber**(unsigned short *pid*, unsigned int *index*)
Get the number of kills at a certain index in a player's latest kill count changes.

Return The number of kills.

Parameters

- `pid`: The player ID whose kill count changes should be used.
- `index`: The index of the kill count.

int **GetReputation**(unsigned short *pid*)
Get the a certain player's reputation.

Return The reputation.

Parameters

- `pid`: The player ID.

void **SendJournalChanges**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a PlayerJournal packet with a player's recorded journal changes.

Return void

Parameters

- `pid`: The player ID whose journal changes should be used.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **SendKillChanges**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a WorldKillCount packet with a player's recorded kill count changes.

Return void

Parameters

- `pid`: The player ID whose kill count changes should be used.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **`SendReputation`**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a PlayerReputation packet with a player's recorded reputation.

Return void

Parameters

- `pid`: The player ID whose reputation should be used.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

1.15 Records Dynamic functions

`class RecordsDynamicFunctions`

Public Static Functions

void **`ClearRecords`**()
Clear the data from the records stored on the server.

Return void

unsigned short **`GetRecordType`**()
Get the type of records in the read worldstate's dynamic records.

Return The type of records (0 for SPELL, 1 for POTION, 2 for ENCHANTMENT, 3 for NPC).

unsigned int **`GetRecordCount`**()
Get the number of records in the read worldstate's dynamic records.

Return The number of records.

unsigned int **`GetRecordEffectCount`**(unsigned int *recordIndex*)
Get the number of effects for the record at a certain index in the read worldstate's current records.

Return The number of effects.

Parameters

- `recordIndex`: The index of the record.

const char ***GetRecordId**(unsigned int *index*)
Get the id of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The id of the record.

Parameters

- `index`: The index of the record.

const char ***GetRecordBaseId**(unsigned int *index*)
Get the base id (i.e. the id this record should inherit default values from) of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The base id of the record.

Parameters

- `index`: The index of the record.

int **GetRecordSubtype**(unsigned int *index*)
Get the subtype of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The type of the record.

Parameters

- `index`: The index of the record.

const char ***GetRecordName**(unsigned int *index*)
Get the name of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The name of the record.

Parameters

- `index`: The index of the record.

const char ***GetRecordModel**(unsigned int *index*)
Get the model of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The model of the record.

Parameters

- `index`: The index of the record.

const char ***GetRecordIcon**(unsigned int *index*)
Get the icon of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The icon of the record.

Parameters

- `index`: The index of the record.

const char ***GetRecordScript**(unsigned int *index*)
Get the script of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The script of the record.

Parameters

- `index`: The index of the record.

const char ***GetRecordEnchantmentId**(unsigned int *index*)

Get the enchantment id of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The enchantment id of the record.

Parameters

- `index`: The index of the record.

int **GetRecordEnchantmentCharge**(unsigned int *index*)

Get the enchantment charge of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The enchantment charge of the record.

Parameters

- `index`: The index of the record.

int **GetRecordAutoCalc**(unsigned int *index*)

Get the auto-calculation flag value of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The auto-calculation flag value of the record.

Parameters

- `index`: The index of the record.

int **GetRecordCharge**(unsigned int *index*)

Get the charge of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The charge of the record.

Parameters

- `index`: The index of the record.

int **GetRecordCost**(unsigned int *index*)

Get the cost of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The cost of the record.

Parameters

- `index`: The index of the record.

int **GetRecordFlags**(unsigned int *index*)

Get the flags of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The flags of the spell as an integer.

Parameters

- `index`: The index of the record.

int **GetRecordValue**(unsigned int *index*)
Get the value of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The value of the record.

Parameters

- `index`: The index of the record.

double **GetRecordWeight**(unsigned int *index*)
Get the weight of the record at a certain index in the read worldstate's dynamic records of the current type.

Return The weight of the record.

Parameters

- `index`: The index of the record.

unsigned int **GetRecordEffectId**(unsigned int *recordIndex*, unsigned int *effectIndex*)
Get the ID of the effect at a certain index in the read worldstate's current records.

Return The ID of the effect.

Parameters

- `recordIndex`: The index of the record.
- `effectIndex`: The index of the effect.

int **GetRecordEffectAttribute**(unsigned int *recordIndex*, unsigned int *effectIndex*)
Get the ID of the attribute modified by the effect at a certain index in the read worldstate's current records.

Return The attribute ID for the effect.

Parameters

- `recordIndex`: The index of the record.
- `effectIndex`: The index of the effect.

int **GetRecordEffectSkill**(unsigned int *recordIndex*, unsigned int *effectIndex*)
Get the ID of the skill modified by the effect at a certain index in the read worldstate's current records.

Return The skill ID for the effect.

Parameters

- `recordIndex`: The index of the record.
- `effectIndex`: The index of the effect.

unsigned int **GetRecordEffectRangeType**(unsigned int *recordIndex*, unsigned int *effectIndex*)
Get the range type of the effect at a certain index in the read worldstate's current records (0 for self, 1 for touch, 2 for target).

Return The range of the effect.

Parameters

- `recordIndex`: The index of the record.
- `effectIndex`: The index of the effect.

int **GetRecordEffectArea**(unsigned int *recordIndex*, unsigned int *effectIndex*)
Get the area of the effect at a certain index in the read worldstate's current records.

Return The area of the effect.

Parameters

- `recordIndex`: The index of the record.
- `effectIndex`: The index of the effect.

int **GetRecordEffectDuration**(unsigned int *recordIndex*, unsigned int *effectIndex*)
Get the duration of the effect at a certain index in the read worldstate's current records.

Return The duration of the effect.

Parameters

- `recordIndex`: The index of the record.
- `effectIndex`: The index of the effect.

int **GetRecordEffectMagnitudeMax**(unsigned int *recordIndex*, unsigned int *effectIndex*)
Get the maximum magnitude of the effect at a certain index in the read worldstate's current records.

Return The maximum magnitude of the effect.

Parameters

- `recordIndex`: The index of the record.
- `effectIndex`: The index of the effect.

int **GetRecordEffectMagnitudeMin**(unsigned int *recordIndex*, unsigned int *effectIndex*)
Get the minimum magnitude of the effect at a certain index in the read worldstate's current records.

Return The minimum magnitude of the effect.

Parameters

- `recordIndex`: The index of the record.
- `effectIndex`: The index of the effect.

void **SetRecordType**(unsigned int *type*)
Set which type of temporary records stored on the server should have their data changed via setter functions.

Return void

Parameters

- `type`: The type of records.

void **SetRecordId**(**const** char **id*)
Set the id of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `id`: The id of the record.

void **SetRecordBaseId**(**const** char **baseId*)
Set the base id (i.e. the id this record should inherit default values from) of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `baseId`: The baseId of the record.

void **SetRecordInventoryBaseId**(**const** char **inventoryBaseId*)
Set the inventory base id (i.e. the id this record should inherit its inventory contents from) of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `inventoryBaseId`: The inventoryBaseId of the record.

void **SetRecordSubtype**(unsigned int *subtype*)
Set the subtype of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `type`: The spell type.

void **SetRecordName**(**const** char **name*)
Set the name of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `name`: The name of the record.

void **SetRecordModel**(**const** char **model*)
Set the model of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `model`: The model of the record.

void **SetRecordIcon**(**const** char **icon*)
Set the icon of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `icon`: The icon of the record.

void **SetRecordScript**(**const** char *_script_)
Set the script of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `script`: The script of the record.

void **SetRecordEnchantmentId**(**const** char *_enchantmentId_)
Set the enchantment id of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `enchantmentId`: The enchantment id of the record.

void **SetRecordEnchantmentCharge**(int _enchantmentCharge_)
Set the enchantment charge of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `enchantmentCharge`: The enchantmentCharge of the record.

void **SetRecordAutoCalc**(int _autoCalc_)
Set the auto-calculation flag value of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `autoCalc`: The auto-calculation flag value of the record.

void **SetRecordCharge**(int _charge_)
Set the charge of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `charge`: The charge of the record.

void **SetRecordCost**(int _cost_)
Set the cost of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `cost`: The cost of the record.

void **`SetRecordFlags`**(int *flags*)
Set the flags of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `flags`: The flags of the record.

void **`SetRecordValue`**(int *value*)
Set the value of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `value`: The value of the record.

void **`SetRecordWeight`**(double *weight*)
Set the weight of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `weight`: The weight of the record.

void **`SetRecordArmorRating`**(int *armorRating*)
Set the armor rating of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `armorRating`: The armor rating of the record.

void **`SetRecordHealth`**(int *health*)
Set the health of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `health`: The health of the record.

void **`SetRecordDamageChop`**(unsigned int *minDamage*, unsigned int *maxDamage*)
Set the chop damage of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `minDamage`: The minimum damage of the record.
- `maxDamage`: The maximum damage of the record.

void **SetRecordDamageSlash**(unsigned int *minDamage*, unsigned int *maxDamage*)
Set the slash damage of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `minDamage`: The minimum damage of the record.
- `maxDamage`: The maximum damage of the record.

void **SetRecordDamageThrust**(unsigned int *minDamage*, unsigned int *maxDamage*)
Set the thrust damage of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `minDamage`: The minimum damage of the record.
- `maxDamage`: The maximum damage of the record.

void **SetRecordReach**(double *reach*)
Set the reach of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `reach`: The reach of the record.

void **SetRecordSpeed**(double *speed*)
Set the speed of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `speed`: The speed of the record.

void **SetRecordKeyState**(bool *keyState*)
Set whether the temporary record stored on the server for the currently specified record type is a key.

Note: This is only applicable to Miscellaneous records.

Return void

Parameters

- `keyState`: Whether the record is a key.

void **SetRecordScrollState**(bool *scrollState*)
Set whether the temporary record stored on the server for the currently specified record type is a scroll.

Note: This is only applicable to Book records.

Return void

Parameters

- `scrollState`: Whether the record is a scroll.

void **SetRecordSkillId**(int *skillId*)
Set the skill ID of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `skillId`: The skill ID of the record.

void **SetRecordText**(**const** char **text*)
Set the text of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `text`: The text of the record.

void **SetRecordHair**(**const** char **hair*)
Set the hair of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `hair`: The hair of the record.

void **SetRecordHead**(**const** char **head*)
Set the head of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `hair`: The head of the record.

void **SetRecordGender**(unsigned int *gender*)
Set the gender of the temporary record stored on the server for the currently specified record type (0 for female, 1 for male).

Return void

Parameters

- `hair`: The race of the record.

void **SetRecordRace**(**const** char **race*)
Set the race of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `hair`: The race of the record.

void **SetRecordClass**(**const** char **charClass*)
Set the character class of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `hair`: The character class of the record.

void **SetRecordFaction**(**const** char **faction*)
Set the faction of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `faction`: The faction of the record.

void **SetRecordLevel**(int *level*)
Set the level of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `level`: The level of the record.

void **SetRecordMagicka**(int *magicka*)
Set the magicka of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `magicka`: The magicka of the record.

void **SetRecordFatigue**(int *fatigue*)
Set the fatigue of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `fatigue`: The fatigue of the record.

void **SetRecordAIFight**(int *aiFight*)
Set the AI fight value of the temporary record stored on the server for the currently specified record type.

Return void

Parameters

- `aiFight`: The AI fight value of the record.

void **SetRecordIdByIndex**(unsigned int *index*, **const** char **id*)
Set the id of the record at a certain index in the records stored on the server.

When resending a received RecordsDynamic packet, this allows you to set the server-generated id of a record without having to clear and recreate the packet.

Return void

Parameters

- `index`: The index of the record.
- `id`: The id of the record.

void **SetRecordEnchantmentIdByIndex**(unsigned int *index*, **const** char **enchantmentId*)
Set the enchantment id of the record at a certain index in the records stored on the server.

When resending a received RecordsDynamic packet, this allows you to set the server-generated enchantment id of a record without having to clear and recreate the packet.

Return void

Parameters

- `index`: The index of the record.
- `enchantmentId`: The enchantment id of the record.

void **SetRecordEffectId**(unsigned int *effectId*)
Set the ID of the temporary effect stored on the server.

Return void

Parameters

- `effectId`: The ID of the effect.

void **SetRecordEffectAttribute**(int *attributeId*)
Set the ID of the attribute modified by the temporary effect stored on the server.

Return void

Parameters

- `attributeId`: The ID of the attribute.

void **SetRecordEffectSkill**(int *skillId*)
Set the ID of the skill modified by the temporary effect stored on the server.

Return void

Parameters

- `skillId`: The ID of the skill.

void **SetRecordEffectRangeType**(unsigned int *rangeType*)
Set the range type of the temporary effect stored on the server (0 for self, 1 for touch, 2 for target).

Return void

Parameters

- `rangeType`: The range type of the effect.

void **SetRecordEffectArea**(int *area*)
Set the area of the temporary effect stored on the server.

Return void

Parameters

- `area`: The area of the effect.

void **SetRecordEffectDuration**(int *duration*)
Set the duration of the temporary effect stored on the server.

Return void

Parameters

- `duration`: The duration of the effect.

void **SetRecordEffectMagnitudeMax**(int *magnitudeMax*)
Set the maximum magnitude of the temporary effect stored on the server.

Return void

Parameters

- `magnitudeMax`: The maximum magnitude of the effect.

void **SetRecordEffectMagnitudeMin**(int *magnitudeMin*)
Set the minimum magnitude of the temporary effect stored on the server.

Return void

Parameters

- `magnitudeMin`: The minimum magnitude of the effect.

void **SetRecordBodyPartType**(unsigned int *partType*)
Set the type of the temporary body part stored on the server.

Return void

Parameters

- `partType`: The type of the body part.

void **SetRecordBodyPartIdForMale**(**const** char **partId*)
Set the id of the male version of the temporary body part stored on the server.

Return void

Parameters

- `partId`: The id of the body part.

void **SetRecordBodyPartIdForFemale**(**const** char **partId*)
Set the id of the female version of the temporary body part stored on the server.

Return void

Parameters

- `partId`: The id of the body part.

void **SetRecordInventoryItemId**(**const** char **itemId*)
Set the id of the of the temporary inventory item stored on the server.

Return void

Parameters

- `partId`: The id of the inventory item.

void **SetRecordInventoryItemCount**(unsigned int *count*)
Set the count of the of the temporary inventory item stored on the server.

Return void

Parameters

- `count`: The count of the inventory item.

void **AddRecord**()
Add a copy of the server's temporary record of the current specified type to the stored records.

In the process, the server's temporary record will automatically be cleared so a new one can be set up.

Return void

void **AddRecordEffect**()
Add a copy of the server's temporary effect to the temporary record of the current specified type.

In the process, the server's temporary effect will automatically be cleared so a new one can be set up.

Return void

void **AddRecordBodyPart**()
Add a copy of the server's temporary body part to the temporary record of the current specified type.

In the process, the server's temporary body part will automatically be cleared so a new one can be set up.

Return void

void **AddRecordInventoryItem**()
Add a copy of the server's temporary inventory item to the temporary record of the current specified type.

In the process, the server's temporary inventory item will automatically be cleared so a new one can be set up.

Note: Any items added this way will be ignored if the record already has a valid inventoryBaseId.

Return void

void **SendRecordDynamic**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a RecordDynamic packet with the current specified record type.

Return void

Parameters

- `pid`: The player ID attached to the packet.

- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

1.16 Server functions

class ServerFunctions

Public Static Functions

void **StopServer**(int *code*)
Shut down the server.

Return void

Parameters

- `code`: The shutdown code.

void **Kick**(unsigned short *pid*)
Kick a certain player from the server.

Return void

Parameters

- `pid`: The player ID.

void **BanAddress**(**const** char **ipAddress*)
Ban a certain IP address from the server.

Return void

Parameters

- `ipAddress`: The IP address.

void **UnbanAddress**(**const** char **ipAddress*)
Unban a certain IP address from the server.

Return void

Parameters

- `ipAddress`: The IP address.

const char ***GetOperatingSystemType**()
Get the type of the operating system used by the server.

Note: Currently, the type can be "Windows", "Linux", "OS X" or "Unknown OS".

Return The type of the operating system.

const char ***GetArchitectureType**()
Get the architecture type used by the server.

Note: Currently, the type can be "64-bit", "32-bit", "ARMv#" or "Unknown architecture".

Return The architecture type.

const char ***GetServerVersion**()
Get the TES3MP version of the server.

Return The server version.

const char ***GetProtocolVersion**()
Get the protocol version of the server.

Return The protocol version.

int **GetAvgPing**(unsigned short *pid*)
Get the average ping of a certain player.

Return The average ping.

Parameters

- `pid`: The player ID.

const char ***GetIP**(unsigned short *pid*)
Get the IP address of a certain player.

Return The IP address.

Parameters

- `pid`: The player ID.

unsigned short **GetPort**()
Get the port used by the server.

Return Port

unsigned int **GetMaxPlayers**()
Get the maximum number of players.

Return Max players

bool **HasPassword**()
Checking if the server requires a password to connect.

Return

bool **GetPluginEnforcementState**()
Get the plugin enforcement state of the server.

If true, clients are required to use the same plugins as set for the server.

Return The enforcement state.

bool **GetScriptErrorIgnoringState**()
Get the script error ignoring state of the server.

If true, script errors will not crash the server.

Return The script error ignoring state.

void **SetGameMode**(**const** char **gameMode*)
Set the game mode of the server, as displayed in the server browser.

Return void

Parameters

- `name`: The new game mode.

void **SetHostname**(**const** char **name*)
Set the name of the server, as displayed in the server browser.

Return void

Parameters

- `name`: The new name.

void **SetServerPassword**(**const** char **password*)
Set the password required to join the server.

Return void

Parameters

- `password`: The password.

void **SetPluginEnforcementState**(bool *state*)
Set the plugin enforcement state of the server.

If true, clients are required to use the same plugins as set for the server.

Return void

Parameters

- `state`: The new enforcement state.

void **SetScriptErrorIgnoringState**(bool *state*)
Set whether script errors should be ignored or not.

If true, script errors will not crash the server, but could have any number of unforeseen consequences, which is why this is a highly experimental setting.

Return void

Parameters

- `state`: The new script error ignoring state.

void **SetRuleString**(**const** char **key*, **const** char **value*)
Set a rule string for the server details displayed in the server browser.

Return void

Parameters

- `key`: The name of the rule.
- `value`: The string value of the rule.

void **SetRuleValue**(**const** char **key*, double *value*)
Set a rule value for the server details displayed in the server browser.

Return void

Parameters

- `key`: The name of the rule.
- `value`: The numerical value of the rule.

void **AddPluginHash**(**const** char **pluginName*, **const** char **hash*)
Adds plugins to the internal server structure to validate players.

Parameters

- `pluginName`: Name with extension of the plugin or master file.
- `hash`: Hash string

1.17 Setting functions

class SettingFunctions

Public Static Functions

void **SetDifficulty**(unsigned short *pid*, int *difficulty*)
Set the difficulty for a player.

This changes the difficulty for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `difficulty`: The difficulty.

void **SetEnforcedLogLevel**(unsigned short *pid*, int *enforcedLogLevel*)
Set the client log level enforced for a player.

This changes the enforced log level for that player in the server memory, but does not by itself send a packet.

Enforcing a certain log level is necessary to prevent players from learning information from their console window that they are otherwise unable to obtain, such as the locations of other players.

If you do not wish to enforce a log level, simply set enforcedLogLevel to -1

Return void

Parameters

- `pid`: The player ID.
- `enforcedLogLevel`: The enforced log level.

void **`SetPhysicsFramerate`**(unsigned short *pid*, double *physicsFramerate*)
Set the physics framerate for a player.

This changes the physics framerate for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `physicsFramerate`: The physics framerate.

void **`SetConsoleAllowed`**(unsigned short *pid*, bool *state*)
Set whether the console is allowed for a player.

This changes the console permission for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `state`: The console permission state.

void **`SetBedRestAllowed`**(unsigned short *pid*, bool *state*)
Set whether resting in beds is allowed for a player.

This changes the resting permission for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `state`: The resting permission state.

void **`SetWildernessRestAllowed`**(unsigned short *pid*, bool *state*)
Set whether resting in the wilderness is allowed for a player.

This changes the resting permission for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `state`: The resting permission state.

void **SetWaitAllowed**(unsigned short *pid*, bool *state*)
Set whether waiting is allowed for a player.

This changes the waiting permission for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `state`: The waiting permission state.

void **SendSettings**(unsigned short *pid*)
Send a PlayerSettings packet to the player affected by it.

Return void

Parameters

- `pid`: The player ID to send it to.

1.18 Shapeshift functions

class ShapeshiftFunctions

Public Static Functions

double **GetScale**(unsigned short *pid*)
Get the scale of a player.

Return The scale.

Parameters

- `pid`: The player ID.

bool **IsWerewolf**(unsigned short *pid*)
Check whether a player is a werewolf.

This is based on the last PlayerShapeshift packet received or sent for that player.

Return The werewolf state.

Parameters

- `pid`: The player ID.

const char ***GetCreatureRefId**(unsigned short *pid*)
Get the refId of the creature the player is disguised as.

Return The creature refId.

Parameters

- `pid`: The player ID.

bool **GetCreatureNameDisplayState**(unsigned short *pid*)
Check whether a player's name is replaced by that of the creature they are disguised as when other players hover over them.

This is based on the last PlayerShapeshift packet received or sent for that player.

Return The creature name display state.

Parameters

- `pid`: The player ID.

void **SetScale**(unsigned short *pid*, double *scale*)
Set the scale of a player.

This changes the scale recorded for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `scale`: The new scale.

void **SetWerewolfState**(unsigned short *pid*, bool *isWerewolf*)
Set the werewolf state of a player.

This changes the werewolf state recorded for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `isWerewolf`: The new werewolf state.

void **SetCreatureRefId**(unsigned short *pid*, **const** char **refId*)
Set the refId of the creature a player is disguised as.

This changes the creature refId recorded for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `refId`: The creature refId.

- `displaysCreatureName`: Whether the player's name appears as that of the creature when hovered over by others.

void **`SetCreatureNameDisplayState`**(unsigned short *pid*, bool *displayState*)
Set whether a player's name is replaced by that of the creature they are disguised as when other players hover over them.

Return void

Parameters

- `pid`: The player ID.
- `displayState`: The creature name display state.

void **`SendShapeshift`**(unsigned short *pid*)
Send a PlayerShapeshift packet about a player.

This sends the packet to all players connected to the server. It is currently used only to communicate werewolf states.

Return void

Parameters

- `pid`: The player ID.

1.19 Spell functions

`class SpellFunctions`

Public Static Functions

void **`ClearSpellbookChanges`**(unsigned short *pid*)
Clear the last recorded spellbook changes for a player.

This is used to initialize the sending of new PlayerSpellbook packets.

Return void

Parameters

- `pid`: The player ID whose spellbook changes should be used.

unsigned int **`GetSpellbookChangesSize`**(unsigned short *pid*)
Get the number of indexes in a player's latest spellbook changes.

Return The number of indexes.

Parameters

- `pid`: The player ID whose spellbook changes should be used.

unsigned int **`GetSpellbookChangesAction`**(unsigned short *pid*)
Get the action type used in a player's latest spellbook changes.

Return The action type (0 for SET, 1 for ADD, 2 for REMOVE).

Parameters

- `pid`: The player ID whose spellbook changes should be used.

void **SetSpellbookChangesAction**(unsigned short *pid*, unsigned char *action*)
Set the action type in a player's spellbook changes.

Return void

Parameters

- `pid`: The player ID whose spellbook changes should be used.
- `action`: The action (0 for SET, 1 for ADD, 2 for REMOVE).

void **AddSpell**(unsigned short *pid*, **const** char **spellId*)
Add a new spell to the spellbook changes for a player.

Return void

Parameters

- `pid`: The player ID whose spellbook changes should be used.
- `spellId`: The spellId of the spell.

const char ***GetSpellId**(unsigned short *pid*, unsigned int *index*)
Get the spellId at a certain index in a player's latest spellbook changes.

Return The spellId.

Parameters

- `pid`: The player ID whose spellbook changes should be used.
- `index`: The index of the spell.

void **SendSpellbookChanges**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a PlayerSpellbook packet with a player's recorded spellbook changes.

Return void

Parameters

- `pid`: The player ID whose spellbook changes should be used.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

1.20 Stats functions

class StatsFunctions

Public Static Functions

int **GetAttributeCount**()
Get the number of attributes.

The number is 8 before any dehardcoding is done in OpenMW.

Return The number of attributes.

int **GetSkillCount**()
Get the number of skills.

The number is 27 before any dehardcoding is done in OpenMW.

Return The number of skills.

int **GetAttributeId**(**const** char ***name*)
Get the numerical ID of an attribute with a certain name.

If an invalid name is used, the ID returned is -1

Return The ID of the attribute.

Parameters

- `name`: The name of the attribute.

int **GetSkillId**(**const** char ***name*)
Get the numerical ID of a skill with a certain name.

If an invalid name is used, the ID returned is -1

Return The ID of the skill.

Parameters

- `name`: The name of the skill.

const char ***GetAttributeName**(unsigned short *attributeId*)
Get the name of the attribute with a certain numerical ID.

If an invalid ID is used, "invalid" is returned.

Return The name of the attribute.

Parameters

- `attributeId`: The ID of the attribute.

const char ***GetSkillName**(unsigned short *skillId*)
Get the name of the skill with a certain numerical ID.

If an invalid ID is used, "invalid" is returned.

Return The name of the skill.

Parameters

- `skillId`: The ID of the skill.

const char ***GetName**(unsigned short *pid*)
Get the name of a player.

Return The name of the player.

Parameters

- `pid`: The player ID.

const char ***GetRace**(unsigned short *pid*)
Get the race of a player.

Return The race of the player.

Parameters

- `pid`: The player ID.

const char ***GetHead**(unsigned short *pid*)
Get the head mesh used by a player.

Return The head mesh of the player.

Parameters

- `pid`: The player ID.

const char ***GetHairstyle**(unsigned short *pid*)
Get the hairstyle mesh used by a player.

Return The hairstyle mesh of the player.

Parameters

- `pid`: The player ID.

int **GetIsMale**(unsigned short *pid*)
Check whether a player is male or not.

Return Whether the player is male.

Parameters

- `pid`: The player ID.

const char ***GetBirthsign**(unsigned short *pid*)
Get the birthsign of a player.

Return The birthsign of the player.

Parameters

- `pid`: The player ID.

int **GetLevel**(unsigned short *pid*)
Get the character level of a player.

Return The level of the player.

Parameters

- `pid`: The player ID.

int **`GetLevelProgress`**(unsigned short *pid*)
Get the player's progress to their next character level.

Return The level progress.

Parameters

- `pid`: The player ID.

double **`GetHealthBase`**(unsigned short *pid*)
Get the base health of the player.

Return The base health.

Parameters

- `pid`: The player ID.

double **`GetHealthCurrent`**(unsigned short *pid*)
Get the current health of the player.

Return The current health.

Parameters

- `pid`: The player ID.

double **`GetMagickaBase`**(unsigned short *pid*)
Get the base magicka of the player.

Return The base magicka.

Parameters

- `pid`: The player ID.

double **`GetMagickaCurrent`**(unsigned short *pid*)
Get the current magicka of the player.

Return The current magicka.

Parameters

- `pid`: The player ID.

double **`GetFatigueBase`**(unsigned short *pid*)
Get the base fatigue of the player.

Return The base fatigue.

Parameters

- `pid`: The player ID.

double **GetFatigueCurrent** (unsigned short *pid*)
Get the current fatigue of the player.

Return The current fatigue.

Parameters

- `pid`: The player ID.

int **GetAttributeBase** (unsigned short *pid*, unsigned short *attributeId*)
Get the base value of a player's attribute.

Return The base value of the attribute.

Parameters

- `pid`: The player ID.
- `attributeId`: The attribute ID.

int **GetAttributeModifier** (unsigned short *pid*, unsigned short *attributeId*)
Get the modifier value of a player's attribute.

Return The modifier value of the attribute.

Parameters

- `pid`: The player ID.
- `attributeId`: The attribute ID.

int **GetSkillBase** (unsigned short *pid*, unsigned short *skillId*)
Get the base value of a player's skill.

Return The base value of the skill.

Parameters

- `pid`: The player ID.
- `skillId`: The skill ID.

int **GetSkillModifier** (unsigned short *pid*, unsigned short *skillId*)
Get the modifier value of a player's skill.

Return The modifier value of the skill.

Parameters

- `pid`: The player ID.
- `skillId`: The skill ID.

double **GetSkillProgress** (unsigned short *pid*, unsigned short *skillId*)
Get the progress the player has made towards increasing a certain skill by 1.

Return The skill progress.

Parameters

- `pid`: The player ID.

- `skillId`: The skill ID.

int **GetSkillIncrease**(unsigned short *pid*, unsigned int *attributeId*)
Get the bonus applied to a certain attribute at the next level up as a result of associated skill increases.

Although confusing, the term "skill increase" for this is taken from OpenMW itself.

Return The increase in the attribute caused by skills.

Parameters

- `pid`: The player ID.
- `skillId`: The attribute ID.

int **GetBounty**(unsigned short *pid*)
Get the bounty of the player.

Return The bounty.

Parameters

- `pid`: The player ID.

void **SetName**(unsigned short *pid*, **const** char **name*)
Set the name of a player.

Return void

Parameters

- `pid`: The player ID.
- `name`: The new name of the player.

void **SetRace**(unsigned short *pid*, **const** char **race*)
Set the race of a player.

Return void

Parameters

- `pid`: The player ID.
- `race`: The new race of the player.

void **SetHead**(unsigned short *pid*, **const** char **head*)
Set the head mesh used by a player.

Return void

Parameters

- `pid`: The player ID.
- `head`: The new head mesh of the player.

void **SetHairstyle**(unsigned short *pid*, **const** char **hairstyle*)
Set the hairstyle mesh used by a player.

Return void

Parameters

- `pid`: The player ID.
- `hairstyle`: The new hairstyle mesh of the player.

void **SetIsMale**(unsigned short *pid*, int *state*)
Set whether a player is male or not.

Return void

Parameters

- `pid`: The player ID.
- `state`: Whether the player is male.

void **SetBirthsign**(unsigned short *pid*, **const** char **name*)
Set the birthsign of a player.

Return void

Parameters

- `pid`: The player ID.
- `name`: The new birthsign of the player.

void **SetResetStats**(unsigned short *pid*, bool *resetStats*)
Set whether the player's stats should be reset based on their current race as the result of a PlayerBaseInfo packet.

This changes the resetState for that player in the server memory, but does not by itself send a packet.

Return void

Parameters

- `pid`: The player ID.
- `resetStats`: The stat reset state.

void **SetLevel**(unsigned short *pid*, int *value*)
Set the character level of a player.

Return void

Parameters

- `pid`: The player ID.
- `value`: The new level of the player.

void **SetLevelProgress**(unsigned short *pid*, int *value*)
Set the player's progress to their next character level.

Return void

Parameters

- `pid`: The player ID.
- `value`: The new level progress of the player.

void **`SetHealthBase`**(unsigned short *pid*, double *value*)
Set the base health of a player.

Return void

Parameters

- `pid`: The player ID.
- `name`: The new base health of the player.

void **`SetHealthCurrent`**(unsigned short *pid*, double *value*)
Set the current health of a player.

Return void

Parameters

- `pid`: The player ID.
- `name`: The new current health of the player.

void **`SetMagickaBase`**(unsigned short *pid*, double *value*)
Set the base magicka of a player.

Return void

Parameters

- `pid`: The player ID.
- `name`: The new base magicka of the player.

void **`SetMagickaCurrent`**(unsigned short *pid*, double *value*)
Set the current magicka of a player.

Return void

Parameters

- `pid`: The player ID.
- `name`: The new current magicka of the player.

void **`SetFatigueBase`**(unsigned short *pid*, double *value*)
Set the base fatigue of a player.

Return void

Parameters

- `pid`: The player ID.
- `name`: The new base fatigue of the player.

void **`SetFatigueCurrent`**(unsigned short *pid*, double *value*)
Set the current fatigue of a player.

Return void

Parameters

- `pid`: The player ID.
- `name`: The new current fatigue of the player.

void **`SetAttributeBase`**(unsigned short *pid*, unsigned short *attributeId*, int *value*)
Set the base value of a player's attribute.

Return void

Parameters

- `pid`: The player ID.
- `attributeId`: The attribute ID.
- `value`: The new base value of the player's attribute.

void **`ClearAttributeModifier`**(unsigned short *pid*, unsigned short *attributeId*)
Clear the modifier value of a player's attribute.

There's no way to set a modifier to a specific value because it can come from multiple different sources, but clearing it is a straightforward process that dispels associated effects on a client and, if necessary, unequips associated items.

Return void

Parameters

- `pid`: The player ID.
- `attributeId`: The attribute ID.

void **`SetSkillBase`**(unsigned short *pid*, unsigned short *skillId*, int *value*)
Set the base value of a player's skill.

Return void

Parameters

- `pid`: The player ID.
- `skillId`: The skill ID.
- `value`: The new base value of the player's skill.

void **`ClearSkillModifier`**(unsigned short *pid*, unsigned short *skillId*)
Clear the modifier value of a player's skill.

There's no way to set a modifier to a specific value because it can come from multiple different sources, but clearing it is a straightforward process that dispels associated effects on a client and, if necessary, unequips associated items.

Return void

Parameters

- `pid`: The player ID.

- `skillId`: The skill ID.

void **SetSkillProgress**(unsigned short *pid*, unsigned short *skillId*, double *value*)
Set the progress the player has made towards increasing a certain skill by 1.

Return void

Parameters

- `pid`: The player ID.
- `skillId`: The skill ID.
- `value`: The progress value.

void **SetSkillIncrease**(unsigned short *pid*, unsigned int *attributeId*, int *value*)
Set the bonus applied to a certain attribute at the next level up as a result of associated skill increases.

Although confusing, the term "skill increase" for this is taken from OpenMW itself.

Return void

Parameters

- `pid`: The player ID.
- `skillId`: The attribute ID.
- `value`: The increase in the attribute caused by skills.

void **SetBounty**(unsigned short *pid*, int *value*)
Set the bounty of a player.

Return void

Parameters

- `pid`: The player ID.
- `value`: The new bounty.

void **SetCharGenStage**(unsigned short *pid*, int *currentStage*, int *endStage*)
Set the current and ending stages of character generation for a player.

This is used to repeat part of character generation or to only go through part of it.

Return void

Parameters

- `pid`: The player ID.
- `currentStage`: The new current stage.
- `endStage`: The new ending stage.

void **SendBaseInfo**(unsigned short *pid*)
Send a PlayerBaseInfo packet with a player's name, race, head mesh, hairstyle mesh, birthsign and stat reset state.

It is always sent to all players.

Return void

Parameters

- `pid`: The player ID.

void **`SendStatsDynamic`**(unsigned short *pid*)
Send a PlayerStatsDynamic packet with a player's dynamic stats (health, magicka and fatigue).

It is always sent to all players.

Return void

Parameters

- `pid`: The player ID.

void **`SendAttributes`**(unsigned short *pid*)
Send a PlayerAttribute packet with a player's attributes and bonuses to those attributes at the next level up (the latter being called "skill increases" as in OpenMW).

It is always sent to all players.

Return void

Parameters

- `pid`: The player ID.

void **`SendSkills`**(unsigned short *pid*)
Send a PlayerSkill packet with a player's skills.

It is always sent to all players.

Return void

Parameters

- `pid`: The player ID.

void **`SendLevel`**(unsigned short *pid*)
Send a PlayerLevel packet with a player's character level and progress towards the next level up.

It is always sent to all players.

Return void

Parameters

- `pid`: The player ID.

void **`SendBounty`**(unsigned short *pid*)
Send a PlayerBounty packet with a player's bounty.

It is always sent to all players.

Return void

Parameters

- `pid`: The player ID.

1.21 Worldstate functions

class WorldstateFunctions

Public Static Functions

void **ReadReceivedWorldstate**()
Use the last worldstate received by the server as the one being read.

Return void

void **CopyReceivedWorldstateToStore**()
Take the contents of the read-only worldstate last received by the server from a player and move its contents to the stored worldstate that can be sent by the server.

Return void

void **ClearMapChanges**()
Clear the map changes for the write-only worldstate.

This is used to initialize the sending of new WorldMap packets.

Return void

unsigned int **GetMapChangesSize**()
Get the number of indexes in the read worldstate's map changes.

Return The number of indexes.

const char ***GetWeatherRegion**()
Get the weather region in the read worldstate.

Return The weather region.

int **GetWeatherCurrent**()
Get the current weather in the read worldstate.

Return The current weather.

int **GetWeatherNext**()
Get the next weather in the read worldstate.

Return The next weather.

int **GetWeatherQueued**()
Get the queued weather in the read worldstate.

Return The queued weather.

double **GetWeatherTransitionFactor**()
Get the transition factor of the weather in the read worldstate.

Return The transition factor of the weather.

int **GetMapTileCellX**(unsigned int *index*)
Get the X coordinate of the cell corresponding to the map tile at a certain index in the read worldstate's map tiles.

Return The X coordinate of the cell.

Parameters

- `index`: The index of the map tile.

int **GetMapTileCellY**(unsigned int *index*)
Get the Y coordinate of the cell corresponding to the map tile at a certain index in the read worldstate's map tiles.

Return The Y coordinate of the cell.

Parameters

- `index`: The index of the map tile.

void **SetAuthorityRegion**(**const** char **authorityRegion*)
Set the region affected by the next WorldRegionAuthority packet sent.

Return void

Parameters

- `region`: The region.

void **SetWeatherRegion**(**const** char **region*)
Set the weather region in the write-only worldstate stored on the server.

Return void

Parameters

- `region`: The region.

void **SetWeatherForceState**(bool *forceState*)
Set the weather forcing state in the write-only worldstate stored on the server.

Players who receive a packet with forced weather will switch to that weather immediately.

Return void

Parameters

- `forceState`: The weather forcing state.

void **SetWeatherCurrent**(int *currentWeather*)
Set the current weather in the write-only worldstate stored on the server.

Return void

Parameters

- `currentWeather`: The current weather.

void **SetWeatherNext**(int *nextWeather*)
Set the next weather in the write-only worldstate stored on the server.

Return void

Parameters

- `nextWeather`: The next weather.

void **SetWeatherQueued**(int *queuedWeather*)
Set the queued weather in the write-only worldstate stored on the server.

Return void

Parameters

- `queuedWeather`: The queued weather.

void **SetWeatherTransitionFactor**(double *transitionFactor*)
Set the transition factor for the weather in the write-only worldstate stored on the server.

Return void

Parameters

- `transitionFactor`: The transition factor.

void **SetHour**(double *hour*)
Set the world's hour in the write-only worldstate stored on the server.

Return void

Parameters

- `hour`: The hour.

void **SetDay**(int *day*)
Set the world's day in the write-only worldstate stored on the server.

Return void

Parameters

- `day`: The day.

void **SetMonth**(int *month*)
Set the world's month in the write-only worldstate stored on the server.

Return void

Parameters

- `month`: The month.

void **SetYear**(int *year*)
Set the world's year in the write-only worldstate stored on the server.

Return void

Parameters

- `year`: The year.

void **`SetDaysPassed`**(int *daysPassed*)
Set the world's days passed in the write-only worldstate stored on the server.

Return void

Parameters

- `daysPassed`: The days passed.

void **`SetTimeScale`**(double *timeScale*)
Set the world's time scale in the write-only worldstate stored on the server.

Return void

Parameters

- `pid`: The player ID.
- `timeScale`: The time scale.

void **`SetPlayerCollisionState`**(bool *state*)
Set the collision state for other players in the write-only worldstate stored on the server.

Return void

Parameters

- `state`: The collision state.

void **`SetActorCollisionState`**(bool *state*)
Set the collision state for actors in the write-only worldstate stored on the server.

Return void

Parameters

- `state`: The collision state.

void **`SetPlacedObjectCollisionState`**(bool *state*)
Set the collision state for placed objects in the write-only worldstate stored on the server.

Return void

Parameters

- `state`: The collision state.

void **`UseActorCollisionForPlacedObjects`**(bool *useActorCollision*)
Whether placed objects with collision turned on should use actor collision, i.e. whether they should be slippery and prevent players from standing on them.

Return void

Parameters

- `useActorCollision`: Whether to use actor collision.

void **AddEnforcedCollisionRefId**(**const** char **refId*)
Add a refId to the list of refIds for which collision should be enforced irrespective of other settings.

Return void

Parameters

- `refId`: The refId.

void **ClearEnforcedCollisionRefIds**()
Clear the list of refIdsd for which collision should be enforced irrespective of other settings.

Return void

void **SaveMapTileImageFile**(unsigned int *index*, **const** char **filePath*)
Save the .png image data of the map tile at a certain index in the read worldstate's map changes.

Return void

Parameters

- `index`: The index of the map tile.
- `filePath`: The file path of the resulting file.

void **LoadMapTileImageFile**(int *cellX*, int *cellY*, **const** char **filePath*)
Load a .png file as the image data for a map tile and add it to the write-only worldstate stored on the server.

Return void

Parameters

- `cellX`: The X coordinate of the cell corresponding to the map tile.
- `cellY`: The Y coordinate of the cell corresponding to the map tile.
- `filePath`: The file path of the loaded file.

void **SendWorldRegionAuthority**(unsigned short *pid*)
Send a WorldRegionAuthority packet establishing a certain player as the only one who should process certain region-specific events (such as weather changes).

It is always sent to all players.

Return void

Parameters

- `pid`: The player ID attached to the packet.

void **SendWorldMap**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a WorldMap packet with the current set of map changes in the write-only worldstate.

Return void

Parameters

- `pid`: The player ID attached to the packet.
- `broadcast`: Whether this packet should be sent only to the attached player or to all players on the server.

void **`SendWorldTime`**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a WorldTime packet with the current time and time scale in the write-only worldstate.

Return void

Parameters

- `pid`: The player ID attached to the packet.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **`SendWorldWeather`**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a WorldWeather packet with the current weather in the write-only worldstate.

Return void

Parameters

- `pid`: The player ID attached to the packet.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

void **`SendWorldCollisionOverride`**(unsigned short *pid*, bool *sendToOtherPlayers*, bool *skipAttachedPlayer*)
Send a WorldCollisionOverride packet with the current collision overrides in the write-only worldstate.

Return void

Parameters

- `pid`: The player ID attached to the packet.
- `sendToOtherPlayers`: Whether this packet should be sent to players other than the player attached to the packet (false by default).
- `skipAttachedPlayer`: Whether the packet should skip being sent to the player attached to the packet (false by default).

Index

A

B

C

D

F

G

I

M

O

P

Q

R

S

W