OpenMW Documentation

Release (0, 40, 0)

Bret Curtis

January 08, 2017

Contents

1 Components **1**
1.1 OpenMW Source Documentation 1
1.2 OpenMW CS user manual 32

2 Indices and tables **43**

CHAPTER 1

Components

1.1 OpenMW Source Documentation

1.1.1 ./mwbase

namespace `MWBase`

class `DialogueManager`

#include <dialoguemanager.hpp> Interface for dialogue manager (implemented in MWDialogue)

Public Functions

`DialogueManager`()

virtual void `clear`() = 0

virtual `~DialogueManager`()

virtual bool `isInChoice`() **const** = 0

virtual void `startDialogue`(**const** *MWWorld*::Ptr &*actor*) = 0

virtual void `addTopic`(**const** std::string &*topic*) = 0

virtual void `askQuestion`(**const** std::string &*question*, int *choice*) = 0

virtual void `goodbye`() = 0

virtual void `say`(**const** *MWWorld*::Ptr &*actor*, **const** std::string &*topic*) **const** = 0

virtual void `keywordSelected`(**const** std::string &*keyword*) = 0

virtual void `goodbyeSelected`() = 0

virtual void `questionAnswered`(int *answer*) = 0

virtual bool `checkServiceRefused`() = 0

virtual void `persuade`(int *type*) = 0

virtual int `getTemporaryDispositionChange`() **const** = 0

virtual void `applyDispositionChange`(int *delta*) = 0

Note

This change is temporary and gets discarded when dialogue ends.

virtual int `countSavedGameRecords`() **const** = 0

virtual void **write**(ESM::ESMWriter &*writer*, Loading::Listener &*progress*) **const** = 0

virtual void **readRecord**(ESM::ESMReader &*reader*, uint32_t *type*) = 0

virtual void **modFactionReaction**(**const** std::string &*faction1*, **const** std::string &*faction2*, int *diff*) = 0
Changes faction1's opinion of faction2 by *diff*.

virtual void **setFactionReaction**(**const** std::string &*faction1*, **const** std::string &*faction2*, int *absolute*) = 0

virtual int **getFactionReaction**(**const** std::string &*faction1*, **const** std::string &*faction2*) **const** = 0
Return faction1's opinion of faction2

virtual void **clearInfoActor**(**const** *MWWorld*::Ptr &*actor*) **const** = 0
Removes the last added topic response for the given actor from the journal.

Private Functions

DialogueManager(**const** *DialogueManager*&)
not implemented

DialogueManager &**operator=**(**const** *DialogueManager*&)
not implemented

namespace MWBase

class Environment
#include <environment.hpp> Central hub for mw-subsystems.

This class allows each mw-subsystem to access any others subsystem's top-level manager class.

Attention *Environment* takes ownership of the manager class instances it is handed over in the set* functions.

Public Functions

Environment()

~Environment()

void **setWorld**(*World* *world)

void **setSoundManager**(*SoundManager* *soundManager)

void **setScriptManager**(*MWBase*::*ScriptManager* *scriptManager)

void **setWindowManager**(*WindowManager* *windowManager)

void **setMechanicsManager**(*MechanicsManager* *mechanicsManager)

void **setDialogueManager**(*DialogueManager* *dialogueManager)

void **setJournal**(*Journal* *journal)

void **setInputManager**(*InputManager* *inputManager)

void **setStateManager**(*StateManager* *stateManager)

void **setFrameDuration**(float *duration*)
Set length of current frame in seconds.

World ***getWorld**() **const**

SoundManager ***getSoundManager**() **const**

ScriptManager ***getScriptManager**() **const**

WindowManager ***getWindowManager**() **const**

MechanicsManager ***getMechanicsManager**() **const**

DialogueManager ***getDialogueManager**() **const**

Journal ***getJournal**() **const**

InputManager ***getInputManager**() **const**

StateManager ***getStateManager**() **const**

float **getFrameDuration**() **const**

void **cleanup**()
Delete all mw*-subsystems.

Public Static Functions

static const *Environment* &**get**()
Return instance of this class.

Private Functions

Environment(**const** *Environment*&)
not implemented

Environment &**operator=**(**const** *Environment*&)
not implemented

Private Members

World ***mWorld**

SoundManager ***mSoundManager**

ScriptManager ***mScriptManager**

WindowManager ***mWindowManager**

MechanicsManager ***mMechanicsManager**

DialogueManager ***mDialogueManager**

Journal ***mJournal**

InputManager ***mInputManager**

StateManager ***mStateManager**

float **mFrameDuration**

Private Static Attributes

Environment ***sThis**

namespace MWBase

class InputManager

#include <inputmanager.hpp> Interface for input manager (implemented in MWInput)

Public Functions

InputManager()

virtual void **clear**() = 0
Clear all savegame-specific data.

virtual ~InputManager()

virtual bool **isWindowVisible**() = 0

virtual void **update**(float *dt*, bool *disableControls*, bool *disableEvents* = false) = 0

virtual void **changeInputMode**(bool *guiMode*) = 0

virtual void **processChangedSettings**(**const** std::set<std::pair<std::string, std::string>> &*changed*) = 0

virtual void **setDragDrop**(bool *dragDrop*) = 0

virtual void **toggleControlSwitch**(**const** std::string &*sw*, bool *value*) = 0

virtual bool **getControlSwitch**(**const** std::string &*sw*) = 0

virtual std::string **getActionDescription**(int *action*) = 0

virtual std::string **getActionKeyBindingName**(int *action*) = 0

virtual std::string **getActionControllerBindingName**(int *action*) = 0

virtual std::string **sdlControllerAxisToString**(int *axis*) = 0

virtual std::string **sdlControllerButtonToString**(int *button*) = 0

virtual std::vector<int> **getActionKeySorting**() = 0
Actions available for binding to keyboard buttons.

virtual std::vector<int> **getActionControllerSorting**() = 0
Actions available for binding to controller buttons.

virtual int **getNumActions**() = 0

virtual void **enableDetectingBindingMode**(int *action*, bool *keyboard*) = 0
If keyboard is true, only pay attention to keyboard events. If false, only pay attention to controller events (excluding esc)

virtual void **resetToDefaultKeyBindings**() = 0

virtual void **resetToDefaultControllerBindings**() = 0

virtual bool **joystickLastUsed**() = 0
Returns if the last used input device was a joystick or a keyboard
Return true if joystick, false otherwise

Private Functions

InputManager(**const** *InputManager*&)
not implemented

InputManager &**operator=**(**const** *InputManager*&)
not implemented

namespace MWBase

class Journal
#include <journal.hpp> Interface for the player's journal (implemented in MWDialogue)

Public Types

typedef std::deque<MWDialogue::StampedJournalEntry> **TEntryContainer**

typedef *TEntryContainer*::const_iterator **TEntryIter**

typedef std::map<std::string, MWDialogue::Quest> **TQuestContainer**

typedef *TQuestContainer*::const_iterator **TQuestIter**

typedef std::map<std::string, MWDialogue::Topic> **TTopicContainer**

typedef *TTopicContainer*::const_iterator **TTopicIter**

Public Functions

Journal()

virtual void **clear**() = 0

virtual ~Journal()

virtual void **addEntry**(**const** std::string &*id*, int *index*, **const** *MWWorld*::Ptr &*actor*) = 0
Add a journal entry.
Parameters
- `actor`: Used as context for replacing of escape sequences (name, etc).

virtual void **setJournalIndex**(**const** std::string &*id*, int *index*) = 0
Set the journal index without adding an entry.

virtual int **getJournalIndex**(**const** std::string &*id*) **const** = 0
Get the journal index.

virtual void **addTopic**(**const** std::string &*topicId*, **const** std::string &*infoId*, **const** *MWWorld*::Ptr &*actor*) = 0

virtual void **removeLastAddedTopicResponse**(**const** std::string &*topicId*, **const** std::string &*actorName*) = 0 **Note** topicId must be lowercase Removes the last topic response added for the given topicId and actor name.

topicId must be lowercase

virtual *TEntryIter* **begin**() **const** = 0
Iterator pointing to the begin of the main journal.

Note Iterators to main journal entries will never become invalid.

virtual *TEntryIter* **end**() **const** = 0
Iterator pointing past the end of the main journal.

virtual *TQuestIter* **questBegin**() **const** = 0
Iterator pointing to the first quest (sorted by topic ID)

virtual *TQuestIter* **questEnd**() **const** = 0
Iterator pointing past the last quest.

virtual *TTopicIter* **topicBegin**() **const** = 0
Iterator pointing to the first topic (sorted by topic ID)

Note The topic ID is identical with the user-visible topic string.

virtual *TTopicIter* **topicEnd**() **const** = 0
Iterator pointing past the last topic.

virtual int **countSavedGameRecords**() **const** = 0

virtual void **write**(ESM::ESMWriter &*writer*, Loading::Listener &*progress*) **const** = 0

virtual void **readRecord**(ESM::ESMReader &*reader*, uint32_t *type*) = 0

Private Functions

Journal(**const** *Journal*&)
not implemented

Journal &**operator=**(**const** *Journal*&)
not implemented

namespace MWBase

class MechanicsManager
#include <mechanicsmanager.hpp> Interface for game mechanics manager (implemented in MWMechanics)

Public Types

enum OffenseType
Values:

OT_Theft

OT_Assault

OT_Murder

OT_Trespassing

OT_SleepingInOwnedBed

OT_Pickpocket

enum PersuasionType
Values:

PT_Admire

PT_Intimidate

PT_Taunt

PT_Bribe10

PT_Bribe100

PT_Bribe1000

Public Functions

MechanicsManager()

virtual ~MechanicsManager()

virtual void **add**(**const** *MWWorld*::Ptr &*ptr*) = 0
Register an object for management.

virtual void **remove**(**const** *MWWorld*::Ptr &*ptr*) = 0
Deregister an object for management.

virtual void **updateCell**(**const** *MWWorld*::Ptr &*old*, **const** *MWWorld*::Ptr &*ptr*) = 0
Moves an object to a new cell.

virtual void **drop**(**const** *MWWorld*::CellStore **cellStore*) = 0
Deregister all objects in the given cell.

virtual void **watchActor**(**const** *MWWorld*::Ptr &*ptr*) = 0
On each update look for changes in a previously registered actor and update the GUI accordingly.

virtual void **update**(float *duration*, bool *paused*) = 0
Update objects

Parameters
- `paused`: In game type does not currently advance (this usually means some GUI component is up).

virtual void **advanceTime**(float *duration*) = 0

virtual void **setPlayerName**(**const** std::string &*name*) = 0
Set player name.

virtual void **setPlayerRace**(**const** std::string &*id*, bool *male*, **const** std::string &*head*, **const** std::string &*hair*) = 0
Set player race.

virtual void **setPlayerBirthsign**(**const** std::string &*id*) = 0
Set player birthsign.

virtual void **setPlayerClass**(**const** std::string &*id*) = 0
Set player class to stock class.

virtual void **setPlayerClass**(**const** ESM::Class &*class_*) = 0
Set player class to custom class.

virtual void **rest**(bool *sleep*) = 0
If the player is sleeping or waiting, this should be called every hour.
Parameters
- `sleep`: is the player sleeping or waiting?

virtual int **getHoursToRest**() **const** = 0
Calculate how many hours the player needs to rest in order to be fully healed.

virtual int **getBarterOffer**(**const** *MWWorld*::Ptr &*ptr*, int *basePrice*, bool *buying*) = 0
This is used by every service to determine the price of objects given the trading skills of the player and NPC.

virtual int **getDerivedDisposition**(**const** *MWWorld*::Ptr &*ptr*, bool *addTemporaryDispositionChange* = true) = 0

Calculate the diposition of an NPC toward the player.

virtual int **countDeaths**(**const** std::string &*id*) **const** = 0

Return the number of deaths for actors with the given ID.

virtual bool **awarenessCheck**(**const** *MWWorld*::Ptr &*ptr*, **const** *MWWorld*::Ptr &*observer*) = 0

Check if *observer* is potentially aware of *ptr*. Does not do a line of sight check!

virtual void **startCombat**(**const** *MWWorld*::Ptr &*ptr*, **const** *MWWorld*::Ptr &*target*) = 0

Makes *ptr* fight *target*. Also shouts a combat taunt.

virtual bool **commitCrime**(**const** *MWWorld*::Ptr &*ptr*, **const** *MWWorld*::Ptr &*victim*, *OffenseType* *type*, int *arg* = 0, bool *victimAware* = false) = 0

Note victim may be empty

Return was the crime seen?

Parameters

- `arg`: Depends on *type*, e.g. for Theft, the value of the item that was stolen.
- `victimAware`: Is the victim already aware of the crime? If this parameter is false, it will be determined by a line-of-sight and awareness check.

virtual bool **actorAttacked**(**const** *MWWorld*::Ptr &*victim*, **const** *MWWorld*::Ptr &*attacker*) = 0

Return false if the attack was considered a "friendly hit" and forgiven

virtual void **actorKilled**(**const** *MWWorld*::Ptr &*victim*, **const** *MWWorld*::Ptr &*attacker*) = 0

Notify that actor was killed, add a murder bounty if applicable

Note No-op for non-player attackers

virtual void **itemTaken**(**const** *MWWorld*::Ptr &*ptr*, **const** *MWWorld*::Ptr &*item*, **const** *MWWorld*::Ptr &*container*, int *count*) = 0

Utility to check if taking this item is illegal and calling commitCrime if so

Parameters

- `container`: The container the item is in; may be empty for an item in the world

virtual void **objectOpened**(**const** *MWWorld*::Ptr &*ptr*, **const** *MWWorld*::Ptr &*item*) = 0

Utility to check if opening (i.e. unlocking) this object is illegal and calling commitCrime if so.

virtual bool **sleepInBed**(**const** *MWWorld*::Ptr &*ptr*, **const** *MWWorld*::Ptr &*bed*) = 0

Attempt sleeping in a bed. If this is illegal, call commitCrime.

Return was it illegal, and someone saw you doing it?

virtual void **getPersuasionDispositionChange**(**const** *MWWorld*::Ptr &*npc*, *PersuasionType* *type*, bool &*success*, float &*tempChange*, float &*permChange*) = 0

Perform a persuasion action on NPC.

virtual void **forceStateUpdate**(**const** *MWWorld*::Ptr &*ptr*) = 0

Forces an object to refresh its animation state.

virtual bool **playAnimationGroup**(**const** *MWWorld*::Ptr &*ptr*, **const** std::string &*groupName*, int *mode*, int *number* = 1) = 0

Run animation for a MW-reference. Calls to this function for references that are currently not in the scene should be ignored.

Return Success or error

Parameters

- `mode`: 0 normal, 1 immediate start, 2 immediate loop
- `count`: How many times the animation should be run

virtual void **skipAnimation** (**const** *MWWorld*::Ptr &*ptr*) = 0
Skip the animation for the given MW-reference for one frame. Calls to this function for references that are currently not in the scene should be ignored.

virtual bool **checkAnimationPlaying** (**const** *MWWorld*::Ptr &*ptr*, **const** std::string &*groupName*) = 0

virtual void **updateMagicEffects** (**const** *MWWorld*::Ptr &*ptr*) = 0
Update magic effects for an actor. Usually done automatically once per frame, but if we're currently paused we may want to do it manually (after equipping permanent enchantment)

virtual bool **toggleAI** () = 0

virtual bool **isAIActive** () = 0

virtual void **getObjectsInRange** (**const** osg::Vec3f &*position*, float *radius*, std::vector<*MWWorld*::Ptr> &*objects*) = 0

virtual void **getActorsInRange** (**const** osg::Vec3f &*position*, float *radius*, std::vector<*MWWorld*::Ptr> &*objects*) = 0

virtual std::list<*MWWorld*::Ptr> **getActorsSidingWith** (**const** *MWWorld*::Ptr &*actor*) = 0
Returns the list of actors which are siding with the given actor in fights.

ie AiFollow or AiEscort is active and the target is the actor

virtual std::list<*MWWorld*::Ptr> **getActorsFollowing** (**const** *MWWorld*::Ptr &*actor*) = 0

virtual std::list<int> **getActorsFollowingIndices** (**const** *MWWorld*::Ptr &*actor*) = 0

virtual std::list<*MWWorld*::Ptr> **getActorsFighting** (**const** *MWWorld*::Ptr &*actor*) = 0
Returns a list of actors who are fighting the given actor within the fAlarmDistance.

ie AiCombat is active and the target is the actor

virtual std::list<*MWWorld*::Ptr> **getEnemiesNearby** (**const** *MWWorld*::Ptr &*actor*) = 0

virtual void **playerLoaded** () = 0

virtual int **countSavedGameRecords** () **const** = 0

virtual void **write** (ESM::ESMWriter &*writer*, Loading::Listener &*listener*) **const** = 0

virtual void **readRecord** (ESM::ESMReader &*reader*, uint32_t *type*) = 0

virtual void **clear** () = 0

virtual bool **isAggressive** (**const** *MWWorld*::Ptr &*ptr*, **const** *MWWorld*::Ptr &*target*) = 0

virtual void **keepPlayerAlive** () = 0
Resurrects the player if necessary.

virtual bool **isReadyToBlock** (**const** *MWWorld*::Ptr &*ptr*) **const** = 0

virtual void **confiscateStolenItems** (**const** *MWWorld*::Ptr &*player*, **const** *MWWorld*::Ptr &*targetContainer*) = 0

virtual std::vector<std::pair<std::string, int>> **getStolenItemOwners** (**const** std::string &*itemid*) = 0
List the owners that the player has stolen this item from (the owner can be an NPC or a faction). <Owner, item count>

virtual bool **isItemStolenFrom** (**const** std::string &*itemid*, **const** std::string &*ownerid*) = 0
Has the player stolen this item from the given owner?

virtual bool **isAllowedToUse** (**const** *MWWorld*::Ptr &*ptr*, **const** *MWWorld*::CellRef &*cellref*, *MWWorld*::Ptr &*victim*) = 0

virtual void **setWerewolf**(**const** *MWWorld*::Ptr &*actor*, bool *werewolf*) = 0
Turn actor into werewolf or normal form.

virtual void **applyWerewolfAcrobatics**(**const** *MWWorld*::Ptr &*actor*) = 0
Sets the NPC's Acrobatics skill to match the fWerewolfAcrobatics GMST. It only applies to the current form the NPC is in.

virtual void **cleanupSummonedCreature**(**const** *MWWorld*::Ptr &*caster*, int *creatureActorId*) = 0

Private Functions

MechanicsManager(**const** *MechanicsManager*&)
not implemented

MechanicsManager &**operator=**(**const** *MechanicsManager*&)
not implemented

namespace MWBase

class ScriptManager
#include <*scriptmanager.hpp*> Interface for script manager (implemented in MWScript)

Public Functions

ScriptManager()

virtual ~ScriptManager()

virtual void **run**(**const** std::string &*name*, Interpreter::Context &*interpreterContext*) = 0
Run the script with the given name (compile first, if not compiled yet)

virtual bool **compile**(**const** std::string &*name*) = 0
Compile script with the given namen
Return Success?

virtual std::pair<int, int> **compileAll**() = 0
Compile all scripts
Return count, success

virtual const Compiler::Locals &**getLocals**(**const** std::string &*name*) = 0
Return locals for script *name*.

virtual MWScript::GlobalScripts &**getGlobalScripts**() = 0

Private Functions

ScriptManager(**const** *ScriptManager*&)
not implemented

ScriptManager &**operator=**(**const** *ScriptManager*&)
not implemented

namespace MWSound

Typedefs

typedef boost::shared_ptr<Sound_Decoder> **DecoderPtr**

namespace MWBase

Typedefs

typedef boost::shared_ptr<*MWSound*::Sound> **SoundPtr**

typedef boost::shared_ptr<*MWSound*::Stream> **SoundStreamPtr**

class SoundManager

#include <soundmanager.hpp> Interface for sound manager (implemented in MWSound)

Public Types

enum PlayMode

Values:

Play_Normal = 0

Play_Loop = 1<<0

Play_NoEnv = 1<<1

Play_RemoveAtDistance = 1<<2

Play_NoPlayerLocal = 1<<3

Play_LoopNoEnv = Play_Loop | Play_NoEnv

Play_LoopRemoveAtDistance = Play_Loop | Play_RemoveAtDistance

enum PlayType

Values:

Play_TypeSfx = 1<<4

Play_TypeVoice = 1<<5

Play_TypeFoot = 1<<6

Play_TypeMusic = 1<<7

Play_TypeMovie = 1<<8

Play_TypeMask = Play_TypeSfx|Play_TypeVoice|Play_TypeFoot|Play_TypeMusic|Play_TypeMovie

Public Functions

SoundManager()

virtual ~SoundManager()

virtual void **processChangedSettings**(**const** std::set<std::pair<std::string, std::string>> &*settings*) = 0

virtual void **stopMusic**() = 0

Stops music if it's playing.

virtual void **streamMusic**(**const** std::string &*filename*) = 0
Play a soundifle
Parameters
- `filename`: name of a sound file in "Music/" in the data directory.

virtual void **startRandomTitle**() = 0
Starts a random track from the current playlist.

virtual bool **isMusicPlaying**() = 0
Returns true if music is playing.

virtual void **playPlaylist**(**const** std::string &*playlist*) = 0
Start playing music from the selected folder
Parameters
- `name`: of the folder that contains the playlist

virtual void **say**(**const** *MWWorld*::ConstPtr &*reference*, **const** std::string &*filename*) = 0
Make an actor say some text.
Parameters
- `filename`: name of a sound file in "Sound/" in the data directory.

virtual void **say**(**const** std::string &*filename*) = 0
Say some text, without an actor ref
Parameters
- `filename`: name of a sound file in "Sound/" in the data directory.

virtual bool **sayDone**(**const** *MWWorld*::ConstPtr &*reference* = MWWorld::ConstPtr()) **const** = 0
Is actor not speaking?

virtual void **stopSay**(**const** *MWWorld*::ConstPtr &*reference* = MWWorld::ConstPtr()) = 0
Stop an actor speaking.

virtual float **getSaySoundLoudness**(**const** *MWWorld*::ConstPtr &*reference*) **const** = 0
Check the currently playing say sound for this actor and get an average loudness value (scale [0,1]) at the current time position. If the actor is not saying anything, returns 0.

virtual *SoundStreamPtr* **playTrack**(**const** *MWSound*::*DecoderPtr* &*decoder*, *PlayType type*) = 0
Play a 2D audio track, using a custom decoder.

virtual void **stopTrack**(*SoundStreamPtr stream*) = 0
Stop the given audio track from playing.

virtual double **getTrackTimeDelay**(*SoundStreamPtr stream*) = 0
Retives the time delay, in seconds, of the audio track (must be a sound returned by *playTrack*). Only intended to be called by the track decoder's read method.

virtual *SoundPtr* **playSound**(**const** std::string &*soundId*, float *volume*, float *pitch*, *PlayType type* = Play_TypeSfx, *PlayMode mode* = Play_Normal, float *offset* = 0) = 0
Play a sound, independently of 3D-position
Parameters
- `offset`: Number of seconds into the sound to start playback.

virtual *MWBase*::*SoundPtr* **playSound3D**(**const** *MWWorld*::ConstPtr &*reference*, **const** std::string &*soundId*, float *volume*, float *pitch*, *PlayType type* = Play_TypeSfx, *PlayMode mode* = Play_Normal, float *offset* = 0) = 0
Play a 3D sound attached to an MWWorld::Ptr. Will be updated automatically with the Ptr's position, unless Play_NoTrack is specified.
Parameters
- `offset`: Number of seconds into the sound to start playback.

virtual *MWBase*::*SoundPtr* **playSound3D** (**const** osg::Vec3f &*initialPos*, **const** std::string &*soundId*, float *volume*, float *pitch*, *PlayType type* = Play_TypeSfx, *PlayMode mode* = Play_Normal, float *offset* = 0) = 0

Play a 3D sound at *initialPos*. If the sound should be moving, it must be updated using Sound::setPosition.

virtual void **stopSound** (*SoundPtr sound*) = 0

Stop the given sound from playing.

virtual void **stopSound3D** (**const** *MWWorld*::ConstPtr &*reference*, **const** std::string &*soundId*) = 0

Stop the given object from playing the given sound,.

virtual void **stopSound3D** (**const** *MWWorld*::ConstPtr &*reference*) = 0

Stop the given object from playing all sounds.

virtual void **stopSound** (**const** *MWWorld*::CellStore **cell*) = 0

Stop all sounds for the given cell.

virtual void **stopSound** (**const** std::string &*soundId*) = 0

Stop a non-3d looping sound.

virtual void **fadeOutSound3D** (**const** *MWWorld*::ConstPtr &*reference*, **const** std::string &*soundId*, float *duration*) = 0

Fade out given sound (that is already playing) of given object

Parameters

- `reference`: Reference to object, whose sound is faded out
- `soundId`: ID of the sound to fade out.
- `duration`: Time until volume reaches 0.

virtual bool **getSoundPlaying** (**const** *MWWorld*::ConstPtr &*reference*, **const** std::string &*soundId*) **const** = 0

Is the given sound currently playing on the given object? If you want to check if sound played with playSound is playing, use empty Ptr

virtual void **pauseSounds** (int *types* = Play_TypeMask) = 0

Pauses all currently playing sounds, including music.

virtual void **resumeSounds** (int *types* = Play_TypeMask) = 0

Resumes all previously paused sounds.

virtual void **update** (float *duration*) = 0

virtual void **setListenerPosDir** (**const** osg::Vec3f &*pos*, **const** osg::Vec3f &*dir*, **const** osg::Vec3f &*up*, bool *underwater*) = 0

virtual void **updatePtr** (**const** *MWWorld*::ConstPtr &*old*, **const** *MWWorld*::ConstPtr &*updated*) = 0

virtual void **clear** () = 0

Private Functions

SoundManager (**const** *SoundManager*&)

not implemented

SoundManager &**operator=** (**const** *SoundManager*&)

not implemented

namespace MWBase

class `StateManager`

#include <*statemanager.hpp*> Interface for game state manager (implemented in MWState)

Public Types

enum `State`

Values:

`State_NoGame`

`State_Ended`

`State_Running`

typedef std::list<MWState::Character>::const_iterator `CharacterIterator`

Public Functions

`StateManager`()

virtual `~StateManager`()

virtual void `requestQuit`() = 0

virtual bool `hasQuitRequest`() **const** = 0

virtual void `askLoadRecent`() = 0

virtual *State* `getState`() **const** = 0

virtual void `newGame`(bool *bypass* = false) = 0

Start a new game.

Parameters

- `bypass`: Skip new game mechanics.

virtual void `endGame`() = 0

virtual void `deleteGame`(**const** MWState::Character ***character*, **const** MWState::Slot ***slot*) = 0

virtual void `saveGame`(**const** std::string &*description*, **const** MWState::Slot ***slot* = 0) = 0

Write a saved game to *slot* or create a new slot if *slot* == 0.

Note Slot must belong to the current character.

virtual void `loadGame`(**const** std::string &*filepath*) = 0

Load a saved game directly from the given file path. This will search the CharacterManager for a Character containing this save file, and set this Character current if one was found. Otherwise, a new Character will be created.

virtual void `loadGame`(**const** MWState::Character ***character*, **const** std::string &*filepath*) = 0

Load a saved game file belonging to the given character.

virtual void `quickSave`(std::string = "Quicksave") = 0

Simple saver, writes over the file if already existing.

Used for quick save and autosave

virtual void `quickLoad`() = 0

Simple loader, loads the last saved file.

Used for quickload

virtual MWState::Character ***getCurrentCharacter**() = 0

Note

May return null.

virtual *CharacterIterator* **characterBegin**() = 0

Any call to SaveGame and getCurrentCharacter can invalidate the returned iterator.

virtual *CharacterIterator* **characterEnd**() = 0

virtual void **update**(float *duration*) = 0

Private Functions

StateManager(**const** *StateManager*&)

not implemented

StateManager &**operator=**(**const** *StateManager*&)

not implemented

namespace MWGui

Enums

enum ShowInDialogueMode

Values:

ShowInDialogueMode_IfPossible

ShowInDialogueMode_Only

ShowInDialogueMode_Never

namespace MWBase

class WindowManager

#include <*windowmanager.hpp*> Interface for widnow manager (implemented in MWGui)

Public Types

typedef std::vector<int> **SkillList**

Public Functions

WindowManager()

virtual ~WindowManager()

virtual void **update**() = 0

Should be called each frame to update windows/gui elements. This could mean updating sizes of gui elements or opening new dialogs.

virtual void **playVideo**(**const** std::string &*name*, bool *allowSkipping*) = 0

Note

This method will block until the video finishes playing (and will continually update the window while doing so)

virtual void **setNewGame**(bool *newgame*) = 0

virtual void **pushGuiMode**(*MWGui*::GuiMode *mode*) = 0

virtual void **popGuiMode** () = 0

virtual void **removeGuiMode** (*MWGui*::GuiMode *mode*) = 0
can be anywhere in the stack

virtual void **goToJail** (int *days*) = 0

virtual void **updatePlayer** () = 0

virtual *MWGui*::GuiMode **getMode** () **const** = 0

virtual bool **containsMode** (*MWGui*::GuiMode) **const** = 0

virtual bool **isGuiMode** () **const** = 0

virtual bool **isConsoleMode** () **const** = 0

virtual void **toggleVisible** (*MWGui*::GuiWindow *wnd*) = 0

virtual void **forceHide** (*MWGui*::GuiWindow *wnd*) = 0

virtual void **unsetForceHide** (*MWGui*::GuiWindow *wnd*) = 0

virtual void **disallowAll** () = 0
Disallow all inventory mode windows.

virtual void **allow** (*MWGui*::GuiWindow *wnd*) = 0
Allow one or more windows.

virtual bool **isAllowed** (*MWGui*::GuiWindow *wnd*) **const** = 0

virtual *MWGui*::DialogueWindow ***getDialogueWindow** () = 0

virtual *MWGui*::InventoryWindow ***getInventoryWindow** () = 0

virtual *MWGui*::CountDialog ***getCountDialog** () = 0

virtual *MWGui*::ConfirmationDialog ***getConfirmationDialog** () = 0

virtual *MWGui*::TradeWindow ***getTradeWindow** () = 0

virtual void **useItem** (**const** *MWWorld*::Ptr &*item*) = 0
Make the player use an item, while updating GUI state accordingly.

virtual void **updateSpellWindow** () = 0

virtual void **setConsoleSelectedObject** (**const** *MWWorld*::Ptr &*object*) = 0

virtual void **setValue** (**const** std::string &*id*, **const** MWMechanics::AttributeValue &*value*) = 0
Set value for the given ID.

virtual void **setValue** (int *parSkill*, **const** MWMechanics::SkillValue &*value*) = 0

virtual void **setValue** (**const** std::string &*id*, **const** MWMechanics::DynamicStat<float> &*value*) = 0

virtual void **setValue** (**const** std::string &*id*, **const** std::string &*value*) = 0

virtual void **setValue** (**const** std::string &*id*, int *value*) = 0

virtual void **setDrowningTimeLeft** (float *time*, float *maxTime*) = 0
Set time left for the player to start drowning (update the drowning bar)
Parameters
- `time`: time left to start drowning
- `maxTime`: how long we can be underwater (in total) until drowning starts

virtual void **setPlayerClass** (**const** ESM::Class &*class_*) = 0
set current class of player

virtual void **configureSkills** (**const** *SkillList* &*major*, **const** *SkillList* &*minor*) = 0
configure skill groups, each set contains the skill ID for that group.

virtual void **updateSkillArea** () = 0
update display of skills, factions, birth sign, reputation and bounty

virtual void **changeCell** (**const** *MWWorld*::CellStore **cell*) = 0
change the active cell

virtual void **setFocusObject** (**const** *MWWorld*::Ptr &*focus*) = 0

virtual void **setFocusObjectScreenCoords** (float *min_x*, float *min_y*, float *max_x*, float *max_y*) = 0

virtual void **setCursorVisible** (bool *visible*) = 0

virtual void **getMousePosition** (int &*x*, int &*y*) = 0

virtual void **getMousePosition** (float &*x*, float &*y*) = 0

virtual void **setDragDrop** (bool *dragDrop*) = 0

virtual bool **getWorldMouseOver** () = 0

virtual bool **toggleFogOfWar** () = 0

virtual bool **toggleFullHelp** () = 0
show extra info in item tooltips (owner, script)

virtual bool **getFullHelp** () **const** = 0

virtual void **setActiveMap** (int *x*, int *y*, bool *interior*) = 0
set the indices of the map texture that should be used

virtual void **setDrowningBarVisibility** (bool *visible*) = 0
sets the visibility of the drowning bar

virtual void **setHMSVisibility** (bool *visible*) = 0
sets the visibility of the hud health/magicka/stamina bars

virtual void **setMinimapVisibility** (bool *visible*) = 0
sets the visibility of the hud minimap

virtual void **setWeaponVisibility** (bool *visible*) = 0

virtual void **setSpellVisibility** (bool *visible*) = 0

virtual void **setSneakVisibility** (bool *visible*) = 0

virtual void **activateQuickKey** (int *index*) = 0

virtual std::string **getSelectedSpell** () = 0

virtual void **setSelectedSpell** (**const** std::string &*spellId*, int *successChancePercent*) = 0

virtual void **setSelectedEnchantItem** (**const** *MWWorld*::Ptr &*item*) = 0

virtual void **setSelectedWeapon** (**const** *MWWorld*::Ptr &*item*) = 0

virtual void **unsetSelectedSpell** () = 0

virtual void **unsetSelectedWeapon** () = 0

virtual void **showCrosshair** (bool *show*) = 0

virtual bool **getSubtitlesEnabled** () = 0

virtual bool **toggleGui** () = 0

virtual void **disallowMouse** () = 0

virtual void **allowMouse** () = 0

virtual void **notifyInputActionBound** () = 0

virtual void **addVisitedLocation** (**const** std::string &*name*, int *x*, int *y*) = 0

virtual void **removeDialog** (*MWGui*::Layout **dialog*) = 0
Hides dialog and schedules dialog to be deleted.

virtual void **exitCurrentGuiMode** () = 0
Gracefully attempts to exit the topmost GUI mode.

No guarentee of actually closing the window

virtual void MWBase::WindowManager::messageBox(const std::string & message, enum MW

virtual void **staticMessageBox** (**const** std::string &*message*) = 0

virtual void **removeStaticMessageBox** () = 0

virtual void **interactiveMessageBox** (**const** std::string &*message*, **const** std::vector<std::string> &*buttons* = std::vector< std::string >(), bool *block* = false) = 0

virtual int **readPressedButton** () = 0
returns the index of the pressed button or -1 if no button was pressed (->MessageBoxmanager->InteractiveMessageBox)

virtual void **onFrame** (float *frameDuration*) = 0

virtual std::map<int, MWMechanics::SkillValue> **getPlayerSkillValues** () = 0

virtual std::map<int, MWMechanics::AttributeValue> **getPlayerAttributeValues** () = 0

virtual *SkillList* **getPlayerMinorSkills** () = 0

virtual *SkillList* **getPlayerMajorSkills** () = 0

virtual std::string **getGameSettingString** (**const** std::string &*id*, **const** std::string &*default_*) = 0
Fetches a GMST string from the store, if there is no setting with the given ID or it is not a string the default string is returned.

Parameters
- `id`: Identifier for the GMST setting, e.g. “aName”
- `default`: Default value if the GMST setting cannot be used.

virtual void **processChangedSettings** (**const** std::set<std::pair<std::string, std::string>> &*changed*) = 0

virtual void **windowResized** (int *x*, int *y*) = 0

virtual void **executeInConsole** (**const** std::string &*path*) = 0

virtual void **enableRest** () = 0

virtual bool **getRestEnabled** () = 0

virtual bool **getJournalAllowed** () = 0

virtual bool **getPlayerSleeping** () = 0

virtual void **wakeUpPlayer** () = 0

virtual void **showCompanionWindow** (*MWWorld*::Ptr *actor*) = 0

virtual void **startSpellMaking** (*MWWorld*::Ptr *actor*) = 0

virtual void **startEnchanting**(*MWWorld*::Ptr *actor*) = 0

virtual void **startRecharge**(*MWWorld*::Ptr *soulgem*) = 0

virtual void **startSelfEnchanting**(*MWWorld*::Ptr *soulgem*) = 0

virtual void **startTraining**(*MWWorld*::Ptr *actor*) = 0

virtual void **startRepair**(*MWWorld*::Ptr *actor*) = 0

virtual void **startRepairItem**(*MWWorld*::Ptr *item*) = 0

virtual void **startTravel**(**const** *MWWorld*::Ptr &*actor*) = 0

virtual void **startSpellBuying**(**const** *MWWorld*::Ptr &*actor*) = 0

virtual void **startTrade**(**const** *MWWorld*::Ptr &*actor*) = 0

virtual void **openContainer**(**const** *MWWorld*::Ptr &*container*, bool *loot*) = 0

virtual void **showBook**(**const** *MWWorld*::Ptr &*item*, bool *showTakeButton*) = 0

virtual void **showScroll**(**const** *MWWorld*::Ptr &*item*, bool *showTakeButton*) = 0

virtual void **showSoulgemDialog**(*MWWorld*::Ptr *item*) = 0

virtual void **changePointer**(**const** std::string &*name*) = 0

virtual void **setEnemy**(**const** *MWWorld*::Ptr &*enemy*) = 0

virtual const Translation::Storage &**getTranslationDataStorage**() **const** = 0

virtual void **setKeyFocusWidget**(MyGUI::Widget **widget*) = 0
Warning: do not use MyGUI::InputManager::setKeyFocusWidget directly. Instead use this.

virtual Loading::Listener ***getLoadingScreen**() = 0

virtual bool **getCursorVisible**() = 0
Should the cursor be visible?

virtual void **clear**() = 0
Clear all savegame-specific data.

virtual void **write**(ESM::ESMWriter &*writer*, Loading::Listener &*progress*) = 0

virtual void **readRecord**(ESM::ESMReader &*reader*, uint32_t *type*) = 0

virtual int **countSavedGameRecords**() **const** = 0

virtual bool **isSavingAllowed**() **const** = 0
Does the current stack of GUI-windows permit saving?

virtual void **exitCurrentModal**() = 0
Send exit command to active Modal window.

virtual void **addCurrentModal**(*MWGui*::WindowModal **input*) = 0
Sets the current Modal.

Used to send exit command to active Modal when Esc is pressed

virtual void **removeCurrentModal**(*MWGui*::WindowModal **input*) = 0
Removes the top Modal.

Used when one Modal adds another Modal
Parameters
- `input`: Pointer to the current modal, to ensure proper modal is removed

virtual void **pinWindow**(*MWGui*::GuiWindow *window*) = 0

virtual void **fadeScreenIn**(**const** float *time*, bool *clearQueue* = true) = 0
Fade the screen in, over *time* seconds.

virtual void **fadeScreenOut**(**const** float *time*, bool *clearQueue* = true) = 0
Fade the screen out to black, over *time* seconds.

virtual void **fadeScreenTo**(**const** int *percent*, **const** float *time*, bool *clearQueue* = true) = 0
Fade the screen to a specified percentage of black, over *time* seconds.

virtual void **setBlindness**(**const** int *percent*) = 0
Darken the screen to a specified percentage.

virtual void **activateHitOverlay**(bool *interrupt* = true) = 0

virtual void **setWerewolfOverlay**(bool *set*) = 0

virtual void **toggleDebugWindow**() = 0

virtual void **cycleSpell**(bool *next*) = 0
Cycle to next or previous spell.

virtual void **cycleWeapon**(bool *next*) = 0
Cycle to next or previous weapon.

virtual std::string **correctIconPath**(**const** std::string &*path*) = 0

virtual std::string **correctBookartPath**(**const** std::string &*path*, int *width*, int *height*) = 0

virtual std::string **correctTexturePath**(**const** std::string &*path*) = 0

virtual bool **textureExists**(**const** std::string &*path*) = 0

virtual void **removeCell**(*MWWorld*::CellStore **cell*) = 0

virtual void **writeFog**(*MWWorld*::CellStore **cell*) = 0

Private Functions

WindowManager(**const** *WindowManager*&)
not implemented

WindowManager &**operator=**(**const** *WindowManager*&)
not implemented

namespace MWWorld

Typedefs

typedef std::vector<std::pair<*MWWorld*::Ptr, MWMechanics::Movement>> **PtrMovementList**

namespace MWBase

class World
#include <world.hpp> Interface for the *World* (implemented in MWWorld)

Public Types

enum DetectionType
Values:

Detect_Enchantment

Detect_Key

Detect_Creature

Public Functions

World()

virtual ~World()

virtual void **preloadCommonAssets**() = 0

virtual void **startNewGame**(bool *bypass*) = 0

Parameters

- `bypass`: Bypass regular game start.

virtual void **clear**() = 0

virtual int **countSavedGameRecords**() **const** = 0

virtual int **countSavedGameCells**() **const** = 0

virtual void **write**(ESM::ESMWriter &*writer*, Loading::Listener &*listener*) **const** = 0

virtual void **readRecord**(ESM::ESMReader &*reader*, uint32_t *type*, **const** std::map<int, int> &*contentFileMap*) = 0

virtual *MWWorld*::CellStore ***getExterior**(int *x*, int *y*) = 0

virtual *MWWorld*::CellStore ***getInterior**(**const** std::string &*name*) = 0

virtual *MWWorld*::CellStore ***getCell**(**const** ESM::CellId &*id*) = 0

virtual void **useDeathCamera**() = 0

virtual void **setWaterHeight**(**const** float *height*) = 0

virtual bool **toggleWater**() = 0

virtual bool **toggleWorld**() = 0

virtual void **adjustSky**() = 0

virtual const Fallback::Map ***getFallback**() **const** = 0

virtual *MWWorld*::Player &**getPlayer**() = 0

virtual *MWWorld*::Ptr **getPlayerPtr**() = 0

virtual const *MWWorld*::ESMStore &**getStore**() **const** = 0

virtual std::vector<ESM::ESMReader> &**getEsmReader**() = 0

virtual *MWWorld*::LocalScripts &**getLocalScripts**() = 0

virtual bool **hasCellChanged**() **const** = 0
Has the set of active cells changed, since the last frame?

virtual bool **isCellExterior**() **const** = 0

virtual bool **isCellQuasiExterior**() **const** = 0

virtual osg::Vec2f **getNorthVector**(**const** *MWWorld*::CellStore **cell*) = 0
get north vector for given interior cell

virtual void **getDoorMarkers** (*MWWorld*::CellStore **cell*, std::vector<*DoorMarker*> &*out*) = 0
get a list of teleport door markers for a given cell, to be displayed on the local map

virtual void **setGlobalInt** (**const** std::string &*name*, int *value*) = 0
Set value independently from real type.

virtual void **setGlobalFloat** (**const** std::string &*name*, float *value*) = 0
Set value independently from real type.

virtual int **getGlobalInt** (**const** std::string &*name*) **const** = 0
Get value independently from real type.

virtual float **getGlobalFloat** (**const** std::string &*name*) **const** = 0
Get value independently from real type.

virtual char **getGlobalVariableType** (**const** std::string &*name*) **const** = 0
Return ' ', if there is no global variable with this name.

virtual std::string **getCellName** (**const** *MWWorld*::CellStore **cell* = 0) **const** = 0
Return name of the cell.

Note If cell==0, the cell the player is currently in will be used instead to generate a name.

virtual void **removeRefScript** (*MWWorld*::RefData **ref*) = 0

virtual *MWWorld*::Ptr **getPtr** (**const** std::string &*name*, bool *activeOnly*) = 0
Return a pointer to a liveCellRef with the given name.
Parameters
- `activeOnly`: do non search inactive cells.

virtual *MWWorld*::Ptr **searchPtr** (**const** std::string &*name*, bool *activeOnly*) = 0
Return a pointer to a liveCellRef with the given name.
Parameters
- `activeOnly`: do non search inactive cells.

virtual *MWWorld*::Ptr **searchPtrViaActorId** (int *actorId*) = 0
Search is limited to the active cells.

virtual *MWWorld*::Ptr **findContainer** (**const** *MWWorld*::ConstPtr &*ptr*) = 0
Return a pointer to a liveCellRef which contains *ptr*.
Note Search is limited to the active cells.

virtual void **enable** (**const** *MWWorld*::Ptr &*ptr*) = 0

virtual void **disable** (**const** *MWWorld*::Ptr &*ptr*) = 0

virtual void **advanceTime** (double *hours*, bool *incremental* = false) = 0
Advance in-game time.

virtual void **setHour** (double *hour*) = 0
Set in-game time hour.

virtual void **setMonth** (int *month*) = 0
Set in-game time month.

virtual void **setDay** (int *day*) = 0
Set in-game time day.

virtual int **getDay** () **const** = 0

virtual int **getMonth** () **const** = 0

virtual int **getYear** () **const** = 0

virtual std::string **getMonthName** (int *month* = -1) **const** = 0
Return name of month (-1: current month)

virtual *MWWorld*::TimeStamp **getTimeStamp** () **const** = 0
Return current in-game time stamp.

virtual bool **toggleSky** () = 0 **Return**
Resulting mode

virtual void **changeWeather** (**const** std::string &*region*, unsigned int *id*) = 0

virtual int **getCurrentWeather** () **const** = 0

virtual int **getMasserPhase** () **const** = 0

virtual int **getSecundaPhase** () **const** = 0

virtual void **setMoonColour** (bool *red*) = 0

virtual void **modRegion** (**const** std::string &*regionid*, **const** std::vector<char> &*chances*) = 0

virtual float **getTimeScaleFactor** () **const** = 0

virtual void **changeToInteriorCell** (**const** std::string &*cellName*, **const** ESM::Position &*position*, bool *adjustPlayerPos*, bool *changeEvent* = true) = 0
Move to interior cell.
Parameters
- `changeEvent`: If false, do not trigger cell change flag or detect worldspace changes

virtual void **changeToExteriorCell** (**const** ESM::Position &*position*, bool *adjustPlayerPos*, bool *changeEvent* = true) = 0
Move to exterior cell.
Parameters
- `changeEvent`: If false, do not trigger cell change flag or detect worldspace changes

virtual void **changeToCell** (**const** ESM::CellId &*cellId*, **const** ESM::Position &*position*, bool *adjustPlayerPos*, bool *changeEvent* = true) = 0 **Parameters**
- `changeEvent`: If false, do not trigger cell change flag or detect worldspace changes

virtual const ESM::Cell ***getExterior** (**const** std::string &*cellName*) **const** = 0
Return a cell matching the given name or a 0-pointer, if there is no such cell.

virtual void **markCellAsUnchanged** () = 0

virtual *MWWorld*::Ptr **getFacedObject** () = 0
Return pointer to the object the player is looking at, if it is within activation range.

virtual float **getDistanceToFacedObject** () = 0

virtual float **getMaxActivationDistance** () = 0

virtual std::pair<*MWWorld*::Ptr, osg::Vec3f> **getHitContact** (**const** *MWWorld*::ConstPtr &*ptr*, float *distance*) = 0
Returns a pointer to the object the provided object would hit (if within the specified distance), and the point where the hit occurs. This will attempt to use the "Head" node, or alternatively the "Bip01 Head" node as a basis.

virtual void **adjustPosition** (**const** *MWWorld*::Ptr &*ptr*, bool *force*) = 0
Adjust position after load to be on ground. Must be called after model load.
Parameters
- `force`: do this even if the ptr is flying

virtual void **fixPosition** (**const** *MWWorld*::Ptr &*actor*) = 0
Attempt to fix position so that the Ptr is no longer inside collision geometry.

virtual void **deleteObject** (**const** *MWWorld*::Ptr &*ptr*) = 0 **Note**
No-op for items in containers. Use ContainerStore::removeItem instead.

virtual void **undeleteObject** (**const** *MWWorld*::Ptr &*ptr*) = 0

virtual *MWWorld*::Ptr **moveObject** (**const** *MWWorld*::Ptr &*ptr*, float *x*, float *y*, float *z*) = 0 **Return**
an updated Ptr in case the Ptr's cell changes

virtual *MWWorld*::Ptr **moveObject** (**const** *MWWorld*::Ptr &*ptr*, *MWWorld*::CellStore **newCell*, float *x*, float *y*, float *z*, bool *movePhysics* = true) = 0 **Return**
an updated Ptr

virtual void **scaleObject** (**const** *MWWorld*::Ptr &*ptr*, float *scale*) = 0

virtual void **rotateObject** (**const** *MWWorld*::Ptr &*ptr*, float *x*, float *y*, float *z*, bool *adjust* = false) = 0

virtual *MWWorld*::Ptr **placeObject** (**const** *MWWorld*::ConstPtr &*ptr*, *MWWorld*::CellStore **cell*, ESM::Position *pos*) = 0
Place an object. Makes a copy of the Ptr.

virtual *MWWorld*::Ptr **safePlaceObject** (**const** *MWWorld*::ConstPtr &*ptr*, **const** *MWWorld*::ConstPtr &*referenceObject*, *MWWorld*::CellStore **referenceCell*, int *direction*, float *distance*) = 0
Place an object in a safe place next to *referenceObject*. *direction* and *distance* specify the wanted placement relative to *referenceObject* (but the object may be placed somewhere else if the wanted location is obstructed).

virtual void **indexToPosition** (int *cellX*, int *cellY*, float &*x*, float &*y*, bool *centre* = false) **const** = 0
Convert cell numbers to position.

virtual void **positionToIndex** (float *x*, float *y*, int &*cellX*, int &*cellY*) **const** = 0
Convert position to cell numbers.

virtual void **queueMovement** (**const** *MWWorld*::Ptr &*ptr*, **const** osg::Vec3f &*velocity*) = 0
Queues movement for *ptr* (in local space), to be applied in the next call to doPhysics.

virtual bool **castRay** (float *x1*, float *y1*, float *z1*, float *x2*, float *y2*, float *z2*) = 0
cast a Ray and return true if there is an object in the ray path.

virtual bool **toggleCollisionMode** () = 0
Toggle collision mode for player. If disabled player object should ignore collisions and gravity.
Return Resulting mode

virtual bool **toggleRenderMode** (MWRender::RenderMode *mode*) = 0
Toggle a render mode.
Return Resulting mode

virtual const ESM::Potion ***createRecord** (**const** ESM::Potion &*record*) = 0
Create a new record (of type potion) in the ESM store.
Return pointer to created record

virtual const ESM::Spell ***createRecord** (**const** ESM::Spell &*record*) = 0
Create a new record (of type spell) in the ESM store.
Return pointer to created record

virtual const ESM::Class ***createRecord** (**const** ESM::Class &*record*) = 0
Create a new record (of type class) in the ESM store.
Return pointer to created record

virtual const ESM::Cell *`createRecord`(**const** ESM::Cell &*record*) = 0
Create a new record (of type cell) in the ESM store.
Return pointer to created record

virtual const ESM::NPC *`createRecord`(**const** ESM::NPC &*record*) = 0
Create a new record (of type npc) in the ESM store.
Return pointer to created record

virtual const ESM::Armor *`createRecord`(**const** ESM::Armor &*record*) = 0
Create a new record (of type armor) in the ESM store.
Return pointer to created record

virtual const ESM::Weapon *`createRecord`(**const** ESM::Weapon &*record*) = 0
Create a new record (of type weapon) in the ESM store.
Return pointer to created record

virtual const ESM::Clothing *`createRecord`(**const** ESM::Clothing &*record*) = 0
Create a new record (of type clothing) in the ESM store.
Return pointer to created record

virtual const ESM::Enchantment *`createRecord`(**const** ESM::Enchantment &*record*) = 0
Create a new record (of type enchantment) in the ESM store.
Return pointer to created record

virtual const ESM::Book *`createRecord`(**const** ESM::Book &*record*) = 0
Create a new record (of type book) in the ESM store.
Return pointer to created record

virtual const ESM::CreatureLevList *`createOverrideRecord`(**const** ESM::CreatureLevList &*record*) = 0
Write this record to the ESM store, allowing it to override a pre-existing record with the same ID.
Return pointer to created record

virtual const ESM::ItemLevList *`createOverrideRecord`(**const** ESM::ItemLevList &*record*) = 0
Write this record to the ESM store, allowing it to override a pre-existing record with the same ID.
Return pointer to created record

virtual void `update`(float *duration*, bool *paused*) = 0

virtual *MWWorld*::Ptr `placeObject`(**const** *MWWorld*::ConstPtr &*object*, float *cursorX*, float *cursorY*, int *amount*) = 0
copy and place an object into the gameworld at the specified cursor position
Parameters
- `object`:
- `cursor`: X (relative 0-1)
- `cursor`: Y (relative 0-1)
- `number`: of objects to place

virtual *MWWorld*::Ptr `dropObjectOnGround`(**const** *MWWorld*::Ptr &*actor*, **const** *MWWorld*::ConstPtr &*object*, int *amount*) = 0
copy and place an object into the gameworld at the given actor's position
Parameters
- `actor`: giving the dropped object position
- `object`:
- `number`: of objects to place

virtual bool `canPlaceObject`(float *cursorX*, float *cursorY*) = 0 **Return** true if it is possible to place on object at specified cursor location

virtual void `processChangedSettings`(**const** std::set<std::pair<std::string, std::string>> &*settings*) = 0

virtual bool **isFlying** (**const** *MWWorld*::Ptr &*ptr*) **const** = 0

virtual bool **isSlowFalling** (**const** *MWWorld*::Ptr &*ptr*) **const** = 0

virtual bool **isSwimming** (**const** *MWWorld*::ConstPtr &*object*) **const** = 0

virtual bool **isWading** (**const** *MWWorld*::ConstPtr &*object*) **const** = 0

virtual bool **isSubmerged** (**const** *MWWorld*::ConstPtr &*object*) **const** = 0
Is the head of the creature underwater?

virtual bool **isUnderwater** (**const** *MWWorld*::CellStore **cell*, **const** osg::Vec3f &*pos*) **const** = 0

virtual bool **isOnGround** (**const** *MWWorld*::Ptr &*ptr*) **const** = 0

virtual osg::Matrixf **getActorHeadTransform** (**const** *MWWorld*::ConstPtr &*actor*) **const** = 0

virtual void **togglePOV** () = 0

virtual bool **isFirstPerson** () **const** = 0

virtual void **togglePreviewMode** (bool *enable*) = 0

virtual bool **toggleVanityMode** (bool *enable*) = 0

virtual void **allowVanityMode** (bool *allow*) = 0

virtual void **togglePlayerLooking** (bool *enable*) = 0

virtual void **changeVanityModeScale** (float *factor*) = 0

virtual bool **vanityRotateCamera** (float **rot*) = 0

virtual void **setCameraDistance** (float *dist*, bool *adjust* = false, bool *override* = true) = 0

virtual void **setupPlayer** () = 0

virtual void **renderPlayer** () = 0

virtual void **activateDoor** (**const** *MWWorld*::Ptr &*door*) = 0
open or close a non-teleport door (depending on current state)

virtual void **activateDoor** (**const** *MWWorld*::Ptr &*door*, int *state*) = 0
update movement state of a non-teleport door as specified
Note throws an exception when invoked on a teleport door
Parameters
- `state`: see MWClass::setDoorState

virtual bool **getPlayerStandingOn** (**const** *MWWorld*::ConstPtr &*object*) = 0 **Return**
true if the player is standing on *object*

virtual bool **getActorStandingOn** (**const** *MWWorld*::ConstPtr &*object*) = 0 **Return**
true if any actor is standing on *object*

virtual bool **getPlayerCollidingWith** (**const** *MWWorld*::ConstPtr &*object*) = 0 **Return**
true if the player is colliding with *object*

virtual bool **getActorCollidingWith** (**const** *MWWorld*::ConstPtr &*object*) = 0 **Return**
true if any actor is colliding with *object*

virtual void **hurtStandingActors** (**const** *MWWorld*::ConstPtr &*object*, float *dmgPerSecond*) = 0
Apply a health difference to any actors standing on *object*. To hurt actors, healthPerSecond should be a positive value. For a negative value, actors will be healed.

virtual void **hurtCollidingActors** (**const** *MWWorld*::ConstPtr &*object*, float *dmgPerSecond*) = 0

Apply a health difference to any actors colliding with *object*. To hurt actors, healthPerSecond should be a positive value. For a negative value, actors will be healed.

virtual float **getWindSpeed** () = 0

virtual void **getContainersOwnedBy** (**const** *MWWorld*::ConstPtr &*npc*, std::vector<*MWWorld*::Ptr> &*out*) = 0

get all containers in active cells owned by this Npc

virtual void **getItemsOwnedBy** (**const** *MWWorld*::ConstPtr &*npc*, std::vector<*MWWorld*::Ptr> &*out*) = 0

get all items in active cells owned by this Npc

virtual bool **getLOS** (**const** *MWWorld*::ConstPtr &*actor*, **const** *MWWorld*::ConstPtr &*targetActor*) = 0

get Line of Sight (morrowind stupid implementation)

virtual float **getDistToNearestRayHit** (**const** osg::Vec3f &*from*, **const** osg::Vec3f &*dir*, float *maxDist*, bool *includeWater* = false) = 0

virtual void **enableActorCollision** (**const** *MWWorld*::Ptr &*actor*, bool *enable*) = 0

virtual int **canRest** () = 0

check if the player is allowed to rest 0 - yes 1 - only waiting 2 - player is underwater 3 - enemies are nearby (not implemented)

virtual MWRender::Animation ***getAnimation** (**const** *MWWorld*::Ptr &*ptr*) = 0

virtual const MWRender::Animation ***getAnimation** (**const** *MWWorld*::ConstPtr &*ptr*) **const** = 0

virtual void **reattachPlayerCamera** () = 0

virtual void **screenshot** (osg::Image **image*, int *w*, int *h*) = 0

virtual bool **findExteriorPosition** (**const** std::string &*name*, ESM::Position &*pos*) = 0

Find default position inside exterior cell specified by name

Return false if exterior with given name not exists, true otherwise

virtual bool **findInteriorPosition** (**const** std::string &*name*, ESM::Position &*pos*) = 0

Find default position inside interior cell specified by name

Return false if interior with given name not exists, true otherwise

virtual void **enableTeleporting** (bool *enable*) = 0

Enables or disables use of teleport spell effects (recall, intervention, etc).

virtual bool **isTeleportingEnabled** () **const** = 0

Returns true if teleport spell effects are allowed.

virtual void **enableLevitation** (bool *enable*) = 0

Enables or disables use of levitation spell effect.

virtual bool **isLevitationEnabled** () **const** = 0

Returns true if levitation spell effect is allowed.

virtual bool **getGodModeState** () = 0

virtual bool **toggleGodMode** () = 0

virtual bool **toggleScripts** () = 0

virtual bool **getScriptsEnabled** () **const** = 0

virtual bool **startSpellCast** (**const** *MWWorld*::Ptr &*actor*) = 0

startSpellCast attempt to start casting a spell. Might fail immediately if conditions are not met.

Return true if the spell can be casted (i.e. the animation should start)
Parameters
- `actor`:

virtual void **castSpell**(**const** *MWWorld*::Ptr &*actor*) = 0

virtual void **launchMagicBolt**(**const** std::string &*model*, **const** std::string &*sound*, **const** std::string &*spellId*, float *speed*, bool *stack*, **const** ESM::EffectList &*effects*, **const** *MWWorld*::Ptr &*caster*, **const** std::string &*sourceName*, **const** osg::Vec3f &*fallbackDirection*) = 0

virtual void **launchProjectile**(*MWWorld*::Ptr *actor*, *MWWorld*::ConstPtr *projectile*, **const** osg::Vec3f &*worldPos*, **const** osg::Quat &*orient*, *MWWorld*::Ptr *bow*, float *speed*, float *attackStrength*) = 0

virtual const std::vector<std::string> &**getContentFiles**() **const** = 0

virtual void **breakInvisibility**(**const** *MWWorld*::Ptr &*actor*) = 0

virtual bool **isDark**() **const** = 0

virtual bool **findInteriorPositionInWorldSpace**(**const** *MWWorld*::CellStore **cell*, osg::Vec3f &*result*) = 0

virtual void **teleportToClosestMarker**(**const** *MWWorld*::Ptr &*ptr*, **const** std::string &*id*) = 0

Teleports *ptr* to the closest reference of *id* (e.g. DivineMarker, PrisonMarker, TtempleMarker)
Note id must be lower case

virtual void **listDetectedReferences**(**const** *MWWorld*::Ptr &*ptr*, std::vector<*MWWorld*::Ptr> &*out*, *DetectionType* *type*) = 0

List all references (filtered by *type*) detected by *ptr*. The range is determined by the current magnitude of the "Detect X" magic effect belonging to *type*.
Note This also works for references in containers.

virtual void **updateDialogueGlobals**() = 0

Update the value of some globals according to the world state, which may be used by dialogue entries. This should be called when initiating a dialogue.

virtual void **confiscateStolenItems**(**const** *MWWorld*::Ptr &*ptr*) = 0

Moves all stolen items from *ptr* to the closest evidence chest.

virtual void **goToJail**() = 0

virtual void **spawnRandomCreature**(**const** std::string &*creatureList*) = 0

Spawn a random creature from a levelled list next to the player.

virtual void **spawnBloodEffect**(**const** *MWWorld*::Ptr &*ptr*, **const** osg::Vec3f &*worldPosition*) = 0

Spawn a blood effect for *ptr* at *worldPosition*.

virtual void **spawnEffect**(**const** std::string &*model*, **const** std::string &*textureOverride*, **const** osg::Vec3f &*worldPos*) = 0

virtual void **explodeSpell**(**const** osg::Vec3f &*origin*, **const** ESM::EffectList &*effects*, **const** *MWWorld*::Ptr &*caster*, **const** *MWWorld*::Ptr &*ignore*, ESM::RangeType *rangeType*, **const** std::string &*id*, **const** std::string &*sourceName*) = 0

virtual void **activate**(**const** *MWWorld*::Ptr &*object*, **const** *MWWorld*::Ptr &*actor*) = 0

virtual bool **isInStorm**() **const** = 0

See MWWorld::WeatherManager::isInStorm

virtual osg::Vec3f **getStormDirection**() **const** = 0 **See** MWWorld::WeatherManager::getStormDirection

virtual void **resetActors**() = 0
Resets all actors in the current active cells to their original location within that cell.

virtual bool **isWalkingOnWater**(**const** *MWWorld*::ConstPtr &*actor*) = 0

virtual osg::Vec3f **aimToTarget**(**const** *MWWorld*::ConstPtr &*actor*, **const** *MWWorld*::ConstPtr &*target*) = 0
Return a vector aiming the actor's weapon towards a target.
Note The length of the vector is the distance between actor and target.

virtual float **getHitDistance**(**const** *MWWorld*::ConstPtr &*actor*, **const** *MWWorld*::ConstPtr &*target*) = 0
Return the distance between actor's weapon and target's collision box.

virtual void **removeContainerScripts**(**const** *MWWorld*::Ptr &*reference*) = 0

virtual bool **isPlayerInJail**() **const** = 0

Private Functions

World(**const** *World*&)
not implemented

World &**operator=**(**const** *World*&)
not implemented

struct DoorMarker

Public Members

std::string **name**

float **x**

float **y**

ESM::CellId **dest**

namespace OMW

class Engine
#include <engine.hpp> Main engine class, that brings together all the components of OpenMW.

Public Functions

Engine(Files::ConfigurationManager &*configurationManager*)

virtual ~Engine()

void **enableFSStrict**(bool *fsStrict*)
Enable strict filesystem mode (do not fold case)

Attention The strict mode must be specified before any path-related settings are given to the engine.

void **setDataDirs**(**const** Files::PathContainer &*dataDirs*)
Set data dirs.

void **addArchive** (**const** std::string &*archive*)
Add BSA archive.

void **setResourceDir** (**const** boost::filesystem::path &*parResDir*)
Set resource dir.

void **setCell** (**const** std::string &*cellName*)
Set start cell name (only interiors for now)

void **addContentFile** (**const** std::string &*file*)
addContentFile - Adds content file (ie. esm/esp, or omwgame/omwaddon) to the content files container.

Parameters
- `file`: - filename (extension is required)

void **setScriptsVerbosity** (bool *scriptsVerbosity*)
Enable or disable verbose script output.

void **setSoundUsage** (bool *soundUsage*)
Disable or enable all sounds.

void **setSkipMenu** (bool *skipMenu*, bool *newGame*)
Skip main menu and go directly into the game

Parameters
- `newGame`: Start a new game instead off dumping the player into the game (ignored if !skipMenu).

void **setGrabMouse** (bool *grab*)

void **go** ()
Initialise and enter main loop.

void **setCompileAll** (bool *all*)
Compile all scripts (excludign dialogue scripts) at startup?

void **setCompileAllDialogue** (bool *all*)
Compile all dialogue scripts at startup?

void **setEncoding** (**const** ToUTF8::FromType &*encoding*)
Font encoding.

void **setFallbackValues** (std::map<std::string, std::string> *map*)

void **setScriptConsoleMode** (bool *enabled*)
Enable console-only script functionality.

void **setStartupScript** (**const** std::string &*path*)
Set path for a script that is run on startup in the console.

void **setActivationDistanceOverride** (int *distance*)
Override the game setting specified activation distance.

void **setWarningsMode** (int *mode*)

void **setScriptBlacklist** (**const** std::vector<std::string> &*list*)

void **setScriptBlacklistUse** (bool *use*)

void **enableFontExport** (bool *exportFonts*)

void **setSaveGameFile** (**const** std::string &*savegame*)
Set the save game file to load after initialising the engine.

Private Functions

Engine (**const** *Engine*&)

Engine &**operator=** (**const** *Engine*&)

void **executeLocalScripts** ()

void **frame** (float *dt*)

std::string **loadSettings** (Settings::Manager &*settings*)
Load settings from various files, returns the path to the user settings file.

void **prepareEngine** (Settings::Manager &*settings*)
Prepare engine for game play.

void **createWindow** (Settings::Manager &*settings*)

void **setWindowIcon** ()

Private Members

SDL_Window ***mWindow**

std::auto_ptr<VFS::Manager> **mVFS**

std::auto_ptr<Resource::ResourceSystem> **mResourceSystem**

MWBase::*Environment* **mEnvironment**

ToUTF8::FromType **mEncoding**

ToUTF8::Utf8Encoder ***mEncoder**

Files::PathContainer **mDataDirs**

std::vector<std::string> **mArchives**

boost::filesystem::path **mResDir**

osg::ref_ptr<osgViewer::Viewer> **mViewer**

osg::ref_ptr<osgViewer::ScreenCaptureHandler> **mScreenCaptureHandler**

std::string **mCellName**

std::vector<std::string> **mContentFiles**

bool **mVerboseScripts**

bool **mSkipMenu**

bool **mUseSound**

bool **mCompileAll**

bool **mCompileAllDialogue**

int **mWarningsMode**

std::string **mFocusName**

std::map<std::string, std::string> **mFallbackMap**

bool **mScriptConsoleMode**

std::string **mStartupScript**

int **mActivationDistanceOverride**

std::string **mSaveGameFile**

bool **mGrab**

bool **mExportFonts**

Compiler::Extensions **mExtensions**

Compiler::Context ***mScriptContext**

Files::Collections **mFileCollections**

bool **mFSStrict**

Translation::Storage **mTranslationDataStorage**

std::vector<std::string> **mScriptBlacklist**

bool **mScriptBlacklistUse**

bool **mNewGame**

osg::Timer_t **mStartTick**

Files::ConfigurationManager &**mCfgMgr**

1.1.2 Indices and tables

- genindex
- search

1.2 OpenMW CS user manual

The following document is the complete user manual for *OpenMW CS*, the construction set for the OpenMW game engine. It is intended to serve both as an introduction and a reference for the application. Even if you are familiar with modding *The Elder Scrolls III: Morrowind* you should at least read the first few chapters to familiarise yourself with the new interface.

> **Warning:** OpenMW CS is still software in development. The manual does not cover any of its shortcomings, it is written as if everything was working as inteded. Please report any software problems as bugs in the software, not errors in the manual.

1.2.1 Foreword

<Some introductory lines here, explain nomenclature and abbreviations>

How to read the manual

The manual can be roughly divided into two parts: a tutorial part consisting of the first two (three) chapters and the reference manual. We recommend all readers to work through the tutorials first, there you will be guided through the creation of a fairly simple mod where you can familiarise yourself with the record-based interface. The tutorials are very simple and teach you only what is necessary for the task, each one can be completed in under half an hour. It is strongly recommended to do the tutorials in order.

When you are familiar with the CS in general and want to write your own mods it is time to move on to the reference part of the manual. The reference chapters can be read out of order, each one covers only one topic.

1.2.2 A Tour through OpenMW CS: making a magic ring

In this first chapter we will create a mod that adds a new ring with a simple enchantment to the game. The ring will give its wearer a permanent Night Vision effect while being worn. You don't need prior knowledge about modding Morrowind, but you should be familiar with the game itself. There will be no scripting necessary, we chan achieve everything using just what the base game offers out of the box. Before continuing make sure that OpenMW is properly installed and playable.

Adding the ring to the game's records

In this first section we will define what our new ring is, what it looks like and what it does. Getting it to work is the first step before we go further.

Starting up OpenMW CS

We will start by launching OpenMW CS, the location of the program depends on your operating system. You will be presented with a dialogue with three options: create a new game, create a new addon, edit a content file.

The first option is for creating an entirely new game, that's not what we want. We want to edit an existing game, so choose the second one. When you save your addon you can use the third option to open it again.

You will be presented with another window where you get to chose the content to edit and the name of your project. We have to chose at least a base game, and optionally a number of other addons we want to depend on. The name of the project is arbitrary, it will be used to identify the addon later in the OpenMW launcher.

Choose Morrowind as your content file and enter *Ring of Night Vision* as the name. We could also chose further content files as dependencies if we wanted to, but for this mod the base game is enough.

Once the addon has been created you will be presented with a table. If you see a blank window rather than a table choose *World* → *Objects* from the menu.

Let's talk about the interface for a second. Every window in OpenMW CS has *panels*, these are often but not always tables. You can close a panel by clicking the small "X" on the title bar of the panel, or you can detach it by either dragging the title bar or clicking the icon with the two windows. A detached panel can be re-attached to a window by dragging it by the title bar on top of the window.

Now let's look at the panel itself: we have a filter text field, a very large table and a status bar. The filter will be very useful when we want to find an entry in the table, but for now it is irrelevant. The table you are looking at contains all objects in the game, these can be items, NPCs, creatures, whatever. Every object is an entry in that table, visible as a row. The columns of the table are the attributes of each object.

Morrowind uses something called a *relational database* for game data. If you are not familiar with the term, it means that every type of thing can be expressed as a *table*: there is a table for objects, a table for enchantments, a table for icons, one for meshes, and so on. Properties of an entry must be simple values, like numbers or text strings. If we want a more complicated property we need to reference an entry from another table. There are a few exceptions to this though, some tables do have subtables. The effects of enchantments are one of those exceptions.

Defining a new record

Enough talk, let's create the new ring now. Right-click anywhere in the objects table, choose *Add Record* and the status bar will change into an input field. We need to enter an *ID* (short for *identifier*) and pick the type. The identifier is a unique name by which the ring can later be identified; I have chosen *ring_night_vision*. For the type choose *Clothing*.

The table should jump right to our newly created record, if not read further below how to use filters to find a record by ID. Notice that the *Modified* column now shows that this record is new. Records can also be *Base* (unmodified), *Modified* and *Deleted*. The other fields are still empty since we created this record from nothing. We can double-click a table cell while holding Shift to edit it (this is a configurable shortcut), but there is a better way: right-click the row of our new record and chose *Edit Record*, a new panel will open.

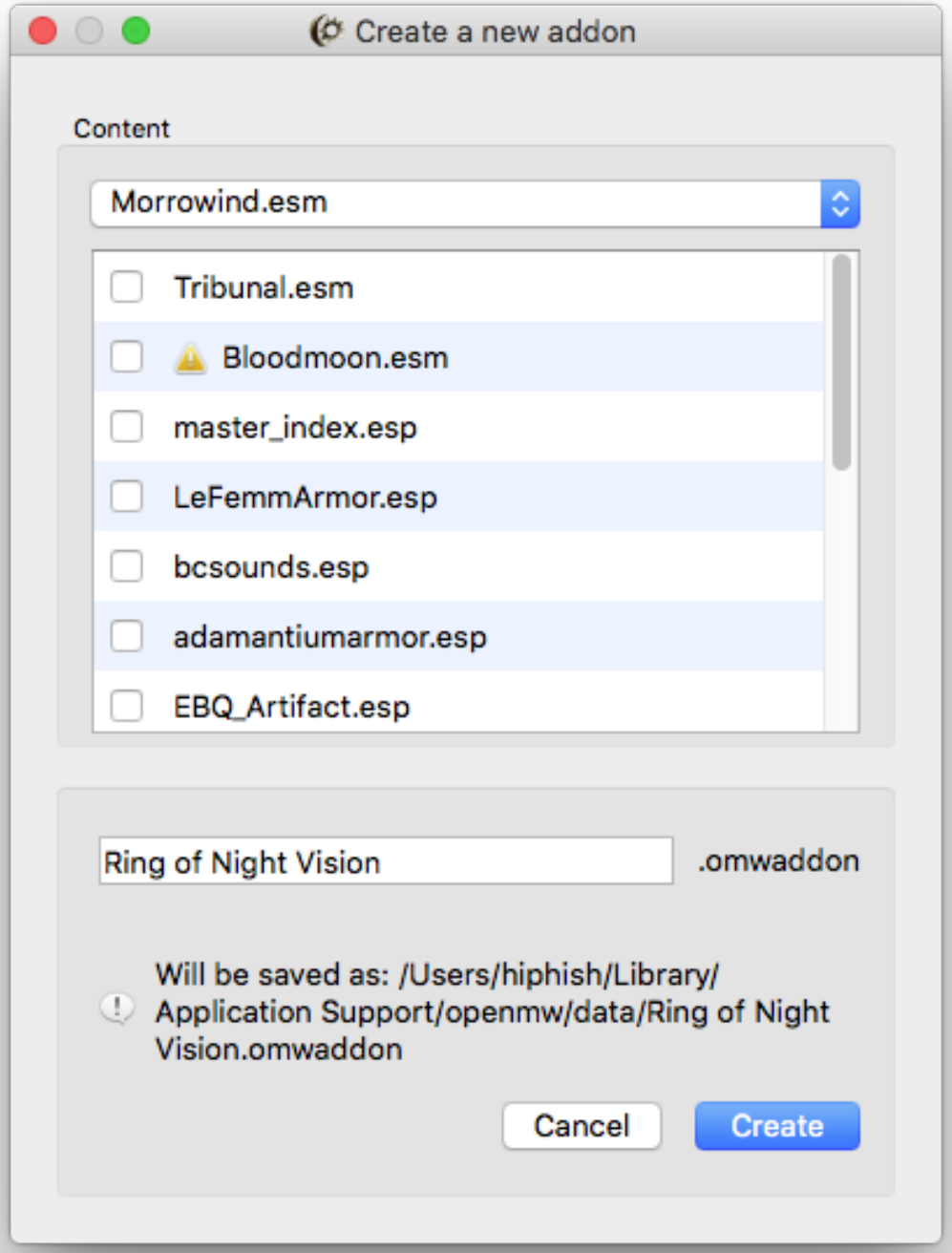
We can right-click the row of our new record and chose *Edit Record*, a new panel will open. Alternatively we can also define a configurable shortcut instead of using the context menu; the default is double-clicking while holding down the shift key.

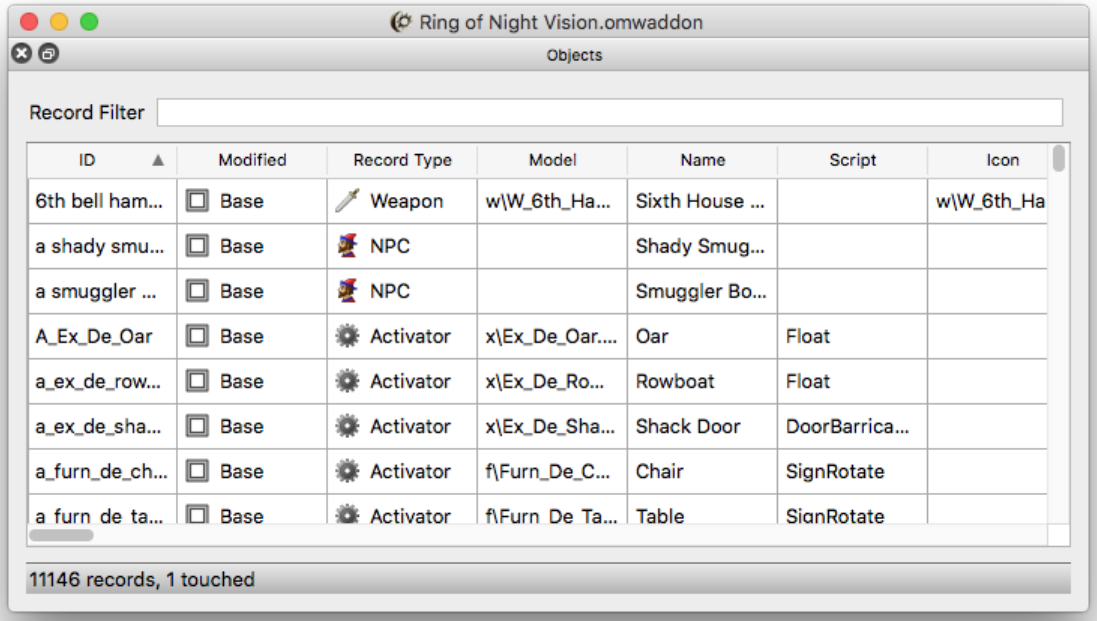
You can set the name, weight and coin value as you like, I chose *Ring of Night Vision*, *0.1* and *2500* respectively. Make sure you set the *Clothing Type* to *Ring*. We could set the other properties manually as well, but unless you have an exceptional memory for identifiers and never make typos that's not feasible. What we are going to do instead is find the records we want in their respective tables and assign them from there.

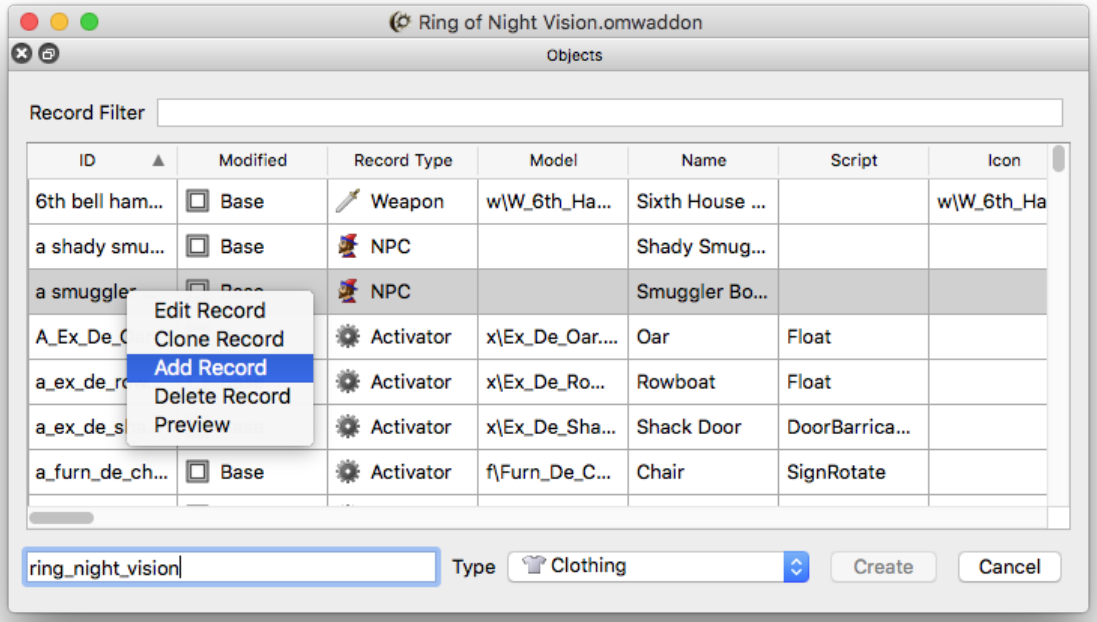
Finding records using filters

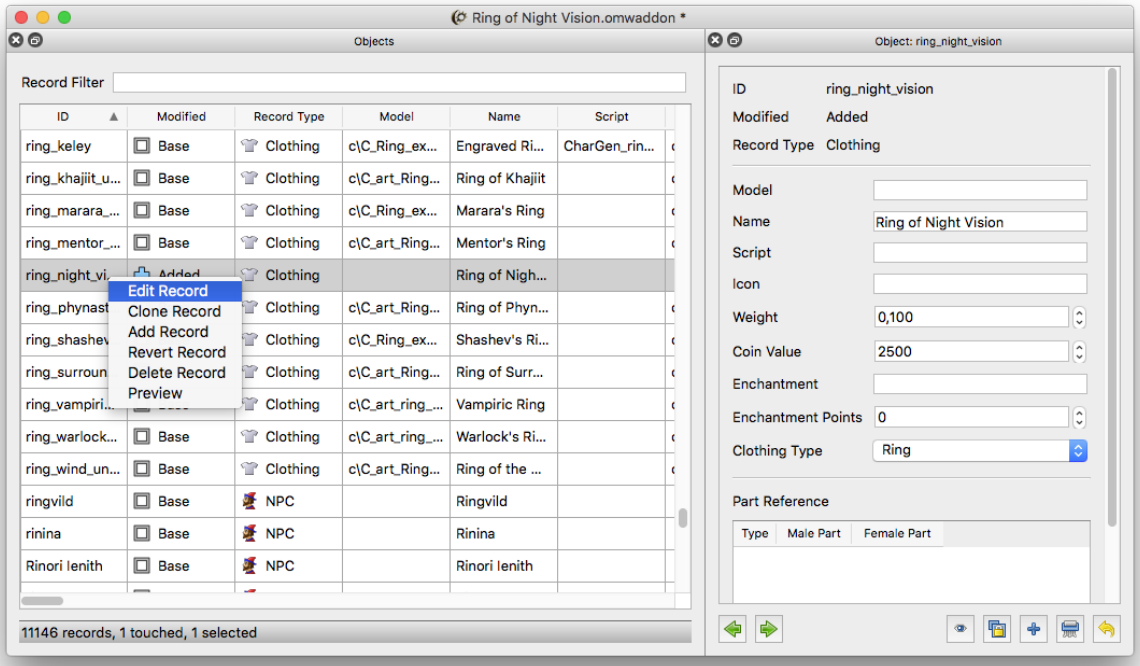
We will add an icon first. Open the *Icons* table the same way you opened the *Objects* table: in the menu click *Assets* → *Icons*. If the window gets too crowded remember that you can detach panels. The table is huge and not every ring icon starts with "ring", so we have to use filters to find what we want.

Filters are a central element of OpenMW CS and a major departure from how the original Morrowind CS was used. In fact, filters are so important that they have their own table as well. We won't be going that far for now though. There


Create a new addon
Content
Morrowind.esm
Tribunal.esm
Bloodmoon.esm
master_index.esp
LeFemmArmor.esp
bcsounds.esp
adamantiumarmor.esp
EBQ_Artifact.esp
Ring of Night Vision
.omwaddon
Will be saved as: /Users/hiphish/Library/
Application Support/openmw/data/Ring of Night
Vision.omwaddon
Cancel
Create


Ring of Night Vision.omwaddon
Objects
Record Filter
ID
Modified
Record Type
Model
Name
Script
Icon
6th bell ham...
Base
Weapon
w\W_6th_Ha...
Sixth House ...
w\W_6th_Ha
a shady smu...
Base
NPC
Shady Smug...
a smuggler ...
Base
NPC
Smuggler Bo...
A_Ex_De_Oar
Base
Activator
x\Ex_De_Oar....
Oar
Float
a_ex_de_row...
Base
Activator
x\Ex_De_Ro...
Rowboat
Float
a_ex_de_sha...
Base
Activator
x\Ex_De_Sha...
Shack Door
DoorBarrica...
a_furn_de_ch...
Base
Activator
f\Furn_De_C...
Chair
SignRotate
a_furn_de_ta...
Base
Activator
f\Furn_De_Ta...
Table
SignRotate
11146 records, 1 touched


Ring of Night Vision.omwaddon
Objects
Record Filter
ID
Modified
Record Type
Model
Name
Script
Icon
6th bell ham...
Base
Weapon
w\W_6th_Ha...
Sixth House ...
w\W_6th_Ha
a shady smu...
Base
NPC
Shady Smug...
a smuggle
NPC
Smuggler Bo...
Edit Record
Clone Record
Add Record
Delete Record
Preview
A_Ex_De_
Activator
x\Ex_De_Oar....
Oar
Float
a_ex_de_r
Activator
x\Ex_De_Ro...
Rowboat
Float
a_ex_de_s
Activator
x\Ex_De_Sha...
Shack Door
DoorBarrica...
a_furn_de_ch...
Base
Activator
f\Furn_De_C...
Chair
SignRotate
ring_night_vision
Type
Clothing
Create
Cancel



are three types of filters: *Project filters* are part of the project and are stored in the project file, *session filter* are only valid until you exit the CS, and finally *instant filter* which are used only once and typed directly into the *Filter* field.

For this tutorial we will only use instant filters. We type the definition of the filter directly into the filter field rather than the name of an existing filter. To signify that we are using an instant filter the have to use *!* as the first character. Type the following into the field:

```
!string("id", ".*ring.*")
```

A filter is defined by a number of *queries* which can be logically linked. For now all that matters is that the *string(<property>, <pattern>)* query will check whether *<propery>* matches *<pattern>*. The pattern is a regular expression, if you don't know about them you should learn their syntax. For now all that matters is that . stands for any character and * stands for any amount, even zero. In other words, we are looking for all entries which have an ID that contains the word "ring" somewhere in it. This is a pretty dumb pattern because it will also match words like "ringmail", but it's good enough for now.

If you have typed the filter definition properly the text should change from red to black and our table will be narrowed down a lot. Browse for an icon you like and drag & drop its table row onto the *Icon* field of our new ring.

That's it, you have now assigned a reference to an entry in another table to the ring entry in the *Objects* table. Repeat the same process for the 3D model, you can find the *Meshes* table under *Assets* → *Meshes*.

Adding the enchantment

Putting everything you have learned so far to practice we can add the final and most important part to our new ring: the enchantment. You know enough to perform the following steps without guidance: Open the *Enchantments* table (*Mechanics* → *Enchantments*) and create a new entry with the ID *Cats Eye*. Edit it so that it has *Constant Effect* enchantment type.

To add an effect to the enchantment right-click the *Magic Effects* table and choose *Add new row*. You can edit the

effects by right-clicking their table cells. Set the effect to *NightEye*, range to *Self*, and both magnitudes to *50*. The other properties are irrelevant.

Once you are done add the new enchantment to our ring. That's it, we now have a complete enchanted ring to play with. Let's take it for a test ride.

Playing your new addon

Launch OpenMW and in the launcher under *Data Files* check your addon. Load a game and open the console. We have only defined the ring, but we haven't placed any instance of it anywhere in the game world, so we have to create one. In the console type:

```
player->AddItem "ring_night_vision" 1
```

The part in quotation marks is the ID of our ring, you have to adjust it if you chose a different ID. Exit the console and you should find a new ring in your inventory. Equip it and you will instantly receive the *Night Vision* effect for your character.

Conclusion

In this tutorial we have learned how to create a new addon, what tables are and how to create new records. We have also taken a very brief glimpse at the syntax of filters, a feature you will be using a lot when creating larger mods.

This mod is a pure addition, it does not change any of the existing records. However, if you want to actually present appealing content to the player rather than just offering abstract definitions you will have to change the game's content. In the next tutorial we will learn how to place the ring in the game world so the player can find it legitimately.

Adding the ring to the game's world

Now that we have defined the ring it is time add it to the game world so the player can find it legitimately. We will add the ring to a merchant, place it in a chest and put it somewhere in plain sight. To this end we will have to actually modify the contents of the game.

Subsection to come...

1.2.3 Files and Directories

In this chapter of the manual we will cover the usage of files and directories by OpenMW CS. Files and directories are file system concepts of your operating system, so we will not be going into specifics about that, we will only focus on what is relevant to OpenMW CS.

Basics

Directories

OpenMW and OpenMW CS us multiple directories on the file system. First of all there is a *user directory* that holds configuration files and a number of different sub-directories. The location of the user directory is hard-coded into the CS and depends on your operating system.

Operating System	User Dircetory
GNU/Linux	`<whatever>`
OS X	`~/Library/Application Support/openmw/`
Windows	`<whatever>`

In addition to to this single hard-coded directory both OpenMW and OpenMW CS need a place to seek for a actuals data files of the game: textures, 3D models, sounds and record files that store objects in game; dialogues an so one. These files are called *content files*. We support multiple such paths (we call them *data paths*) as specified in the configuration. Usually one data path points to the directory where the original Morrowind game is either installed or unpacked to. You are free to specify as many data paths as you would like, however, there is one special data path that, as described later, which is used to store newly created content files.

Content files

The original Morrowind engine by Bethesda Softworks uses two types of content files: *esm* (master) and *esp* (plugin). The distinction between those two is not clear, and often confusing. One would expect the *esm* (master) file to be used to specify one master, which is then modified by the *esp* plugins. And indeed: this is the basic idea. However, the official expansions were also made as ESM files, even though they could essentially be described as really large plugins, and therefore would rather use *esp* files. There were technical reasons behind this decision – somewhat valid in the case of the original engine, but clearly it is better to create a system that can be used in a more sensible way. OpenMW achieves this with our own content file types.

We support both ESM and ESP files, but in order to make use of new features in OpenMW one should consider using new file types designed with our engine in mind: *game* files and *addon* files, collectively called *content files*.

OpenMW content files The concepts of *Game* and *Addon* files are somewhat similar to the old concept of *ESM* and *ESP*, but more strictly enforced. It is quite straight-formward: If you want to make new game using OpenMW as the engine (a so called *total conversion*) you should create a game file. If you want to create an addon for an existing game file create an addon file. Nothing else matters; the only distinction you should consider is if your project is about changing another game or creating a new one. Simple as that.

Another simple thing about content files are the extensions: we are using `.omwaddon` for addon files and `.omwgame` for game files.

Morrowind content files Using our content files is recommended for projects that are intended to used with the OpenMW engine. However, some players might wish to still use the original Morrowind engine. In addition thousands of *ESP*/*ESM* files were created since 2002, some of them with really outstanding content. Because of this OpenMW CS simply has no other choice but to support *ESP*/*ESM* files. If you decid to choose *ESP*/*ESM* file instead of using our own content file types you are most likely aimng at compatibility with the original engine. This subject is covered in it own chapter of this manual.

The actual creation of new files is described in the next chapter. Here we are going to focus only on the details you need to know in order to create your first OpenMW CS file while fully understanding your needs. For now let's jut remember that content files are created inside the user directory in the the `data` subdirectory (that is the one special data directory mentioned earlier).

Dependencies Since an addon is supposed to change the game it follows that it also depends on the said game to work. We can conceptualise this with an examples: your modification is the changing prize of an iron sword, but what if there is no iron sword in game? That's right: we get nonsense. What you want to do is tie your addon to the files you are changing. Those can be either game files (for example when making an expansion island for a game) or other addon files (making a house on said island). Obviously It is a good idea to be dependent only on files that are really changed in your addon, but sadly there is no other way to achieve this than knowing what you want to do. Again, please remember that this section of the manual does not cover creating the content files – it is only a theoretical

introduction to the subject. For now just keep in mind that dependencies exist, and is up to you to decide whether your content file should depend on other content files.

Game files are not intend to have any dependencies for a very simple reasons: the player is using only one game file (excluding original and the dirty ESP/ESM system) at a time and therefore no game file can depend on other game file, and since a game file makes the base for addon files it can not depend on addon files.

Project files Project files act as containers for data not used by the OpenMW game engine itself, but still useful for OpenMW CS. The shining example of this data category are without doubt record filters (described in a later chapter of the manual). As a mod author you probably do not need or want to distribute project files at all, they are meant to be used only by you and your team.

As you would imagine, project files make sense only in combination with actual content files. In fact, each time you start to work on new content file and a project file was not found, one will be created. The extensio of project files is `.project`. The whole name of the project file is the whole name of the content file with appended extension. For instance a `swords.omwaddon` file is associated with a `swords.omwaddon.project` file.

Project files are stored inside the user directory, in the `projects` subdirectory. This is the path location for both freshly created project files, and a place where OpenMW CS looks for already existing files.

Resource files

Unless we are talking about a fully text based game, like Zork or Rogue, one would expect that a video game is using some media files: 3D models with textures, images acting as icons, sounds and anything else. Since content files, no matter whether they are *ESP*, *ESM* or new OpenMW file type, do not contain any of those, it is clear that they have to be delivered with a different file. It is also clear that this, let's call it "resources file", has to be supported by the engine. Without code handling those files it is nothing more than a mathematical abstraction – something, that lacks meaning for human beings. Therefore this section must cover ways to add resources files to your content file, and point out what is supported. We are going to do just that. Later, you will learn how to make use of those files in your content.

Audio OpenMW uses FFmpeg for audio playback, and so we support every audio type supported by that library. This makes a huge list. Below is only small portion of the supported file types.

mp3 (MPEG-1 Part 3 Layer 3) A popular audio file format and de facto standard for storing audio. Used by the Morrowind game.

ogg An open source, multimedia container file using a high quality Vorbis audio codec. Recommended.

Video Video As in the case of audio files, we are using FFmepg to decode video files. The list of supported files is long, we will cover only the most significant.

bik Videos used by the original Morrowind game.

mp4 A multimedia container which use more advanced codecs (MPEG-4 Parts 2, 3 and 10) with a better audio and video compression rate, but also requiring more CPU intensive decoding – this makes it probably less suited for storing sounds in computer games, but good for videos.

webm A new, shiny and open source video format with excellent compression. It needs quite a lot of processing power to be decoded, but since game logic is not running during cutscenes we can recommend it for use with OpenMW.

ogv Alternative, open source container using Theora codec for video and Vorbis for audio.

Textures and images The original Morrowind game uses *DDS* and *TGA* files for all kinds of two dimensional images and textures alike. In addition, the engine supported *BMP* files for some reason (*BMP* is a terrible format for a video game). We also support an extended set of image files – including *JPEG* and *PNG*. *JPEG* and *PNG* files can be useful in some cases, for instance a *JPEG* file is a valid option for skybox texture and *PNG* can useful for masks. However, please keep in mind that *JPEG* images can grow to large sizes quickly and are not the best option with a DirectX rendering backend. You probably still want to use *DDS* files for textures.

1.2.4 OpenMW CS Starting Dialog

In this chapter we will cover starting up OpenMW CS and the starting interface. Start the CS the way intended for your operating system and you will be presented with window and three main buttons and a small button with a wrench-icon. The wrench will open the configuration dialog which we will cover later. The three main buttons are the following:

Create A New Game Choose this option if you want to create an original game that does not depend on any other content files. The distinction between game and addon in the original Morrowind engine was somewhat blurry, but OpenMW is very strict about it: regardless of how large your addon is, if it depends on another content file it is not an original game.

Create A New Addon Choose this option if you want to create an extension to an existing game. An addon can depend on other addons as well optionally, but it *must* depend on a game.

Edit A Content File Choose this option is you wish to edit an existing content file, regardless of whether it is a game or an addon.

Whether you create a game or an addon, a data file and a project file will be generated for you in you user directory.

You will have to choose a name for your content file and if you chose to create an addon you will also have to chose a number of dependencies. You have to choose exactly one game and you can choose an arbitrary amount of addon dependencies. For the sake of simplicity and maintainability choose only the addons you actually want to depend on. Also keep in mind that your dependencies might have dependencies of their own, you have to depend on those as well. If one of your dependencies nees something it will be indicated by a warning sign and automatically include its dependencies when you choose it.

If you want to edit an existing content file you will be presented with a similar dialog, except you don't get to choose a file name (because you are editing files that already exist).

CHAPTER 2

Indices and tables

- genindex
- search

M

MWBase (C++ type), 1, 2, 4–6, 10, 11, 13, 15, 20
MWBase::Detect_Creature (C++ class), 21
MWBase::Detect_Enchantment (C++ class), 20
MWBase::Detect_Key (C++ class), 21
MWBase::DetectionType (C++ type), 20
MWBase::DialogueManager (C++ class), 1
MWBase::DialogueManager::~DialogueManager (C++ function), 1
MWBase::DialogueManager::addTopic (C++ function), 1
MWBase::DialogueManager::applyDispositionChange (C++ function), 1
MWBase::DialogueManager::askQuestion (C++ function), 1
MWBase::DialogueManager::checkServiceRefused (C++ function), 1
MWBase::DialogueManager::clear (C++ function), 1
MWBase::DialogueManager::clearInfoActor (C++ function), 2
MWBase::DialogueManager::countSavedGameRecords (C++ function), 1
MWBase::DialogueManager::DialogueManager (C++ function), 1, 2
MWBase::DialogueManager::getFactionReaction (C++ function), 2
MWBase::DialogueManager::getTemporaryDispositionChange (C++ function), 1
MWBase::DialogueManager::goodbye (C++ function), 1
MWBase::DialogueManager::goodbyeSelected (C++ function), 1
MWBase::DialogueManager::isInChoice (C++ function), 1
MWBase::DialogueManager::keywordSelected (C++ function), 1
MWBase::DialogueManager::modFactionReaction (C++ function), 2
MWBase::DialogueManager::operator= (C++ function), 2
MWBase::DialogueManager::persuade (C++ function), 1
MWBase::DialogueManager::questionAnswered (C++ function), 1
MWBase::DialogueManager::readRecord (C++ function), 2
MWBase::DialogueManager::say (C++ function), 1
MWBase::DialogueManager::setFactionReaction (C++ function), 2
MWBase::DialogueManager::startDialogue (C++ function), 1
MWBase::DialogueManager::write (C++ function), 1
MWBase::Environment (C++ class), 2
MWBase::Environment::~Environment (C++ function), 2
MWBase::Environment::cleanup (C++ function), 3
MWBase::Environment::Environment (C++ function), 2, 3
MWBase::Environment::get (C++ function), 3
MWBase::Environment::getDialogueManager (C++ function), 3
MWBase::Environment::getFrameDuration (C++ function), 3
MWBase::Environment::getInputManager (C++ function), 3
MWBase::Environment::getJournal (C++ function), 3
MWBase::Environment::getMechanicsManager (C++ function), 3
MWBase::Environment::getScriptManager (C++ function), 3
MWBase::Environment::getSoundManager (C++ function), 3
MWBase::Environment::getStateManager (C++ function), 3
MWBase::Environment::getWindowManager (C++ function), 3
MWBase::Environment::getWorld (C++ function), 2
MWBase::Environment::mDialogueManager (C++ member), 3
MWBase::Environment::mFrameDuration (C++ member), 3
MWBase::Environment::mInputManager (C++ member), 3
MWBase::Environment::mJournal (C++ member), 3

MWBase::Environment::mMechanicsManager (C++ member), 3
MWBase::Environment::mScriptManager (C++ member), 3
MWBase::Environment::mSoundManager (C++ member), 3
MWBase::Environment::mStateManager (C++ member), 3
MWBase::Environment::mWindowManager (C++ member), 3
MWBase::Environment::mWorld (C++ member), 3
MWBase::Environment::operator= (C++ function), 3
MWBase::Environment::setDialogueManager (C++ function), 2
MWBase::Environment::setFrameDuration (C++ function), 2
MWBase::Environment::setInputManager (C++ function), 2
MWBase::Environment::setJournal (C++ function), 2
MWBase::Environment::setMechanicsManager (C++ function), 2
MWBase::Environment::setScriptManager (C++ function), 2
MWBase::Environment::setSoundManager (C++ function), 2
MWBase::Environment::setStateManager (C++ function), 2
MWBase::Environment::setWindowManager (C++ function), 2
MWBase::Environment::setWorld (C++ function), 2
MWBase::Environment::sThis (C++ member), 4
MWBase::InputManager (C++ class), 4
MWBase::InputManager::~InputManager (C++ function), 4
MWBase::InputManager::changeInputMode (C++ function), 4
MWBase::InputManager::clear (C++ function), 4
MWBase::InputManager::enableDetectingBindingMode (C++ function), 4
MWBase::InputManager::getActionControllerBindingName (C++ function), 4
MWBase::InputManager::getActionControllerSorting (C++ function), 4
MWBase::InputManager::getActionDescription (C++ function), 4
MWBase::InputManager::getActionKeyBindingName (C++ function), 4
MWBase::InputManager::getActionKeySorting (C++ function), 4
MWBase::InputManager::getControlSwitch (C++ function), 4
MWBase::InputManager::getNumActions (C++ function), 4
MWBase::InputManager::InputManager (C++ function), 4, 5
MWBase::InputManager::isWindowVisible (C++ function), 4
MWBase::InputManager::joystickLastUsed (C++ function), 4
MWBase::InputManager::operator= (C++ function), 5
MWBase::InputManager::processChangedSettings (C++ function), 4
MWBase::InputManager::resetToDefaultControllerBindings (C++ function), 4
MWBase::InputManager::resetToDefaultKeyBindings (C++ function), 4
MWBase::InputManager::sdlControllerAxisToString (C++ function), 4
MWBase::InputManager::sdlControllerButtonToString (C++ function), 4
MWBase::InputManager::setDragDrop (C++ function), 4
MWBase::InputManager::toggleControlSwitch (C++ function), 4
MWBase::InputManager::update (C++ function), 4
MWBase::Journal (C++ class), 5
MWBase::Journal::~Journal (C++ function), 5
MWBase::Journal::addEntry (C++ function), 5
MWBase::Journal::addTopic (C++ function), 5
MWBase::Journal::begin (C++ function), 5
MWBase::Journal::clear (C++ function), 5
MWBase::Journal::countSavedGameRecords (C++ function), 6
MWBase::Journal::end (C++ function), 5
MWBase::Journal::getJournalIndex (C++ function), 5
MWBase::Journal::Journal (C++ function), 5, 6
MWBase::Journal::operator= (C++ function), 6
MWBase::Journal::questBegin (C++ function), 6
MWBase::Journal::questEnd (C++ function), 6
MWBase::Journal::readRecord (C++ function), 6
MWBase::Journal::removeLastAddedTopicResponse (C++ function), 5
MWBase::Journal::setJournalIndex (C++ function), 5
MWBase::Journal::TEntryContainer (C++ type), 5
MWBase::Journal::TEntryIter (C++ type), 5
MWBase::Journal::topicBegin (C++ function), 6
MWBase::Journal::topicEnd (C++ function), 6
MWBase::Journal::TQuestContainer (C++ type), 5
MWBase::Journal::TQuestIter (C++ type), 5
MWBase::Journal::TTopicContainer (C++ type), 5
MWBase::Journal::TTopicIter (C++ type), 5
MWBase::Journal::write (C++ function), 6
MWBase::MechanicsManager (C++ class), 6
MWBase::MechanicsManager::~MechanicsManager (C++ function), 7
MWBase::MechanicsManager::actorAttacked (C++ function), 8

MWBase::MechanicsManager::actorKilled (C++ function), 8
MWBase::MechanicsManager::add (C++ function), 7
MWBase::MechanicsManager::advanceTime (C++ function), 7
MWBase::MechanicsManager::applyWerewolfAcrobatics (C++ function), 10
MWBase::MechanicsManager::awarenessCheck (C++ function), 8
MWBase::MechanicsManager::checkAnimationPlaying (C++ function), 9
MWBase::MechanicsManager::cleanupSummonedCreature (C++ function), 10
MWBase::MechanicsManager::clear (C++ function), 9
MWBase::MechanicsManager::commitCrime (C++ function), 8
MWBase::MechanicsManager::confiscateStolenItems (C++ function), 9
MWBase::MechanicsManager::countDeaths (C++ function), 8
MWBase::MechanicsManager::countSavedGameRecords (C++ function), 9
MWBase::MechanicsManager::drop (C++ function), 7
MWBase::MechanicsManager::forceStateUpdate (C++ function), 8
MWBase::MechanicsManager::getActorsFighting (C++ function), 9
MWBase::MechanicsManager::getActorsFollowing (C++ function), 9
MWBase::MechanicsManager::getActorsFollowingIndices (C++ function), 9
MWBase::MechanicsManager::getActorsInRange (C++ function), 9
MWBase::MechanicsManager::getActorsSidingWith (C++ function), 9
MWBase::MechanicsManager::getBarterOffer (C++ function), 7
MWBase::MechanicsManager::getDerivedDisposition (C++ function), 7
MWBase::MechanicsManager::getEnemiesNearby (C++ function), 9
MWBase::MechanicsManager::getHoursToRest (C++ function), 7
MWBase::MechanicsManager::getObjectsInRange (C++ function), 9
MWBase::MechanicsManager::getPersuasionDispositionChange (C++ function), 8
MWBase::MechanicsManager::getStolenItemOwners (C++ function), 9
MWBase::MechanicsManager::isAggressive (C++ function), 9
MWBase::MechanicsManager::isAIActive (C++ function), 9
MWBase::MechanicsManager::isAllowedToUse (C++ function), 9
MWBase::MechanicsManager::isItemStolenFrom (C++ function), 9
MWBase::MechanicsManager::isReadyToBlock (C++ function), 9
MWBase::MechanicsManager::itemTaken (C++ function), 8
MWBase::MechanicsManager::keepPlayerAlive (C++ function), 9
MWBase::MechanicsManager::MechanicsManager (C++ function), 7, 10
MWBase::MechanicsManager::objectOpened (C++ function), 8
MWBase::MechanicsManager::operator= (C++ function), 10
MWBase::MechanicsManager::playAnimationGroup (C++ function), 8
MWBase::MechanicsManager::playerLoaded (C++ function), 9
MWBase::MechanicsManager::readRecord (C++ function), 9
MWBase::MechanicsManager::remove (C++ function), 7
MWBase::MechanicsManager::rest (C++ function), 7
MWBase::MechanicsManager::setPlayerBirthsign (C++ function), 7
MWBase::MechanicsManager::setPlayerClass (C++ function), 7
MWBase::MechanicsManager::setPlayerName (C++ function), 7
MWBase::MechanicsManager::setPlayerRace (C++ function), 7
MWBase::MechanicsManager::setWerewolf (C++ function), 9
MWBase::MechanicsManager::skipAnimation (C++ function), 8
MWBase::MechanicsManager::sleepInBed (C++ function), 8
MWBase::MechanicsManager::startCombat (C++ function), 8
MWBase::MechanicsManager::toggleAI (C++ function), 9
MWBase::MechanicsManager::update (C++ function), 7
MWBase::MechanicsManager::updateCell (C++ function), 7
MWBase::MechanicsManager::updateMagicEffects (C++ function), 9
MWBase::MechanicsManager::watchActor (C++ function), 7
MWBase::MechanicsManager::write (C++ function), 9
MWBase::OffenseType (C++ type), 6
MWBase::OT_Assault (C++ class), 6
MWBase::OT_Murder (C++ class), 6
MWBase::OT_Pickpocket (C++ class), 6

MWBase::OT_SleepingInOwnedBed (C++ class), 6
MWBase::OT_Theft (C++ class), 6
MWBase::OT_Trespassing (C++ class), 6
MWBase::PersuasionType (C++ type), 6
MWBase::Play_Loop (C++ class), 11
MWBase::Play_LoopNoEnv (C++ class), 11
MWBase::Play_LoopRemoveAtDistance (C++ class), 11
MWBase::Play_NoEnv (C++ class), 11
MWBase::Play_NoPlayerLocal (C++ class), 11
MWBase::Play_Normal (C++ class), 11
MWBase::Play_RemoveAtDistance (C++ class), 11
MWBase::Play_TypeFoot (C++ class), 11
MWBase::Play_TypeMask (C++ class), 11
MWBase::Play_TypeMovie (C++ class), 11
MWBase::Play_TypeMusic (C++ class), 11
MWBase::Play_TypeSfx (C++ class), 11
MWBase::Play_TypeVoice (C++ class), 11
MWBase::PlayMode (C++ type), 11
MWBase::PlayType (C++ type), 11
MWBase::PT_Admire (C++ class), 6
MWBase::PT_Bribe10 (C++ class), 7
MWBase::PT_Bribe100 (C++ class), 7
MWBase::PT_Bribe1000 (C++ class), 7
MWBase::PT_Intimidate (C++ class), 6
MWBase::PT_Taunt (C++ class), 6
MWBase::ScriptManager (C++ class), 10
MWBase::ScriptManager::~ScriptManager (C++ function), 10
MWBase::ScriptManager::compile (C++ function), 10
MWBase::ScriptManager::compileAll (C++ function), 10
MWBase::ScriptManager::getGlobalScripts (C++ function), 10
MWBase::ScriptManager::getLocals (C++ function), 10
MWBase::ScriptManager::operator= (C++ function), 10
MWBase::ScriptManager::run (C++ function), 10
MWBase::ScriptManager::ScriptManager (C++ function), 10
MWBase::SoundManager (C++ class), 11
MWBase::SoundManager::~SoundManager (C++ function), 11
MWBase::SoundManager::clear (C++ function), 13
MWBase::SoundManager::fadeOutSound3D (C++ function), 13
MWBase::SoundManager::getSaySoundLoudness (C++ function), 12
MWBase::SoundManager::getSoundPlaying (C++ function), 13
MWBase::SoundManager::getTrackTimeDelay (C++ function), 12
MWBase::SoundManager::isMusicPlaying (C++ function), 12
MWBase::SoundManager::operator= (C++ function), 13
MWBase::SoundManager::pauseSounds (C++ function), 13
MWBase::SoundManager::playPlaylist (C++ function), 12
MWBase::SoundManager::playSound (C++ function), 12
MWBase::SoundManager::playSound3D (C++ function), 12
MWBase::SoundManager::playTrack (C++ function), 12
MWBase::SoundManager::processChangedSettings (C++ function), 11
MWBase::SoundManager::resumeSounds (C++ function), 13
MWBase::SoundManager::say (C++ function), 12
MWBase::SoundManager::sayDone (C++ function), 12
MWBase::SoundManager::setListenerPosDir (C++ function), 13
MWBase::SoundManager::SoundManager (C++ function), 11, 13
MWBase::SoundManager::startRandomTitle (C++ function), 12
MWBase::SoundManager::stopMusic (C++ function), 11
MWBase::SoundManager::stopSay (C++ function), 12
MWBase::SoundManager::stopSound (C++ function), 13
MWBase::SoundManager::stopSound3D (C++ function), 13
MWBase::SoundManager::stopTrack (C++ function), 12
MWBase::SoundManager::streamMusic (C++ function), 11
MWBase::SoundManager::update (C++ function), 13
MWBase::SoundManager::updatePtr (C++ function), 13
MWBase::SoundPtr (C++ type), 11
MWBase::SoundStreamPtr (C++ type), 11
MWBase::State (C++ type), 14
MWBase::State_Ended (C++ class), 14
MWBase::State_NoGame (C++ class), 14
MWBase::State_Running (C++ class), 14
MWBase::StateManager (C++ class), 13
MWBase::StateManager::~StateManager (C++ function), 14
MWBase::StateManager::askLoadRecent (C++ function), 14
MWBase::StateManager::characterBegin (C++ function), 15
MWBase::StateManager::characterEnd (C++ function), 15
MWBase::StateManager::CharacterIterator (C++ type), 14
MWBase::StateManager::deleteGame (C++ function), 14
MWBase::StateManager::endGame (C++ function), 14
MWBase::StateManager::getCurrentCharacter (C++ function), 14
MWBase::StateManager::getState (C++ function), 14
MWBase::StateManager::hasQuitRequest (C++ function), 14
MWBase::StateManager::loadGame (C++ function), 14
MWBase::StateManager::newGame (C++ function), 14

MWBase::StateManager::operator= (C++ function), 15
MWBase::StateManager::quickLoad (C++ function), 14
MWBase::StateManager::quickSave (C++ function), 14
MWBase::StateManager::requestQuit (C++ function), 14
MWBase::StateManager::saveGame (C++ function), 14
MWBase::StateManager::StateManager (C++ function), 14, 15
MWBase::StateManager::update (C++ function), 15
MWBase::WindowManager (C++ class), 15
MWBase::WindowManager::~WindowManager (C++ function), 15
MWBase::WindowManager::activateHitOverlay (C++ function), 20
MWBase::WindowManager::activateQuickKey (C++ function), 17
MWBase::WindowManager::addCurrentModal (C++ function), 19
MWBase::WindowManager::addVisitedLocation (C++ function), 18
MWBase::WindowManager::allow (C++ function), 16
MWBase::WindowManager::allowMouse (C++ function), 18
MWBase::WindowManager::changeCell (C++ function), 17
MWBase::WindowManager::changePointer (C++ function), 19
MWBase::WindowManager::clear (C++ function), 19
MWBase::WindowManager::configureSkills (C++ function), 16
MWBase::WindowManager::containsMode (C++ function), 16
MWBase::WindowManager::correctBookartPath (C++ function), 20
MWBase::WindowManager::correctIconPath (C++ function), 20
MWBase::WindowManager::correctTexturePath (C++ function), 20
MWBase::WindowManager::countSavedGameRecords (C++ function), 19
MWBase::WindowManager::cycleSpell (C++ function), 20
MWBase::WindowManager::cycleWeapon (C++ function), 20
MWBase::WindowManager::disallowAll (C++ function), 16
MWBase::WindowManager::disallowMouse (C++ function), 17
MWBase::WindowManager::enableRest (C++ function), 18
MWBase::WindowManager::executeInConsole (C++ function), 18
MWBase::WindowManager::exitCurrentGuiMode (C++ function), 18
MWBase::WindowManager::exitCurrentModal (C++ function), 19
MWBase::WindowManager::fadeScreenIn (C++ function), 19
MWBase::WindowManager::fadeScreenOut (C++ function), 20
MWBase::WindowManager::fadeScreenTo (C++ function), 20
MWBase::WindowManager::forceHide (C++ function), 16
MWBase::WindowManager::getConfirmationDialog (C++ function), 16
MWBase::WindowManager::getCountDialog (C++ function), 16
MWBase::WindowManager::getCursorVisible (C++ function), 19
MWBase::WindowManager::getDialogueWindow (C++ function), 16
MWBase::WindowManager::getFullHelp (C++ function), 17
MWBase::WindowManager::getGameSettingString (C++ function), 18
MWBase::WindowManager::getInventoryWindow (C++ function), 16
MWBase::WindowManager::getJournalAllowed (C++ function), 18
MWBase::WindowManager::getLoadingScreen (C++ function), 19
MWBase::WindowManager::getMode (C++ function), 16
MWBase::WindowManager::getMousePosition (C++ function), 17
MWBase::WindowManager::getPlayerAttributeValues (C++ function), 18
MWBase::WindowManager::getPlayerMajorSkills (C++ function), 18
MWBase::WindowManager::getPlayerMinorSkills (C++ function), 18
MWBase::WindowManager::getPlayerSkillValues (C++ function), 18
MWBase::WindowManager::getPlayerSleeping (C++ function), 18
MWBase::WindowManager::getRestEnabled (C++ function), 18
MWBase::WindowManager::getSelectedSpell (C++ function), 17
MWBase::WindowManager::getSubtitlesEnabled (C++ function), 17
MWBase::WindowManager::getTradeWindow (C++ function), 16
MWBase::WindowManager::getTranslationDataStorage (C++ function), 19
MWBase::WindowManager::getWorldMouseOver (C++ function), 17

MWBase::WindowManager::goToJail (C++ function), 16
MWBase::WindowManager::interactiveMessageBox (C++ function), 18
MWBase::WindowManager::isAllowed (C++ function), 16
MWBase::WindowManager::isConsoleMode (C++ function), 16
MWBase::WindowManager::isGuiMode (C++ function), 16
MWBase::WindowManager::isSavingAllowed (C++ function), 19
MWBase::WindowManager::notifyInputActionBound (C++ function), 18
MWBase::WindowManager::onFrame (C++ function), 18
MWBase::WindowManager::openContainer (C++ function), 19
MWBase::WindowManager::operator= (C++ function), 20
MWBase::WindowManager::pinWindow (C++ function), 19
MWBase::WindowManager::playVideo (C++ function), 15
MWBase::WindowManager::popGuiMode (C++ function), 15
MWBase::WindowManager::processChangedSettings (C++ function), 18
MWBase::WindowManager::pushGuiMode (C++ function), 15
MWBase::WindowManager::readPressedButton (C++ function), 18
MWBase::WindowManager::readRecord (C++ function), 19
MWBase::WindowManager::removeCell (C++ function), 20
MWBase::WindowManager::removeCurrentModal (C++ function), 19
MWBase::WindowManager::removeDialog (C++ function), 18
MWBase::WindowManager::removeGuiMode (C++ function), 16
MWBase::WindowManager::removeStaticMessageBox (C++ function), 18
MWBase::WindowManager::setActiveMap (C++ function), 17
MWBase::WindowManager::setBlindness (C++ function), 20
MWBase::WindowManager::setConsoleSelectedObject (C++ function), 16
MWBase::WindowManager::setCursorVisible (C++ function), 17
MWBase::WindowManager::setDragDrop (C++ function), 17
MWBase::WindowManager::setDrowningBarVisibility (C++ function), 17
MWBase::WindowManager::setDrowningTimeLeft (C++ function), 16
MWBase::WindowManager::setEnemy (C++ function), 19
MWBase::WindowManager::setFocusObject (C++ function), 17
MWBase::WindowManager::setFocusObjectScreenCoords (C++ function), 17
MWBase::WindowManager::setHMSVisibility (C++ function), 17
MWBase::WindowManager::setKeyFocusWidget (C++ function), 19
MWBase::WindowManager::setMinimapVisibility (C++ function), 17
MWBase::WindowManager::setNewGame (C++ function), 15
MWBase::WindowManager::setPlayerClass (C++ function), 16
MWBase::WindowManager::setSelectedEnchantItem (C++ function), 17
MWBase::WindowManager::setSelectedSpell (C++ function), 17
MWBase::WindowManager::setSelectedWeapon (C++ function), 17
MWBase::WindowManager::setSneakVisibility (C++ function), 17
MWBase::WindowManager::setSpellVisibility (C++ function), 17
MWBase::WindowManager::setValue (C++ function), 16
MWBase::WindowManager::setWeaponVisibility (C++ function), 17
MWBase::WindowManager::setWerewolfOverlay (C++ function), 20
MWBase::WindowManager::showBook (C++ function), 19
MWBase::WindowManager::showCompanionWindow (C++ function), 18
MWBase::WindowManager::showCrosshair (C++ function), 17
MWBase::WindowManager::showScroll (C++ function), 19
MWBase::WindowManager::showSoulgemDialog (C++ function), 19
MWBase::WindowManager::SkillList (C++ type), 15
MWBase::WindowManager::startEnchanting (C++ function), 19
MWBase::WindowManager::startRecharge (C++ function), 19
MWBase::WindowManager::startRepair (C++ function), 19
MWBase::WindowManager::startRepairItem (C++ function), 19

MWBase::WindowManager::startSelfEnchanting (C++ function), 19
MWBase::WindowManager::startSpellBuying (C++ function), 19
MWBase::WindowManager::startSpellMaking (C++ function), 18
MWBase::WindowManager::startTrade (C++ function), 19
MWBase::WindowManager::startTraining (C++ function), 19
MWBase::WindowManager::startTravel (C++ function), 19
MWBase::WindowManager::staticMessageBox (C++ function), 18
MWBase::WindowManager::textureExists (C++ function), 20
MWBase::WindowManager::toggleDebugWindow (C++ function), 20
MWBase::WindowManager::toggleFogOfWar (C++ function), 17
MWBase::WindowManager::toggleFullHelp (C++ function), 17
MWBase::WindowManager::toggleGui (C++ function), 17
MWBase::WindowManager::toggleVisible (C++ function), 16
MWBase::WindowManager::unsetForceHide (C++ function), 16
MWBase::WindowManager::unsetSelectedSpell (C++ function), 17
MWBase::WindowManager::unsetSelectedWeapon (C++ function), 17
MWBase::WindowManager::update (C++ function), 15
MWBase::WindowManager::updatePlayer (C++ function), 16
MWBase::WindowManager::updateSkillArea (C++ function), 17
MWBase::WindowManager::updateSpellWindow (C++ function), 16
MWBase::WindowManager::useItem (C++ function), 16
MWBase::WindowManager::wakeUpPlayer (C++ function), 18
MWBase::WindowManager::WindowManager (C++ function), 15, 20
MWBase::WindowManager::windowResized (C++ function), 18
MWBase::WindowManager::write (C++ function), 19
MWBase::WindowManager::writeFog (C++ function), 20
MWBase::World (C++ class), 20
MWBase::World::~World (C++ function), 21
MWBase::World::activate (C++ function), 28
MWBase::World::activateDoor (C++ function), 26
MWBase::World::adjustPosition (C++ function), 23
MWBase::World::adjustSky (C++ function), 21
MWBase::World::advanceTime (C++ function), 22
MWBase::World::aimToTarget (C++ function), 29
MWBase::World::allowVanityMode (C++ function), 26
MWBase::World::breakInvisibility (C++ function), 28
MWBase::World::canPlaceObject (C++ function), 25
MWBase::World::canRest (C++ function), 27
MWBase::World::castRay (C++ function), 24
MWBase::World::castSpell (C++ function), 28
MWBase::World::changeToCell (C++ function), 23
MWBase::World::changeToExteriorCell (C++ function), 23
MWBase::World::changeToInteriorCell (C++ function), 23
MWBase::World::changeVanityModeScale (C++ function), 26
MWBase::World::changeWeather (C++ function), 23
MWBase::World::clear (C++ function), 21
MWBase::World::confiscateStolenItems (C++ function), 28
MWBase::World::countSavedGameCells (C++ function), 21
MWBase::World::countSavedGameRecords (C++ function), 21
MWBase::World::createOverrideRecord (C++ function), 25
MWBase::World::createRecord (C++ function), 24, 25
MWBase::World::deleteObject (C++ function), 24
MWBase::World::disable (C++ function), 22
MWBase::World::DoorMarker (C++ class), 29
MWBase::World::DoorMarker::dest (C++ member), 29
MWBase::World::DoorMarker::name (C++ member), 29
MWBase::World::DoorMarker::x (C++ member), 29
MWBase::World::DoorMarker::y (C++ member), 29
MWBase::World::dropObjectOnGround (C++ function), 25
MWBase::World::enable (C++ function), 22
MWBase::World::enableActorCollision (C++ function), 27
MWBase::World::enableLevitation (C++ function), 27
MWBase::World::enableTeleporting (C++ function), 27
MWBase::World::explodeSpell (C++ function), 28
MWBase::World::findContainer (C++ function), 22
MWBase::World::findExteriorPosition (C++ function), 27
MWBase::World::findInteriorPosition (C++ function), 27
MWBase::World::findInteriorPositionInWorldSpace (C++ function), 28
MWBase::World::fixPosition (C++ function), 23
MWBase::World::getActorCollidingWith (C++ function), 26
MWBase::World::getActorHeadTransform (C++ function), 26

MWBase::World::getActorStandingOn (C++ function), 26
MWBase::World::getAnimation (C++ function), 27
MWBase::World::getCell (C++ function), 21
MWBase::World::getCellName (C++ function), 22
MWBase::World::getContainersOwnedBy (C++ function), 27
MWBase::World::getContentFiles (C++ function), 28
MWBase::World::getCurrentWeather (C++ function), 23
MWBase::World::getDay (C++ function), 22
MWBase::World::getDistanceToFacedObject (C++ function), 23
MWBase::World::getDistToNearestRayHit (C++ function), 27
MWBase::World::getDoorMarkers (C++ function), 21
MWBase::World::getEsmReader (C++ function), 21
MWBase::World::getExterior (C++ function), 21, 23
MWBase::World::getFacedObject (C++ function), 23
MWBase::World::getFallback (C++ function), 21
MWBase::World::getGlobalFloat (C++ function), 22
MWBase::World::getGlobalInt (C++ function), 22
MWBase::World::getGlobalVariableType (C++ function), 22
MWBase::World::getGodModeState (C++ function), 27
MWBase::World::getHitContact (C++ function), 23
MWBase::World::getHitDistance (C++ function), 29
MWBase::World::getInterior (C++ function), 21
MWBase::World::getItemsOwnedBy (C++ function), 27
MWBase::World::getLocalScripts (C++ function), 21
MWBase::World::getLOS (C++ function), 27
MWBase::World::getMasserPhase (C++ function), 23
MWBase::World::getMaxActivationDistance (C++ function), 23
MWBase::World::getMonth (C++ function), 22
MWBase::World::getMonthName (C++ function), 22
MWBase::World::getNorthVector (C++ function), 21
MWBase::World::getPlayer (C++ function), 21
MWBase::World::getPlayerCollidingWith (C++ function), 26
MWBase::World::getPlayerPtr (C++ function), 21
MWBase::World::getPlayerStandingOn (C++ function), 26
MWBase::World::getPtr (C++ function), 22
MWBase::World::getScriptsEnabled (C++ function), 27
MWBase::World::getSecundaPhase (C++ function), 23
MWBase::World::getStore (C++ function), 21
MWBase::World::getStormDirection (C++ function), 28
MWBase::World::getTimeScaleFactor (C++ function), 23
MWBase::World::getTimeStamp (C++ function), 23
MWBase::World::getWindSpeed (C++ function), 27
MWBase::World::getYear (C++ function), 22
MWBase::World::goToJail (C++ function), 28
MWBase::World::hasCellChanged (C++ function), 21
MWBase::World::hurtCollidingActors (C++ function), 26
MWBase::World::hurtStandingActors (C++ function), 26
MWBase::World::indexToPosition (C++ function), 24
MWBase::World::isCellExterior (C++ function), 21
MWBase::World::isCellQuasiExterior (C++ function), 21
MWBase::World::isDark (C++ function), 28
MWBase::World::isFirstPerson (C++ function), 26
MWBase::World::isFlying (C++ function), 26
MWBase::World::isInStorm (C++ function), 28
MWBase::World::isLevitationEnabled (C++ function), 27
MWBase::World::isOnGround (C++ function), 26
MWBase::World::isPlayerInJail (C++ function), 29
MWBase::World::isSlowFalling (C++ function), 26
MWBase::World::isSubmerged (C++ function), 26
MWBase::World::isSwimming (C++ function), 26
MWBase::World::isTeleportingEnabled (C++ function), 27
MWBase::World::isUnderwater (C++ function), 26
MWBase::World::isWading (C++ function), 26
MWBase::World::isWalkingOnWater (C++ function), 29
MWBase::World::launchMagicBolt (C++ function), 28
MWBase::World::launchProjectile (C++ function), 28
MWBase::World::listDetectedReferences (C++ function), 28
MWBase::World::markCellAsUnchanged (C++ function), 23
MWBase::World::modRegion (C++ function), 23
MWBase::World::moveObject (C++ function), 24
MWBase::World::operator= (C++ function), 29
MWBase::World::placeObject (C++ function), 24, 25
MWBase::World::positionToIndex (C++ function), 24
MWBase::World::preloadCommonAssets (C++ function), 21
MWBase::World::processChangedSettings (C++ function), 25
MWBase::World::queueMovement (C++ function), 24
MWBase::World::readRecord (C++ function), 21
MWBase::World::reattachPlayerCamera (C++ function), 27
MWBase::World::removeContainerScripts (C++ function), 29
MWBase::World::removeRefScript (C++ function), 22
MWBase::World::renderPlayer (C++ function), 26
MWBase::World::resetActors (C++ function), 29
MWBase::World::rotateObject (C++ function), 24
MWBase::World::safePlaceObject (C++ function), 24
MWBase::World::scaleObject (C++ function), 24
MWBase::World::screenshot (C++ function), 27
MWBase::World::searchPtr (C++ function), 22
MWBase::World::searchPtrViaActorId (C++ function), 22
MWBase::World::setCameraDistance (C++ function), 26

MWBase::World::setDay (C++ function), 22
MWBase::World::setGlobalFloat (C++ function), 22
MWBase::World::setGlobalInt (C++ function), 22
MWBase::World::setHour (C++ function), 22
MWBase::World::setMonth (C++ function), 22
MWBase::World::setMoonColour (C++ function), 23
MWBase::World::setupPlayer (C++ function), 26
MWBase::World::setWaterHeight (C++ function), 21
MWBase::World::spawnBloodEffect (C++ function), 28
MWBase::World::spawnEffect (C++ function), 28
MWBase::World::spawnRandomCreature (C++ function), 28
MWBase::World::startNewGame (C++ function), 21
MWBase::World::startSpellCast (C++ function), 27
MWBase::World::teleportToClosestMarker (C++ function), 28
MWBase::World::toggleCollisionMode (C++ function), 24
MWBase::World::toggleGodMode (C++ function), 27
MWBase::World::togglePlayerLooking (C++ function), 26
MWBase::World::togglePOV (C++ function), 26
MWBase::World::togglePreviewMode (C++ function), 26
MWBase::World::toggleRenderMode (C++ function), 24
MWBase::World::toggleScripts (C++ function), 27
MWBase::World::toggleSky (C++ function), 23
MWBase::World::toggleVanityMode (C++ function), 26
MWBase::World::toggleWater (C++ function), 21
MWBase::World::toggleWorld (C++ function), 21
MWBase::World::undeleteObject (C++ function), 24
MWBase::World::update (C++ function), 25
MWBase::World::updateDialogueGlobals (C++ function), 28
MWBase::World::useDeathCamera (C++ function), 21
MWBase::World::vanityRotateCamera (C++ function), 26
MWBase::World::World (C++ function), 21, 29
MWBase::World::write (C++ function), 21
MWGui (C++ type), 15
MWGui::ShowInDialogueMode (C++ type), 15
MWGui::ShowInDialogueMode_IfPossible (C++ class), 15
MWGui::ShowInDialogueMode_Never (C++ class), 15
MWGui::ShowInDialogueMode_Only (C++ class), 15
MWSound (C++ type), 10
MWSound::DecoderPtr (C++ type), 11
MWWorld (C++ type), 20
MWWorld::PtrMovementList (C++ type), 20

O

OMW (C++ type), 29
OMW::Engine (C++ class), 29
OMW::Engine::~Engine (C++ function), 29
OMW::Engine::addArchive (C++ function), 29
OMW::Engine::addContentFile (C++ function), 30
OMW::Engine::createWindow (C++ function), 31
OMW::Engine::enableFontExport (C++ function), 30
OMW::Engine::enableFSStrict (C++ function), 29
OMW::Engine::Engine (C++ function), 29, 31
OMW::Engine::executeLocalScripts (C++ function), 31
OMW::Engine::frame (C++ function), 31
OMW::Engine::go (C++ function), 30
OMW::Engine::loadSettings (C++ function), 31
OMW::Engine::mActivationDistanceOverride (C++ member), 31
OMW::Engine::mArchives (C++ member), 31
OMW::Engine::mCellName (C++ member), 31
OMW::Engine::mCfgMgr (C++ member), 32
OMW::Engine::mCompileAll (C++ member), 31
OMW::Engine::mCompileAllDialogue (C++ member), 31
OMW::Engine::mContentFiles (C++ member), 31
OMW::Engine::mDataDirs (C++ member), 31
OMW::Engine::mEncoder (C++ member), 31
OMW::Engine::mEncoding (C++ member), 31
OMW::Engine::mEnvironment (C++ member), 31
OMW::Engine::mExportFonts (C++ member), 32
OMW::Engine::mExtensions (C++ member), 32
OMW::Engine::mFallbackMap (C++ member), 31
OMW::Engine::mFileCollections (C++ member), 32
OMW::Engine::mFocusName (C++ member), 31
OMW::Engine::mFSStrict (C++ member), 32
OMW::Engine::mGrab (C++ member), 32
OMW::Engine::mNewGame (C++ member), 32
OMW::Engine::mResDir (C++ member), 31
OMW::Engine::mResourceSystem (C++ member), 31
OMW::Engine::mSaveGameFile (C++ member), 32
OMW::Engine::mScreenCaptureHandler (C++ member), 31
OMW::Engine::mScriptBlacklist (C++ member), 32
OMW::Engine::mScriptBlacklistUse (C++ member), 32
OMW::Engine::mScriptConsoleMode (C++ member), 31
OMW::Engine::mScriptContext (C++ member), 32
OMW::Engine::mSkipMenu (C++ member), 31
OMW::Engine::mStartTick (C++ member), 32
OMW::Engine::mStartupScript (C++ member), 31
OMW::Engine::mTranslationDataStorage (C++ member), 32
OMW::Engine::mUseSound (C++ member), 31
OMW::Engine::mVerboseScripts (C++ member), 31
OMW::Engine::mVFS (C++ member), 31
OMW::Engine::mViewer (C++ member), 31
OMW::Engine::mWarningsMode (C++ member), 31
OMW::Engine::mWindow (C++ member), 31
OMW::Engine::operator= (C++ function), 31
OMW::Engine::prepareEngine (C++ function), 31

OMW::Engine::setActivationDistanceOverride (C++ function), 30
OMW::Engine::setCell (C++ function), 30
OMW::Engine::setCompileAll (C++ function), 30
OMW::Engine::setCompileAllDialogue (C++ function), 30
OMW::Engine::setDataDirs (C++ function), 29
OMW::Engine::setEncoding (C++ function), 30
OMW::Engine::setFallbackValues (C++ function), 30
OMW::Engine::setGrabMouse (C++ function), 30
OMW::Engine::setResourceDir (C++ function), 30
OMW::Engine::setSaveGameFile (C++ function), 30
OMW::Engine::setScriptBlacklist (C++ function), 30
OMW::Engine::setScriptBlacklistUse (C++ function), 30
OMW::Engine::setScriptConsoleMode (C++ function), 30
OMW::Engine::setScriptsVerbosity (C++ function), 30
OMW::Engine::setSkipMenu (C++ function), 30
OMW::Engine::setSoundUsage (C++ function), 30
OMW::Engine::setStartupScript (C++ function), 30
OMW::Engine::setWarningsMode (C++ function), 30
OMW::Engine::setWindowIcon (C++ function), 31