libloot Documentation

Release latest

WrinklyNinja

May 06, 2023

API DOCUMENTATION

CHAPTER

ONE

INTRODUCTION

LOOT is a utility that helps users avoid serious conflicts between their mods by setting their plugins in an optimal load order. It also provides tens of thousands of plugin-specific messages, including usage notes, requirements, incompatibilities, bug warnings and installation mistake notifications, and thousands of Bash Tag suggestions.

This metadata that LOOT supplies is stored in its masterlist, which is maintained by the LOOT team using information provided by mod authors and users. Users can also add to and modify the metadata used by LOOT through the use of userlist files. libloot provides all of LOOT's non-UI-related functionality, and can be used by third-party developers to access this metadata for use in their own programs.

CHAPTER

TWO

MISCELLANEOUS DETAILS

2.1 String Encoding

- All output strings are encoded in UTF-8.
- Metadata files are written encoded in UTF-8.
- Input strings are expected to be encoded in UTF-8.
- Metadata files read are expected to be encoded in UTF-8.
- File paths are case-sensitive if and only if the underlying file system is case-sensitive.

2.2 Language Codes

All language strings in the API are codes of the form `ll` or `ll_CC`, where `ll` is an ISO 639-1 language code and `CC` is an ISO 3166 country code. For example, the default language for metadata message content is English, identified by the code `en`, and Brazilian Portuguese is `pt_BR`.

2.3 Errors

All errors encountered are thrown as exceptions that inherit from `std::exception`.

2.4 Metadata Files

LOOT stores plugin metadata in YAML files. It distinguishes between three types of metadata file:

- *masterlist* files: each game has a single masterlist, which is a public, curated metadata store
- *masterlist prelude* files: there is a single masterlist prelude, which is a public store of common metadata templates that can be shared across all masterlists
- *userlist* files: each game has a userlist, which is a private user-specific metadata store containing metadata added by the LOOT user.

All three files use the same syntax, but the masterlist prelude file is used to replace part of a masterlist file before it is parsed, and metadata in the userlist extends or replaces metadata sourced from the masterlist.

LOOT's plugin metadata can be conditional, eg. a plugin may require a patch only if another plugin is also present. The API's *loot::DatabaseInterface::LoadLists()* method parses metadata files into memory, but does not

evaluate these conditions, so the loaded metadata may contain metadata that is invalid for the installed game that the *loot::DatabaseInterface* object being operated on was created for.

2.5 Caching

All unevaluated metadata is cached between calls to *loot::DatabaseInterface::LoadLists()*.

The results of evaluating metadata conditions are cached between calls to *loot::GameInterface::LoadPlugins()*, *loot::GameInterface::SortPlugins()* and *loot::DatabaseInterface::GetGeneralMessages()*.

Plugin content is cached between calls to *loot::GameInterface::LoadPlugins()* and *loot::GameInterface::SortPlugins()*.

Load order is cached between calls to *loot::GameInterface::LoadCurrentLoadOrderState()*.

2.6 Performance

The following may involve filesystem access and reading/parsing or writing of data from the filesystem:

- Any function that takes a `std::filesystem::path`
- *loot::GameInterface::IsValidPlugin()*
- *loot::GameInterface::LoadPlugins()*
- *loot::GameInterface::LoadCurrentLoadOrderState()*
- *loot::GameInterface::SetLoadOrder()*

Evaluating conditions may also involve filesystem read access.

loot::GameInterface::SortPlugins() is expensive, as it involves loading all the content of all the plugins, apart from the game's main master file, which is skipped as an optimisation (it doesn't depend on anything else and is much bigger than any other plugin, so is unnecessary and slow to load).

loot::DatabaseInterface::GetGroupsPath() involves building a graph of all defined groups and then using it to search for the shortest path between the two given groups, which may be relatively slow given a sufficiently large and/or complex set of group definitions.

All other API functions should be relatively fast.

CHAPTER

THREE

LOOT'S SORTING ALGORITHM

LOOT's sorting algorithm consists of the following stages:

3.1 Load plugin data

In this first stage, the plugins to be sorted are parsed and their record IDs (which are FormIDs for all games apart from Morrowind) are stored. When parsing plugins, all subrecords are skipped over for efficiency, apart from the subrecords of the `TES4` header record.

Loading plugin data also involves loading any metadata that the plugin may have in the masterlist and userlist.

3.2 Create plugin graph vertices

Once the plugins have been loaded, they are sorted into lexicographical order.

After that, two graphs are created, and the plugins are added to them as vertices in their sorted order. Plugins that have their master flag set go in one graph, and plugins that do not have the flag set go in the other.

Two graphs are used because master-flagged plugins must always load before non-master-flagged plugins, and it's much more efficient to sort them separately and then combine their load orders than to enforce those relationships within a single graph.

A consequence of using two separate graphs is that any plugin data or metadata that involves a pair of plugins with and without their master flag set will be silently ignored. For example: if plugin A is master-flagged and plugin B is

not, and plugin A has metadata saying it must load after plugin B, then that metadata will be ignored because the two plugins are sorted independently, as if the other plugin is not installed.

3.3 Create plugin graph edges

The steps described in this section are run on both graphs independently.

In this section, the terms *vertex* and *plugin* are used interchangeably, and the iteration order 'for each plugin' is the order in which the vertices were added to the graph.

For each plugin:

1. If the plugin is a master file, add edges going to all non-master files. If the plugin is a non-master file, add edges coming from all master files. This shouldn't result in any edges being added, since masters and non-masters are sorted in separate graphs, but is done for completeness.
2. Add edges coming from all the plugin's masters. Missing masters have no edges added.
3. Add edges coming from all the plugin's requirements. Missing requirements have no edges added.
4. Add edges coming from all the plugin's load after files that are installed plugins.

3.3.1 Hardcoded edges

Some games hardcode certain plugins to load in certain positions, and this section adds edges in the correct order between those plugins, and between those plugins and the rest of the plugins in the graph.

3.3.2 Group edges

For each plugin, the plugins that are members of groups that the current plugin's group loads after are iterated over and individually checked to see if adding an edge from the other group's plugin to the current plugin would cause a cycle. If not, the edge is queued for addition. If it would cause a cycle and one of the plugins is in the default group and the other group's plugin is master-flagged or the current plugin is not master flagged, then the plugin in the default group is recorded as one to skip adding edges to or from when the prospective edge involves any of the groups in the path from the other plugin to the current plugin.

Once all the plugins have been iterated over, all the queued edges are added, skipping those edges identified in the earlier loop.

At this point the plugin graph is checked for cycles, and an error is thrown if any are encountered, so that metadata (or indeed plugin data) that cause them can be corrected.

3.3.3 Overlap edges

Plugin overlap edges are then added. Two plugins overlap if they contain the same record, i.e. if they both edit the same record or if one edits a record the other plugin adds. Plugins also overlap if they both load one or more BSAs (BA2s for Fallout 4) and the BSAs loaded by one plugin contain data for a file path that is also included in the BSAs loaded by the other plugin.

For each plugin, skip it if it overrides no records, otherwise iterate over all other plugins.

- If the plugin and other plugin override the same number of records and the same number of assets, or do not overlap, skip the other plugin.

- Otherwise, add an edge from the plugin which overrides more records to the plugin that overrides fewer records, unless that edge would cause a cycle. If the plugins don't have overlapping records or override the same number of records, the edge is added from the plugin that loads more assets via its BSAs to the plugin that loads fewer assets.

For Morrowind, identifying which records override others requires all of a plugin's masters to be installed, so if a plugin has missing masters, its total record count is used in place of its override record count. Morrowind plugins also can't load BSAs, so they can't have overlapping assets.

3.3.4 Tie-break edges

Finally, tie-break edges are added to ensure that sorting is consistent. The graph's vertices are sorted into their current load order:

- If both plugins have positions in the current load order, the function preserves their existing relative order.
- If one plugin has a position and the other does not, the plugin with a position goes before the plugin without a position.
- If neither plugin has a load order position, a case-insensitive lexicographical comparison of their filenames without file extensions is used to decide their order. If they are equal, a case-insensitive lexicographical comparison of their file extensions is used.

Once sorted, they are iterated over. Each loop looks at the current vertex and the next one following it (e.g. the first iteration is for vertices 0 and 1, the second is for 1 and 2, etc.).

For each (`current`, `next`) pair of vertices, try to find a path from `next` to `current`.

If sorting makes no changes, then there won't be any paths found and it'll therefore be possible to add an edge from `current` to `next` without causing a cycle, producing the old load order.

If no path is found then that means the old load order can be used for those two plugins. If the `current` vertex has not already been processed (which will be the case unless it appeared in a path found earlier and had its position pinned, see below), append it to a list representing the new load order and record the vertex as having been processed.

If no path is found but the `current` vertex has been processed and is not the last vertex in the new load order list, pin the position of the `next` vertex (see below).

If a path is found then that means the old load order for those two plugins (which is `current` before `next`) can't be used. If `current` is the first vertex in the iteration order, then `next` is simply treated as the start of the new load order. If `current` is not the first vertex, iterate over the vertices in the path found, going from `next` to `current`, and pin each vertex's position.

Pinning vertex positions

A vertex's position needs to be pinned when it must go somewhere before the last plugin in the new load order list, because although it has a fixed position relative to that last plugin, it doesn't necessarily have a fixed position relative to the plugins that come before the last plugin. I.e. it needs to load earlier, but how much earlier?

To pin a vertex's position, iterate over the new load order list in reverse order, going from the last vertex towards the first, and stop at the first load order vertex for which there is no path going from the unpinned vertex to the load order vertex. This is equivalent to finding the last plugin that the unpinned vertex's plugin can load after (which is not necessarily the same as the last plugin it *must* load after).

If such a load order vertex is found, add an edge going from it to the unpinned vertex. If the found vertex is not the last vertex in the load order list, also add an edge going from the unpinned vertex to the vertex after the found vertex. Then record the unpinned vertex's new position in the new load order list: the vertex is now pinned.

3.4 Topologically sort the plugin graphs

This is done for both graphs independently.

Note that edges for explicit interdependencies are the only edges allowed to create cycles. However, the graph is again checked for cycles to guard against potential logic bugs, and if a cycle is encountered an error is thrown.

Once the graph is confirmed to be cycle-free, a topological sort is performed on the graph, outputting a list of plugins in their newly-sorted load order.

3.5 Combine the two load orders

Finally, the sorted load order for non-master-flagged plugins is appended to the sorted load order for master-flagged plugins to give the complete sorted load order.

CHAPTER

FOUR

API REFERENCE

Contents

4.1 Constants

constexpr unsigned int `loot::LIBLOOT_VERSION_MAJOR` = 0
: libloot's major version number.

constexpr unsigned int `loot::LIBLOOT_VERSION_MINOR` = 19
: libloot's minor version number.

constexpr unsigned int `loot::LIBLOOT_VERSION_PATCH` = 4
: libloot's patch version number.

4.2 Enumerations

enum loot::**EdgeType**
An enum representing the different possible types of interactions between plugins or groups.

Values:

enumerator **hardcoded**

enumerator **masterFlag**

enumerator **master**

enumerator **masterlistRequirement**

enumerator **userRequirement**

enumerator **masterlistLoadAfter**

enumerator **userLoadAfter**

enumerator **masterlistGroup**

enumerator **userGroup**

enumerator **recordOverlap**

enumerator **assetOverlap**

enumerator **tieBreak**

enum loot::**GameType**
Codes used to create database handles for specific games.

Values:

enumerator **tes4**
The Elder Scrolls IV: Oblivion

enumerator **tes5**
The Elder Scrolls V: Skyrim

enumerator **fo3**
Fallout 3

enumerator **fonv**
Fallout: New Vegas

enumerator **fo4**
Fallout 4

enumerator **tes5se**
The Elder Scrolls V: Skyrim Special Edition

enumerator **fo4vr**
Fallout 4 VR

enumerator **tes5vr**
Skyrim VR

enumerator **tes3**
The Elder Scrolls III: Morrowind

enum `loot::LogLevel`
Codes used to specify different levels of API logging.

Values:

enumerator **trace**

enumerator **debug**

enumerator **info**

enumerator **warning**

enumerator **error**

enumerator **fatal**

enum `loot::MessageType`
Codes used to indicate the type of a message.

Values:

enumerator **say**
A notification message that is of no significant severity.

enumerator **warn**
A warning message, used to indicate that an issue may be present that the user may wish to act on.

enumerator **error**
An error message, used to indicate that an issue that requires user action is present.

4.3 Public-Field Data Structures

struct `loot::SimpleMessage`
A structure that holds the type of a message and the message string itself.

Public Members

MessageType **type** = {*MessageType*::*say*}
The type of the message.

std::string **language**
The language the message string is written in.

std::string **text**
The message string, which may be formatted using CommonMark.

std::string **condition**
The message's condition string.

4.4 Functions

void `loot::SetLoggingCallback`(std::function<void(*LogLevel*, const char*)> callback)
Set the callback function that is called when logging.

If this function is not called, the default behaviour is to print messages to the console.

Parameters **callback** – The function called when logging. The first parameter is the level of the message being logged, and the second is the message.

bool `loot::IsCompatible`(const unsigned int major, const unsigned int minor, const unsigned int patch)
Checks for API compatibility.

Checks whether the loaded API is compatible with the given version of the API, abstracting API stability policy away from clients. The version numbering used is major.minor.patch.

Parameters

- **major** – The major version number to check.
- **minor** – The minor version number to check.
- **patch** – The patch version number to check.

Returns True if the API versions are compatible, false otherwise.

std::unique_ptr<*GameInterface*> `loot::CreateGameHandle`(const *GameType* game, const std::filesystem::path &game_path, const std::filesystem::path &game_local_path = "")
Initialise a new game handle.

Creates a handle for a game, which is then used by all game-specific functions.

Parameters

- **game** – A game code for which to create the handle.
- **game_path** – The relative or absolute path to the directory containing the game's executable.
- **game_local_path** – The relative or absolute path to the game's folder in `%LOCALAPPDATA%` or an empty path. If an empty path, the API will attempt to look up the path that `%LOCALAPPDATA%` corresponds to. This parameter is provided so that systems lacking that environmental variable (eg. Linux) can still use the API.

Returns The new game handle.

std::string `loot::GetLiblootVersion`()
Get the library version.

Returns A string of the form "major.minor.patch".

std::string `loot::GetLiblootRevision`()
Get the source control revision that libloot was built from.

Returns A string containing the revision ID.

std::optional<*MessageContent*> `loot::SelectMessageContent`(const std::vector<*MessageContent*> content, const std::string &language)
Choose a *MessageContent* object from a vector given a language.

Parameters

- **content** – The *MessageContent* objects to choose between.
- **language** – The locale or language code for the preferred language to select. Locale codes are of the form `[language code]_[country code]`.

Returns A *MessageContent* object.

- If the vector only contains a single element, that element is returned.
- If content with a language that exactly matches the given locale or language code is present, that content is returned.
- If a locale code is given and there is no exact match but content for that locale's language is present, that content is returned.
- If a language code is given and there is no exact match but content for a locale in that langauge is present, that content is returned.
- If no locale or language code matches are found and content in the default language is present, that content is returned.
- Otherwise, an empty optional is returned.

std::optional<*SimpleMessage*> `loot::ToSimpleMessage`(const *Message* &message, const std::string &language)

Get a given *Message* as a *SimpleMessage* given a language.

Parameters

- **message** – The message to convert.
- **language** – The preferred language for the message content.

Returns A *SimpleMessage* object for the preferred language, or for English if message text is not available for the given language.

std::vector<*SimpleMessage*> `loot::ToSimpleMessages`(const std::vector<*Message*> &messages, const std::string &language)

Get the given messages as simple messages given a language.

Parameters

- **messages** – The messages to convert.
- **language** – The preferred language for the message content.

Returns A vector of *SimpleMessage* objects for the preferred language, or for English if message text is not available for the given language. The order of the input *Message* objects is preserved, though any messages without the preferred language or English content will be omitted.

4.5 Interfaces

class `loot::DatabaseInterface`

The interface provided by API's database handle.

Data Reading & Writing

virtual void **LoadLists**(const std::filesystem::path &masterlist_path, const std::filesystem::path &userlist_path = "", const std::filesystem::path &masterlist_prelude_path = "") = 0

Loads the masterlist, userlist and masterlist prelude from the paths specified.

Can be called multiple times, each time replacing the previously-loaded data.

Parameters

- **masterlist_path** – The relative or absolute path to the masterlist file that should be loaded.
- **userlist_path** – The relative or absolute path to the userlist file that should be loaded, or an empty path. If an empty path, no userlist will be loaded.
- **masterlist_prelude_path** – The relative or absolute path to the masterlist prelude file that should be loaded. If an empty path, no masterlist prelude will be loaded.

virtual void **WriteUserMetadata**(const std::filesystem::path &outputFile, const bool overwrite) const = 0

Writes a metadata file containing all loaded user-added metadata.

Parameters

- **outputFile** – The path to which the file shall be written.
- **overwrite** – If `false` and `outputFile` already exists, no data will be written. Otherwise, data will be written.

virtual void **WriteMinimalList**(const std::filesystem::path &outputFile, const bool overwrite) const = 0

Writes a minimal metadata file that only contains plugins with Bash *Tag* suggestions and/or dirty info, plus the suggestions and info themselves.

Parameters

- **outputFile** – The path to which the file shall be written.
- **overwrite** – If `false` and `outputFile` already exists, no data will be written. Otherwise, data will be written.

Non-plugin Data Access

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

Gets the Bash Tags that are listed in the loaded metadata lists.

Bash *Tag* suggestions can include plugins not in this list.

Returns A set of Bash *Tag* names.

virtual std::vector<*Message*> **GetGeneralMessages**(bool evaluateConditions = false) const = 0

Get all general messages listen in the loaded metadata lists.

Parameters **evaluateConditions** – If true, any metadata conditions are evaluated before the metadata is returned, otherwise unevaluated metadata is returned. Evaluating general message conditions also clears the condition cache before evaluating conditions.

Returns A vector of messages supplied in the metadata lists but not attached to any particular plugin.

virtual std::vector<*Group*> **GetGroups**(bool includeUserMetadata = true) const = 0

Gets the groups that are defined in the loaded metadata lists.

Parameters **includeUserMetadata** – If true, any group metadata present in the userlist is included in the returned metadata, otherwise the metadata returned only includes metadata from the masterlist.

Returns An vector of *Group* objects. Each *Group*'s name is unique, if a group has masterlist and user metadata the two are merged into a single group object.

virtual std::vector<*Group*> **GetUserGroups**() const = 0
Gets the groups that are defined or extended in the loaded userlist.

Returns An unordered set of *Group* objects.

virtual void **SetUserGroups**(const std::vector<*Group*> &groups) = 0
Sets the group definitions to store in the userlist, overwriting any existing definitions there.

Parameters **groups** – The unordered set of *Group* objects to set.

virtual std::vector<*Vertex*> **GetGroupsPath**(const std::string &fromGroupName, const std::string &toGroupName) const = 0
Get the "shortest" path between the two given groups according to their load after metadata.

The "shortest" path is defined as the path that maximises the amount of user metadata involved while minimising the amount of masterlist metadata involved. It's not the path involving the fewest groups.

Parameters

- **fromGroupName** – The name of the source group, that loads earlier.
- **toGroupName** – The name of the destination group, that loads later.

Returns A vector of *Vertex* elements representing the path from the source group to the destination group, or an empty vector if no path exists.

Plugin Data Access

virtual std::optional<*PluginMetadata*> **GetPluginMetadata**(const std::string &plugin, bool includeUserMetadata = true, bool evaluateConditions = false) const = 0
Get all a plugin's loaded metadata.

Parameters

- **plugin** – The filename of the plugin to look up metadata for.
- **includeUserMetadata** – If true, any user metadata the plugin has is included in the returned metadata, otherwise the metadata returned only includes metadata from the masterlist.
- **evaluateConditions** – If true, any metadata conditions are evaluated before the metadata is returned, otherwise unevaluated metadata is returned. Evaluating plugin metadata conditions does not clear the condition cache.

Returns If the plugin has metadata, an optional containing that metadata, otherwise an optional containing no value.

virtual std::optional<*PluginMetadata*> **GetPluginUserMetadata**(const std::string &plugin, bool evaluateConditions = false) const = 0
Get a plugin's metadata loaded from the given userlist.

Parameters

- **plugin** – The filename of the plugin to look up user-added metadata for.

- **evaluateConditions** – If true, any metadata conditions are evaluated before the metadata is returned, otherwise unevaluated metadata is returned. Evaluating plugin metadata conditions does not clear the condition cache.

Returns If the plugin has user-added metadata, an optional containing that metadata, otherwise an optional containing no value.

virtual void **SetPluginUserMetadata**(const *PluginMetadata* &pluginMetadata) = 0
Sets a plugin's user metadata, overwriting any existing user metadata.

Parameters **pluginMetadata** – The user metadata you want to set, with plugin.Name() being the filename of the plugin the metadata is for.

virtual void **DiscardPluginUserMetadata**(const std::string &plugin) = 0
Discards all loaded user metadata for the plugin with the given filename.

Parameters **plugin** – The filename of the plugin for which all user-added metadata should be deleted.

virtual void **DiscardAllUserMetadata**() = 0
Discards all loaded user metadata for all plugins, and any user-added general messages and known bash tags.

class loot::**GameInterface**
The interface provided for accessing game-specific functionality.

Metadata Access

virtual *DatabaseInterface* &**GetDatabase**() = 0
Get the database interface used for accessing metadata-related functionality.

Returns A reference to the game's *DatabaseInterface*. The reference remains valid for the lifetime of the *GameInterface* instance.

Plugin Data Access

virtual bool **IsValidPlugin**(const std::string &pluginPath) const = 0
Check if a file is a valid plugin.

The validity check is not exhaustive: it checks that the file extension is `.esm` or `.esp` (after trimming any `.ghost` extension), and that the `TES4` header can be parsed.

Parameters **pluginPath** – The path to the file to check. Relative paths are resolved relative to the game's plugins directory, while absolute paths are used as given.

Returns True if the file is a valid plugin, false otherwise.

virtual void **LoadPlugins**(const std::vector<std::string> &pluginPaths, bool loadHeadersOnly) = 0
Parses plugins and loads their data.

Any previously-loaded plugin data is discarded when this function is called.

Parameters

- **pluginPaths** – The plugin paths to load. Relative paths are resolved relative to the game's plugins directory, while absolute paths are used as given. Each plugin filename must be unique within the vector.

- **loadHeadersOnly** – If true, only the plugins' TES4 headers are loaded. If false, all records in the plugins are parsed, apart from the main master file if it has been identified by a previous call to *IdentifyMainMasterFile()*.

virtual const *PluginInterface* ***GetPlugin**(const std::string &pluginName) const = 0
Get data for a loaded plugin.

Parameters **pluginName** – The filename of the plugin to get data for.

Returns A shared pointer to a const *PluginInterface* implementation. The pointer is null if the given plugin has not been loaded.

virtual std::vector<const *PluginInterface**> **GetLoadedPlugins**() const = 0
Get a set of const references to all loaded plugins' *PluginInterface* objects.

Returns A set of const *PluginInterface* references. The references remain valid until the *LoadPlugins()* or *SortPlugins()* functions are next called or this *GameInterface* is destroyed.

Sorting

virtual void **IdentifyMainMasterFile**(const std::string &masterFile) = 0
Identify the game's main master file.

When sorting, LOOT always only loads the headers of the game's main master file as a performance optimisation.

virtual std::vector<std::string> **SortPlugins**(const std::vector<std::string> &pluginPaths) = 0
Calculates a new load order for the game's installed plugins (including inactive plugins) and outputs the sorted order.

Pulls metadata from the masterlist and userlist if they are loaded, and reads the contents of each plugin. No changes are applied to the load order used by the game. This function does not load or evaluate the masterlist or userlist.

Parameters **plugins** – The plugin paths to sort, in their current load order. Relative paths are resolved relative to the game's plugins directory, while absolute paths are used as given. Each plugin filename must be unique within the vector.

Returns A vector of the given plugin filenames in their sorted load order.

Load Order Interaction

virtual void **LoadCurrentLoadOrderState**() = 0
Load the current load order state, discarding any previously held state.

This function should be called whenever the load order or active state of plugins "on disk" changes, so that the cached state is updated to reflect the changes.

virtual bool **IsLoadOrderAmbiguous**() const = 0
Check if the load order is ambiguous.

This checks that all plugins in the current load order state have a well-defined position in the "on disk" state, and that all data sources are consistent. If the load order is ambiguous, different applications may read different load orders from the same source data.

Returns True if the load order is ambiguous, false otherwise.

virtual std::filesystem::path **GetActivePluginsFilePath**() const = 0
Gets the path to the file that holds the list of active plugins.

The active plugins file path is often within the game's local path, but its name and location varies by game and game configuration, so this function exposes the path that libloot uses.

Returns The file path.

virtual bool **IsPluginActive**(const std::string &plugin) const = 0
Check if a plugin is active.

Parameters **plugin** – The filename of the plugin for which to check the active state.

Returns True if the plugin is active, false otherwise.

virtual std::vector<std::string> **GetLoadOrder**() const = 0
Get the current load order.

Returns A vector of plugin filenames in their load order.

virtual void **SetLoadOrder**(const std::vector<std::string> &loadOrder) = 0
Set the game's load order.

Parameters **loadOrder** – A vector of plugin filenames sorted in the load order to set.

class loot::**PluginInterface**
Represents a plugin file that has been parsed by LOOT.

Public Functions

virtual std::string **GetName**() const = 0
Get the plugin's filename.

Returns The plugin filename.

virtual std::optional<float> **GetHeaderVersion**() const = 0
Get the value of the version field in the HEDR subrecord of the plugin's TES4 record.

Returns The value of the version field, or an empty optional if that value is NaN or could not be found.

virtual std::optional<std::string> **GetVersion**() const = 0
Get the plugin's version number from its description field.

The description field may not contain a version number, or LOOT may be unable to detect it. The description field parsing may fail to extract the version number correctly, though it functions correctly in all known cases.

Returns An optional containing a version string if one is found, otherwise an optional containing no value.

virtual std::vector<std::string> **GetMasters**() const = 0
Get the plugin's masters.

Returns The plugin's masters in the same order they are listed in the file.

virtual std::vector<*Tag*> **GetBashTags**() const = 0
Get any Bash Tags found in the plugin's description field.

Returns A set of Bash Tags. The order of elements in the set holds no semantics.

virtual std::optional<uint32_t> **GetCRC**() const = 0
Get the plugin's CRC-32 checksum.

Returns An optional containing the plugin's CRC-32 checksum if the plugin has been fully loaded, otherwise an optional containing no value.

virtual bool **IsMaster**() const = 0
Check if the plugin's master flag is set.

Returns True if the master flag is set, false otherwise.

virtual bool **IsLightPlugin**() const = 0
Check if the plugin is a light plugin.

Returns True if plugin is a light plugin, false otherwise.

virtual bool **IsValidAsLightPlugin**() const = 0
Check if the plugin is or would be valid as a light plugin.

Returns True if the plugin is a valid light plugin or would be a valid light plugin, false otherwise.

virtual bool **IsEmpty**() const = 0
Check if the plugin contains any records other than its TES4 header.

Returns True if the plugin only contains a TES4 header, false otherwise.

virtual bool **LoadsArchive**() const = 0
Check if the plugin loads an archive (BSA/BA2 depending on the game).

Returns True if the plugin loads an archive, false otherwise.

virtual bool **DoRecordsOverlap**(const *PluginInterface* &plugin) const = 0
Check if two plugins contain a record with the same ID.

Parameters **plugin** – The other plugin to check for overlap with.

Returns True if the plugins both contain at least one record with the same ID, false otherwise. FormIDs are compared for all games apart from Morrowind, which doesn't have FormIDs and so has other identifying data compared.

4.6 Classes

class loot::**ConditionalMetadata**
A base class for metadata that can be conditional based on the result of evaluating a condition string.

Subclassed by *File*, *Message*, *Tag*

Public Functions

ConditionalMetadata() = default
Construct a *ConditionalMetadata* object with an empty condition string.

Returns A *ConditionalMetadata* object.

explicit **ConditionalMetadata**(const std::string &condition)
Construct a *ConditionalMetadata* object with a given condition string.

Parameters **condition** – A condition string, as defined in the LOOT metadata syntax documentation.

Returns A *ConditionalMetadata* object.

bool **IsConditional**() const
Check if the condition string is non-empty.

Returns True if the condition string is not empty, false otherwise.

std::string **GetCondition**() const
Get the condition string.

Returns The object's condition string.

class loot::**Filename**
Represents a case-insensitive filename.

Public Functions

Filename() = default
Construct a *Filename* using an empty string.

Returns A *Filename* object.

explicit **Filename**(const std::string &filename)
Construct a *Filename* using the given string.

Returns A *Filename* object.

explicit **operator std::string**() const
Get this *Filename* as a string.

class loot::**File** : public *ConditionalMetadata*
Represents a file in a game's Data folder, including files in subdirectories.

Public Functions

File() = default
Construct a *File* with blank name, display and condition strings.

Returns A *File* object.

explicit **File**(const std::string &name, const std::string &display = "", const std::string &condition = "", const std::vector<*MessageContent*> &detail = {})
Construct a *File* with the given name, display name and condition strings.

Parameters

- **name** – The filename of the file.
- **display** – The name to be displayed for the file in messages, formatted using CommonMark.
- **condition** – The *File*'s condition string.
- **detail** – The detail message content, which may be appended to any messages generated for this file. If multilingual, one language must be English.

Returns A *File* object.

Filename **GetName**() const
Get the filename of the file.

Returns The file's filename.

std::string **GetDisplayName**() const
Get the display name of the file.

Returns The file's display name.

std::vector<*MessageContent*> **GetDetail**() const
Get the detail message content of the file.

If this file causes an error message to be displayed, the detail message content should be appended to that message, as it provides more detail about the error (e.g. suggestions for how to resolve it).

class loot::**Group**
Represents a group to which plugin metadata objects can belong.

Public Functions

Group() = default
Construct a *Group* with the name "default" and an empty set of groups to load after.

Returns A *Group* object.

explicit **Group**(const std::string &name, const std::vector<std::string> &afterGroups = {}, const std::string &description = "")
Construct a *Group* with the given name, description and set of groups to load after.

Parameters

- **name** – The group name.
- **afterGroups** – The names of groups this group loads after.
- **description** – A description of the group.

Returns A *Group* object.

std::string **GetName**() const
Get the name of the group.

Returns The group's name.

std::string **GetDescription**() const
Get the description of the group.

Returns The group's description.

std::vector<std::string> **GetAfterGroups**() const
Get the set of groups this group loads after.

Returns A set of group names.

Public Static Attributes

static constexpr const char ***DEFAULT_NAME** = "default"
The name of the group to which all plugins belong by default.

class loot::**Location**
Represents a URL at which the parent plugin can be found.

Public Functions

Location() = default
Construct a *Location* with empty URL and name strings.

Returns A *Location* object.

explicit **Location**(const std::string &url, const std::string &name = "")
Construct a *Location* with the given URL and name.

Parameters

- **url** – The URL at which the plugin can be found.
- **name** – A name for the URL, eg. the page or site name.

Returns A *Location* object.

std::string **GetURL**() const
Get the object's URL.

Returns A URL string.

std::string **GetName**() const
Get the object's name.

Returns The name of the location.

class loot::**MessageContent**
Represents a message's localised text content.

Public Functions

MessageContent() = default
Construct a *MessageContent* object with an empty English message string.

Returns A *MessageContent* object.

explicit **MessageContent**(const std::string &text, const std::string &language = *DEFAULT_LANGUAGE*)
Construct a *Message* object with the given text in the given language.

Parameters

- **text** – The message text.
- **language** – The language that the message is written in.

Returns A *MessageContent* object.

std::string **GetText**() const
Get the message text.

Returns A string containing the message text.

std::string **GetLanguage**() const
Get the message language.

Returns A code representing the language that the message is written in.

Public Static Attributes

static constexpr const char *DEFAULT_LANGUAGE = "en"
The code for the default language assumed for message content, which is "en" (English).

class loot::Message : public *ConditionalMetadata*
Represents a message with localisable text content.

Public Functions

Message() = default
Construct a *Message* object of type 'say' with blank content and condition strings.

Returns A *Message* object.

explicit Message(const *MessageType* type, const std::string &content, const std::string &condition = "")
Construct a *Message* object with the given type, English content and condition string.

Parameters

- **type** – The message type.
- **content** – The English message content text.
- **condition** – A condition string.

Returns A *Message* object.

explicit Message(const *MessageType* type, const std::vector<*MessageContent*> &content, const std::string &condition = "")
Construct a *Message* object with the given type, content and condition string.

Parameters

- **type** – The message type.
- **content** – The message content. If multilingual, one language must be English.
- **condition** – A condition string.

Returns A *Message* object.

explicit Message(const *SimpleMessage* &message)
Construct a *Message* object from a *SimpleMessage* object.

Parameters **message** – The *SimpleMessage* object.

Returns A *Message* object.

MessageType GetType() const
Get the message type.

Returns The message type.

std::vector<*MessageContent*> GetContent() const
Get the message content.

Returns The message's *MessageContent* objects.

class loot::PluginCleaningData
Represents data identifying the plugin under which it is stored as dirty or clean.

Public Functions

PluginCleaningData() = default
Construct a *PluginCleaningData* object with zero CRC, ITM count, deleted reference count and deleted navmesh count values, an empty utility string and no detail.

Returns A *PluginCleaningData* object.

explicit **PluginCleaningData**(uint32_t crc, const std::string &utility)
Construct a *PluginCleaningData* object with the given CRC and utility, zero ITM count, deleted reference count and deleted navmesh count values and no detail.

Parameters

- **crc** – The CRC of a plugin.
- **utility** – The utility that the plugin cleanliness was checked with.

Returns A *PluginCleaningData* object.

explicit **PluginCleaningData**(uint32_t crc, const std::string &utility, const std::vector<*MessageContent*> &detail, unsigned int itm, unsigned int ref, unsigned int nav)
Construct a *PluginCleaningData* object with the given values.

Parameters

- **crc** – A clean or dirty plugin's CRC.
- **utility** – The utility that the plugin cleanliness was checked with.
- **detail** – A vector of localised information message strings about the plugin cleanliness.
- **itm** – The number of Identical To Master records found in the plugin.
- **ref** – The number of deleted references found in the plugin.
- **nav** – The number of deleted navmeshes found in the plugin.

Returns A *PluginCleaningData* object.

uint32_t **GetCRC**() const
Get the CRC that identifies the plugin that the cleaning data is for.

Returns A CRC-32 checksum.

unsigned int **GetITMCount**() const
Get the number of Identical To Master records in the plugin.

Returns The number of Identical To Master records in the plugin.

unsigned int **GetDeletedReferenceCount**() const
Get the number of deleted references in the plugin.

Returns The number of deleted references in the plugin.

unsigned int **GetDeletedNavmeshCount**() const
Get the number of deleted navmeshes in the plugin.

Returns The number of deleted navmeshes in the plugin.

std::string **GetCleaningUtility**() const
Get the name of the cleaning utility that was used to check the plugin.

Returns A cleaning utility name, possibly related information such as a version number and/or a CommonMark-formatted URL to the utility's download location.

std::vector<*MessageContent*> **GetDetail**() const
Get any additional informative message content supplied with the cleaning data, eg. a link to a cleaning guide or information on wild edits or manual cleaning steps.

Returns A vector of localised *MessageContent* objects.

class `loot::`**PluginMetadata**
Represents a plugin's metadata.

Public Functions

PluginMetadata() = default
Construct a *PluginMetadata* object with a blank plugin name and no metadata.

Returns A *PluginMetadata* object.

explicit **PluginMetadata**(const std::string &name)
Construct a *PluginMetadata* object with no metadata for a plugin with the given filename.

Parameters **name** – The filename of the plugin that the object is constructed for.

Returns A *PluginMetadata* object.

void **MergeMetadata**(const *PluginMetadata* &plugin)
Merge metadata from the given *PluginMetadata* object into this object.

If an equal metadata object already exists in this *PluginMetadata* object, it is not duplicated. This object's group is replaced by the given object's group if the latter is explicit.

Parameters **plugin** – The plugin metadata to merge.

std::string **GetName**() const
Get the plugin name.

Returns The plugin name.

std::optional<std::string> **GetGroup**() const
Get the plugin's group.

Returns An optional containing the name of the group this plugin belongs to if it was explicitly set, otherwise an optional containing no value.

std::vector<*File*> **GetLoadAfterFiles**() const
Get the plugins that the plugin must load after.

Returns The plugins that the plugin must load after.

std::vector<*File*> **GetRequirements**() const
Get the files that the plugin requires to be installed.

Returns The files that the plugin requires to be installed.

std::vector<*File*> **GetIncompatibilities**() const
Get the files that the plugin is incompatible with.

Returns The files that the plugin is incompatible with.

std::vector<*Message*> **GetMessages**() const
Get the plugin's messages.

Returns The plugin's messages.

std::vector<*Tag*> **GetTags**() const
Get the plugin's Bash *Tag* suggestions.

Returns The plugin's Bash *Tag* suggestions.

std::vector<*PluginCleaningData*> **GetDirtyInfo**() const
Get the plugin's dirty plugin information.

Returns The *PluginCleaningData* objects that identify the plugin as dirty.

std::vector<*PluginCleaningData*> **GetCleanInfo**() const
Get the plugin's clean plugin information.

Returns The *PluginCleaningData* objects that identify the plugin as clean.

std::vector<*Location*> **GetLocations**() const
Get the locations at which this plugin can be found.

Returns The locations at which this plugin can be found.

void **SetGroup**(const std::string &group)
Set the plugin's group.

Parameters **group** – The name of the group this plugin belongs to.

void **UnsetGroup**()
Unsets the plugin's group.

void **SetLoadAfterFiles**(const std::vector<*File*> &after)
Set the files that the plugin must load after.

Parameters **after** – The files to set.

void **SetRequirements**(const std::vector<*File*> &requirements)
Set the files that the plugin requires to be installed.

Parameters **requirements** – The files to set.

void **SetIncompatibilities**(const std::vector<*File*> &incompatibilities)
Set the files that the plugin must load after.

Parameters **incompatibilities** – The files to set.

void **SetMessages**(const std::vector<*Message*> &messages)
Set the plugin's messages.

Parameters **messages** – The messages to set.

void **SetTags**(const std::vector<*Tag*> &tags)
Set the plugin's Bash *Tag* suggestions.

Parameters **tags** – The Bash *Tag* suggestions to set.

void **SetDirtyInfo**(const std::vector<*PluginCleaningData*> &info)
Set the plugin's dirty information.

Parameters **info** – The dirty information to set.

void **SetCleanInfo**(const std::vector<*PluginCleaningData*> &info)
Set the plugin's clean information.

Parameters **info** – The clean information to set.

void **SetLocations**(const std::vector<*Location*> &locations)
Set the plugin's locations.

Parameters **locations** – The locations to set.

bool **HasNameOnly**() const
Check if no plugin metadata is set.

Returns True if the group is implicit and the metadata containers are all empty, false otherwise.

bool **IsRegexPlugin**() const
Check if the plugin name is a regular expression.

Returns True if the plugin name contains any of the characters `:\*?|`, false otherwise.

bool **NameMatches**(const std::string &pluginName) const
Check if the given plugin name matches this plugin metadata object's name field.

If the name field is a regular expression, the given plugin name will be matched against it, otherwise the strings will be compared case-insensitively. The given plugin name must be literal, i.e. not a regular expression.

Returns True if the given plugin name matches this metadata's plugin name, false otherwise.

std::string **AsYaml**() const
Serialises the plugin metadata as YAML.

Returns The serialised plugin metadata.

class `loot::Tag` : public *ConditionalMetadata*
Represents a Bash *Tag* suggestion for a plugin.

Public Functions

explicit **Tag**() = default
Construct a *Tag* object with an empty tag name suggested for addition, with an empty condition string.

Returns A *Tag* object.

explicit **Tag**(const std::string &tag, const bool isAddition = true, const std::string &condition = "")
Construct a *Tag* object with the given name, for addition or removal, with the given condition string.

Parameters

- **tag** – The name of the Bash *Tag*.
- **isAddition** – True if the tag should be added, false if it should be removed.
- **condition** – A condition string.

Returns A *Tag* object.

bool **IsAddition**() const
Check if the tag should be added.

Returns True if the tag should be added, false if it should be removed.

std::string **GetName**() const
Get the tag's name.

Returns The tag's name.

class `loot::Vertex`
A class representing a plugin or group vertex in a path, and the type of the edge to the next vertex in the path if one exists.

Public Functions

explicit **Vertex**(std::string name)
Construct a *Vertex* with the given name and no out edge.

Parameters **name** – The name of the plugin or group that this vertex represents.

explicit **Vertex**(std::string name, *EdgeType* outEdgeType)
Construct a *Vertex* with the given name and out edge type.

Parameters

- **name** – The name of the plugin or group that this vertex represents.
- **outEdgeType** – The type of the edge going out from this vertex.

std::string **GetName**() const
Get the name of the plugin or group.

Returns The name of the plugin or group.

std::optional<*EdgeType*> **GetTypeOfEdgeToNextVertex**() const
Get the type of the edge going to the next vertex.

Each edge goes from the vertex that loads earlier to the vertex that loads later.

Returns The edge type.

4.7 Exceptions

class loot::**CyclicInteractionError** : public runtime_error
An exception class thrown if a cyclic interaction is detected when sorting a load order.

Public Functions

CyclicInteractionError(std::vector<*Vertex*> cycle)
Construct an exception detailing a plugin or group graph cycle.

Parameters **cycle** – A representation of the cyclic path.

std::vector<*Vertex*> **GetCycle**() const
Get a representation of the cyclic path.

Each *Vertex* is the name of a graph element (plugin or group) and the type of the edge going to the next *Vertex*. The last *Vertex* has an edge going to the first *Vertex*.

Returns A vector of *Vertex* elements representing the cyclic path.

class **ConditionSyntaxError** : public runtime_error
An exception class thrown if invalid syntax is encountered when parsing a metadata condition.

class **FileAccessError** : public runtime_error
An exception class thrown if an error is encountered while reading or writing a file.

class loot::**UndefinedGroupError** : public runtime_error
An exception class thrown if group is referenced but is undefined.

Public Functions

UndefinedGroupError(const std::string &groupName)
Construct an exception for an undefined group.

Parameters **groupName** – The name of the group that is undefined.

std::string **GetGroupName**() const
Get the name of the undefined group.

Returns A group name.

4.8 Error Categories

LOOT uses error category objects to identify errors with codes that originate in lower-level libraries.

const std::error_category &**loot::libloadorder_category**()
Get the error category that can be used to identify system_error exceptions that are due to libloadorder errors.

Returns A reference to the static object of unspecified runtime type, derived from std::error_category.

CHAPTER

FIVE

CREDITS

libloot is written by Ortham in C++ and makes use of the Boost, esplugin, libloadorder, loot-condition-interpreter, spdlog and yaml-cpp libraries. The copyright licenses for all of these and libloot itself in *Copyright License Texts*.

CHAPTER

SIX

VERSION HISTORY

6.1 0.19.4 - 2023-05-06

6.1.1 Added

- Support for the Microsoft Store's Fallout 4 DLC. The Microsoft Store installs Fallout 4's DLC to separate directories outside of the Fallout 4 install path, and the Microsoft Store's version of Fallout 4 knows to look for plugins and resources to load in those other directories.
  - libloot detects if a copy of Fallout 4 is from the Microsoft Store by checking for the existence of an `appxmanifest.xml` file in the given install path, and if found will look for Fallout 4 DLC directories at their install paths. The DLC install paths used are relative to the game install path because those relative paths are assumed by the game.
  - If a DLC data path exists, load order operations will include plugins in that directory, i.e. DLC plugins will appear as part of the load order that libloot reads and writes.
  - Metadata conditions will check for files in DLC data paths as well as the game's data path, with DLC paths checked before the game's data path to match the order in which the game checks paths.

6.1.2 Changed

- *loot::GameInterface::IsValidPlugin()*, *loot::GameInterface::LoadPlugins()* and *loot::GameInterface::SortPlugins()* now take plugin paths instead of plugin filenames. Relative paths are interpreted as relative to the game's data path, so this change is backwards-compatible. Absolute paths are used as given. The functions take plugin paths as strings to avoid breaking libloot's binary interface, but they will be changed to take ``std::filesystem::path``s in a future release.
- *loot::GameInterface::LoadPlugins()* and *loot::GameInterface::SortPlugins()* now check that all filenames in the given paths are unique. This was previously implicitly required for correct behaviour but not explicitly enforced.

6.2 0.19.3 - 2023-03-18

6.2.1 Added

- Support for the Steam and GOG distributions of Enderal: Forgotten Stories and Enderal: Forgotten Stories (Special Edition), which are total conversion mods for Skyrim and Skyrim Special Edition respectively. This support means that the game local path does not need to be specified when creating a game handle: when libloot is given the path to a Skyrim or Skyrim SE installation that is actually an Enderal installation, it is now able to look up the correct game local path. Via libloadorder.

6.2.2 Fixed

- libloot would deactivate plugins when setting the load order if too many plugins were active. This could cause unexpected behaviour if later-loading active plugins were sorted to load earlier.
- The path returned by *loot::CyclicInteractionError::GetCycle()* could include too many vertices, including repeated vertices.

6.2.3 Changed

- Updated Boost to v1.81.0.
- Updated libloadorder to v14.0.0.

6.3 0.19.2 - 2023-01-13

6.3.1 Fixed

- libloot v0.19.1 did not take user groups into account when avoiding cycles during sorting, causing unnecessary cyclic interaction errors.

6.3.2 Changed

- Sorting will once more throw a cyclic interaction error if there is any plugin data or metadata that would try to load a master-flagged plugin after a non-master-flagged plugin. This behaviour was removed as a side-effect of sorting changes made in libloot v0.19.0.

6.4 0.19.1 - 2023-01-09

6.4.1 Fixed

- Sorting and applying and then sorting again will no longer give a different result for the second sort. libloot v0.19.0 changed the order in which group and overlap edges were processed to be the current load order: it has now reverted back to the lexicographical order of plugin filenames.

6.5 0.19.0 - 2023-01-07

6.5.1 Added

- Sorting now takes into account overlapping assets in BSAs/BA2s that are loaded by plugins. If two plugins don't make changes to the same record but load BSAs (or BA2s for Fallout 4) that contain data for the same asset path, the plugin that loads more assets will load first (unless that's contradicted by higher-priority data and metadata).
- *loot::GameInterface::GetActivePluginsFilePath()*, which returns the path of the file libloot reads to determine which plugins are active.
- *loot::EdgeType::masterlistGroup*, *loot::EdgeType::userGroup*, *loot::EdgeType::recordOverlap* and *loot::EdgeType::assetOverlap*.

6.5.2 Fixed

- Building libloot using CMake versions older than 3.24.
- A few potential null pointer dereferences.

6.5.3 Changed

- Sorting has been heavily optimised, leading to sorting being about 58 times faster than libloot 0.18.3 with large load orders:
  - The plugin graph used during sorting has been split in two. As a result, any plugin data or metadata that would previously caused a cyclic interaction error due to contradicting a plugin's master flag being set is now silently ignored instead.
  - The tie-breaking stage has been completely overhauled. As a result, some ties may now be broken differently to how they were broken in previous versions of libloot.
  - *loot::GameInterface::LoadPlugins()* now checks plugin validity in parallel.
- Cyclic interaction errors now distinguish between group edges that involve user metadata and those that don't.
- `PluginInterface::DoFormIDsOverlap()` has been renamed to *loot::PluginInterface::DoRecordsOverlap()*.
- *loot::CyclicInteractionError::GetCycle()* is now `const`.
- *loot::UndefinedGroupError::GetGroupName()* is now `const`.
- Linux builds are now built using GCC 10 and now link against the `tbb` library.

6.5.4 Removed

- `EdgeType::group`
- `EdgeType::overlap`

6.6 0.18.3 - 2022-12-13

6.6.1 Fixed

- Resolved a CMake warning relating to policy CMP0135 when building libloot.
- Some of the documentation on `not` operators in the metadata syntax was outdated.
- The libloot Windows DLL did not include some file info fields that are required according to Microsoft's documentation. The `CompanyName`, `FileDescription`, `InternalName`, `OriginalFilename` and `ProductName` fields have been added.
- The libloot Windows DLL advertised its `FILETYPE` as `VFT_APP`, which has been changed to `VFT_DLL`.

6.6.2 Changed

- Sorting optimisations mean that sorting is now significantly faster (over 5 times faster in testing).
- Log message severities have been adjusted to reduce the verbosity at the "info" level and to move some messages between "debug" and "trace".
- Release build archive names no longer include the output of `git describe`.
- Updated spdlog to v1.11.0.

6.7 0.18.2 - 2022-10-11

6.7.1 Fixed

- libloot will now use the correct local app data path for the Epic Games Store distribution of Skyrim Special Edition when no local app data path is passed to *`loot::CreateGameHandle()`*. Via libloadorder.

6.7.2 Changed

- Updated libloadorder to v13.3.0.

6.8 0.18.1 - 2022-10-01

6.8.1 Fixed

- libloot will now use the correct local app data path for the GOG distribution of Skyrim Special Edition when no local app data path is passed to *`loot::CreateGameHandle()`*. Via libloadorder.
- If Oblivion's `Oblivion.ini` could not be found or read, or if it did not contain the `bUseMyGamesDirectory` setting, the game's install path would be used as the parent directory for `plugins.txt`. libloot now correctly defaults to using the game's local app data directory, and only uses the install path if `bUseMyGamesDirectory=0` is found. Via libloadorder.

6.8.2 Changed

- When serialising plugin metadata as YAML, LOOT now:
  - Puts `url` before `group`
  - Serialises single-element lists using the flow style if the element would be serialised as a scalar value
  - Pads CRC hexadecimal values to always be 8 characters long (excluding the `0x` prefix)
  - Uses uppercase letters in CRC hexadecimal values.
- Updated esplugin to v4.0.0.
- Updated Google Test to v1.12.1.
- Updated libloadorder to v13.2.0.
- Updated loot-condition-interpreter to v2.3.1.
- Updated spdlog to v1.10.0.

6.9 0.18.0 - 2022-02-27

6.9.1 Added

- *loot::Group::DEFAULT_NAME* gives the default group name as a compile-time constant.
- *loot::ToSimpleMessages()* turns a `std::vector<Message>` into a `std::vector<SimpleMessage>` for a given language.
- *loot::GameInterface::IsLoadOrderAmbiguous()* exposes libloadorder's `lo_is_ambiguous()` function.

6.9.2 Fixed

- *loot::SimpleMessage* now uses an in-class initialiser to ensure that its `type` member variable is always initialised.
- Added missing virtual destructors to *loot::GameInterface*, *loot::DatabaseInterface* and *loot::PluginInterface*.
- Two versions that only differ by the presence and absence of pre-release identifiers were not correctly compared according to Semantic Versioning, which states that 1.0.0-alpha is less than 1.0.0. Via loot-condition-interpreter.
- Some missing API documentation and formatting issues.

6.9.3 Changed

- *loot::CreateGameHandle()* now returns a `std::unique_ptr<GameInterface>` instead of a `std::shared_ptr<GameInterface>`.
- *loot::GameInterface::GetDatabase()* now returns a `DatabaseInterface&` instead of a `std::shared_ptr<DatabaseInterface>`.
- *loot::GameInterface::GetPlugin()* now returns a `const PluginInterface*` instead of a `std::shared_ptr<const PluginInterface>`.
- *loot::GameInterface::GetLoadedPlugins()* now returns a `std::vector<const PluginInterface*>` instead of a `std::vector<std::shared_ptr<const PluginInterface>>`.

- `MessageContent::defaultLanguage` has been replaced with *loot::MessageContent::DEFAULT_LANGUAGE*, which is a compile-time constant.
- `File::ChooseDetail()`, `Message::GetContent(const std::string& language)`, `MessageContent::Choose()` and `PluginCleaningData::ChooseDetail()` have been replaced with *loot::SelectMessageContent()*.
- `Message::ToSimpleMessage()` has been replaced with *loot::ToSimpleMessage()*.
- `LootVersion` has been replaced with *loot::LIBLOOT_VERSION_MAJOR*, *loot::LIBLOOT_VERSION_MINOR*, *loot::LIBLOOT_VERSION_PATCH*, *loot::GetLiblootVersion()* and *loot::GetLiblootRevision()*.
- *loot::File::GetDisplayName()* is now a trivial accessor that only ever returns the value of the display name member variable and performs no character escaping.
- *loot::CyclicInteractionError* and *loot::UndefinedGroupError* have had their `const` member variables made non-`const`.
- *loot::ConditionalMetadata*, *loot::File*, *loot::Filename*, *loot::Group*, *loot::Location*, *loot::Message*, *loot::MessageContent*, *loot::PluginCleaningData*, *loot::PluginMetadata* and *loot::Tag* have had their user-defined default constructors replaced by use of in-class initialisers and defaulted default constructors.
- The < and == operator overloads for *loot::File*, *loot::Group*, *loot::Location*, *loot::Message*, *loot::MessageContent*, *loot::PluginCleaningData* and *loot::Tag* have become non-member functions.
- The performance of *loot::PluginMetadata::NameMatches()* has been greatly improved by not constructing a new regex object every time the function is called.
- Mentions of GitHub Flavored Markdown have been replaced with CommonMark, as LOOT now uses the latter instead of the former.
- Updated loot-condition-interpreter to v2.3.0.

6.9.4 Removed

- `ConditionalMetadata::ParseCondition()`
- `PluginMetadata::NewMetadata()`
- All Git-related functionality has been removed, including the libgit2 dependency and the following API items:
  - `loot::UpdateFile()`
  - `loot::GetFileRevision()`
  - `loot::IsLatestFile()`
  - `loot::libgit2_category()`
  - `loot::GitStateError`
  - `loot::FileRevision`

6.10 0.17.3 - 2022-01-02

6.10.1 Added

- `PluginMetadata::AsYaml` can be used to serialise plugin metadata as YAML.

6.10.2 Changed

- Plugin name regular expression objects are now cached between calls to `DatabaseInterface::LoadLists`.

6.11 0.17.2 - 2021-12-24

6.11.1 Fixed

- A missing <string> include in `include/loot/struct/simple_message.h`.
- Invalid configuration causing Read The Docs to fail to build the documentation.

6.11.2 Changed

- Updated libgit2 to v1.3.0.

6.12 0.17.1 - 2021-11-13

6.12.1 Fixed

- Out-of-bounds array access that could occur in some situations and which could cause crashes in Linux builds.

6.13 0.17.0 - 2021-09-24

6.13.1 Added

- `DatabaseInterface::LoadLists` now accepts an optional third parameter that is the path to a masterlist prelude file to load. If loaded, it will be used to replace the value of the prelude in the loaded masterlist (if the masterlist has a prelude).
- The `Message` class has gained a constructor that takes a `SimpleMessage`.
- The `File` class has been gained support for the metadata structure's new `detail` field, adding:
  - An optional `const std::vector<MessageContent>&` parameter to the multiple-parameter constructor.
  - A new `File::GetDetail` member function.
  - A new `File::ChooseDetail` member function.

6.13.2 Changed

- `MasterlistInfo` has been renamed to `FileRevision`, and its `revision_id` and `revision_date` fields are now named `id` and `date` respectively.
- The `UpdateMasterlist`, `GetMasterlistRevision` and `IsLatestMasterlist` member functions have been moved out of `DatabaseInterface` and are now free functions named `UpdateFile`, `GetFileRevision` and `IsLatestFile` respectively.
- `PluginInterface::GetHeaderVersion` now returns a `std::optional<float>` instead of a `float`. The return value is `std::nullopt` if no header version field was found or if its value was NaN.
- Sorting now checks for cycles before adding overlap edges, so that any cycles are caught before the slowest steps in the sorting process.
- `PluginCleaningData::GetInfo()` has been renamed to `PluginCleaningData::GetDetail()`.
- `PluginCleaningData::ChooseInfo()` has been renamed to `PluginCleaningData::ChooseDetail()`.
- All API functions that returned a `MessageContent` or `SimpleMessage` now return a `std::optional<MessageContent>` or `std::optional<SimpleMessage>` respectively. This affects the following member functions:
  - `Message::GetContent`
  - `Message::ToSimpleMessage`
  - `MessageContent::Choose`
  - `PluginCleaningData::ChooseDetail`
- Updated libgit2 to v1.1.1.
- Updated Google Test to v1.11.0.
- Updated spdlog to v1.9.2.
- Updated yaml-cpp to v0.7.0+merge-key-support.1.

6.13.3 Removed

- `PluginInterface::IsLightMaster`
- `PluginInterface::IsValidAsLightMaster`
- Updating the masterlist no longer reloads it, the masterlist must now be reloaded separately.
- Masterlist update no longer supports rolling back through revisions until a revision that can be successfully loaded is found.

6.14 0.16.3 - 2021-05-06

6.14.1 Added

- `PluginInterface::IsLightPlugin` as a more accurately named equivalent to `PluginInterface::IsLightMaster`.
- `PluginInterface::IsValidAsLightPlugin` as a more accurately named equivalent to `PluginInterface::IsValidAsLightMaster`.

- Support for parsing inverted metadata conditions (`not (<expression>)`). Note however that this is not yet part of any released version of LOOT's metadata syntax and must not be used where compatibility with older releases of LOOT is required. Via loot-condition-interpreter.

6.14.2 Changed

- `loot::MessageContent::Choose` now compares locale and language codes so that if an exact match is not present but a more or less specific match is present, that will be preferred over the default language message content.
- Regular expression functions in metadata conditions now handle ghosted plugins in the same way as their path function counterparts.
- Updated esplugin to v3.5.0.
- Updated libloadorder to v13.0.0.
- Updated loot-condition-interpreter to v2.2.1.
- Updated spdlog to v1.8.5.

6.14.3 Fixed

- `.ghost` file extensionms are no longer recursively trimmed when checking if a file has a valid plugin file extension during metadata condition evaluation. Via loot-condition-interpreter.
- When looking for a plugin file matching a path during metadata condition evaluation, a `.ghost` extension is only added to the path if one was not already present. Via loot-condition-interpreter.
- When comparing versions during metadata condition evaluation, the comparison now compares numeric against non-numeric release identifiers (and vice versa) by comparing the numeric value against the numeric value of leading digits in the non-numeric value, and treating the latter as greater if the two numeric values are equal. The numeric value is treated as less than the non-numeric value if the latter has no leading digits. Previously all non-numeric identifiers were always greater than any numeric identifier. For example, 78b was previously considered to be greater than 86, but is now considered to be less than 86. Via loot-condition-interpreter.
- Linux builds did not correctly handle case-insensitivity of plugin names during sorting on filesystems with case folding enabled.

6.14.4 Deprecated

- `PluginInterface::IsLightMaster`: use `PluginInterface::IsLightPlugin` instead.
- `PluginInterface::IsValidAsLightMaster`: use `PluginInterface::IsValidAsLightPlugin` instead.

6.15 0.16.2 - 2021-02-13

6.15.1 Changed

- Updated libgit2 to v1.1.0.
- Updated loot-condition-interpreter to v2.1.2.
- Updated Boost to v1.72.0.

- Linux releases are now built on GitHub Actions.
- Masterlist updates can no longer be fetched using SSH URLs. This support was never tested or documented.

6.16 0.16.1 - 2020-08-22

6.16.1 Fixed

- `File::GetDisplayName()` now escapes ASCII punctuation characters when returning the file's name, i.e. when no display name is explicitly set. For example, `File("plugin.esp").GetDisplayName()` will now return `plugin\.esp`.

6.17 0.16.0 - 2020-07-12

6.17.1 Added

- The !=, >, <= and >= comparison operators are now implemented for *loot::File*, *loot::Location*, *loot::Message*, *loot::MessageContent*, *loot::PluginCleaningData* and *loot::Tag*.
- The !=, <, >, <= and >= comparison operators are now implemented for *loot::Group*.
- A new `Filename` class for representing strings handled as case-insensitive filenames.
- `PluginMetadata::NameMatches()` checks if the given plugin filename matches the plugin name of the metadata object it is called on. If the plugin metadata name is a regular expression, the given plugin filename will be matched against it, otherwise the comparison is case-insensitive equality.

6.17.2 Changed

- `File::GetName()` now returns a `Filename` instead of a `std::string`.
- `GetGroups` and `GetUserGroups` now return `std::vector<Group>` instead of `std::unordered_set<Group>`.
- `SetUserGroups` now takes a `const std::vector<Group>&` instead of a `const std::unordered_set<std::string>&`.
- *loot::Group*'s three-argument constructor now takes a `const std::vector<std::string>&` instead of a `const std::unordered_set<std::string>&` as its second parameter.
- `GetAfterGroups` now returns a `std::vector<std::string>` instead of a `std::unordered_set<std::string>`.
- `std::set<>` usage has been replaced by `std::vector<>` throughout the public API. This affects the following functions:
  - `PluginInterface::GetBashTags()`
  - `DatabaseInterface::GetKnownBashTags()`
  - `GameInterface::GetLoadedPlugins()`
  - `PluginMetadata::GetLoadAfterFiles()`
  - `PluginMetadata::SetLoadAfterFiles()`
  - `PluginMetadata::GetRequirements()`

  - `PluginMetadata::SetRequirements()`
  - `PluginMetadata::GetIncompatibilities()`
  - `PluginMetadata::SetIncompatibilities()`
  - `PluginMetadata::GetTags()`
  - `PluginMetadata::SetTags()`
  - `PluginMetadata::GetDirtyInfo()`
  - `PluginMetadata::SetDirtyInfo()`
  - `PluginMetadata::GetCleanInfo()`
  - `PluginMetadata::SetCleanInfo()`
  - `PluginMetadata::GetLocations()`
  - `PluginMetadata::SetLocations()`
- *loot::File*, *loot::Location*, *loot::Message*, *loot::MessageContent*, *loot::PluginCleaningData*, *loot::Tag* and *loot::Group* now implement their comparison operators by comparing all their fields (including inherited fields), using the same operator for the fields. For example, comparing two *loot::File* objects using == will now compare each of their fields using ==.
- When loading plugins, the speed at which LOOT identifies their corresponding archive files (`*.bsa` or `.ba2`, depending on the game) has been improved.

6.17.3 Removed

- `PluginMetadata::IsEnabled()` and `PluginMetadata::SetEnabled()`, as it is no longer possible to disable plugin metadata (though doing so never had any effect).
- `PluginMetadata` no longer implements the == or `!=` comparison operators.
- `std::hash` is no longer specialised for *loot::Group*.

6.17.4 Fixed

- `LoadsArchive` now correctly identifies the BSAs that a Skyrim SE or Skyrim VR loads. This assumes that Skyrim VR plugins load BSAs in the same way as Skyrim SE. Previously LOOT would use the same rules as the Fallout games for Skyrim SE or VR, which was incorrect.
- Some operations involving loaded plugins or copies of game interface objects could potentially cause data races due to a lack of mutex locking in some data read operations.
- Copying a game interface object did not copy its cached archive files, leaving the new copy with no cached archive files.

6.18 0.15.2 - 2020-06-14

6.18.1 Changed

- `MergeMetadata` now only uses the group value of the given metadata object if there is not already one set, matching the behaviour for all other merged metadata.
- Updated esplugin to v3.3.1.
- Updated libgit2 to v1.0.1.
- Updated loot-condition-interpreter to v2.1.1.
- Updated spdlog to v1.6.1.

6.18.2 Fixed

- `GetPluginMetadata` preferred masterlist metadata over userlist metadata when merging them, which was the opposite of the intended behaviour.

6.19 0.15.1 - 2019-12-07

6.19.1 Changed

- The range of FormIDs that are recognised as valid in light masters has been extended for Fallout 4 plugins, from between 0x800 and 0xFFF inclusive to between 0x001 and 0xFFF inclusive, to reflect the extended range supported by Fallout 4 v1.10.162.0.0. The valid range for Skyrim Special Edition plugins is unchanged. Via esplugin.
- Updated esplugin to v3.3.0.

6.20 0.15.0 - 2019-11-05

6.20.1 Changed

- libloot now supports v0.15 of the metadata syntax.
- The order of the plugins passed to `SortPlugins` is now used as the current load order during sorting. The order of plugins passed in did not previously have any impact.
- Constructors for the following classes and structs are now `explicit`:
  - *loot::ConditionalMetadata*
  - *loot::File*
  - *loot::Group*
  - *loot::Location*
  - *loot::Message*
  - *loot::MessageContent*
  - *loot::PluginCleaningData*

  - *loot::PluginMetadata*
  - *loot::Tag*
  - loot::MasterlistInfo
  - *loot::Vertex*
- Updated loot-condition-interpreter to v2.1.0.
- Updated spdlog to v1.4.2.

6.20.2 Removed

- `InitialiseLocale()`
- `PluginMetadata::GetLowercasedName()`
- `PluginMetadata::GetNormalizedName()`

6.20.3 Fixed

- libloot was unable to extract versions from plugin descriptions containing `version:` followed by whitespace and one or more digits.
- libloot did not error if masterlist metadata defined a group that loaded after another group that was not defined in the masterlist, but which was defined in user metadata. This was unintentional, and now all groups mentioned in masterlist metadata must now be defined in the masterlist.
- Build errors on Linux using GCC 9 and ICU 61+.

6.21 0.14.10 - 2019-09-06

6.21.1 Changed

- Improved the sorting process for Morrowind. Previously, sorting was unable to determine if a Morrowind plugin contained any records overriding those of its masters, and so added no overlap edges between Morrowind plugins when sorting. Sorting now counts override records by comparing plugins against their masters, giving the same results as for other games.

  However, unlike for other games, this requires all a plugin's masters to be installed. If a plugin's masters are missing, the plugin's total record count will be used as if it was the plugin's override record count to ensure that sorting can still proceed, albeit with potentially reduced accuracy.
- Updated esplugin to v3.2.0.
- Updated libgit2 to v0.28.3.

6.22 0.14.9 - 2019-07-23

6.22.1 Fixed

- Regular expressions in condition strings are now prefixed with `^` and suffixed with `$` before evaluation to ensure that only exact matches to the given expression are found. Via loot-condition-interpreter.

6.22.2 Changed

- Updated loot-condition-interpreter to v2.0.0.

6.23 0.14.8 - 2019-06-30

6.23.1 Fixed

- Evaluating `version()` and `product_version()` conditions will no longer error if the given executable has no version fields. Instead, it will be evaluated as having no version. Via loot-condition-interpreter.
- Sorting would not preserve the existing relative positions of plugins that had no relative positioning enforced by plugin data or metadata, if one or both of their filenames were not case-sensitively equal to their entries in `plugins.txt` / `loadorder.txt`. Load order position comparison is now correctly case-insensitive.

6.23.2 Changed

- Improved load order sorting performance.
- Updated loot-condition-interpreter to v2.0.0.

6.24 0.14.7 - 2019-06-13

6.24.1 Fixed

- Filename comparisons on Windows now has the same locale-invariant case insensitivity behaviour as Windows itself, instead of being locale-dependent.
- Filename comparisons on Linux now use ICU case folding to give locale-invariant results that are much closer to Windows' case insensitivity, though still not identical.

6.24.2 Changed

- Updated libgit2 to v0.28.2.

6.25 0.14.6 - 2019-04-24

6.25.1 Added

- Support for TES III: Morrowind using `GameType::tes3`. The sorting process for Morrowind is slightly different than for other games, because LOOT cannot currently detect when plugins overlap. As a result, LOOT is much less likely to suggest load order changes.

6.25.2 Changed

- Updated esplugin to v2.1.2.
- Updated loot-condition-interpreter to v1.3.0.

6.25.3 Fixed

- LOOT would unnecessarily ignore intermediate plugins in a non-master to master cycle involving groups, leading to unexpected results when sorting plugins.

6.26 0.14.5 - 2019-02-27

6.26.1 Changed

- Updated libgit2 to v0.28.1.
- Updated libloadorder to v12.0.1.
- Updated spdlog to v1.3.1.

6.26.2 Fixed

- `HearthFires.esm` was not recognised as a hardcoded plugin on case-sensitive filesystems, causing a cyclic interaction error when sorting Skyrim or Skyrim SE (via libloadorder).

6.27 0.14.4 - 2019-01-27

6.27.1 Added

- Added `UnsetGroup` to `PluginMetadata`.

6.28 0.14.3 - 2019-01-27

6.28.1 Changed

- Condition parsing now errors if it does not consume the whole condition string. Via loot-condition-interpreter.
- Removed a few unhelpful log statements and changed the verbosity level of others.
- Updated loot-condition-interpreter to v1.2.2.

6.28.2 Fixed

- Conditions were not parsed past the first instance of `file(<regex>)`, `active(<regex>)`, `many(<regex>)` or `many_active(<regex>)`. Via loot-condition-interpreter.
- *loot::CreateGameHandle()* could crash when trying to check if the given paths are symlinks. If a check fails, LOOT will assume the path is not a symlink.

6.29 0.14.2 - 2019-01-20

6.29.1 Changed

- Updated loot-condition-interpreter to v1.2.1.
- Updated spdlog to v1.3.0.

6.29.2 Fixed

- An error when loading plugins with a file present in the plugins directory that has a filename containing characters that cannot be represented in the system code page.
- An error when trying to read the version of an executable that does not have a US English version information resource. Executable versions are now read from the file's first version information resource, whatever its language. Via loot-condition-interpreter.

6.30 0.14.1 - 2018-12-23

6.30.1 Changed

- Updated loot-condition-interpreter to v1.2.0.

6.30.2 Fixed

- Product version conditions read from executables' `VS_FIXEDFILEINFO` structure, so the versions read did not match the versions displayed by Windows' File Explorer. Product versions are now read from executables' `VS_VERSIONINFO` structure, using the `ProductVersion` key. Via loot-condition-interpreter.
- The release date in the metadata syntax changelog for v0.14 was "Unreleased".

6.31 0.14.0 - 2018-12-09

6.31.1 Added

- `GetHeaderVersion` to get the value of the version field in the `HEDR` subrecord of a plugin's `TES4` record.
- `IsValidAsLightMaster` to check if a light master is valid or if a non-light-master plugin would be valid with the light master flag or `.esl` extension. Validity is defined as having no new records with a FormID object index greater than 0xFFF.
- `GetGroupsPath` to return the path between two given groups that maximises the user metadata and minimises the masterlist metadata involved.
- *loot::Vertex* to represent a plugin or group vertex in a sorting graph path.
- *loot::EdgeType* to represent the type of the edge between two vertices in a sorting graph. Each edge type indicates the type of data it was sourced from.

6.31.2 Changed

- Renamed the library from "the LOOT API" to "libloot" to avoid confusion between the name of the library and the API that it provides. The library filename is changed so that the `loot_api` part is now `loot`, e.g. `loot.dll` on Windows and `libloot.so` on Linux.
- `CyclicInteractionError` has had its constructor and methods completely replaced to provide a more detailed and flexible representation of the cyclic path that it reports.
- `UndefinedGroupError::getGroupName()` has been renamed to `UndefinedGroupError::GetGroupName()` for consistency with other API method names.
- `LootVersion::string()` has been renamed to `LootVersion::GetVersionString()` for consistency with other API method names.
- `GetPluginMetadata` and `GetPluginUserMetadata` now return `std::optional<PluginMetadata>` to differentiate metadata being found or not. Note that the `PluginMetadata` value may still return true for `HasNameOnly` if a metadata entry exists but has no content other than the plugin name.
- `GetGroup` now returns `std::optional<std::string>` to indicate when there is no group metadata explicitly set, to simplify distinguishing between explicit and implicit default group membership.
- `GetVersion` now returns `std::optional<std::string>` to differentiate between there being no version and the version being an empty string, though the latter should never occur.
- `GetCRC` now returns `std::optional<uint32_t>` to differentiate between there being no CRC calculated and the CRC somehow being zero (which should never occur).
- Filesystem paths are now represented in the API by `std::filesystem::path` values instead of `std::string` values. This affects the following functions:
  - *loot::CreateGameHandle()*

  - `LoadLists`
  - `WriteUserMetadata`
  - `WriteMinimalList`
  - `UpdateMasterlist`
  - `GetMasterlistRevision`
  - `IsLatestMasterlist`
- The metadata condition parsing, evaluation and caching code and the pseudosem dependency have been replaced by a dependency on loot-condition-interpreter, which provides more granular caching and more opportunity for future enhancements.
- The API now supports v0.14 of the metadata syntax.
- Updated C++ version required to C++17. This means that Windows builds now require the MSVC 2017 runtime redistributable to be installed.
- Updated esplugin to v2.1.1.
- Updated libloadorder to v12.0.0.
- Updated libgit2 to v0.27.7.
- Updated spdlog to v1.2.1.

6.31.3 Removed

- `PluginInterface::GetLowercasedName()`, as the case folding behaviour LOOT uses is not necessarily appropriate for all use cases, so it's up to the client to lowercase according to their own needs.

6.31.4 Fixed

- BSAs/BA2s loaded by non-ASCII plugins for Oblivion, Fallout 3, Fallout: New Vegas and Fallout 4 may not have been detected due to incorrect case-insensitivity handling.
- Fixed incorrect case-insensitivity handling for non-ASCII plugin filenames and `File` metadata names.
- `FileVersion` and `ProductVersion` properties were not set in the DLL since v0.11.0.
- Path equivalence checks could be inaccurate as they were using case-insensitive string comparisons, which may not match filesystem behaviour. Filesystem equivalence checks are now used to improve correctness.
- Errors due to filesystem permissions when cloning a new masterlist repository into an existing game directory. Deleting the temporary directory is now deferred until after its contents have been copied into the game directory, and if an error is encountered when deleting the temporary directory, it is logged but does not cause the masterlist update to fail.
- An error creating a game handle for Skyrim if `loadorder.txt` is not encoded in UTF-8. In this case, libloadorder will now fall back to interpreting its contents as encoded in Windows-1252, to match the behaviour when reading the load order state.

6.32 0.13.8 - 2018-09-24

6.32.1 Fixed

- Filesystem errors when trying to set permissions during a masterlist update that clones a new repository.

6.33 0.13.7 - 2018-09-10

6.33.1 Changed

- Significantly improve plugin loading performance by scanning for BSAs/BA2s once instead of for each plugin.
- Improve performance of metadata evaluation by caching CRCs with the same cache lifetime as condition results.
- Improve performance of sorting when it involves long plugin interaction chains.
- Updated esplugin to v2.0.1.
- Updated libgit2 to v0.27.4.
- Updated libloadorder v11.4.1.
- Updated spdlog to v1.1.0.
- Updated yaml-cpp to 0.6.2+merge-key-support.2.

6.33.2 Fixed

- Fallout 4's *DLCUltraHighResolution.esm* is now handled as a hardcoded plugin (via libloadorder).

6.34 0.13.6 - 2018-06-29

6.34.1 Changed

- Tweaked masterlist repository cloning to avoid undefined behaviour.
- Updated Boost to v1.67.0.
- Updated esplugin to v2.0.0.
- Updated libgit2 to v0.27.2.
- Updated libloadorder to v11.4.0.

6.35 0.13.5 - 2018-06-02

6.35.1 Changed

- Sorting now enforces hardcoded plugin positions, sourcing them through libloadorder. This avoids the need for often very verbose metadata entries, particularly for Creation Club plugins.
- Updated libgit2 to v0.27.1. This includes a security fix for CVE-2018-11235, but LOOT API's usage is not susceptible. libgit2 is not susceptible to CVE-2018-11233, another Git vulnerability which was published on the same day.
- Updated libloadorder to v11.3.0.
- Updated spdlog to v0.17.0.
- Updated esplugin to v1.0.10.

6.36 0.13.4 - 2018-06-02

6.36.1 Fixed

- `NewMetadata` now uses the passed plugin's group if the calling plugin's group is implicit, and sets the group to be implicit if the two plugins' groups are equal.

6.37 0.13.3 - 2018-05-26

6.37.1 Changed

- Improved cycle avoidance when resolving evaluating plugin groups during sorting. If enforcing the group difference between two plugins would cause a cycle and one of the plugins' groups is the default group, that plugin's group will be ignored for all plugins in groups between default and the other plugin's group.
- The masterlist repository cloning process no longer moves LOOT's game folders, so if something goes wrong the process fails more safely.
- The LOOT API is now built with debugging information on Windows, and its PDB is included in build archives.
- Updated libloadorder to v11.2.2.

6.37.2 Fixed

- Various filesystem-related issues that could be encountered when updating masterlists, including failure due to file handles being left open while attempting to remove.
- Building the esplugin and libloadorder dependencies using Rust 1.26.0, which included a regression to workspace builds.

6.38 0.13.2 - 2018-04-29

6.38.1 Changed

- Updated libloadorder to v11.2.1.

6.38.2 Fixed

- Incorrect load order positions were given for light-master-flagged `.esp` plugins when getting the load order (via libloadorder).

6.39 0.13.1 - 2018-04-09

6.39.1 Added

- Support for Skyrim VR using `GameType::tes5vr`.

6.39.2 Changed

- Updated libloadorder to v11.2.0.

6.40 0.13.0 - 2018-04-02

6.40.1 Added

- Group metadata as a replacement for priority metadata. Each plugin belongs to a group, and a group can load after other groups. Plugins belong to the `default` group by default.
  - Added the *loot::Group* class to represent a group.
  - Added *loot::UndefinedGroupError*.
  - Added `GetGroups`, `GetUserGroups` and `SetUserGroups`.
  - Added `GetGroup`, `IsGroupExplicit` and `SetGroup`.
  - Updated `MergeMetadata` to replace the existing group with the given object's group if the latter is explicit.
  - Updated `NewMetadata` to return an object using the called object's group.
  - Updated `HasNameOnly` to check the group is implicit.
  - Updated `SortPlugins` to take into account plugin groups.

6.40.2 Changed

- `LoadPlugins` and `SortPlugins` no longer load the current load order state, so `LoadCurrentLoadOrderState` must be called separately.
- Updated libgit2 to v0.27.0.
- Updated libloadorder to v11.1.0.

6.40.3 Removed

- Support for local and global plugin priorities.
  - Removed the `loot::Priority` class.
  - Removed `PluginMetadata::GetLocalPriority()`, `PluginMetadata::GetGlobalPriority()`, `PluginMetadata::SetLocalPriority()` and `PluginMetadata::SetGlobalPriority()`
  - Priorities are no longer taken into account when sorting plugins.

6.40.4 Fixed

- An error when applying a load order for Morrowind, Oblivion, Fallout 3 or Fallout: New Vegas when a plugin had a timestamp earlier than 1970-01-01 00:00:00 UTC (via libloadorder).
- An error when loading the current load order for Skyrim with a `loadorder.txt` incorrectly encoded in Windows-1252 (via libloadorder).

6.41 0.12.5 - 2018-02-17

6.41.1 Changed

- Updated esplugin to v1.0.9.
- Updated libgit2 to v0.26.3. This enables TLS 1.2 support on Windows 7, so users shouldn't need to manually enable it themselves.

6.42 0.12.4 - 2018-02-17

6.42.1 Fixed

- Loading or saving a load order could be very slow because the plugins directory was scanned recursively, which is unnecessary. In the reported case, this fix caused saving a load order to go from 23 seconds to 43 milliseconds (via libloadorder).
- Plugin parsing errors were being logged with trace severity, they are now logged as errors.
- Saving a load order for Oblivion, Fallout 3 or Fallout: New Vegas now updates plugin access times to the current time for correctness (via libloadorder).

6.42.2 Changed

- `GameInterface::SetLoadOrder()` now errors if passed a load order that does not contain all installed plugins. The previous behaviour was to append any missing plugins, but this was undefined and could cause unexpected results (via libloadorder).
- Performance improvements for load order operations, benchmarked at 2x to 150x faster (via libloadorder).
- Updated mentions of libespm in error messages to mention esplugin instead.
- Updated libloadorder to v11.0.1.
- Updated spdlog to v0.16.3.

6.43 0.12.3 - 2018-02-04

6.43.1 Added

- Support for Fallout 4 VR via the new *`loot::GameType::fo4vr`* game type.

6.43.2 Fixed

- *`loot::CreateGameHandle()`* no longer accepts an empty game path string, and no longer has a default value for its game path parameter, as using an empty string as the game path is invalid and always causes an exception to be thrown.

6.43.3 Changed

- Added an empty string as the default value of `loot::InitialiseLocale`'s string parameter.
- Updated esplugin to v1.0.8.
- Updated libloadorder to v10.1.0.

6.44 0.12.2 - 2017-12-24

6.44.1 Fixed

- Plugins with a `.esp` file extension that have the light master flag set are no longer treated as masters when sorting, so they can have other `.esp` files as masters without causing cyclic interaction sorting errors.

6.44.2 Changed

- Downgraded Boost to 1.63.0 to take advantage of pre-built binaries on AppVeyor.

6.45 0.12.1 - 2017-11-23

6.45.1 Added

- Support for identifying Creation Club plugins using `Skyrim.ccc` and `Fallout4.ccc` (via libloadorder).

6.45.2 Changed

- Update esplugin to v1.0.7.
- Update libloadorder to v10.0.4.

6.46 0.12.0 - 2017-11-03

6.46.1 Added

- Support for light master (`.esl`) plugins.
- `LoadCurrentLoadOrderState` in *`loot::GameInterface`* to expose load order cache management to clients, as libloadorder no longer internally manages it.
- *`loot::SetLoggingCallback()`* to allow clients to handle the LOOT API's logging statements themselves.
- Logging of libloadorder error details.

6.46.2 Changed

- `LoadPlugins` now loads the current load order state before loading plugins.
- Added a *condition* string field to `SimpleMessage`.
- Replaced libespm dependency with esplugin v1.0.6. This significantly improves safety and sorting performance, especially for large load orders.
- Updated libloadorder to v10.0.3. This significantly improves safety and the performance of load order operations, at the expense of exposing cache management to the client.
- Replaced Boost.Log with spdlog v0.14.0, removing dependencies on several other Boost libraries in the process.
- Updated libgit2 to v0.26.0.
- Update Boost to v1.65.1.

6.46.3 Removed

- `DatabaseInterface::EvalLists()` as it was superseded in v0.11.0 by the ability to evaluate conditions when getting general messages and individual plugins' metadata, which is more efficient.
- `SetLoggingVerbosity()` and `SetLogFile()` as they have been superseded by the new *loot::SetLoggingCallback()* function.
- The `loot/yaml/*` headers containing LOOT's internal YAML conversion functions are no longer exposed alongside the API headers.
- The `loot/windows_encoding_converters.h` header is no longer exposed alongside the API headers.

6.46.4 Fixed

- Formatting in metadata documentation.
- Saving metadata wrote entries in an inconsistent order.
- Clang build errors.

6.47 0.11.1 - 2017-06-19

6.47.1 Fixed

- A crash would occur when loading an plugin that had invalid data past its header. Such plugins are now just silently ignored.
- *loot::CreateGameHandle()* would not resolve game or local data paths that are junction links correctly, which caused problems later when trying to perform actions such as loading plugins.
- Performing a masterlist update on a branch where the remote and local histories had diverged would fail. The existing local branch is now discarded and the remote branch checked out anew, as intended.

6.48 0.11.0 - 2017-05-13

6.48.1 Added

- New functions to *loot::DatabaseInterface*:
  - `WriteUserMetadata`
  - `GetKnownBashTags`
  - `GetGeneralMessages`
  - `GetPluginMetadata`
  - `GetPluginUserMetadata`
  - `SetPluginUserMetadata`
  - `DiscardPluginUserMetadata`
  - `DiscardAllUserMetadata`
  - `IsLatestMasterlist`

- A *loot::GameInterface* pure abstract class that exposes methods for accessing game-specific functionality.
- A *loot::PluginInterface* pure abstract class that exposes methods for accessing plugin file data.
- The `loot::SetLoggingVerbosity` and `loot::SetLogFile` functions and `loot::LogVerbosity` enum for controlling the API's logging behaviour.
- An `loot::InitialiseLocale` function that must be called to configure the API's locale before any of its other functionality is used.
- LOOT's internal metadata classes are now exposed as part of the API.

6.48.2 Changed

- Renamed `loot::CreateDatabase()` to *loot::CreateGameHandle()*, and changed its signature so that it returns a shared pointer to a *loot::GameInterface* instead of a shared pointer to a *loot::DatabaseInterface*.
- Moved `SortPlugins` into *loot::GameInterface*.
- Some *loot::DatabaseInterface* methods are now const:
  - `WriteMinimalList`
  - `GetMasterlistRevision`
- LOOT's internal YAML conversion functions have been refactored into the `include/loot/yaml` directory, but they are not really part of the API. They're only exposed so that they can be shared between the API and LOOT application without introducing another component.
- LOOT's internal string encoding conversion functions have been refactored into the `include/loot/windows_encoding_converters.h` header, but are not really part of the API. They're only exposed so that they can be shared between the API and LOOT application without introducing another component.
- Metadata is now cached more efficiently, reducing the API's memory footprint.
- Log timestamps now have microsecond precision.
- Updated to libgit2 v0.25.1.
- Refactored code only useful to the LOOT application out of the API internals and into the application source code.

6.48.3 Removed

- `DatabaseInterface::GetPluginTags()`, `DatabaseInterface::GetPluginMessages()` and `DatabaseInterface::GetPluginCleanliness()` have been removed as they have been superseded by `DatabaseInterface::GetPluginMetadata()`.
- The `GameDetectionError` class, as it is no longer thrown by the API.
- The `PluginTags` struct, as it is no longer used.
- The `LanguageCode` enum, as the API now uses ISO language codes directly instead.
- The `PluginCleanliness` enum. as it's no longer used. Plugin cleanliness should now be checked by getting a plugin's evaluated metadata and checking if any dirty info is present. If none is present, the cleanliness is unknown. If dirty info is present, check if any of the English info strings contain the text "Do not clean": if not, the plugin is dirty.
- The LOOT API no longer caches the load order, as this is already done more accurately by libloadorder (which is used internally).

6.48.4 Fixed

- Libgit2 error details were not being logged.
- A FileAccessError was thrown when the masterlist path was an empty string. The API now just skips trying to load the masterlist in this case.
- Updating the masterlist did not update the cached metadata, requiring a call to `LoadLists`.
- The reference documentation was broken due to an incompatibility between Sphinx 1.5.x and Breathe 4.4.

6.49 0.10.3 - 2017-01-08

6.49.1 Added

- Automated 64-bit API builds.

6.49.2 Changed

- Replaced `std::invalid_argument` exceptions thrown during condition evaluation with `ConditionSyntaxError` exceptions.
- Improved robustness of error handling when calculating file CRCs.

6.49.3 Fixed

- Documentation was not generated correctly for enums, exceptions and structs exposed by the API.
- Added missing documentation for `CyclicInteractionError` methods.

6.50 0.10.2 - 2016-12-03

6.50.1 Changed

- Updated libgit2 to 0.24.3.

6.50.2 Fixed

- A crash could occur if some plugins that are hardcoded to always load were missing. Fixed by updating to libloadorder v9.5.4.
- Plugin cleaning metadata with no `info` value generated a warning message with no text.

6.51 0.10.1 - 2016-11-12

No API changes.

6.52 0.10.0 - 2016-11-06

6.52.1 Added

- Support for TES V: Skyrim Special Edition.

6.52.2 Changed

- Completely rewrote the API as a C++ API. The C API has been reimplemented as a wrapper around the C++ API, and can be found in a separate repository.
- Windows builds now have a runtime dependency on the MSVC 2015 runtime redistributable.
- Rewrote the API documentation, which is now hosted online at Read The Docs.
- The Windows release archive includes the `.lib` file for compile-time linking.
- LOOT now supports v0.10 of the metadata syntax. This breaks compatibility with existing syntax. See *the syntax version history* for the details.
- Updated libgit2 to 0.24.2.

6.52.3 Removed

- The `loot_get_tag_map()` function has no equivalent in the new C++ API as it is obsolete.
- The `loot_apply_load_order()` function has no equivalent in the new C++ API as it just passed through to libloadorder, which clients can use directly instead.

6.52.4 Fixed

- Database creation was failing when passing paths to symlinks that point to the game and/or game local paths.
- Cached plugin CRCs causing checksum conditions to always evaluate to false.
- Updating the masterlist when the user's `TEMP` and `TMP` environmental variables point to a different drive than the one LOOT is installed on.

6.53 0.9.2 - 2016-08-03

6.53.1 Changed

- libespm (2.5.5) and Pseudosem (1.1.0) dependencies have been updated to the versions given in brackets.

6.53.2 Fixed

- The packaging script used to create API archives was packaging the wrong binary, which caused the v0.9.0 and v0.9.1 API releases to actually be re-releases of a snapshot build made at some point between v0.8.1 and v0.9.0: the affected API releases were taken offline once this was discovered.
- `loot_get_plugin_tags()` remembering results and including them in the results of subsequent calls.
- An error occurred when the user's temporary files directory didn't exist and updating the masterlist tried to create a directory there.
- Errors when reading some Oblivion plugins during sorting, including the official DLC.

6.54 0.9.1 - 2016-06-23

No API changes.

6.55 0.9.0 - 2016-05-21

6.55.1 Changed

- Moved API header location to the more standard `include/loot/api.h`.
- Documented LOOT's masterlist versioning system.
- Made all API outputs fully const to make it clear they should not be modified and to avoid internal const casting.
- The `loot_db` type is now an opaque struct, and functions that used to take it as a value now take a pointer to it.

6.55.2 Removed

- The `loot_cleanup()` function, as the one string it used to destroy is now stored on the stack and so destroyed when the API is unloaded.
- The `loot_lang_any` constant. The `loot_lang_english` constant should be used instead.

6.56 0.8.1 - 2015-09-27

6.56.1 Changed

- Safety checks are now performed on file paths when parsing conditions (paths must not reference a location outside the game folder).
- Updated Boost (1.59.0), libgit2 (0.23.2) and CEF (branch 2454) dependencies.

6.56.2 Fixed

- A crash when loading plugins due to lack of thread safety.
- The masterlist updater and validator not checking for valid condition and regex syntax.
- The masterlist updater not working correctly on Windows Vista.

6.57 0.8.0 - 2015-07-22

6.57.1 Added

- Support for metadata syntax v0.8.

6.57.2 Changed

- Improved plugin loading performance for computers with weaker multithreading capabilities (eg. non-hyperthreaded dual-core or single-core CPUs).
- LOOT no longer outputs validity warnings for inactive plugins.
- Updated libgit2 to v0.23.0.

6.57.3 Fixed

- Many miscellaneous bugs, including initialisation crashes and incorrect metadata input/output handling.
- LOOT silently discarding some non-unique metadata: an error will now occur when loading or attempting to apply such metadata.
- LOOT's version comparison behaviour for a wide variety of version string formats.

6.58 0.7.1 - 2015-06-22

6.58.1 Fixed

- "No existing load order position" errors when sorting.
- Output of Bash Tag removal suggestions in `loot_write_minimal_list()`.

6.59 0.7.0 - 2015-05-20

Initial API release.

CHAPTER

SEVEN

INTRODUCTION

The metadata syntax is what LOOT's masterlists and userlists are written in. If you know YAML, good news: the syntax is essentially just YAML 1.2. If you don't know YAML, then its Wikipedia page is a good introduction. All you really need to know is:

- How lists and associative arrays (key-value maps) are written.
- That whitespace is important, and that only normal spaces (ie. no non-breaking spaces or tabs) count as such.
- That data entries that are siblings must be indented by the same amount, and child data nodes must be indented further than their parents (see the example later in this document if you don't understand).
- That YAML files must be written in a Unicode encoding.
- That each key in a key-value map must only appear once per map object.

An important point that is more specific to how LOOT uses YAML:

- Strings are case-sensitive, apart from file paths, regular expressions and checksums.
- File paths are evaluated relative to the game's Data folder.
- File paths cannot reference a path outside of the game's folder structure, ie. they cannot contain the substring `../../`.

In this document, where a value's type is given as `X list` this is equivalent to a YAML sequence of values which are of the data type `X`. Where a value's type is given as `X set`, this is equivalent to a YAML sequence of **unique** values which are of the data type `X`. Uniqueness is determined using the equality criteria for that data type. All the non-standard data types that LOOT's metadata syntax uses have their equality criteria defined later in this document.

Some strings are interpreted as CommonMark: where this is the case, the strings are interpreted according to version `0.29` of the specification.

CHAPTER

EIGHT

METADATA FILE STRUCTURE

The root of a metadata file is a key-value map. LOOT will recognise the following keys, none of which are required. Other keys may also be present, but are not processed by LOOT.

`prelude`
: The prelude can have any value, but if a masterlist prelude path is provided when loading metadata, the masterlist's `prelude` value will be replaced by the parsed content of the masterlist prelude file. The prelude exists so that metadata that is common across different masterlists can be shared without duplication.

  Note that prelude replacement is only supported when using YAML's block style and an unquoted `prelude` key that is not preceded by a mapping key indicator and that is immediately followed by a colon separator, i.e. `prelude:`.

`bash_tags`
: string list

  A list of Bash Tags that are supported by the game. These Bash Tags are used to provide autocomplete suggestions in LOOT's metadata editor.

`globals`
: message list

  A list of message data structures for messages that are displayed independently of any plugin.

`groups`
: group set

  A set of group data structures that represent the groups that plugins can belong to.

`plugins`
: plugin list *and* plugin set

  The plugin data structures that hold all the plugin metadata within the file. It is a mixture of a list and a set because **no non-regex plugin value may be equal to any other non-regex plugin value** , but there may be any number of equal regex plugin values, and non-regex plugin values may be equal to regex plugin values. If multiple plugin values match a single plugin, their metadata is merged in the order the values are listed, and as defined in *Merging Behaviour*.

The message and plugin data structures are detailed in the next section.

8.1 Example

```yaml
prelude:
  - &thanksForUsing
    type: say
    content: 'Thanks for using LOOT!'
    condition: 'file("LOOT")'

bash_tags:
  - 'C.Climate'
  - 'Relev'

globals:
  - *thanksForUsing

groups:
  - name: 'Map Markers'
    after:
      - 'default'

plugins:
  - name: 'Armamentarium.esm'
    tag:
      - Relev
  - name: 'ArmamentariumFran.esm'
    tag:
      - Relev
  - name: 'Beautiful People 2ch-Ed.esm'
    tag:
      - Eyes
      - Graphics
      - Hair
      - R.Relations
  - name: 'More Map Markers.esp'
    group: 'Map Markers'
```

CHAPTER

NINE

DATA STRUCTURES

LOOT expects metadata to be laid out using a certain set of data structures, described in this section.

9.1 Tag

LOOT metadata files can contain suggestions for the addition or removal of Bash Tags, and this is the structure used for them. It has two forms: a key-value string map and a scalar string.

9.1.1 Map Form

`name`
: **Required.** A Bash Tag, prefixed with a minus sign if it is suggested for removal.

`condition`
: A condition string that is evaluated to determine whether this Bash Tag should be suggested: if it evaluates to true, the Tag is suggested, otherwise it is ignored. See *Condition Strings* for details. If undefined, defaults to an empty string.

9.1.2 Scalar Form

The scalar form is simply the value of the map form's `name` key. Using the scalar form is equivalent to using the map form with an undefined `condition` key.

9.1.3 Equality

Two tag data structures are equal if all their fields are equal. String equality is case-sensitive.

9.1.4 Examples

Scalar form:

```
Relations
```

Map form:

```
name: -Relations
condition: 'file("Mart''s Monster Mod for OOO.esm") or file("FCOM_Convergence.esm")'
```

9.2 File

This structure can be used to hold file paths. It has two forms: a key-value string map and a scalar string.

9.2.1 Map Form

`name`
: **Required.** An exact (ie. not regex) file path or name.

`display`
: A CommonMark string, to be displayed instead of the file path in any generated messages, eg. the name of the mod the file belongs to.

`detail`
: `string` or `localised content list`

  if this file causes an error message to be displayed (e.g. because it's a missing requirement), this detail message content will be appended to that error message. If a string is provided, it will be interpreted as CommonMark. If a localised content list is provided, one of the structures must be for English. If undefined, defaults to an empty string.

`condition`
: A condition string that is evaluated to determine whether this file data should be used: if it evaluates to true, the data is used, otherwise it is ignored. See *Condition Strings* for details.

9.2.2 Scalar Form

The scalar form is simply the value of the map form's `name` key. Using the scalar form is equivalent to using the map form with undefined `display` and `condition` keys.

9.2.3 Equality

Two file data structures are equal if all their fields are equal. `name` field equality is case-insensitive, the other fields use case-sensitive equality.

9.2.4 Examples

Scalar form:

```
'../obse_loader.exe'
```

Map form:

```
name: '../obse_loader.exe'
condition: 'version("../obse_loader.exe", "0.0.18.0", >=)'
display: 'OBSE v18+'
```

9.3 Group

Groups represent sets of plugins, and are a way to concisely and extensibly load sets of plugins after other sets of plugins.

This structure can be used to hold group definitions. It is a key-value map.

`name`
: `string`

  Required. A case-sensitive name that identifies the group.

`description`
: `string`

  A CommonMark description of the group, e.g. what sort of plugins it contains. If undefined, the description is an empty string.

`after`
: `string set`

  The names of groups that this group loads after. Group names are case-sensitive. If undefined, the set is empty. The named groups must be defined when LOOT sorts plugins, but they don't need to be defined in the same metadata file.

  Sorting errors will occur if:

  - A group loads after another group that does not exist.
  - Group loading is cyclic (e.g. A loads after B and B loads after A).

9.3.1 Merging Groups

When a group definition for an already-defined group is encountered, the `description` field is replaced if the new value is not an empty string, and the `after` sets of the two definitions are merged.

9.3.2 The `default` Group

There is one predefined group named `default` that all plugins belong to by default. It is defined with an empty `after` set, as no other predefined groups exist for it to load after.

Like any other group, the `default` group can be redefined to add group names to its `after` set.

9.3.3 Equality

Two group data structures are equal if the values of their `name` keys are identical.

9.3.4 Examples

```yaml
# Create a group for map marker plugins that loads after the predefined
# 'default' group.
name: 'Map Markers'
description: 'A group for map marker plugins that need to load late.'
after:
  - 'default'
```

```yaml
# Extend the predefined 'default' group to load after an 'Unofficial Patches'
# group that is defined elsewhere.
name: 'default'
after:
  - 'Unofficial Patches'
```

9.4 Localised Content

The localised content data structure is a key-value string map.

`text`
: **Required.** The CommonMark message content string.

`lang`
: **Required.** The language that `text` is written in, given as a code of the form `ll` or `ll_CC`, where `ll` is an ISO 639-1 language code and `CC` is an ISO 3166 country code. For example,

Language	Code
Bulgarian	bg
Chinese (Simplified)	zh_CN
Czech	cs
Danish	da
English	en
Finnish	fi
French	fr
German	de
Italian	it
Japanese	ja
Korean	ko
Polish	pl
Portuguese	pt_PT
Portuguese (Brazil)	pt_BR
Russian	ru
Spanish	es
Swedish	sv
Ukrainian	uk_UA

9.4.1 Equality

Two localised content data structures are equal if all their fields are equal. Field equality is case-sensitive.

9.5 Message

Messages are given as key-value maps.

type
: `string`

  Required. The type string can be one of three keywords.

  say
  : A generic message, useful for miscellaneous notes.

  warn
  : A warning message, describing a non-critical issue with the user's mods (eg. dirty mods).

  error
  : An error message, decribing a critical installation issue (eg. missing masters, corrupt plugins).

content
: `string` or `localised content list`

  Required. Either simply a CommonMark string, or a list of localised content data structures. If the latter, one of the structures must be for English.

condition
: `string`

  A condition string that is evaluated to determine whether the message should be displayed: if it evaluates to true, the message is displayed, otherwise it is not. See *Condition Strings* for details.

subs
: `string list`

  A list of CommonMark strings to be substituted into the message content string. The content string must use numbered specifiers (`%1%`, `%2%`, etc.), where the numbers correspond to the position of the substitution string in this list to use, to denote where these strings are to be substituted.

9.5.1 Language Support

If a message's `content` value is a string, the message will use the string as its content if displayed. Otherwise, the first localised content structure with a language or locale code that matches LOOT's current language will be used as the message's content if displayed. If there are no exact matches, LOOT will try to find a close match. If LOOT's current language uses a locale code, it will display the first structure with the same language code, but not another locale code with the same language code. For example, if LOOT's current language has locale code `pt_BR`, it will display the first structure with language code `pt` (but not locale code `pt_PT`) if none exist with locale code `pt_BR`. If LOOT's current language has a language code, it will display the first structure with a locale code that contains that language code. For example, if LOOT's current language has language code `pt`, it will display the first structure with locale code `pt_PT` or `pt_BR` if none exist with language code `pt`. If there are no exact or close matches, then the first structure in English will be used.

9.5.2 Equality

Two message data structures are equal if their *type*, *content* and *condition* fields are equal, after any *subs* values have been substituted into *content* strings. If the *content* field is a string, it is treated as a localised content list containing a single English-language string. String equality is case sensitive.

9.5.3 Examples

```
type: say
content:
  - lang: en
    text: 'An example link: <http://www.example.com>'
  - lang: zh_CN
    text: ': <http://www.example.com>'
condition: 'file("foo.esp")'
```

would be displayed as

- : http://www.example.com

if the current language was Simplified Chinese and `foo.esp` was installed, while

```
type: say
content: 'An alternative [example link](http://www.example.com), with no translations.'
```

would be displayed as

- An alternative example link, with no translations.

In English,

```
type: say
content: 'A newer version of %1% [is available](%2%).'
subs:
  - 'this plugin'
  - 'http://www.example.com'
```

would be displayed as

- A newer version of this plugin is available.

9.6 Location

This data structure is used to hold information on where a plugin is hosted online. It has two forms: a key-value string map and a scalar string.

9.6.1 Map Form

`link`
: **Required.** A URL at which the plugin is found.

`name`
: A descriptive name for the URL, which may be used as hyperlink text. If undefined, defaults to an empty string.

9.6.2 Scalar Form

The scalar form is simply the value of the map form's `link` key. Using the scalar form is equivalent to using the map form with an undefined `name` key.

9.6.3 Equality

Two location data structures are equal if all their fields are equal. Field equality is case-sensitive.

9.6.4 Examples

Scalar form:

```
'https://www.nexusmods.com/skyrim/mods/71214'
```

Map form:

```
link: 'https://www.nexusmods.com/skyrim/mods/71214'
name: 'USLEEP on Skyrim Nexus'
```

9.7 Cleaning Data

This structure holds information on which versions of a plugin are dirty or clean, and if dirty, how many identical-to-master records, deleted records and deleted navmeshes (if applicable) it contains. Cleaning data is given as a key-value map.

`crc`
: `hexadecimal integer`

  Required. The CRC-32 checksum of the plugin. If the plugin is dirty, this needs to be the CRC of the plugin before before cleaning. LOOT displays the CRCs of installed plugins in its report. The 8-character CRC should be preceded by `0x` so that it is interpreted correctly.

`util`
: `string`

  Required. The utility that was used to check the plugin for dirty edits. If available, the version of the utility used should also be included (e.g. `TES5Edit v3.11`). The string will be interpreted as CommonMark.

`detail`
: `string` or `localised content list`

  A message that will be displayed to the user. If a string is provided, it will be interpreted as CommonMark. If a localised content list is provided, one of the structures must be for English. This is only used if the plugin is dirty, and is intended for providing cleaning instructions to the user. If undefined, defaults to an empty string.

itm
: `integer`

  The number of identical-to-master records reported for the dirty plugin. If undefined, defaults to zero.

udr
: `integer`

  The number of undeleted records reported for the dirty plugin. If undefined, defaults to zero.

nav
: `integer`

  The number of deleted navmeshes reported for the dirty plugin. If undefined, defaults to zero.

9.7.1 Equality

Two plugin cleaning data structures are equal if all their fields are equal. *util* field equality is case-sensitive. If the *detail* field is a string, it is treated as a localised content data structure.

9.7.2 Examples

A dirty plugin:

```
crc: 0x3DF62ABC
util: '[TES5Edit](http://www.nexusmods.com/skyrim/mods/25859) v3.1.1'
detail: 'A cleaning guide is available [here](http://www.creationkit.com/index.php?
↪title=TES5Edit_Cleaning_Guide_-_TES5Edit).'
itm: 4
udr: 160
```

A clean plugin:

```
crc: 0x2ABC3DF6
util: '[TES5Edit](http://www.nexusmods.com/skyrim/mods/25859) v3.1.1'
```

9.8 Plugin

This is the structure that brings all the others together, and forms the main component of a metadata file. It is a key-value map.

name
: `string`

  Required. Can be an exact plugin filename or a regular expression plugin filename. If the filename contains any of the characters `:\*?|`, the string will be treated as a regular expression, otherwise it will be treated as an exact filename. For example, `Example\.esm` will be treated as a regular expression, as it contains a `\` character.

  Regular expression plugin filenames must be written in modified ECMAScript syntax.

group
: `string`

  The name of the group the plugin belongs to. If unspecified, defaults to `default`.

The named group must be exist when LOOT sorts plugins, but doesn't need to be defined in the same metadata file. If at sort time the group does not exist, a sorting error will occur.

The plugin must load after all the plugins in the groups its group is defined to load after, resolving them recursively. An exception exists if doing so would introduce a cyclic dependency between two plugins without any other group loading rules applied.

For example, if for plugins A.esp, B.esp, C.esp and D.esp:

- B.esp has A.esp as a master
- A.esp is in group A
- B.esp and C.esp are in the default group
- D.esp is in group D
- group A loads after the default group
- the default group loads after group D

Then the load order must be D.esp, C.esp, A.esp, B.esp. Although A.esp's group must load after B.esp's group, this would cause a cycle between A.esp and B.esp, so the requirement is ignored for that pair of plugins.

However, if for plugins A.esp, B.esp and C.esp in groups of the same names:

1. group B loads after group A
2. group C loads after group B
3. A.esp has C.esp as a master

This will cause a sorting error, as neither group rule introduces a cyclic dependency when combined in isolation with the third rule, but having all three rules applied causes a cycle.

`after`
`file set`

Plugins that this plugin must load after, but which are not dependencies. Used to resolve specific compatibility issues. If undefined, the set is empty.

Note: since an `after` entry uses a `file` structure, its `name` value can't be a regex. This applies to `req` & `inc` entries too.

`req`
`file set`

Files that this plugin requires to be present. This plugin will load after any plugins listed. If any of these files are missing, an error message will be displayed. Intended for use specifying implicit dependencies, as LOOT will detect a plugin's explicit masters itself. If undefined, the set is empty.

Note: if a `condition` is used in a `req` entry, a requirement message will only be displayed if the file isn't present *and* the `condition` is true.

`inc`
`file set`

Files that this plugin is incompatible with. If any of these files are present, an error message will be displayed. If undefined, the set is empty.

`msg`
`message list`

The messages attached to this plugin. The messages will be displayed in the order that they are listed. If undefined, the list is empty.

`tag`
: `tag set`

  Bash Tags suggested for this plugin. If a Bash Tag is suggested for both addition and removal, the latter will override the former when the list is evaluated. If undefined, the set is empty.

`url`
: `location set`

  An unordered set of locations for this plugin. If the same version can be found at multiple locations, only one location should be recorded. If undefined, the set is empty. This metadata is not currently used by LOOT.

`dirty`
: `cleaning data set`

  An unordered set of cleaning data structures for this plugin, identifying dirty plugins.

`clean`
: `cleaning data set`

  An unordered set of cleaning data structures for this plugin, identifying clean plugins. The `itm`, `udr` and `nav` fields are unused in this context, as they're assumed to be zero.

9.8.1 Equality

The equality of two plugin data structures is determined by comparing the values of their `name` keys.

- If neither or both values are regular expressions, then the plugin data structures are equal if the lowercased values are identical.
- If one value is a regular expression, then the plugin data structures are equal if the other value is an exact match for it.

9.8.2 Merging Behaviour

Key	Merge Behaviour (merging B into A)
name	Not merged.
group	Replaced by B's value only if A has no value set.
after	Merged. If B's file set contains an item that is equal to one already present in A's file set, B's item is discarded.
req	Merged. If B's file set contains an item that is equal to one already present in A's file set, B's item is discarded.
inc	Merged. If B's file set contains an item that is equal to one already present in A's file set, B's item is discarded.
msg	Merged. B's message list is appended to A's message list.
tag	Merged. If B's tag set contains an item that is equal to one already present in A's tag set, B's item is discarded.
url	Merged. If B's location set contains an item that is equal to one already present in A's location set, B's item is discarded.
dirty	Merged. If B's dirty data set contain an item that is equal to one already present in A's dirty data set, B's item is discarded.
clean	Merged. If B's clean data set contain an item that is equal to one already present in A's clean data set, B's item is discarded.

9.8.3 Examples

```yaml
name: 'Oscuro''s_Oblivion_Overhaul.esm'
req:
  - 'Oblivion.esm'  # Don't do this, Oblivion.esm is a master of Oscuro's_Oblivion_
↪Overhaul.esm, so LOOT already knows it's required.
  - name: 'example.esp'
    display: '[Example Mod](http://www.example.com)'
    condition: 'version("Oscuro''s_Oblivion_Overhaul.esm", "15.0", ==)'
tag:
  - Actors.Spells
  - Graphics
  - Invent
  - Relations
  - Scripts
  - Stats
  - name: -Relations
    condition: 'file("Mart''s Monster Mod for OOO.esm") or file("FCOM_Convergence.esm")'
msg:
  - type: say
    content: 'Do not clean. "Dirty" edits are intentional and required for the mod to␣
↪function.'
```

CHAPTER

TEN

CONDITION STRINGS

Condition strings can be used to ensure that data is only acted on by LOOT under certain circumstances. They are very similar to boolean conditional expressions in programming languages such as Python, though more limited.

Omitting optional parentheses (see below), their EBNF grammar is:

```
expression          ::=   condition, { logical_or, compound_condition }
compound_condition  ::=   condition, { logical_and, condition }
condition           ::=   ( [ logical_not ], function ) | ( [ logical_not ], "(", expression, ")" )
logical_and         ::=   "and"
logical_or          ::=   "or"
logical_not         ::=   "not"
```

10.1 Types

filesystem_path
: A double-quoted filesystem path, or `"LOOT"`, which resolves to *LOOT.exe* in the current working directory. Bear in mind that *LOOT.exe* may not be present if the condition is being evaluated by an application other than LOOT.

file_path
: A double-quoted file path, or `"LOOT"`, which resolves to *LOOT.exe* in the current working directory. Bear in mind that *LOOT.exe* may not be present if the condition is being evaluated by an application other than LOOT.

regular_expression
: A double-quoted file path, with a regular expression in place of a filename. The path must use `/` for directory separators, not `\`. The regular expression must be written in a modified Perl syntax.

  Only the filename path component will be evaluated as a regular expression. For example, given the regex file path `Meshes/Resources(1|2)/(upperclass)?table.nif`, LOOT will look for a file named `table.nif` or `upperclasstable.nif` in the `Meshes\Resources(1|2)` folder, rather than looking in the `Meshes\Resources1` and `Meshes\Resources2` folders.

checksum
: A string of hexadecimal digits representing an unsigned integer that is the data checksum of a file. LOOT displays the checksums of plugins in its user interface after running.

version
: A double-quoted string of characters representing the version of a plugin or executable. LOOT displays the versions of plugins in its user interface after running.

comparison_operator
: One of the following comparison operators.

`==`
: Is equal to

`!=`
: Is not equal to

`<`
: Is less than

`>`
: Is greater than

`<=`
: Is less than or equal to

`>=`
: Is greater than or equal to

10.2 Functions

There are several conditions that can be tested for using the functions detailed below. All functions return a boolean. For functions that take a path or regex, the argument is treated as regex if it contains any of the characters `:\*?|`.

`file(filesystem_path path)`
: Returns true if `path` is installed, and false otherwise.

`file(regular_expression regex)`
: Returns true if a file matching `regex` is found, and false otherwise.

`readable(filesystem_path path)`
: Returns true if `path` is a readable directory or file, and false otherwise.

  This is particularly useful when writing conditions for games that are available from the Microsoft Store and/or Xbox app, as games installed using them have executables that have heavily restricted permissions, and attempts to read them result in permission denied errors. You can use this function to guard against such errors by calling it before the `checksum`, `version` or `product_version` functions.

`active(file_path path)`
: Returns true if `path` is an active plugin, and false otherwise.

`active(regular_expression regex)`
: Returns true if an active plugin matching `regex` is found, and false otherwise.

`many(regular_expression regex)`
: Returns true if more than one file matching `regex` is found, and false otherwise.

`many_active(regular_expression regex)`
: Returns true if more than one active plugin matching `regex` is found, and false otherwise.

`is_master(file_path path)`
: Returns true if `path` is an installed master plugin, and false otherwise.

`checksum(file_path path, checksum expected_checksum)`
: Returns true if the calculated CRC-32 checksum of `path` matches `expected_checksum`, and false otherwise. Returns false if `path` does not exist.

`version(file_path path, version given_version, comparison_operator comparator)`
: Returns true if the boolean expression:

```
actual_version comparator given_version
```

(where `actual version` is the version read from `path`) holds true, and false otherwise.

- If `path` is a plugin, its version is read from its description field.
- If `path` is not a plugin, it will be assumed to be an executable (e.g. `*.exe` or `*.dll`), and its version is read from its File Version field.
- If `path` does not exist or does not have a version number, the condition evaluates to true for the `!=`, `<` and `<=` comparators, i.e. a missing version is always less than the given version.
- If `path` is not readable or is not a plugin or an executable, an error will occur.

The supported version syntax and precedence rules are detailed in the section below.

product_version(file_path path, version given_version, comparison_operator comparator)

Returns true if the boolean expression:

```
actual_version comparator given_version
```

(where `actual version` is the version read from `path`) holds true, and false otherwise. `path` must be an executable (e.g. `*.exe` or `*.dll`), and its version is read from its Product Version field.

- If `path` does not exist or does not have a version number, the condition evaluates to true for the `!=`, `<` and `<=` comparators, i.e. a missing version is always less than the given version.
- If `path` is not a readable executable, an error will occur.

The supported version syntax and precedence rules are detailed in the section below.

10.2.1 Version Syntax & Comparison Rules

Version parsing and comparison is compatible with Semantic Versioning, with the following exceptions:

- Pre-release identifiers may not include hyphens (`-`), as they are treated as separators. For example, a SemVer-compliant parser would treat `1.0.0-alpha.1.x-y-z.--` as `([1, 0, 0], ["alpha", 1, "x-y-z", "--"])` but libloot treats it as `([1, 0, 0], ["alpha", 1, "x", "y", "z", "", ""])`.
- Identifiers that contain non-digit characters are lowercased before being compared lexically, so that their comparison is case-insensitive instead of case-sensitive. For example, SemVer specifies that `1.0.0-alpha` is greater than `1.0.0-Beta`, but libloot compares them with the opposite result.

These exceptions are necessary to support an extended range of real-world versions that do not conform to SemVer. The supported extensions are:

- Leading zeroes are allowed and ignored in major, minor and patch version numbers and numeric pre-release IDs. For example, `01.02.03` and `1.2.3` are equal.
- An arbitrary number of version numbers is allowed. To support this, the major, minor and patch version numbers are treated as a sequence of numeric release IDs, and any subsequent version numbers are just additional release IDs that get appended to the sequence. For example, `1.2.3` may be represented as the sequence `[1, 2, 3]`, and `1.2.3.4` would be represented as `[1, 2, 3, 4]`.

  If two versions with a different number of release identifiers are compared, the version with fewer release identifiers is padded with zero values until they are the same length. Each release identifier in one version is then compared against the release identifier in the same position in the other version. For example, `1-beta` is padded to `1.0.0-beta` before being compared against `1.0.1-beta`, and the result is that `1.0.1-beta` is greater than `1-beta`.
- Release IDs may be separated by a period (`.`) or a comma (`,`). For example, `1.2.3.4` and `1,2,3,4` are equal.

- The separator between release IDs and pre-release IDs may be a hyphen (`-`), a space (" "), a colon (`:`) or an underscore (`_`). For example, `1.2.3-alpha`, `1.2.3 alpha`, `1.2.3:alpha` and `1.2.3_alpha` are all equal.
- Pre-release IDs may be separated by a period (`.`), a hyphen (`-`), a space (" "), a colon (`:`) or an underscore (`_`). For example, `1.2.3-alpha.1`, `1.2.3-alpha-1`, `1.2.3-alpha 1`, `1.2.3-alpha:1` and `1.2.3-alpha_1` are all equal.
- Non-numeric release IDs are allowed. A non-numeric release ID may contain any character (not just ASCII characters) that is not one of the separators listed above or a plus sign (+). For example, `0.78b.1` is allowed.

  Non-numeric release IDs use the same comparison rules as non-numeric pre-release IDs, with the exception that a non-numeric release ID is not always greater than a numeric release ID:

  - If the non-numeric release ID has no leading digits, it is greater than the numeric release ID. For example, `1.A` is greater than `1.1`.
  - If the non-numeric release ID has leading digits, they are parsed as a number, and this is compared against the numeric release ID:
    - If the two numbers are equal then the non-numeric release ID is greater than the numeric release ID. For example, `1.1A` is greater than `1.1`.
    - Otherwise, the result of comparing the two numbers is used as the result of comparing the two release IDs. For example, `1.2` is greater than `1.1A` and `1.1A` is greater than `1.0`.
- Pre-release IDs may contain any character (not just ASCII characters) that is not one of the pre-release ID separators listed above or a plus sign (+).
- Before non-numeric IDs (release or pre-release) are compared, they are lowercased according to Unicode's lowercasing rules.
- As a special case, version strings that are four comma-and-space-separated sequences of digits are interpreted as if the comma-and-space separators were periods (`.`). For example, `0, 2, 0, 12` and `0.2.0.12` are equal.

10.3 Logical Operators

The `and`, `or` and `not` operators have their usual definitions.

10.3.1 Order of Evaluation

Condition strings are evaluated according to the usual C-style operator precedence rules, and parentheses can be used to override these rules. For example:

```
function and function or not function
```

is evaluated as:

```
( function and function ) or ( not function )
```

but:

```
function and ( function or not function )
```

is evaluated as:

```
function and ( function or ( not function ) )
```

10.4 Performance

LOOT caches the results of condition evaluations. A regular expression check will still take longer than a file check though, so use the former only when appropriate to do so.

CHAPTER

ELEVEN

VERSION HISTORY

The version history of the metadata syntax is given below.

11.1 0.18 - 2022-02-27

11.1.1 Added

- The condition function `readable(filesystem_path path)`, which checks if the given path is a readable directory or file.

11.1.2 Changed

- The documentation for the version comparison condition functions has been updated to detail the supported version syntax and semantics.
- Mentions of GitHub Flavored Markdown have been replaced with CommonMark, as LOOT now uses the latter instead of the former.

11.1.3 Fixed

- Support for `not (<expression>)` syntax was not properly documented.
- The documentation for the version comparison functions stated that missing versions would be treated as if they were `0`, which was not accurate.

11.2 0.17 - 2021-09-24

11.2.1 Added

- The `File` data structure now has a `detail` key that takes a string or localised content list.
- The top-level `prelude` key can be used to supply common data structure definitions, and in masterlists its value is replaced by the contents of the masterlist prelude file, if present.
- Support for parsing inverted metadata conditions (`not (<expression>)`).

11.2.2 Changed

- The cleaning data structure's `info` key has been renamed to `detail` for consistency.

11.3 0.16 - 2020-07-12

11.3.1 Changed

- Equality for all metadata data structures is now determined by comparison of all their fields. String comparison is case-sensitive, with the exception of `File`'s `name` field.

11.3.2 Removed

- The `enabled` field has been removed from plugin metadata objects.

11.4 0.15 - 2019-11-05

11.4.1 Added

- The condition function `is_master(file_path path)`, which checks if the given file is an installed master plugin.

11.5 0.14 - 2018-12-09

11.5.1 Added

- The `Group` data structure now has a `description` key that takes a string value.
- The condition function `product_version(file_path path, version given_version, comparison_operator comparator)`, which checks against the Product Version field of an executable.

11.5.2 Changed

- `clean` and `dirty` metadata are now allowed in regex plugin entries.
- `Location`, `Message`, `MessageContent` and `Tag` equality comparisons are now case-sensitive.
- Regular expressions in condition strings now use a modified Perl grammar instead of a modified ECMAScript grammar. `Plugin` object `name` fields still use the modified ECMAScript grammar for regex values. To improve portability and avoid mistakes, it's best to stick to using the subset of regular expression features that are common to both grammars.

11.5.3 Removed

- The change in regular expression grammar means that the following regular expression features are no longer supported in condition strings:
  - `\c<letter>` control code escape sequences, use `\x<hex>` instead
  - The `\0` null escape sequence, - use `\x00` instead
  - The `[:d:]`, `[:w:]` and `[:s:]` character classes, use `[:digit:]`, `[:alnum:]` and `[:space:]` instead respectively.
  - `\<number>` backreferences
  - `(?=<subpattern>)` and `(?!<subpattern>)` positive and negative lookahead

11.6 0.13 - 2018-04-02

11.6.1 Added

- The `Group` data structure.
- The `groups` list to the root of the metadata file format.
- The `group` key to the plugin data structure.

11.6.2 Removed

- The `priority` field from the plugin data structure.
- The `global_priority` field from the plugin data structure.

11.7 0.10 - 2016-11-06

11.7.1 Added

- The `clean` key to the plugin data structure.
- The `global_priority` field to the plugin data structure.
- The `many_active()` condition function.
- The `info` key to the cleaning data structure.

11.7.2 Changed

- Renamed the `str` key in the localised content data structure to `text` .
- The `priority` field of the plugin data structure now stores values between -127 and 127 inclusive.
- Regular expressions no longer accept \ as a directory separator: / must now be used.
- The `file()` condition function now also accepts a regular expression.
- The `active()` condition function to also accept a regular expression.

- Renamed the dirty info data structure to the cleaning data structure.

11.7.3 Removed

- The `regex()` condition function, as it has been obsoleted by the `file()` function's new regex support.

11.8 0.8 - 2015-07-22

11.8.1 Added

- The `name` key to the location data structure.
- The `many("regex")` condition function.
- The documentation now defines the equality criteria for all of the metadata syntax's non-standard data structures.

11.8.2 Changed

- Detection of regular expression plugin entries. Previously, a plugin entry was treated as having a regular expression filename if the filename ended with `\.esp` or `\.esp` . Now, a plugin entry is treated as having a regular expression filename if the filename contains one or more of `:\*?|` .

11.8.3 Removed

- Removed the `ver` key in the location data structure.

11.8.4 Fixed

- The documentation gave the values of the `after` , `req` , `inc` , `tag` , `url` and `dirty` keys as lists, when they have always been sets.

11.9 0.7 - 2015-05-20

11.9.1 Added

- The message string substitution key, i.e. `sub` , in the message data structure.
- Support for YAML merge keys, i.e. `<<` .

11.9.2 Changed

- Messages may now be formatted using most of GitHub Flavored Markdown, minus the GitHub-specific features (like @mentions, issue/repo linking and emoji).

11.10 0.6 - 2014-07-05

No changes.

11.11 0.5 - 2014-03-31

Initial release.

CHAPTER

TWELVE

COPYRIGHT NOTICE

LOOT and its API are distributed under the GNU General Public License v3.0. The documentation is distributed under the GNU Free Documentation License v1.3. The full texts of both licenses are included in *Copyright License Texts*.

While the GPL license allows anyone to make derivative works of LOOT, the LOOT Team encourages those thinking of doing so to first discuss their reasoning for such an endeavour with the Team. It may be that what the derivative work would do differently is already planned for a future version of LOOT or would be happily integrated into LOOT, thus avoiding any extra effort by others.

LOOT has been specifically designed to prevent it being locked into the LOOT Team's official masterlist repositories. Nevertheless, the LOOT Team appeals to the community to avoid the distribution of unofficial masterlists, as this would only hamper the effort to create one set of stores for load order information. Any issues with a masterlist are best brought to the attention of the LOOT Team so that they may be remedied.

GNU Free Documentation License Version 1.3 Notice:

> Copyright (C) 2012—2016 WrinklyNinja
>
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
>
> A copy of the license is included in *Copyright License Texts*.

CHAPTER

THIRTEEN

COPYRIGHT LICENSE TEXTS

Contents

13.1 Boost

```
Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization
obtaining a copy of the software and accompanying documentation covered by
this license (the "Software") to use, reproduce, display, distribute,
execute, and transmit the Software, and to prepare derivative works of the
Software, and to permit third-parties to whom the Software is furnished to
do so, all subject to the following:

The copyright notices in the Software and this entire statement, including
the above license grant, this restriction and the following disclaimer,
must be included in all copies of the Software, in whole or in part, and
all derivative works of the Software, unless such copies or derivative
works are solely in the form of machine-executable object code generated by
a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT
SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE
FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

13.2 libloot, esplugin & Libloadorder

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
```

(continues on next page)

(continued from previous page)

```
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
modification follow.

                       TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License.  Each licensee is addressed as "you".  "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy.  The resulting work is called a "modified version" of the
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
on the Program.

  To "propagate" a work means to do anything with it that, without
permission, would make you directly or secondarily liable for
infringement under applicable copyright law, except executing it on a
computer or modifying a private copy.  Propagation includes copying,
distribution (with or without modification), making available to the
public, and in some countries other activities as well.

  To "convey" a work means any kind of propagation that enables other
```

(continues on next page)

(continued from previous page)

```
parties to make or receive copies.  Mere interaction with a user through
a computer network, with no transfer of a copy, is not conveying.

  An interactive user interface displays "Appropriate Legal Notices"
to the extent that it includes a convenient and prominently visible
feature that (1) displays an appropriate copyright notice, and (2)
tells the user that there is no warranty for the work (except to the
extent that warranties are provided), that licensees may convey the
work under this License, and how to view a copy of this License.  If
the interface presents a list of user commands or options, such as a
menu, a prominent item in the list meets this criterion.

  1. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it.  "Object code" means any non-source
form of a work.

  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.

  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form.  A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

  The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities.  However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but
which are not part of the work.  For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.

  The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

  The Corresponding Source for a work in source code form is that
```

(continues on next page)

(continued from previous page)

```
same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.
```

(continues on next page)

(continued from previous page)

```
  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.
```

(continues on next page)

(continued from previous page)

```
    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
    you inform other peers where the object code and Corresponding
    Source of the work are being offered to the general public at no
    charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.

  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
```

(continues on next page)

(continued from previous page)

```
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.

  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.
```

(continues on next page)

(continued from previous page)

```
  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
```

(continues on next page)

(continued from previous page)

```
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
```

(continues on next page)

(continued from previous page)

```
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
```

(continues on next page)

(continued from previous page)

```
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

  A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License.  You may not convey a covered
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.

  Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

  12. No Surrender of Others' Freedom.

  If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you may
not convey it at all.  For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

  13. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
```

(continues on next page)

(continued from previous page)

```
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.

  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.
```

(continues on next page)

(continued from previous page)

```
                    END OF TERMS AND CONDITIONS
```

13.3 libloot Documentation

```
                GNU Free Documentation License
                 Version 1.3, 3 November 2008


 Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.
     <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other
functional and useful document "free" in the sense of freedom: to
assure everyone the effective freedom to copy and redistribute it,
with or without modifying it, either commercially or noncommercially.
Secondarily, this License preserves for the author and publisher a way
to get credit for their work, while not being considered responsible
for modifications made by others.

This License is a kind of "copyleft", which means that derivative
works of the document must themselves be free in the same sense.  It
complements the GNU General Public License, which is a copyleft
license designed for free software.

We have designed this License in order to use it for manuals for free
software, because free software needs free documentation: a free
program should come with manuals providing the same freedoms that the
software does.  But this License is not limited to software manuals;
it can be used for any textual work, regardless of subject matter or
whether it is published as a printed book.  We recommend this License
principally for works whose purpose is instruction or reference.


1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that
contains a notice placed by the copyright holder saying it can be
distributed under the terms of this License.  Such a notice grants a
world-wide, royalty-free license, unlimited in duration, to use that
work under the conditions stated herein.  The "Document", below,
refers to any such manual or work.  Any member of the public is a
licensee, and is addressed as "you".  You accept the license if you
```

(continues on next page)

(continued from previous page)

```
copy, modify or distribute the work in a way requiring permission
under copyright law.

A "Modified Version" of the Document means any work containing the
Document or a portion of it, either copied verbatim, or with
modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of
the Document that deals exclusively with the relationship of the
publishers or authors of the Document to the Document's overall
subject (or to related matters) and contains nothing that could fall
directly within that overall subject.  (Thus, if the Document is in
part a textbook of mathematics, a Secondary Section may not explain
any mathematics.)  The relationship could be a matter of historical
connection with the subject or with related matters, or of legal,
commercial, philosophical, ethical or political position regarding
them.

The "Invariant Sections" are certain Secondary Sections whose titles
are designated, as being those of Invariant Sections, in the notice
that says that the Document is released under this License.  If a
section does not fit the above definition of Secondary then it is not
allowed to be designated as Invariant.  The Document may contain zero
Invariant Sections.  If the Document does not identify any Invariant
Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed,
as Front-Cover Texts or Back-Cover Texts, in the notice that says that
the Document is released under this License.  A Front-Cover Text may
be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy,
represented in a format whose specification is available to the
general public, that is suitable for revising the document
straightforwardly with generic text editors or (for images composed of
pixels) generic paint programs or (for drawings) some widely available
drawing editor, and that is suitable for input to text formatters or
for automatic translation to a variety of formats suitable for input
to text formatters.  A copy made in an otherwise Transparent file
format whose markup, or absence of markup, has been arranged to thwart
or discourage subsequent modification by readers is not Transparent.
An image format is not Transparent if used for any substantial amount
of text.  A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain
ASCII without markup, Texinfo input format, LaTeX input format, SGML
or XML using a publicly available DTD, and standard-conforming simple
HTML, PostScript or PDF designed for human modification.  Examples of
transparent image formats include PNG, XCF and JPG.  Opaque formats
include proprietary formats that can be read and edited only by
proprietary word processors, SGML or XML for which the DTD and/or
processing tools are not generally available, and the
```

(continues on next page)

(continued from previous page)

```
machine-generated HTML, PostScript or PDF produced by some word
processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself,
plus such following pages as are needed to hold, legibly, the material
this License requires to appear in the title page.  For works in
formats which do not have any title page as such, "Title Page" means
the text near the most prominent appearance of the work's title,
preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of
the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose
title either is precisely XYZ or contains XYZ in parentheses following
text that translates XYZ in another language.  (Here XYZ stands for a
specific section name mentioned below, such as "Acknowledgements",
"Dedications", "Endorsements", or "History".)  To "Preserve the Title"
of such a section when you modify the Document means that it remains a
section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which
states that this License applies to the Document.  These Warranty
Disclaimers are considered to be included by reference in this
License, but only as regards disclaiming warranties: any other
implication that these Warranty Disclaimers may have is void and has
no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either
commercially or noncommercially, provided that this License, the
copyright notices, and the license notice saying this License applies
to the Document are reproduced in all copies, and that you add no
other conditions whatsoever to those of this License.  You may not use
technical measures to obstruct or control the reading or further
copying of the copies you make or distribute.  However, you may accept
compensation in exchange for copies.  If you distribute a large enough
number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and
you may publicly display copies.


3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have
printed covers) of the Document, numbering more than 100, and the
Document's license notice requires Cover Texts, you must enclose the
copies in covers that carry, clearly and legibly, all these Cover
Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on
the back cover.  Both covers must also clearly and legibly identify
```

(continues on next page)

(continued from previous page)

```
you as the publisher of these copies.  The front cover must present
the full title with all words of the title equally prominent and
visible.  You may add other material on the covers in addition.
Copying with changes limited to the covers, as long as they preserve
the title of the Document and satisfy these conditions, can be treated
as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit
legibly, you should put the first ones listed (as many as fit
reasonably) on the actual cover, and continue the rest onto adjacent
pages.

If you publish or distribute Opaque copies of the Document numbering
more than 100, you must either include a machine-readable Transparent
copy along with each Opaque copy, or state in or with each Opaque copy
a computer-network location from which the general network-using
public has access to download using public-standard network protocols
a complete Transparent copy of the Document, free of added material.
If you use the latter option, you must take reasonably prudent steps,
when you begin distribution of Opaque copies in quantity, to ensure
that this Transparent copy will remain thus accessible at the stated
location until at least one year after the last time you distribute an
Opaque copy (directly or through your agents or retailers) of that
edition to the public.

It is requested, but not required, that you contact the authors of the
Document well before redistributing any large number of copies, to
give them a chance to provide you with an updated version of the
Document.


4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under
the conditions of sections 2 and 3 above, provided that you release
the Modified Version under precisely this License, with the Modified
Version filling the role of the Document, thus licensing distribution
and modification of the Modified Version to whoever possesses a copy
of it.  In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct
   from that of the Document, and from those of previous versions
   (which should, if there were any, be listed in the History section
   of the Document).  You may use the same title as a previous version
   if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities
   responsible for authorship of the modifications in the Modified
   Version, together with at least five of the principal authors of the
   Document (all of its principal authors, if it has fewer than five),
   unless they release you from this requirement.
C. State on the Title page the name of the publisher of the
   Modified Version, as the publisher.
```

(continues on next page)

(continued from previous page)

```
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications
   adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice
   giving the public permission to use the Modified Version under the
   terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections
   and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add
   to it an item stating at least the title, year, new authors, and
   publisher of the Modified Version as given on the Title Page.  If
   there is no section Entitled "History" in the Document, create one
   stating the title, year, authors, and publisher of the Document as
   given on its Title Page, then add an item describing the Modified
   Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for
   public access to a Transparent copy of the Document, and likewise
   the network locations given in the Document for previous versions
   it was based on.  These may be placed in the "History" section.
   You may omit a network location for a work that was published at
   least four years before the Document itself, or if the original
   publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications",
   Preserve the Title of the section, and preserve in the section all
   the substance and tone of each of the contributor acknowledgements
   and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document,
   unaltered in their text and in their titles.  Section numbers
   or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements".  Such a section
   may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements"
   or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or
appendices that qualify as Secondary Sections and contain no material
copied from the Document, you may at your option designate some or all
of these sections as invariant.  To do this, add their titles to the
list of Invariant Sections in the Modified Version's license notice.
These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains
nothing but endorsements of your Modified Version by various
parties--for example, statements of peer review or that the text has
been approved by an organization as the authoritative definition of a
standard.

You may add a passage of up to five words as a Front-Cover Text, and a
passage of up to 25 words as a Back-Cover Text, to the end of the list
of Cover Texts in the Modified Version.  Only one passage of
```

(continues on next page)

(continued from previous page)

```
Front-Cover Text and one of Back-Cover Text may be added by (or
through arrangements made by) any one entity.  If the Document already
includes a cover text for the same cover, previously added by you or
by arrangement made by the same entity you are acting on behalf of,
you may not add another; but you may replace the old one, on explicit
permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License
give permission to use their names for publicity for or to assert or
imply endorsement of any Modified Version.


5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this
License, under the terms defined in section 4 above for modified
versions, provided that you include in the combination all of the
Invariant Sections of all of the original documents, unmodified, and
list them all as Invariant Sections of your combined work in its
license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and
multiple identical Invariant Sections may be replaced with a single
copy.  If there are multiple Invariant Sections with the same name but
different contents, make the title of each such section unique by
adding at the end of it, in parentheses, the name of the original
author or publisher of that section if known, or else a unique number.
Make the same adjustment to the section titles in the list of
Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History"
in the various original documents, forming one section Entitled
"History"; likewise combine any sections Entitled "Acknowledgements",
and any sections Entitled "Dedications".  You must delete all sections
Entitled "Endorsements".


6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other
documents released under this License, and replace the individual
copies of this License in the various documents with a single copy
that is included in the collection, provided that you follow the rules
of this License for verbatim copying of each of the documents in all
other respects.

You may extract a single document from such a collection, and
distribute it individually under this License, provided you insert a
copy of this License into the extracted document, and follow this
License in all other respects regarding verbatim copying of that
document.
```

(continues on next page)

(continued from previous page)

```
7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate
and independent documents or works, in or on a volume of a storage or
distribution medium, is called an "aggregate" if the copyright
resulting from the compilation is not used to limit the legal rights
of the compilation's users beyond what the individual works permit.
When the Document is included in an aggregate, this License does not
apply to the other works in the aggregate which are not themselves
derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these
copies of the Document, then if the Document is less than one half of
the entire aggregate, the Document's Cover Texts may be placed on
covers that bracket the Document within the aggregate, or the
electronic equivalent of covers if the Document is in electronic form.
Otherwise they must appear on printed covers that bracket the whole
aggregate.


8. TRANSLATION

Translation is considered a kind of modification, so you may
distribute translations of the Document under the terms of section 4.
Replacing Invariant Sections with translations requires special
permission from their copyright holders, but you may include
translations of some or all Invariant Sections in addition to the
original versions of these Invariant Sections.  You may include a
translation of this License, and all the license notices in the
Document, and any Warranty Disclaimers, provided that you also include
the original English version of this License and the original versions
of those notices and disclaimers.  In case of a disagreement between
the translation and the original version of this License or a notice
or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements",
"Dedications", or "History", the requirement (section 4) to Preserve
its Title (section 1) will typically require changing the actual
title.


9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense, or distribute it is void, and
will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license
from a particular copyright holder is reinstated (a) provisionally,
unless and until the copyright holder explicitly and finally
```

(continues on next page)

(continued from previous page)

```
terminates your license, and (b) permanently, if the copyright holder
fails to notify you of the violation by some reasonable means prior to
60 days after the cessation.

Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, receipt of a copy of some or all of the same material does
not give you any rights to use it.


10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the
GNU Free Documentation License from time to time.  Such new versions
will be similar in spirit to the present version, but may differ in
detail to address new problems or concerns.  See
http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number.
If the Document specifies that a particular numbered version of this
License "or any later version" applies to it, you have the option of
following the terms and conditions either of that specified version or
of any later version that has been published (not as a draft) by the
Free Software Foundation.  If the Document does not specify a version
number of this License, you may choose any version ever published (not
as a draft) by the Free Software Foundation.  If the Document
specifies that a proxy can decide which future versions of this
License can be used, that proxy's public statement of acceptance of a
version permanently authorizes you to choose that version for the
Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any
World Wide Web server that publishes copyrightable works and also
provides prominent facilities for anybody to edit those works.  A
public wiki that anybody can edit is an example of such a server.  A
"Massive Multiauthor Collaboration" (or "MMC") contained in the site
means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0
license published by Creative Commons Corporation, a not-for-profit
corporation with a principal place of business in San Francisco,
California, as well as future copyleft versions of that license
```

(continues on next page)

(continued from previous page)

```
published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in
part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this
License, and if all works that were first published under this License
somewhere other than this MMC, and subsequently incorporated in whole or
in part into the MMC, (1) had no cover texts or invariant sections, and
(2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site
under CC-BY-SA on the same site at any time before August 1, 2009,
provided the MMC is eligible for relicensing.
```

13.4 spdlog

```
The MIT License (MIT)

Copyright (c) 2016 Gabi Melman.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

13.5 yaml-cpp

```
Copyright (c) 2008 Jesse Beder.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
```

(continues on next page)

(continued from previous page)

```
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

INDEX

L