Class GXFExtensionManager

Table of contents

- Defined in File gxf_extension_manager.hpp

Inheritance Relationships

Base Type

- `public holoscan::ExtensionManager` (Class ExtensionManager)

Class Documentation

class GXFExtensionManager : public holoscan::ExtensionManager

Class to manage GXF extensions.

This class is a helper class to manage GXF extensions.

Since GXF API doesn't provide a way to ignore duplicate extensions, this class is used to manage the extensions, prevent duplicate extensions from being loaded, and unload the extension handlers when the class is destroyed.

Example:

```
#include <yaml-cpp/yaml.h> #include
"holoscan/core/gxf/gxf_extension_manager.hpp" ...
holoscan::gxf::GXFExtensionManager
extension_manager(reinterpret_cast<gxf_context_t>(context)); // Load
extensions from a file for (const auto& extension_filename :
extension_filenames) {
extension_manager.load_extension(extension_filename); } // Load extensions
from a yaml node (YAML::Node object) for (const auto& manifest_filename :
manifest_filenames) { auto node = YAML::LoadFile(manifest_filename);
extension_manager.load_extensions_from_yaml(node); }
```

Public Functions

explicit GXFExtensionManager(gxf_context_t context)

Construct a new GXFExtensionManager object.

Parameters

context – The GXF context

~GXFExtensionManager() override

Destroy the GXFExtensionManager object.

This method closes all the extension handles that are loaded by this class. Note that the shared library is opened with `RTLD_NODELETE` flag, so the library is not unloaded when the handle is closed.

virtual void refresh() override

Refresh the extension list.

Based on the current GXF context, it gets the list of extensions and stores the type IDs of the extensions so that duplicate extensions can be ignored.

virtual bool load_extension(const std::string &file_name, bool no_error_message = false, const std::string &search_path_envs = "HOLOSCAN_LIB_PATH") override

Load an extension.

This method loads an extension and stores the extension handler so that it can be unloaded when the class is destroyed.

Parameters

- **file_name** – The file name of the extension (e.g. libgxf_std.so).
- **no_error_message** – If true, no error message will be printed if the extension is not found.
- **search_path_envs** – The environment variable names that contains the search paths for the extension. The environment variable names are separated by a comma (,). (default: “HOLOSCAN_LIB_PATH”).

Returns

true if the extension is loaded successfully, false otherwise.

virtual bool load_extensions_from_yaml(const YAML::Node &node, bool no_error_message = false, const std::string &search_path_envs = "HOLOSCAN_LIB_PATH", const std::string &key = "extensions") override

Load extensions from a yaml file.

The yaml file should contain a list of extension file names under the key “extensions”.

For example:

```
extensions: - /path/to/extension1.so - /path/to/extension2.so - /path/to/extension3.so
```

Parameters

- **node** – The yaml node.
- **no_error_message** – If true, no error message will be printed if the extension is not found.
- **search_path_envs** – The environment variable names that contains the search paths for the extension. The environment variable names are separated by a comma (,). (default: “HOLOSCAN_LIB_PATH”).
- **key** – The key in the yaml node that contains the extension file names (default: “extensions”).

Returns

true if the extension is loaded successfully, false otherwise.

bool load_extension(nvidia::gxf::Extension *extension, void *handle = nullptr)

Load an extension from a pointer.

`GxfLoadExtensionFromPointer()` API is used to register the extension programmatically.

Parameters

- **extension** – The extension pointer to load.
- **handle** – The handle of the extension library.

Returns

true if the extension is loaded successfully.

bool is_extension_loaded(gxf_tid_t tid)

Check if the extension is loaded.

Parameters

tid – The type ID of the extension.

Returns

true if the extension is loaded. false otherwise.

Public Static Functions

static std::vector<std::string> tokenize(const std::string &str, const std::string &delimiters)

Tokenize a string.

Parameters

- **str** – The string to tokenize.
- **delimiters** – The delimiters.

Returns

The vector of tokens.

Protected Attributes

gxf_tid_t extension_tid_list_[kGXFExtensionsMaxSize] = {}

Storage for the extension TIDs.

gxf_runtime_info runtime_info_ = {nullptr, kGXFExtensionsMaxSize, extension_tid_list_}

request/response structure for the runtime info

std::set<gxf_tid_t> extension_tids_

Set of extension TIDs.

std::set<void*> extension_handles_

Set of extension handles.