SimpleDynamicLinking

0.0.1

Создано системой Doxygen 1.9.7

1 Concept Index 1
1.1 Concepts . . . 1

2 Иерархический список классов 3
2.1 Иерархия классов . . . 3

3 Алфавитный указатель классов 5
3.1 Классы . . . 5

4 Список файлов 7
4.1 Файлы . . . 7

5 Concept Documentation 9
5.1 NotPointerConcept Concept Reference . . . 9
5.1.1 Concept definition . . . 9
5.2 PointerConcept Concept Reference . . . 9
5.2.1 Concept definition . . . 9

6 Классы 11
6.1 Класс DynamicLibController . . . 11
6.1.1 Подробное описание . . . 12
6.2 Структура DynamicLibController::DynamicLibControllerImpl . . . 12
6.3 Шаблон класса ImportedFunction< ResultType, ArgsType > . . . 12
6.3.1 Подробное описание . . . 13
6.4 Шаблон класса ImportedObject< ImportedObjectType > . . . 13
6.4.1 Подробное описание . . . 13
6.5 Структура ImportError . . . 14
6.6 Шаблон класса PackedPointer< T > . . . 14

7 Файлы 15
7.1 dynamicLib.h . . . 15

Предметный указатель 19

Глава 1

Concept Index

1.1 Concepts

Here is a list of all documented concepts with brief descriptions:

NotPointerConcept 9
PointerConcept 9

Глава 2

Иерархический список классов

2.1 Иерархия классов

Иерархия классов.

DynamicLibController::DynamicLibControllerImpl 12
std::enable_shared_from_this
    DynamicLibController 11
ImportedFunction< ResultType, ArgsType > 12
ImportedObject< ImportedObjectType > 13
ImportError 14
PackedPointer< T > 14

Глава 3

Алфавитный указатель классов

3.1 Классы

Классы с их кратким описанием.

DynamicLibController
: Класс контроллера динамической библиотеки 11

DynamicLibController::DynamicLibControllerImpl 12

ImportedFunction< ResultType, ArgsType >
: Класс импортированной функции 12

ImportedObject< ImportedObjectType >
: Класс импортированного объекта 13

ImportError 14

PackedPointer< T > 14

Глава 4

Список файлов

4.1 Файлы

Полный список документированных файлов.

/home/houdini/Cpp/plugins/dynamicLib/simpleDynamicLinking/src/dynamicLib.h 15

Глава 5

Concept Documentation

5.1 NotPointerConcept Concept Reference

5.1.1 Concept definition

```
template<class T>
concept NotPointerConcept = !std::is_pointer_v<T>
```

5.2 PointerConcept Concept Reference

5.2.1 Concept definition

```
template<class T>
concept PointerConcept = std::is_pointer_v<T>
```

Глава 6

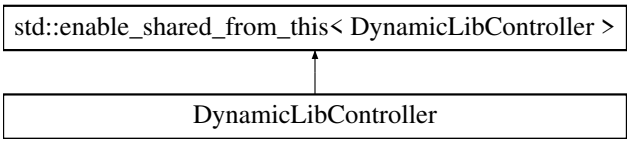
Классы

6.1 Класс DynamicLibController

Класс контроллера динамической библиотеки

#include <dynamicLib.h>

Граф наследования:DynamicLibController:



Классы

- struct DynamicLibControllerImpl

Открытые члены

- DynamicLibController (const DynamicLibController &another)=delete
- DynamicLibController (DynamicLibController &&another)=delete
- DynamicLibController & operator= (const DynamicLibController &another)=delete
- DynamicLibController & operator= (DynamicLibController &&another)=delete
- bool isActive () const
- template<class ResultType , class... ArgsTypes>
  ImportedFunction< ResultType, ArgsTypes... > GetFuncFromName (const std::string &funcName)
- template<NotPointerConcept ResultType>
  std::shared_ptr< ImportedObject< ResultType > > CallFuncFromNameSafe (const std::string &funcName)

Открытые статические члены

- static std::shared_ptr< DynamicLibController > CreateController (const std::string &path)

6.1.1 Подробное описание

Класс контроллера динамической библиотеки

Класс контроллера динамической библиотеки. Предоставляет API для безопасного импорта функции, обернутой с помощью макроса GENERATE_SAFE_EXTERN_VOID, а также ручной импорт функции, соблюдающей C-шные требования. Экземпляры данного класса порождаются с помощью фабрики, а также данный класс имеет примесь enable_shared_from_this.

Объявления и описания членов классов находятся в файлах:

- /home/houdini/Cpp/plugins/dynamicLib/simpleDynamicLinking/src/dynamicLib.h
- /home/houdini/Cpp/plugins/dynamicLib/simpleDynamicLinking/src/dynamicLib.cpp

6.2 Структура DynamicLibController::DynamicLibControllerImpl

Открытые члены

- void CreateImpl (const std::string &path)
- void DestroyImpl ()
- bool isActive () const

Открытые атрибуты

- void ∗ sharedLib
- std::string path

Объявления и описания членов структуры находятся в файле:

- /home/houdini/Cpp/plugins/dynamicLib/simpleDynamicLinking/src/dynamicLib.cpp

6.3 Шаблон класса ImportedFunction< ResultType, ArgsType >

Класс импортированной функции

#include <dynamicLib.h>

Открытые члены

- ImportedFunction (ResultType(∗importedFunc)(ArgsType...), SharedLibController imported=nullptr)
- ResultType operator() (ArgsType... args)
- auto GetLib ()

6.3.1 Подробное описание

template<class ResultType, class... ArgsType>
class ImportedFunction< ResultType, ArgsType >

Класс импортированной функции

Класс импортированной функции. Содержит указатель на вызываемую функцию, а также экземпляр shared_ptr контроллера динамической библиотеки, ассоциированной с этой функции

Объявления и описания членов класса находятся в файле:

- /home/houdini/Cpp/plugins/dynamicLib/simpleDynamicLinking/src/dynamicLib.h

6.4 Шаблон класса ImportedObject< ImportedObjectType >

Класс импортированного объекта

#include <dynamicLib.h>

Открытые члены

- ImportedObject (const ImportedObjectType &another)=delete
- ImportedObject (ImportedObjectType &&another)=delete
- ImportedObjectType & operator= (const ImportedObjectType &another)=delete
- ImportedObjectType & operator= (ImportedObjectType &&another)=delete
- auto GetLib ()
- auto GetImported ()

Открытые статические члены

- static std::shared_ptr< ImportedObject > CreateImportedObject (ImportedObjectType *importedObject, SharedLibController lib=nullptr)

6.4.1 Подробное описание

template<NotPointerConcept ImportedObjectType>
class ImportedObject< ImportedObjectType >

Класс импортированного объекта

Класс импортированного объекта. Содержит указатель на импортированный объект, а также экземпляр shared_ptr контроллера динамической библиотеки, ассоциированной с этой функции. Невозможно создать экземпляр данного класса, зато можно создать shared_ptr с помощью фабрики.

Объявления и описания членов класса находятся в файле:

- /home/houdini/Cpp/plugins/dynamicLib/simpleDynamicLinking/src/dynamicLib.h

6.5 Структура ImportError

Открытые атрибуты

- std::string errorStr

Объявления и описания членов структуры находятся в файле:

- /home/houdini/Cpp/plugins/dynamicLib/simpleDynamicLinking/src/dynamicLib.h

6.6 Шаблон класса PackedPointer< T >

Открытые члены

- PackedPointer (T val)
- PackedPointer (PackedPointer &&another)
- T Unpack ()

Объявления и описания членов класса находятся в файле:

- /home/houdini/Cpp/plugins/dynamicLib/simpleDynamicLinking/src/dynamicLib.h

Глава 7

Файлы

7.1 dynamicLib.h

```
00001 #include <iostream>
00002 #include <any>
00003 #include <memory>
00004 #include <type_traits>
00005 #include <functional>
00006 #include <string>
00007 #include <optional>
00008
00009 using namespace std::string_literals;
00010
00011
00012 template <class T>
00013 concept NotPointerConcept = !std::is_pointer_v<T>;
00014
00015
00016 template <class T>
00017 concept PointerConcept = std::is_pointer_v<T>;
00018
00019 class DynamicLibController;
00020 using SharedLibController = std::shared_ptr<DynamicLibController>;
00021
00022
00023 struct ImportError{
00024     std::string errorStr;
00025 };
00026
00033 template <class ResultType, class... ArgsType>
00034 class ImportedFunction{
00035 private:
00036     ResultType (*importedFunc)(ArgsType...);
00037     SharedLibController importedLib;
00038 public:
00039     explicit ImportedFunction(ResultType (*importedFunc)(ArgsType...), SharedLibController imported = nullptr)
00040     : importedFunc(importedFunc), importedLib(std::move(imported)) {}
00041     ResultType operator()(ArgsType... args){
00042         return this->importedFunc(args...);
00043     }
00044     auto GetLib(){
00045         return this->importedLib;
00046     }
00047 };
00055 template <NotPointerConcept ImportedObjectType>
00056 class ImportedObject{
00057 private:
00058     std::shared_ptr<ImportedObjectType> importedObject;
00059     SharedLibController libController;
00060     explicit ImportedObject(ImportedObjectType* importedObject, SharedLibController lib = nullptr) :
      libController(std::move(lib)){
00061         this->importedObject = std::shared_ptr<ImportedObjectType>(importedObject);
00062     }
00063 public:
00064     ~ImportedObject() = default;
00065     ImportedObject(const ImportedObjectType& another) = delete;
00066     ImportedObject(ImportedObjectType&& another) = delete;
00067     ImportedObjectType& operator=(const ImportedObjectType& another) = delete;
00068     ImportedObjectType& operator=(ImportedObjectType&& another) = delete;
00069     [[nodiscard]] static std::shared_ptr<ImportedObject> CreateImportedObject(ImportedObjectType* importedObject,
      SharedLibController lib = nullptr){
```

```
00070         return std::shared_ptr<ImportedObject>(new ImportedObject(importedObject, lib));
00071     }
00072     auto GetLib() {
00073         return this->libController;
00074     }
00075 
00076     //Не сохранять возвращаемый объект в полях!
00077     auto GetImported(){
00078         return this->importedObject;
00079     }
00080 };
00081 
00082 
00083 
00084 
00085 
00086 template <PointerConcept T>
00087 class PackedPointer
00088 {
00089 private:
00090     using DelType = std::optional<void (*)(T val)>;
00091     T val;
00092     DelType deleter;
00093 
00094 public:
00095     explicit PackedPointer(T val) : val(val) {
00096 
00097         this->deleter = [](T deletableObject) mutable{
00098             std::cout « "Deleter called" « std::endl;
00099             delete deletableObject;
00100         };
00101     }
00102     PackedPointer(PackedPointer&& another) {
00103         this->val = another.val;
00104         this->deleter = another.deleter;
00105         another.deleter.reset();
00106     }
00107     ~PackedPointer()
00108     {
00109         std::cout « "Destructor called" « std::endl;
00110         if (deleter.has_value())
00111         {
00112             deleter.value()(val);
00113         }
00114     }
00115     [[nodiscard]] T Unpack()
00116     {
00117         deleter.reset();
00118         return val;
00119     }
00120 };
00121 
00122 template<class Func>
00123 constexpr bool IsItNonArgsFunc(Func functionName){
00124     constexpr bool isVoidArgs = requires{functionName();};
00125     return isVoidArgs;
00126 }
00127 
00153 #define GENERATE_SAFE_EXTERN_VOID(function_name)                                    \
00154     extern "C"                                                                      \
00155     {                                                                               \
00156         void *function_name##_ExternC_ViaDynamicLib()                               \
00157         {                                                                           \
00158             void* result;                                                           \
00159             constexpr bool isVoidArgs = IsItNonArgsFunc(&function_name);            \
00160             static_assert(isVoidArgs, "This is not void args in function");         \
00161             static_assert(!std::is_same_v<std::decay_t<decltype(function_name())>, void*>, "Return type can‘t be
      void*");  \
00162             static_assert(std::is_pointer_v<decltype(function_name())>, "This function must return a pointer");     \
00163             try{                                                                    \
00164                 auto pFromFunc = function_name();                                   \
00165                 if(pFromFunc != nullptr){                                           \
00166                     using ResultType = decltype(PackedPointer(pFromFunc));          \
00167                     std::shared_ptr<ResultType> shared_packed_ptr =
      std::make_shared<ResultType>(PackedPointer(pFromFunc));     \
00168                     std::any *preRes = new std::any(shared_packed_ptr); \
00169                     result = reinterpret_cast<void *>(preRes);              \
00170                 } else{                                                             \
00171                     ImportError error; error.errorStr = "returned nullptr pointer"s; \
00172                     std::any* preRes = new std::any(error);\
00173                     result = reinterpret_cast<void*>(preRes);               \
00174                 }                                                                   \
00175             } catch(std::exception& ex){                                            \
00176                     ImportError error; error.errorStr = "Catched exception:\n"s + ex.what();          \
00177                     std::any* preRes = new std::any(error);\
00178                     result = reinterpret_cast<void*>(preRes);               \
00179                 } catch(...){                                                       \
```

```
00180                   ImportError error; error.errorStr = "Catched exception:\n"s;           \
00181                   std::any* preRes = new std::any(error);\
00182                   result = reinterpret_cast<void*>(preRes); \
00183               }                                                                          \
00184                                                                                              \
00185                                                                                          \
00186           return result;                                                                \
00187       }                                                                                 \
00188   }\
00189
00198 class DynamicLibController : public std::enable_shared_from_this<DynamicLibController>
00199 {
00200 private:
00201     struct DynamicLibControllerImpl;
00202     std::unique_ptr<DynamicLibControllerImpl> impl;
00203     void *GetRawFuncCaller(const std::string &funcName);
00204     explicit DynamicLibController(const std::string &path);
00205     std::shared_ptr<DynamicLibController> get_ptr()
00206     {
00207         return this->shared_from_this();
00208     }
00209
00210     [[nodiscard]] auto GetFuncRawPointerFromNameInternal(const std::string& funcName){
00211         if (!this->isActive())
00212         {
00213             throw std::runtime_error("Controller is not active");
00214         }
00215         void *rawFuncPointer = this->GetRawFuncCaller(funcName);
00216         if (rawFuncPointer == nullptr)
00217         {
00218             throw std::runtime_error("Function with name "s + funcName + " does not imported");
00219         }
00220         return rawFuncPointer;
00221     }
00222
00223
00224     template <NotPointerConcept ResultType>
00225     [[nodiscard]] auto CallFuncFromNameSafeInternal(const std::string &funcName)
00226     {
00227         ResultType* result;
00228         std::string fullFuncName = funcName + "_ExternC_ViaDynamicLib";
00229         if (!this->isActive())
00230         {
00231             throw std::runtime_error("Controller is not active");
00232         }
00233         std::any *preResult;
00234         void *rawFuncPointer = this->GetRawFuncCaller(fullFuncName);
00235         if (rawFuncPointer == nullptr)
00236         {
00237             throw std::runtime_error("Function with name "s + funcName + " does not imported");
00238         }
00239         auto creater = (void *(*)())(rawFuncPointer);
00240         void *rawImported = creater();
00241         if (rawImported == nullptr)
00242         {
00243             throw std::runtime_error("Function with name "s + funcName + " return nullptr pointer");
00244         }
00245         preResult = reinterpret_cast<std::any *>(rawImported);
00246         if (!preResult->has_value())
00247         {
00248             throw std::runtime_error("Something is wrong with imported pointer");
00249         }
00250         try
00251         {
00252             auto packedPointer = std::any_cast<std::shared_ptr<PackedPointer<ResultType *»>(*preResult);
00253             result = packedPointer->Unpack();
00254         }
00255         catch (std::bad_any_cast &_)
00256         {
00257             try{
00258                 auto err = std::any_cast<ImportError>(*preResult);
00259                 delete preResult;
00260                 throw std::runtime_error(err.errorStr);
00261             } catch(std::bad_any_cast& _){
00262                 delete preResult;
00263                 throw std::runtime_error("Uncorrect result type");
00264             }
00265
00266         }
00267         delete preResult;
00268         return result;
00269     }
00270
00271 public:
00272     ~DynamicLibController();
00273     DynamicLibController(const DynamicLibController &another) = delete;
00274     DynamicLibController(DynamicLibController &&another) = delete;
```

```
00275     DynamicLibController &operator=(const DynamicLibController &another) = delete;
00276     DynamicLibController &operator=(DynamicLibController &&another) = delete;
00277     bool isActive() const;
00278 
00279     [[nodiscard]] static std::shared_ptr<DynamicLibController> CreateController(const std::string &path)
00280     {
00281         std::shared_ptr<DynamicLibController> result(new DynamicLibController(path));
00282         if (!result->isActive())
00283         {
00284             throw std::runtime_error("Uncorrect path to shared lib. Controller don‘t create correctly");
00285         }
00286         return result;
00287     }
00288 
00289     template <class ResultType, class... ArgsTypes>
00290     [[nodiscard]] ImportedFunction<ResultType, ArgsTypes...> GetFuncFromName(const std::string& funcName){
00291         try{
00292             void* rawFuncPointer = this->GetFuncRawPointerFromNameInternal(funcName);
00293             ResultType (*castedFuncPointer)(ArgsTypes...) = (ResultType(*)(ArgsTypes...))(rawFuncPointer);
00294             return ImportedFunction<ResultType, ArgsTypes...>(castedFuncPointer, this->get_ptr());
00295         } catch(std::runtime_error& err){
00296             throw err;
00297         }
00298     }
00299 
00300 
00301     template <NotPointerConcept ResultType>
00302     [[nodiscard]] std::shared_ptr<ImportedObject<ResultType» CallFuncFromNameSafe(const std::string &funcName)
00303     {
00304         auto self = this->get_ptr();
00305         try{
00306             ResultType* rawRes = this->CallFuncFromNameSafeInternal<ResultType>(funcName);
00307             return ImportedObject<ResultType>::CreateImportedObject(rawRes, self);
00308         } catch(std::runtime_error& e){
00309             throw e;
00310         }
00311     }
00312 };
00313 
00314 
```

Предметный указатель

/home/houdini/Cpp/plugins/dynamicLib/simpleDynamicLinking/src/dynamicLib.h, 15

DynamicLibController, 11
DynamicLibController::DynamicLibControllerImpl, 12

ImportedFunction< ResultType, ArgsType >, 12
ImportedObject< ImportedObjectType >, 13
ImportError, 14

NotPointerConcept, 9

PackedPointer< T >, 14
PointerConcept, 9