cpptoolkit

Generated by Doxygen 1.9.1

Chapter 1

C++ Toolkit Documentation

1.1 Description

This toolkit provides a collection of utilities and tools to enhance C++ development. It includes various modules for file handling, string manipulation, and more.

1.2 Build Process

To build the toolkit, follow these steps:

1. Ensure you have CMake installed on your system.
2. Clone the repository to your local machine.
3. Navigate to the root directory of the repository.
4. Create a build directory: `mkdir build && cd build`
5. Run CMake to configure the project: `cmake ..`
6. Build the project using Make: `make`
7. The compiled binaries will be located in the `build` directory.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

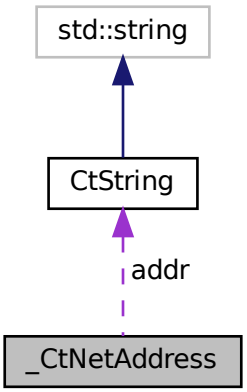
Class Documentation

5.1 _CtNetAddress Struct Reference

Struct describing a network address.

`#include <CtTypes.hpp>`

Collaboration diagram for _CtNetAddress:



Public Attributes

- CtString addr
- CtUInt16 port

5.1.1 Detailed Description

Struct describing a network address.

The network address is described by the IP address and the port number.

Definition at line 120 of file CtTypes.hpp.

5.1.2 Member Data Documentation

5.1.2.1 addr

```
CtString _CtNetAddress::addr
```

Definition at line 121 of file CtTypes.hpp.

5.1.2.2 port

```
CtUInt16 _CtNetAddress::port
```

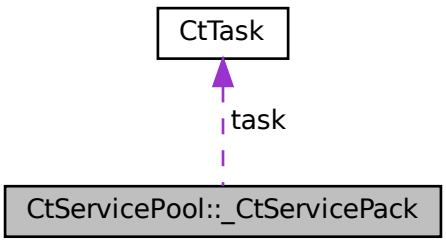
Definition at line 122 of file CtTypes.hpp.

The documentation for this struct was generated from the following file:

- include/CtTypes.hpp

5.2 CtServicePool::_CtServicePack Struct Reference

Collaboration diagram for CtServicePool::_CtServicePack:



Public Attributes

- CtTask task
- std::string id
- CtUInt32 nslots

5.2.1 Detailed Description

Definition at line 76 of file CtServicePool.hpp.

5.2.2 Member Data Documentation

5.2.2.1 id

```
std::string CtServicePool::_CtServicePack::id
```

Definition at line 78 of file CtServicePool.hpp.

5.2.2.2 nslots

```
CtUInt32 CtServicePool::_CtServicePack::nslots
```

Definition at line 79 of file CtServicePool.hpp.

5.2.2.3 task

```
CtTask CtServicePool::_CtServicePack::task
```

Definition at line 77 of file CtServicePool.hpp.

The documentation for this struct was generated from the following file:

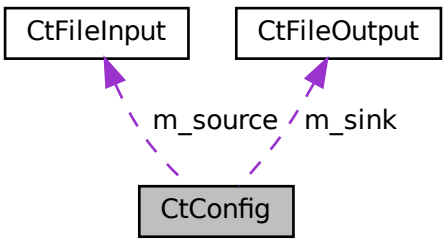
- include/threading/CtServicePool.hpp

5.3 CtConfig Class Reference

A configuration file parser class for extracting various data types from configuration values.

```
#include <CtConfig.hpp>
```

Collaboration diagram for CtConfig:



Public Member Functions

- EXPORTED_API CtConfig (const std::string &p_configFile)

  Constructor for CtConfig.
- EXPORTED_API ~CtConfig ()

  Destructor for cleaning up resources.
- EXPORTED_API void read ()

  Read data from config file. This method can throw CtFileParseError if file cannot be parsed. This method can throw CtFileError if there is a problem with the file.
- EXPORTED_API void write ()

  Write data to config file.
- EXPORTED_API int32_t parseAsInt (const std::string &p_key)

  Parse a value as a 32-bit signed integer or throw CtKeyNotFoundError if key is not found in the map or throw Ct↩ ParseError if key value cannot be parsed as int.
- EXPORTED_API uint32_t parseAsUInt (const std::string &p_key)

  Parse a value as a 32-bit unsigned integer or throw CtKeyNotFoundError if key is not found in the map or throw CtParseError if key value cannot be parsed as uint.
- EXPORTED_API float parseAsFloat (const std::string &p_key)

  Parse a value as a float or throw CtKeyNotFoundError if key is not found in the map or throw CtParseError if key value cannot be parsed as float.
- EXPORTED_API double parseAsDouble (const std::string &p_key)

  Parse a value as a double-precision floating-point number or throw CtKeyNotFoundError if key is not found in the map or throw CtParseError if key value cannot be parsed as double.
- EXPORTED_API std::string parseAsString (const std::string &p_key)

  Parse a value as a standard C++ string or throw CtKeyNotFoundError if key is not found in the map.
- EXPORTED_API void writeInt (const std::string &p_key, const int32_t &p_value)

  Write value to key as int.
- EXPORTED_API void writeUInt (const std::string &p_key, const uint32_t &p_value)

  Write value to key as uint.
- EXPORTED_API void writeFloat (const std::string &p_key, const float &p_value)

  Write value to key as float.
- EXPORTED_API void writeDouble (const std::string &p_key, const double &p_value)

  Write value to key as double.
- EXPORTED_API void writeString (const std::string &p_key, const std::string &p_value)

  Write value to key as string.

Private Member Functions

- std::string getValue (const std::string &p_key)

  This method returns the value assosiated with the given key or throw CtKeyNotFoundError if key is not found in the map.
- void parseLine (const std::string &p_line)

  This method gets a line as input and parse it in order to find the key and value of configured item. These values are stored in the std::map m_configValues. This method can throw CtFileParseError if file cannot be parsed.

Private Attributes

- std::mutex m_mtx_control
- CtFileInput * m_source
- CtFileOutput * m_sink
- std::string m_configFile
- std::map< std::string, std::string > m_configValues

5.3.1 Detailed Description

A configuration file parser class for extracting various data types from configuration values.

The CtConfig class provides a mechanism for reading and writing configuration files providing key-value pairs of various data types. The class can parse integer, unsigned integer, float, double, and string values. The class is thread-safe and can be used in multi-threaded environments.

Definition at line 56 of file CtConfig.hpp.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 CtConfig()

```
CtConfig::CtConfig (
            const std::string & p_configFile )  [explicit]
```

Constructor for CtConfig.

Parameters

configFile	The path to the configuration file to be parsed.

Definition at line 45 of file CtConfig.cpp.

5.3.2.2 ~CtConfig()

```
CtConfig::~CtConfig ( )
```

Destructor for cleaning up resources.

Definition at line 50 of file CtConfig.cpp.

5.3.3 Member Function Documentation

5.3.3.1 getValue()

```
std::string CtConfig::getValue (
            const std::string & p_key )  [private]
```

This method returns the value assosiated with the given key or throw CtKeyNotFoundError if key is not found in the map.

Parameters

key	The key value to be parsed.

Returns

std::string The string value assosiated with the given key.

Definition at line 171 of file CtConfig.cpp.

5.3.3.2 parseAsDouble()

```
double CtConfig::parseAsDouble (
            const std::string & p_key )
```

Parse a value as a double-precision floating-point number or throw CtKeyNotFoundError if key is not found in the map or throw CtParseError if key value cannot be parsed as double.

Parameters

key	The key value to be parsed.

Returns

The parsed double value.

Definition at line 156 of file CtConfig.cpp.

5.3.3.3 parseAsFloat()

```
float CtConfig::parseAsFloat (
            const std::string & p_key )
```

Parse a value as a float or throw CtKeyNotFoundError if key is not found in the map or throw CtParseError if key value cannot be parsed as float.

Parameters

key	The key value to be parsed.

Returns

The parsed floating-point value.

Definition at line 145 of file CtConfig.cpp.

5.3.3.4 parseAsInt()

```
int32_t CtConfig::parseAsInt (
            const std::string & p_key )
```

Parse a value as a 32-bit signed integer or throw CtKeyNotFoundError if key is not found in the map or throw CtParseError if key value cannot be parsed as int.

Parameters

key	The key value to be parsed.

Returns

The parsed integer value.

Definition at line 123 of file CtConfig.cpp.

5.3.3.5 parseAsString()

```
std::string CtConfig::parseAsString (
            const std::string & p_key )
```

Parse a value as a standard C++ string or throw CtKeyNotFoundError if key is not found in the map.

Parameters

key	The key value to be parsed.

Returns

The parsed string.

Definition at line 167 of file CtConfig.cpp.

5.3.3.6 parseAsUInt()

```
uint32_t CtConfig::parseAsUInt (
            const std::string & p_key )
```

Parse a value as a 32-bit unsigned integer or throw CtKeyNotFoundError if key is not found in the map or throw CtParseError if key value cannot be parsed as uint.

Parameters

key	The key value to be parsed.

Returns

The parsed unsigned integer value.

Definition at line 134 of file CtConfig.cpp.

5.3.3.7 parseLine()

```
void CtConfig::parseLine (
            const std::string & p_line )  [private]
```

This method gets a line as input and parse it in order to find the key and value of configured item. These values are stored in the std::map m_configValues. This method can throw CtFileParseError if file cannot be parsed.

Parameters

line	

Definition at line 93 of file CtConfig.cpp.

5.3.3.8 read()

```
void CtConfig::read ( )
```

Read data from config file. This method can throw CtFileParseError if file cannot be parsed. This method can throw CtFileError if there is a problem with the file.

Definition at line 59 of file CtConfig.cpp.

5.3.3.9 write()

```
void CtConfig::write ( )
```

Write data to config file.

Definition at line 75 of file CtConfig.cpp.

5.3.3.10 writeDouble()

```
void CtConfig::writeDouble (
            const std::string & p_key,
            const double & p_value )
```

Write value to key as double.

Parameters

p_key	The key value.
p_value	The value to be written for this key.

Definition at line 191 of file CtConfig.cpp.

5.3.3.11 writeFloat()

```
void CtConfig::writeFloat (
            const std::string & p_key,
            const float & p_value )
```

Write value to key as float.

Parameters

p_key	The key value.
p_value	The value to be written for this key.

Definition at line 187 of file CtConfig.cpp.

5.3.3.12 writeInt()

```
void CtConfig::writeInt (
            const std::string & p_key,
            const int32_t & p_value )
```

Write value to key as int.

Parameters

p_key	The key value.
p_value	The value to be written for this key.

Definition at line 179 of file CtConfig.cpp.

5.3.3.13 writeString()

```
void CtConfig::writeString (
            const std::string & p_key,
            const std::string & p_value )
```

Write value to key as string.

Parameters

p_key	The key value.
p_value	The value to be written for this key.

Definition at line 195 of file CtConfig.cpp.

5.3.3.14 writeUInt()

```
void CtConfig::writeUInt (
            const std::string & p_key,
            const uint32_t & p_value )
```

Write value to key as uint.

Parameters

p_key	The key value.
p_value	The value to be written for this key.

Definition at line 183 of file CtConfig.cpp.

5.3.4 Member Data Documentation

5.3.4.1 m_configFile

```
std::string CtConfig::m_configFile  [private]
```

The path to the configuration file.

Definition at line 193 of file CtConfig.hpp.

5.3.4.2 m_configValues

```
std::map<std::string, std::string> CtConfig::m_configValues  [private]
```

A map to store configuration key-value pairs.

Definition at line 194 of file CtConfig.hpp.

5.3.4.3 m_mtx_control

```
std::mutex CtConfig::m_mtx_control  [private]
```

Internal mutex for synchronization.

Definition at line 190 of file CtConfig.hpp.

5.3.4.4 m_sink

```
CtFileOutput* CtConfig::m_sink  [private]
```

The sink file for writing configuration values.

Definition at line 192 of file CtConfig.hpp.

5.3.4.5 m_source

```
CtFileInput* CtConfig::m_source  [private]
```

The source file for reading configuration values.

Definition at line 191 of file CtConfig.hpp.

The documentation for this class was generated from the following files:

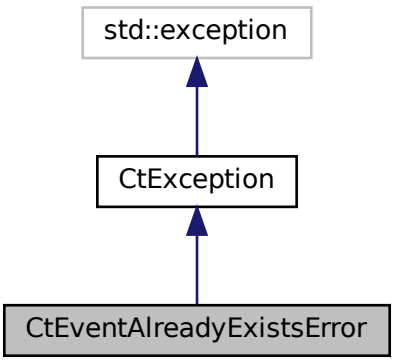
- include/utils/CtConfig.hpp
- src/utils/CtConfig.cpp

5.4 CtEventAlreadyExistsError Class Reference

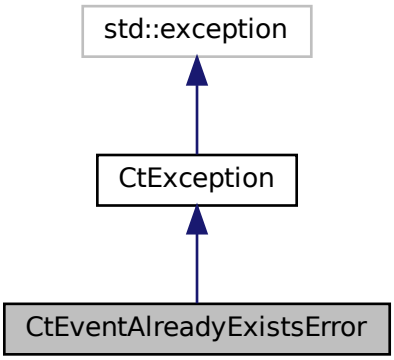
This exception is thrown when an event already exists in the event manager.

```
#include <CtEventExceptions.hpp>
```

Inheritance diagram for CtEventAlreadyExistsError:



Collaboration diagram for CtEventAlreadyExistsError:



Public Member Functions

- CtEventAlreadyExistsError (const std::string &msg)

Additional Inherited Members

5.4.1 Detailed Description

This exception is thrown when an event already exists in the event manager.

Definition at line 50 of file CtEventExceptions.hpp.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 CtEventAlreadyExistsError()

```
CtEventAlreadyExistsError::CtEventAlreadyExistsError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 52 of file CtEventExceptions.hpp.

The documentation for this class was generated from the following file:

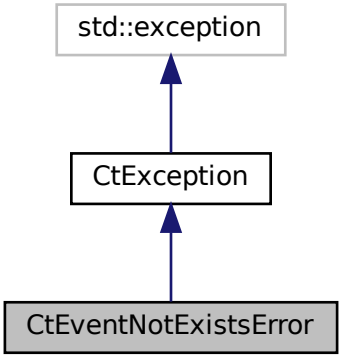
- include/exceptions/CtEventExceptions.hpp

5.5 CtEventNotExistsError Class Reference

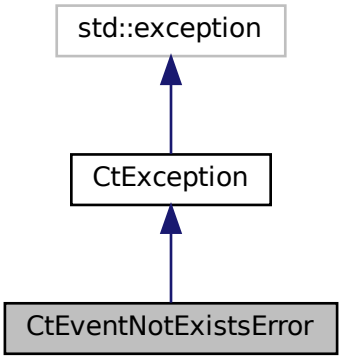
This exception is thrown when an event does not exist in the event manager.

`#include <CtEventExceptions.hpp>`

Inheritance diagram for CtEventNotExistsError:



Collaboration diagram for CtEventNotExistsError:



Public Member Functions

- CtEventNotExistsError (const std::string &msg)

Additional Inherited Members

5.5.1 Detailed Description

This exception is thrown when an event does not exist in the event manager.

Definition at line 41 of file CtEventExceptions.hpp.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 CtEventNotExistsError()

```
CtEventNotExistsError::CtEventNotExistsError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 43 of file CtEventExceptions.hpp.

The documentation for this class was generated from the following file:

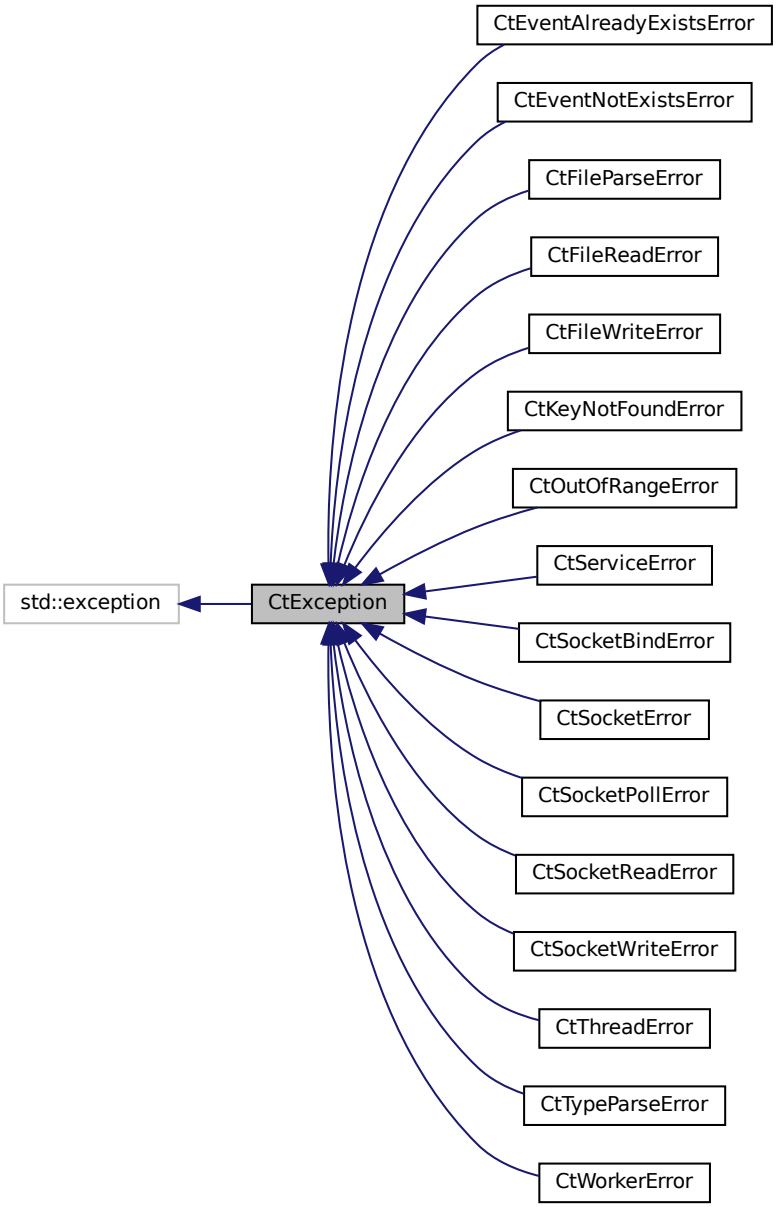
- include/exceptions/CtEventExceptions.hpp

5.6 CtException Class Reference

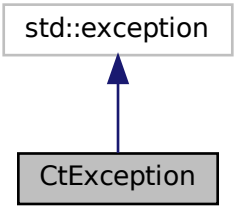
An exception class for the cpptoolkit library.

```
#include <CtException.hpp>
```

Inheritance diagram for CtException:



Collaboration diagram for CtException:



Public Member Functions

- const char * what () const noexcept override

  This method returns the message stored in the exception.

Protected Member Functions

- CtException (const std::string &msg)

  Construct a new Ct Exception object.

Private Attributes

- std::string m_msg

5.6.1 Detailed Description

An exception class for the cpptoolkit library.

This is an abstract class derived from std::exception and is used as a base class for all the exceptions in the library.

Definition at line 45 of file CtException.hpp.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 CtException()

```
CtException::CtException (
            const std::string & msg )  [inline], [explicit], [protected]
```

Construct a new Ct Exception object.

Parameters

msg	Message to be stored in the exception.

Definition at line 52 of file CtException.hpp.

5.6.3 Member Function Documentation

5.6.3.1 what()

```
const char* CtException::what ( ) const  [inline], [override], [noexcept]
```

This method returns the message stored in the exception.

Returns

const char* the message stored in the exception.

Definition at line 60 of file CtException.hpp.

5.6.4 Member Data Documentation

5.6.4.1 m_msg

```
std::string CtException::m_msg  [private]
```

The message stored in the exception.

Definition at line 65 of file CtException.hpp.

The documentation for this class was generated from the following file:

- include/exceptions/CtException.hpp

5.7 CtFileInput Class Reference

CtFileInput class for reading data from file.

```
#include <CtFileInput.hpp>
```

Public Member Functions

- EXPORTED_API CtFileInput (const std::string &p_fileName)

  Constructs the CtFileInput object.
- EXPORTED_API ~CtFileInput ()

  Destructor for CtFileInput.
- EXPORTED_API void setDelimiter (const char *p_delim, CtUInt8 p_delim_size)

  Set the the delimiter of read() method.
- EXPORTED_API bool read (CtRawData *p_data)

  This method read data from the file.

Private Attributes

- std::ifstream m_file
- char * m_delim
- CtUInt8 m_delim_size

5.7.1 Detailed Description

CtFileInput class for reading data from file.

This class provides an interface for reading data from a file. The data can be read in batches or one by one. The class is thread-safe and can be used in multi-threaded environments.

```
// create a file input object
CtFileInput fileInput("input.txt");
CtRawData data;
while (fileInput.read(&data)) {
   // process data
}
```

Definition at line 59 of file CtFileInput.hpp.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 CtFileInput()

```
CtFileInput::CtFileInput (
            const std::string & p_fileName )  [explicit]
```

Constructs the CtFileInput object.

Parameters

p_fileName	Filename.

Definition at line 36 of file CtFileInput.cpp.

5.7.2.2 ~CtFileInput()

```
CtFileInput::~CtFileInput ( )
```

Destructor for CtFileInput.

Performs any necessary cleanup.

Definition at line 45 of file CtFileInput.cpp.

5.7.3 Member Function Documentation

5.7.3.1 read()

```
bool CtFileInput::read (
            CtRawData * p_data )
```

This method read data from the file.

Parameters

p_data	Where to store the data read

Returns

bool Returns True on success or False on EOF.

Definition at line 62 of file CtFileInput.cpp.

5.7.3.2 setDelimiter()

```
void CtFileInput::setDelimiter (
            const char * p_delim,
            CtUInt8 p_delim_size )
```

Set the the delimiter of read() method.

Parameters

p_delim	The delimiter.
p_delim_size	The delimiter size.

Definition at line 54 of file CtFileInput.cpp.

5.7.4 Member Data Documentation

5.7.4.1 m_delim

```
char* CtFileInput::m_delim  [private]
```

Batch read delimiter.

Definition at line 93 of file CtFileInput.hpp.

5.7.4.2 m_delim_size

```
CtUInt8 CtFileInput::m_delim_size  [private]
```

Delimeter size.

Definition at line 94 of file CtFileInput.hpp.

5.7.4.3 m_file

```
std::ifstream CtFileInput::m_file  [private]
```

File stream.

Definition at line 92 of file CtFileInput.hpp.

The documentation for this class was generated from the following files:

- include/io/CtFileInput.hpp
- src/io/CtFileInput.cpp

5.8 CtFileOutput Class Reference

CtFileOutput class for writing data to file.

```
#include <CtFileOutput.hpp>
```

Public Types

- enum class WriteMode { Append , Truncate }

  Enum representing write mode.

Public Member Functions

- EXPORTED_API CtFileOutput (const std::string &p_fileName, WriteMode p_mode=WriteMode::Append)

  Constructs the CtFileOutput object.
- EXPORTED_API ~CtFileOutput ()

  Destructor for CtFileOutput.
- EXPORTED_API void setDelimiter (const char *p_delim, CtUInt8 p_delim_size)

  Set the the delimiter of write() method.
- EXPORTED_API void write (CtRawData *p_data)

  This method writes to file.

Private Attributes

- std::ofstream m_file
- std::unique_ptr< char[]> m_delim
- CtUInt8 m_delim_size

5.8.1 Detailed Description

CtFileOutput class for writing data to file.

This class provides an interface for writing data to a file. The data can be written in batches or one by one. The class is thread-safe and can be used in multi-threaded environments.

```
// create a file output object
CtFileOutput fileOutput("output.txt");
fileOutput.write("Hello, World!");
```

Definition at line 56 of file CtFileOutput.hpp.

5.8.2 Member Enumeration Documentation

5.8.2.1 WriteMode

```
enum CtFileOutput::WriteMode  [strong]
```

Enum representing write mode.

Enumerator

Append	
Truncate	

Definition at line 61 of file CtFileOutput.hpp.

5.8.3 Constructor & Destructor Documentation

5.8.3.1 CtFileOutput()

```
CtFileOutput::CtFileOutput (
            const std::string & p_fileName,
            WriteMode p_mode = WriteMode::Append )  [explicit]
```

Constructs the CtFileOutput object.

Parameters

p_fileName	Filename.

Definition at line 38 of file CtFileOutput.cpp.

5.8.3.2 ~CtFileOutput()

```
CtFileOutput::~CtFileOutput ( )
```

Destructor for CtFileOutput.

Performs any necessary cleanup.

Definition at line 54 of file CtFileOutput.cpp.

5.8.4 Member Function Documentation

5.8.4.1 setDelimiter()

```
void CtFileOutput::setDelimiter (
            const char * p_delim,
            CtUInt8 p_delim_size )
```

Set the the delimiter of write() method.

Parameters

p_delim	The delimiter.
p_delim_size	The delimiter size.

Definition at line 60 of file CtFileOutput.cpp.

5.8.4.2 write()

```
void CtFileOutput::write (
            CtRawData * p_data )
```

This method writes to file.

Parameters

p_data	The data to be written.

Returns

void

Definition at line 69 of file CtFileOutput.cpp.

5.8.5 Member Data Documentation

5.8.5.1 m_delim

```
std::unique_ptr<char[]> CtFileOutput::m_delim  [private]
```

Batch write delimiter.

Definition at line 95 of file CtFileOutput.hpp.

5.8.5.2 m_delim_size

```
CtUInt8 CtFileOutput::m_delim_size  [private]
```

Delimeter size.

Definition at line 96 of file CtFileOutput.hpp.

5.8.5.3 m_file

```
std::ofstream CtFileOutput::m_file  [private]
```

File stream.

Definition at line 94 of file CtFileOutput.hpp.

The documentation for this class was generated from the following files:

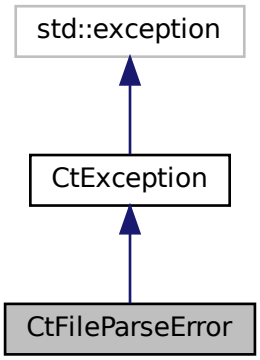
- include/io/CtFileOutput.hpp
- src/io/CtFileOutput.cpp

5.9 CtFileParseError Class Reference

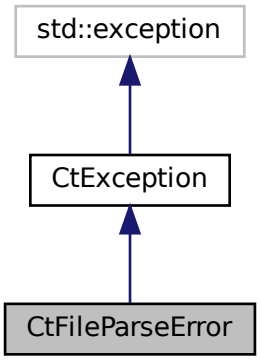
This exception is thrown when a file cannot be parsed.

```
#include <CtFileExceptions.hpp>
```

Inheritance diagram for CtFileParseError:



Collaboration diagram for CtFileParseError:



Public Member Functions

- CtFileParseError (const std::string &msg)

Additional Inherited Members

5.9.1 Detailed Description

This exception is thrown when a file cannot be parsed.

Definition at line 59 of file CtFileExceptions.hpp.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 CtFileParseError()

```
CtFileParseError::CtFileParseError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 61 of file CtFileExceptions.hpp.

The documentation for this class was generated from the following file:

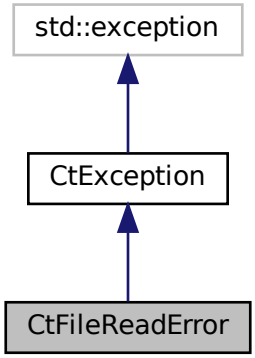
- include/exceptions/CtFileExceptions.hpp

5.10 CtFileReadError Class Reference

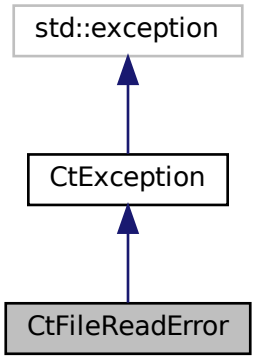
This exception is thrown when a file cannot be read.

`#include <CtFileExceptions.hpp>`

Inheritance diagram for CtFileReadError:



Collaboration diagram for CtFileReadError:



Public Member Functions

- CtFileReadError (const std::string &msg)

Additional Inherited Members

5.10.1 Detailed Description

This exception is thrown when a file cannot be read.

Definition at line 41 of file CtFileExceptions.hpp.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 CtFileReadError()

```
CtFileReadError::CtFileReadError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 43 of file CtFileExceptions.hpp.

The documentation for this class was generated from the following file:

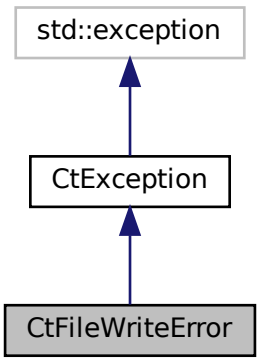
- include/exceptions/CtFileExceptions.hpp

5.11 CtFileWriteError Class Reference

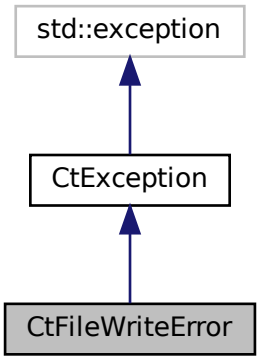
This exception is thrown when a file cannot be written.

```
#include <CtFileExceptions.hpp>
```

Inheritance diagram for CtFileWriteError:



Collaboration diagram for CtFileWriteError:



Public Member Functions

- CtFileWriteError (const std::string &msg)

Additional Inherited Members

5.11.1 Detailed Description

This exception is thrown when a file cannot be written.

Definition at line 50 of file CtFileExceptions.hpp.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 CtFileWriteError()

```
CtFileWriteError::CtFileWriteError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 52 of file CtFileExceptions.hpp.

The documentation for this class was generated from the following file:

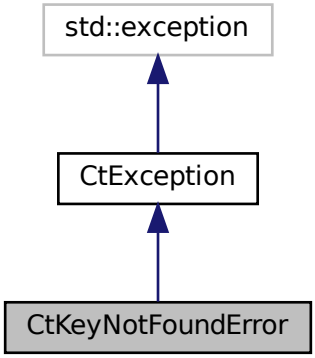
- include/exceptions/CtFileExceptions.hpp

5.12 CtKeyNotFoundError Class Reference

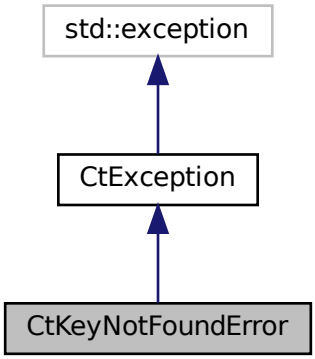
This exception is thrown when a key is not found in a container.

`#include <CtTypeExceptions.hpp>`

Inheritance diagram for CtKeyNotFoundError:



Collaboration diagram for CtKeyNotFoundError:



Public Member Functions

- CtKeyNotFoundError (const std::string &msg)

Additional Inherited Members

5.12.1 Detailed Description

This exception is thrown when a key is not found in a container.

Definition at line 50 of file CtTypeExceptions.hpp.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 CtKeyNotFoundError()

```
CtKeyNotFoundError::CtKeyNotFoundError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 52 of file CtTypeExceptions.hpp.

The documentation for this class was generated from the following file:

- include/exceptions/CtTypeExceptions.hpp

5.13 CtLogger Class Reference

A simple logger with log levels and timestamp.

```
#include <CtLogger.hpp>
```

Public Types

- enum class Level {
  DEBUG , INFO , WARNING , ERROR ,
  CRITICAL }

  Enum representing log levels.

Public Member Functions

- EXPORTED_API CtLogger (CtLogger::Level level=CtLogger::Level::DEBUG, const std::string &componentName="")

  Constructs a CtLogger with a component name.
- EXPORTED_API ~CtLogger ()

  Destructor.
- EXPORTED_API void log_debug (const std::string &message)

  Log a message with debug log level.
- EXPORTED_API void log_info (const std::string &message)

  Log a message with info log level.
- EXPORTED_API void log_warning (const std::string &message)

  Log a message with warning log level.
- EXPORTED_API void log_error (const std::string &message)

  Log a message with error log level.
- EXPORTED_API void log_critical (const std::string &message)

  Log a message with critical log level.

Static Public Member Functions

- static EXPORTED_API CtLogger::Level stringToLevel (const std::string &level_str)

  Given the logger output level in string format this method returns the enum CtLogger::Level format.

Private Member Functions

- void log (CtLogger::Level level, const std::string &message)

  Log a message with the specified log level.

Static Private Member Functions

- static const std::string levelToString (CtLogger::Level level)

  Given the logger output level in enum CtLogger::Level format this method returns it in a string format.
- static const std::string generateLoggerMsg (CtLogger::Level level, const std::string &component_name, const std::string &message)

  This method generates the message to be printed via logger.

Private Attributes

- std::mutex m_mtx_control
- CtLogger::Level m_level
- std::string m_componentName

5.13.1 Detailed Description

A simple logger with log levels and timestamp.

The CtLogger class provides a mechanism for logging messages with different log levels. The log levels are DEBUG, INFO, WARNING, ERROR, and CRITICAL and can be used to filter messages. The logger also provides a timestamp for each message. It is thread-safe and can be used in multi-threaded environments.

Definition at line 54 of file CtLogger.hpp.

5.13.2 Member Enumeration Documentation

5.13.2.1 Level

```
enum CtLogger::Level  [strong]
```

Enum representing log levels.

Enumerator

DEBUG	
INFO	
WARNING	
ERROR	
CRITICAL	

Definition at line 59 of file CtLogger.hpp.

5.13.3 Constructor & Destructor Documentation

5.13.3.1 CtLogger()

```
CtLogger::CtLogger (
            CtLogger::Level level = CtLogger::Level::DEBUG,
            const std::string & componentName = "" )  [explicit]
```

Constructs a CtLogger with a component name.

Parameters

level	The selected level given as CtLogger::Level. All messages that have level above or equal to this value will be logged.
componentName	The name of the component or module.

Definition at line 34 of file CtLogger.cpp.

5.13.3.2 ~CtLogger()

```
CtLogger::~CtLogger ( )
```

Destructor.

Definition at line 37 of file CtLogger.cpp.

5.13.4 Member Function Documentation

5.13.4.1 generateLoggerMsg()

```
const std::string CtLogger::generateLoggerMsg (
            CtLogger::Level level,
            const std::string & component_name,
            const std::string & message )  [static], [private]
```

This method generates the message to be printed via logger.

Parameters

level	The level of the message.
component_name	The component's name.
message	The message.

Returns

const std::string The generated message to be printed via logger.

Definition at line 68 of file CtLogger.cpp.

5.13.4.2 levelToString()

```
const std::string CtLogger::levelToString (
            CtLogger::Level level )  [static], [private]
```

Given the logger output level in enum CtLogger::Level format this method returns it in a string format.

Parameters

level	The level in enum CtLogger::Level format.

Returns

std::string The level in string format.

Definition at line 77 of file CtLogger.cpp.

5.13.4.3 log()

```
void CtLogger::log (
            CtLogger::Level level,
            const std::string & message )  [private]
```

Log a message with the specified log level.

Parameters

level	The log level.
componentName	The name of the component or module.
message	The log message.

Definition at line 60 of file CtLogger.cpp.

5.13.4.4 log_critical()

```
void CtLogger::log_critical (
            const std::string & message )
```

Log a message with critical log level.

Parameters

message	The log message.

Definition at line 56 of file CtLogger.cpp.

5.13.4.5 log_debug()

```
void CtLogger::log_debug (
            const std::string & message )
```

Log a message with debug log level.

Parameters

message	The log message.

Definition at line 40 of file CtLogger.cpp.

5.13.4.6 log_error()

```
void CtLogger::log_error (
            const std::string & message )
```

Log a message with error log level.

Parameters

message	The log message.

Definition at line 52 of file CtLogger.cpp.

5.13.4.7 log_info()

```
void CtLogger::log_info (
            const std::string & message )
```

Log a message with info log level.

Parameters

message	The log message.

Definition at line 44 of file CtLogger.cpp.

5.13.4.8 log_warning()

```
void CtLogger::log_warning (
            const std::string & message )
```

Log a message with warning log level.

Parameters

message	The log message.

Definition at line 48 of file CtLogger.cpp.

5.13.4.9 stringToLevel()

```
CtLogger::Level CtLogger::stringToLevel (
            const std::string & level_str )  [static]
```

Given the logger output level in string format this method returns the enum CtLogger::Level format.

Parameters

level_str	The level in string format.

Returns

CtLogger::Level The level in enum format.

Definition at line 103 of file CtLogger.cpp.

5.13.5 Member Data Documentation

5.13.5.1 m_componentName

```
std::string CtLogger::m_componentName  [private]
```

Component name.

Definition at line 147 of file CtLogger.hpp.

5.13.5.2 m_level

```
CtLogger::Level CtLogger::m_level  [private]
```

Level of message logging.

Definition at line 146 of file CtLogger.hpp.

5.13.5.3 m_mtx_control

```
std::mutex CtLogger::m_mtx_control  [private]
```

Mutex for controlling access to shared resources.

Definition at line 145 of file CtLogger.hpp.

The documentation for this class was generated from the following files:

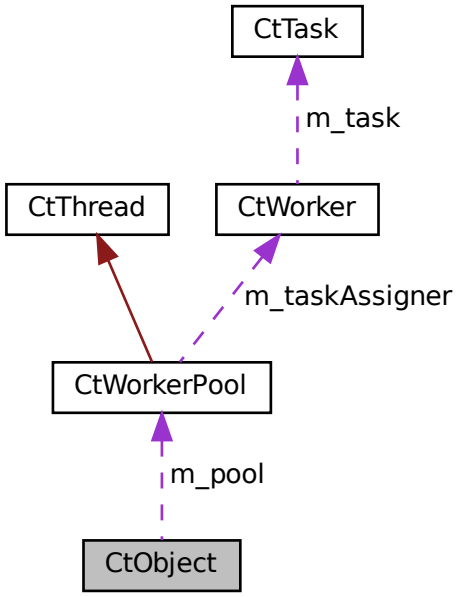
- include/utils/CtLogger.hpp
- src/utils/CtLogger.cpp

5.14 CtObject Class Reference

This abstract class can be used as a base class for objects that can trigger events.

```
#include <CtObject.hpp>
```

Collaboration diagram for CtObject:



Public Member Functions

- template<typename F , typename... FArgs>
  EXPORTED_API void connectEvent (CtUInt32 p_eventCode, F &&func, FArgs &&... fargs)

  This method connects an event code with a function that should be triggered.
- EXPORTED_API void connectEvent (CtUInt32 p_eventCode, CtTask &p_task)

  This method connects an event code with a function that should be triggered.
- EXPORTED_API void waitPendingEvents ()

  This method holds current thread waiting for all the pending events of this object to finish.
- template<typename F , typename... FArgs>
  void connectEvent (CtObject *p_obj, CtUInt32 p_eventCode, F &&func, FArgs &&... fargs)
- template<typename F , typename... FArgs>
  void connectEvent (CtUInt32 p_eventCode, F &&func, FArgs &&... fargs)

Static Public Member Functions

- template<typename F , typename... FArgs>
  static EXPORTED_API void connectEvent (CtObject *p_obj, CtUInt32 p_eventCode, F &&func, FArgs &&... fargs)

  This method connects an event code with a function that should be triggered.
- static EXPORTED_API void connectEvent (CtObject *p_obj, CtUInt32 p_eventCode, CtTask &p_task)

  This method connects an event code with a function that should be triggered.

Protected Member Functions

- EXPORTED_API CtObject ()

  The constructor of the CtObject class.
- EXPORTED_API ~CtObject ()

  The destructor of the CtObject class.
- EXPORTED_API void triggerEvent (CtUInt32 p_eventCode)

  This method triggers a specific event code.
- EXPORTED_API void registerEvent (CtUInt32 p_eventCode)

  This event registers a specific event code.

Private Member Functions

- EXPORTED_API bool hasEvent (CtUInt32 p_eventCode)

  This methods checks if a specific event code is already registered in this object.

Private Attributes

- std::mutex m_mtx_control
- std::vector< CtUInt32 > m_events
- std::multimap< CtUInt32, CtTask > m_triggers
- CtWorkerPool m_pool

5.14.1 Detailed Description

This abstract class can be used as a base class for objects that can trigger events.

The CtObject class provides a mechanism for connecting events with functions that should be triggered. This class is thread-safe and can be used in multi-threaded environments.

```
triggerEvent(100);
connectEvent(obj, 100, [](){});
connectEvent(obj, 100, [](){});
```

Definition at line 61 of file CtObject.hpp.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 CtObject()

```
CtObject::CtObject ( )  [protected]
```

The constructor of the CtObject class.

Definition at line 38 of file CtObject.cpp.

5.14.2.2 ~CtObject()

```
CtObject::~CtObject ( )  [protected]
```

The destructor of the CtObject class.

Definition at line 42 of file CtObject.cpp.

5.14.3 Member Function Documentation

5.14.3.1 connectEvent() [1/6]

```
void CtObject::connectEvent (
            CtObject * p_obj,
            CtUInt32 p_eventCode,
            CtTask & p_task )  [static]
```

This method connects an event code with a function that should be triggered.

Parameters

p_obj	The object that hosts the event.
p_eventCode	The event code.
p_task	The task to be executed.

Returns

void

Definition at line 46 of file CtObject.cpp.

5.14.3.2 connectEvent() [2/6]

```
template<typename F , typename...  FArgs>
static EXPORTED_API void CtObject::connectEvent (
            CtObject * p_obj,
            CtUInt32 p_eventCode,
            F && func,
            FArgs &&...  fargs )  [static]
```

This method connects an event code with a function that should be triggered.

Template Parameters

F	Type of the callable function.
FArgs	Types of the arguments for the callable function.

Parameters

p_obj	The object that hosts the event.
p_eventCode	The event code.
func	The function to be executed.
fargs	The parameters of the function that will be executed.

Returns

void

5.14.3.3 connectEvent() [3/6]

```
template<typename F , typename...  FArgs>
void CtObject::connectEvent (
            CtObject * p_obj,
            CtUInt32 p_eventCode,
            F && func,
            FArgs &&...  fargs )
```

Definition at line 172 of file CtObject.hpp.

5.14.3.4 connectEvent() [4/6]

```
void CtObject::connectEvent (
            CtUInt32 p_eventCode,
            CtTask & p_task )
```

This method connects an event code with a function that should be triggered.

Parameters

p_eventCode	The event code.
p_task	The task to be executed.

Returns

void

Definition at line 54 of file CtObject.cpp.

5.14.3.5 connectEvent() [5/6]

```
template<typename F , typename...  FArgs>
EXPORTED_API void CtObject::connectEvent (
            CtUInt32 p_eventCode,
            F && func,
            FArgs &&...  fargs )
```

This method connects an event code with a function that should be triggered.

Template Parameters

F	Type of the callable function.
FArgs	Types of the arguments for the callable function.

Parameters

p_eventCode	The event code.
func	The function to be executed.
fargs	The parameters of the function that will be executed.

Returns

void

5.14.3.6 connectEvent() [6/6]

```
template<typename F , typename...  FArgs>
void CtObject::connectEvent (
            CtUInt32 p_eventCode,
            F && func,
            FArgs &&...  fargs )
```

Definition at line 179 of file CtObject.hpp.

5.14.3.7 hasEvent()

```
bool CtObject::hasEvent (
            CtUInt32 p_eventCode )  [private]
```

This methods checks if a specific event code is already registered in this object.

Parameters

p_eventCode	The event code to be checked.

Returns

bool True if the event code is registered, false otherwise.

Definition at line 84 of file CtObject.cpp.

5.14.3.8 registerEvent()

```
void CtObject::registerEvent (
            CtUInt32 p_eventCode )  [protected]
```

This event registers a specific event code.

Parameters

p_eventCode	The event code to be registered.

Returns

void

Definition at line 76 of file CtObject.cpp.

5.14.3.9 triggerEvent()

```
void CtObject::triggerEvent (
            CtUInt32 p_eventCode )  [protected]
```

This method triggers a specific event code.

Parameters

p_eventCode	The event code to be triggered.

Returns

void

Definition at line 62 of file CtObject.cpp.

5.14.3.10 waitPendingEvents()

```
void CtObject::waitPendingEvents ( )
```

This method holds current thread waiting for all the pending events of this object to finish.

Returns

void

Definition at line 50 of file CtObject.cpp.

5.14.4 Member Data Documentation

5.14.4.1 m_events

```
std::vector<CtUInt32> CtObject::m_events  [private]
```

This vector contains all the registered event codes.

Definition at line 166 of file CtObject.hpp.

5.14.4.2 m_mtx_control

```
std::mutex CtObject::m_mtx_control  [private]
```

Mutex for controlling access to shared resources.

Definition at line 165 of file CtObject.hpp.

5.14.4.3 m_pool

```
CtWorkerPool CtObject::m_pool  [private]
```

This CtWorkerPool executes the triggered tasks.

Definition at line 168 of file CtObject.hpp.

5.14.4.4 m_triggers

```
std::multimap<CtUInt32, CtTask> CtObject::m_triggers  [private]
```

This map represents a list of tasks that should be triggered for each event code.

Definition at line 167 of file CtObject.hpp.

The documentation for this class was generated from the following files:

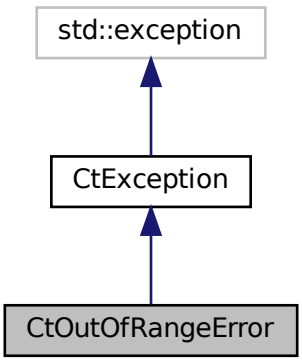
- include/utils/CtObject.hpp
- src/utils/CtObject.cpp

5.15 CtOutOfRangeError Class Reference

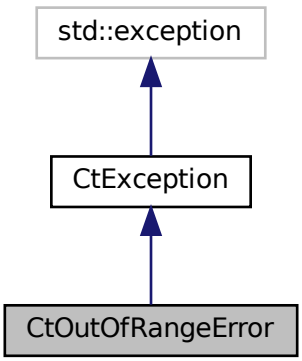
This exception is thrown when an index is out of bounds.

`#include <CtTypeExceptions.hpp>`

Inheritance diagram for CtOutOfRangeError:



Collaboration diagram for CtOutOfRangeError:



Public Member Functions

- CtOutOfRangeError (const std::string &msg)

Additional Inherited Members

5.15.1 Detailed Description

This exception is thrown when an index is out of bounds.

Definition at line 59 of file CtTypeExceptions.hpp.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 CtOutOfRangeError()

```
CtOutOfRangeError::CtOutOfRangeError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 61 of file CtTypeExceptions.hpp.

The documentation for this class was generated from the following file:

- include/exceptions/CtTypeExceptions.hpp

5.16 CtRawData Class Reference

Struct describing raw data buffer.

```
#include <CtTypes.hpp>
```

Public Member Functions

- EXPORTED_API CtRawData (CtUInt32 p_size=CT_BUFFER_SIZE)

  CtRawData constructor.
- EXPORTED_API CtRawData (CtRawData &p_data)

  CtRawData copy constructor.
- virtual EXPORTED_API ~CtRawData ()

  Destructor.
- EXPORTED_API void nextByte (char byte)

  Sets the next byte of the buffer. It also raises the size of the buffer. If the buffer is full an exception will be thrown - CtOutOfRangeError()
- EXPORTED_API CtUInt8 * getNLastBytes (CtUInt32 p_num)

  This method returns a pointer to the last N bytes of the buffer. If the number of bytes is greater than the buffer size an exception will be thrown - CtOutOfRangeError()
- EXPORTED_API void removeNLastBytes (CtUInt32 p_num)

  This method removes the last N bytes of the buffer. It also reduces the size of the buffer. If the number of bytes is greater than the buffer size an exception will be thrown - CtOutOfRangeError()
- EXPORTED_API CtUInt32 size ()

The actual size of the buffer.

- EXPORTED_API CtUInt32 maxSize ()

  The max size of the buffer.

- EXPORTED_API CtUInt8 * get ()

  This method returns a pointer to the buffer data.

- EXPORTED_API void clone (const CtUInt8 *p_data, CtUInt32 p_size)

  This method fills the buffer with the data given in the parameters. This method overwrites the buffer and the actual size. The maximum size of the buffer is preserved. If the size of the data is greater than the buffer size an exception will be thrown - CtOutOfRangeError()

- EXPORTED_API void clone (CtRawData &p_data)

  This method fills the buffer with the data given in the parameters. This method overwrites the buffer and the actual size. The maximum size of the buffer is preserved. If the size of the data is greater than the buffer size an exception will be thrown - CtOutOfRangeError()

- EXPORTED_API void reset ()

  This method resets the buffer to 0 size. The allocated memory is not freed. The actual size of the buffer is set to 0.

- EXPORTED_API CtRawData & operator= (CtRawData &other)

  Assignment operator for CtRawData. Copies the data from a CtRawData to another CtRawData object.

Private Attributes

- CtUInt8 * m_data
- CtUInt32 m_size
- const CtUInt32 m_maxSize

5.16.1 Detailed Description

Struct describing raw data buffer.

The default buffer size is defined as CT_BUFFER_SIZE bytes. The buffer can be set either by another buffer object or given the size of the buffer. The buffer has a prespecified size that can be filled with bytes. It can monitor the size of the buffer that it is currently used and it ensures that the buffer will not overflow. If an overflow occurs an exception will be thrown - CtOutOfRangeError().

```
CtRawData data;
data.nextByte('a');
data.nextByte('b');
data.nextByte('c');
CtUInt8* buffer = data.get(); // returns "abc"
data.removeNLastBytes(1);
buffer = data.get(); // returns "ab"
data.reset(); // resets the buffer
buffer = data.get(); // returns ""
```

Definition at line 147 of file CtTypes.hpp.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 CtRawData() [1/2]

```
EXPORTED_API CtRawData::CtRawData (
            CtUInt32 p_size = CT_BUFFER_SIZE )  [inline], [explicit]
```

CtRawData constructor.

Parameters

p_size	The size of the buffer. The default size is defined as CT_BUFFER_SIZE bytes.

Definition at line 155 of file CtTypes.hpp.

5.16.2.2 CtRawData() [2/2]

```
EXPORTED_API CtRawData::CtRawData (
            CtRawData & p_data )  [inline]
```

CtRawData copy constructor.

Parameters

p_data	Another CtRawData object that it is used to init the currently created.

Definition at line 165 of file CtTypes.hpp.

5.16.2.3 ~CtRawData()

```
virtual EXPORTED_API CtRawData::~CtRawData ( )  [inline], [virtual]
```

Destructor.

Definition at line 174 of file CtTypes.hpp.

5.16.3 Member Function Documentation

5.16.3.1 clone() [1/2]

```
EXPORTED_API void CtRawData::clone (
            const CtUInt8 * p_data,
            CtUInt32 p_size )  [inline]
```

This method fills the buffer with the data given in the parameters. This method overwrites the buffer and the actual size. The maximum size of the buffer is preserved. If the size of the data is greater than the buffer size an exception will be thrown - CtOutOfRangeError()

Parameters

p_data	A pointer to the data to be cloned.
p_size	The size of the gven buffer.

Returns

void

Definition at line 255 of file CtTypes.hpp.

5.16.3.2 clone() [2/2]

```
EXPORTED_API void CtRawData::clone (
            CtRawData & p_data )  [inline]
```

This method fills the buffer with the data given in the parameters. This method overwrites the buffer and the actual size. The maximum size of the buffer is preserved. If the size of the data is greater than the buffer size an exception will be thrown - CtOutOfRangeError()

Parameters

p_data	A CtRawData object to be cloned.

Returns

void

Definition at line 272 of file CtTypes.hpp.

5.16.3.3 get()

```
EXPORTED_API CtUInt8* CtRawData::get ( )  [inline]
```

This method returns a pointer to the buffer data.

Returns

CtUInt8* Pointer to the buffer data.

Definition at line 242 of file CtTypes.hpp.

5.16.3.4 getNLastBytes()

```
EXPORTED_API CtUInt8* CtRawData::getNLastBytes (
            CtUInt32 p_num )  [inline]
```

This method returns a pointer to the last N bytes of the buffer. If the number of bytes is greater than the buffer size an exception will be thrown - CtOutOfRangeError()

Parameters

p_num	Number of bytes to be returned.

Returns

CtUInt8* Pointer to the last N bytes of the buffer.

Definition at line 198 of file CtTypes.hpp.

5.16.3.5 maxSize()

```
EXPORTED_API CtUInt32 CtRawData::maxSize ( )  [inline]
```

The max size of the buffer.

Returns

CtUInt32 The max size of the buffer.

Definition at line 233 of file CtTypes.hpp.

5.16.3.6 nextByte()

```
EXPORTED_API void CtRawData::nextByte (
            char byte )  [inline]
```

Sets the next byte of the buffer. It also raises the size of the buffer. If the buffer is full an exception will be thrown - CtOutOfRangeError()

Parameters

byte	The byte to be added.

Returns

void

Definition at line 185 of file CtTypes.hpp.

5.16.3.7 operator=()

```
EXPORTED_API CtRawData& CtRawData::operator= (
            CtRawData & other )  [inline]
```

Assignment operator for CtRawData. Copies the data from a CtRawData to another CtRawData object.

Parameters

other	The CtRawData object to copy.

Returns

CtRawData& Reference to the current CtRawData object.

Definition at line 298 of file CtTypes.hpp.

5.16.3.8 removeNLastBytes()

```
EXPORTED_API void CtRawData::removeNLastBytes (
            CtUInt32 p_num )  [inline]
```

This method removes the last N bytes of the buffer. It also reduces the size of the buffer. If the number of bytes is greater than the buffer size an exception will be thrown - CtOutOfRangeError()

Parameters

p_num	Number of bytes to be returned.

Returns

EXPORTED_API

Definition at line 212 of file CtTypes.hpp.

5.16.3.9 reset()

```
EXPORTED_API void CtRawData::reset ( )  [inline]
```

This method resets the buffer to 0 size. The allocated memory is not freed. The actual size of the buffer is set to 0.

Returns

void

Definition at line 286 of file CtTypes.hpp.

5.16.3.10 size()

```
EXPORTED_API CtUInt32 CtRawData::size ( )  [inline]
```

The actual size of the buffer.

Returns

CtUInt32 The actual size of the buffer.

Definition at line 224 of file CtTypes.hpp.

5.16.4 Member Data Documentation

5.16.4.1 m_data

```
CtUInt8* CtRawData::m_data  [private]
```

The buffer data.

Definition at line 306 of file CtTypes.hpp.

5.16.4.2 m_maxSize

```
const CtUInt32 CtRawData::m_maxSize  [private]
```

The maximum size of the buffer.

Definition at line 308 of file CtTypes.hpp.

5.16.4.3 m_size

```
CtUInt32 CtRawData::m_size  [private]
```

The actual size of the buffer.

Definition at line 307 of file CtTypes.hpp.

The documentation for this class was generated from the following file:

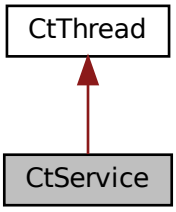
- include/CtTypes.hpp

5.17 CtService Class Reference

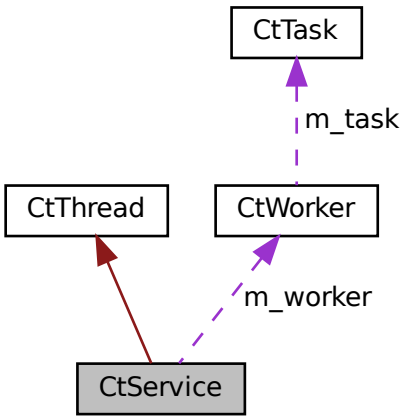
A class representing a service that runs a given task at regular intervals using a worker thread.

`#include <CtService.hpp>`

Inheritance diagram for CtService:



Collaboration diagram for CtService:



Public Member Functions

- EXPORTED_API CtService (CtUInt64 nslots, const CtTask &task)

  Constructor for CtService.
- template<typename F , typename... FArgs>
  EXPORTED_API CtService (CtUInt64 nslots, const F &&func, FArgs &&... fargs)

  Constructor for CtService.
- EXPORTED_API ~CtService ()

Destructor for CtService.

- EXPORTED_API void runService ()

  Run the task provided by the service.

- EXPORTED_API void stopService ()

  Stop the task provided by the service.

- template<typename F , typename... FArgs>
  CtService (CtUInt64 nslots, const F &&func, FArgs &&... fargs)

Static Public Attributes

- static CtUInt32 m_slot_time = 10

Private Member Functions

- void loop () override

  Overridden run function from CtThread, representing the main logic of the service.

Private Attributes

- CtWorker m_worker
- uint64_t m_nslots

Additional Inherited Members

5.17.1 Detailed Description

A class representing a service that runs a given task at regular intervals using a worker thread.

The CtService class provides a mechanism for running a task at regular intervals using a worker thread. The service can be configured to run the task immediately or after a certain number of time slots. The service can be stopped and started at any time. The service is thread-safe and can be used in multi-threaded environments.

```
// create a service that runs a task every 1000 milliseconds
CtService service(1000, [](){ std::cout « "Hello from service!" « std::endl; });
service.runService();
// do something else
service.stopService();
```

Definition at line 60 of file CtService.hpp.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 CtService() [1/3]

```
CtService::CtService (
            CtUInt64 nslots,
            const CtTask & task )
```

Constructor for CtService.

Parameters

nslots	The time slots between task executions in milliseconds. Default is 0 (run immediately).
task	The task to be executed by the service.

Definition at line 37 of file CtService.cpp.

5.17.2.2 CtService() [2/3]

```
template<typename F , typename...  FArgs>
EXPORTED_API CtService::CtService (
            CtUInt64 nslots,
            const F && func,
            FArgs &&...  fargs )
```

Constructor for CtService.

Parameters

nslots	The time slots between task executions in milliseconds. Default is 0 (run immediately).
func	The task function to be executed by the service.
fargs	The task function's parameters.

5.17.2.3 ~CtService()

```
CtService::~CtService ( )
```

Destructor for CtService.

Definition at line 42 of file CtService.cpp.

5.17.2.4 CtService() [3/3]

```
template<typename F , typename...  FArgs>
CtService::CtService (
            CtUInt64 nslots,
            const F && func,
            FArgs &&...  fargs )
```

Definition at line 108 of file CtService.hpp.

5.17.3 Member Function Documentation

5.17.3.1 loop()

```
void CtService::loop ( )  [override], [private], [virtual]
```

Overridden run function from CtThread, representing the main logic of the service.

Implements CtThread.

Definition at line 59 of file CtService.cpp.

5.17.3.2 runService()

```
void CtService::runService ( )
```

Run the task provided by the service.

Definition at line 46 of file CtService.cpp.

5.17.3.3 stopService()

```
void CtService::stopService ( )
```

Stop the task provided by the service.

Definition at line 54 of file CtService.cpp.

5.17.4 Member Data Documentation

5.17.4.1 m_nslots

```
uint64_t CtService::m_nslots  [private]
```

The number of slots to wait before rerunning the service.

Definition at line 104 of file CtService.hpp.

5.17.4.2 m_slot_time

```
CtUInt32 CtService::m_slot_time = 10  [static]
```

The time interval for each "slot" in milliseconds.

Definition at line 94 of file CtService.hpp.

5.17.4.3 m_worker

```
CtWorker CtService::m_worker  [private]
```

Worker for executing the task.

Definition at line 103 of file CtService.hpp.

The documentation for this class was generated from the following files:

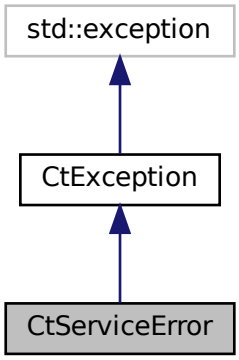
- include/threading/CtService.hpp
- src/threading/CtService.cpp

5.18 CtServiceError Class Reference

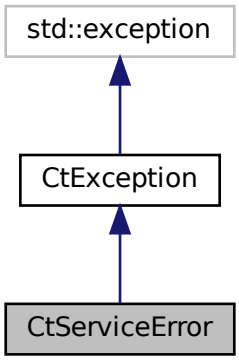
This exception is thrown when a service pool error occurs.

```
#include <CtThreadExceptions.hpp>
```

Inheritance diagram for CtServiceError:



Collaboration diagram for CtServiceError:



Public Member Functions

- CtServiceError (const std::string &msg)

Additional Inherited Members

5.18.1 Detailed Description

This exception is thrown when a service pool error occurs.

Definition at line 50 of file CtThreadExceptions.hpp.

5.18.2 Constructor & Destructor Documentation

5.18.2.1 CtServiceError()

```
CtServiceError::CtServiceError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 52 of file CtThreadExceptions.hpp.

The documentation for this class was generated from the following file:

- include/exceptions/CtThreadExceptions.hpp

5.19 CtServicePack Struct Reference

Represents a pack containing a task, an ID, and an interval for execution.

5.19.1 Detailed Description

Represents a pack containing a task, an ID, and an interval for execution.

The documentation for this struct was generated from the following file:

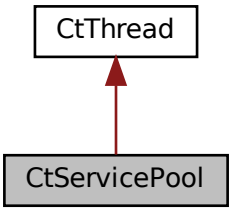
- include/threading/CtServicePool.hpp

5.20 CtServicePool Class Reference

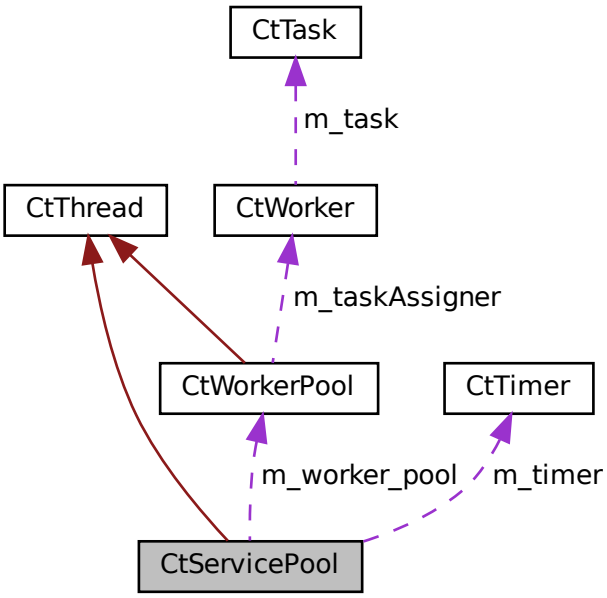
A service pool for managing and executing tasks at specified intervals using a worker pool.

`#include <CtServicePool.hpp>`

Inheritance diagram for CtServicePool:



Collaboration diagram for CtServicePool:



Classes

- struct _CtServicePack

Public Member Functions

- EXPORTED_API CtServicePool (CtUInt32 nworkers)
  Constructor for CtServicePool.
- EXPORTED_API ~CtServicePool ()
  Destructor for CtServicePool.
- EXPORTED_API void addTask (CtUInt32 nslots, const std::string &id, CtTask &task)
  Add a task to the service pool with a specified interval and an optional ID.
- template<typename F , typename... FArgs>
  EXPORTED_API void addTaskFunc (CtUInt32 nslots, const std::string &id, F &&func, FArgs &&... fargs)
  Add a task to the service pool with a specified interval and an optional ID.
- EXPORTED_API void removeTask (const std::string &id)
  Remove a task from the service pool based on its ID.
- EXPORTED_API void startServices ()
  Start the services provided by the service pool.
- EXPORTED_API void shutdownServices ()
  Shutdown the services provided by the service pool.
- EXPORTED_API CtUInt32 getSlotTime ()
  Get slot time.
- EXPORTED_API void setSlotTime (CtUInt32 nslots)
  Set slot time.
- template<typename F , typename... FArgs>
  void addTaskFunc (CtUInt32 nslots, const std::string &id, F &&func, FArgs &&... fargs)

Private Types

- typedef struct CtServicePool::_CtServicePack CtServicePack

Private Member Functions

- void loop () override

  Overridden loop function from CtThread, representing the main thread logic.

Private Attributes

- CtUInt32 m_nworkers
- CtUInt32 m_slot_cnt
- std::vector< CtServicePack > m_tasks
- std::mutex m_mtx_control
- CtWorkerPool m_worker_pool
- CtTimer m_timer
- uint64_t m_exec_time

Additional Inherited Members

5.20.1 Detailed Description

A service pool for managing and executing tasks at specified intervals using a worker pool.

The CtServicePool class provides a mechanism for managing and executing tasks at specified intervals using a worker pool. CtService::m_slot_time is used to determine the interval at which tasks are executed. The default slot time is 10 ms. You can modify the slot time using the setSlotTime() method. The class uses a worker pool to execute tasks concurrently. The class is thread-safe and can be used in multi-threaded environments.

```
// create a service pool with 4 worker threads
CtServicePool pool(4);
// add tasks to the pool
// add a lambda function
pool.addTask(100, [](){ std::cout << "Hello from worker thread!" << std::endl; });
// add a function with arguments
pool.addTask(100, func, arg1, arg2);
// start the services
pool.startServices();
// stop the services
pool.shutdownServices();
```

Definition at line 70 of file CtServicePool.hpp.

5.20.2 Member Typedef Documentation

5.20.2.1 CtServicePack

```
typedef struct CtServicePool::_CtServicePack CtServicePool::CtServicePack  [private]
```

5.20.3 Constructor & Destructor Documentation

5.20.3.1 CtServicePool()

```
CtServicePool::CtServicePool (
            CtUInt32 nworkers )  [explicit]
```

Constructor for CtServicePool.

Parameters

nworkers	The number of worker threads in the service pool.

Definition at line 35 of file CtServicePool.cpp.

5.20.3.2 ~CtServicePool()

```
CtServicePool::~CtServicePool ( )
```

Destructor for CtServicePool.

Definition at line 41 of file CtServicePool.cpp.

5.20.4 Member Function Documentation

5.20.4.1 addTask()

```
void CtServicePool::addTask (
            CtUInt32 nslots,
            const std::string & id,
            CtTask & task )
```

Add a task to the service pool with a specified interval and an optional ID.

Parameters

nslots	The interval in slots for executing the task.
id	An optional ID for the task.
task	The task to be added.

Definition at line 46 of file CtServicePool.cpp.

5.20.4.2 addTaskFunc() [1/2]

```
template<typename F , typename...  FArgs>
EXPORTED_API void CtServicePool::addTaskFunc (
            CtUInt32 nslots,
            const std::string & id,
            F && func,
            FArgs &&...  fargs )
```

Add a task to the service pool with a specified interval and an optional ID.

Parameters

nslots	The interval in slots for executing the task.
id	An optional ID for the task.
func	The task function to be added.
fargs	The task function's arguments to be added.

5.20.4.3 addTaskFunc() [2/2]

```
template<typename F , typename...  FArgs>
void CtServicePool::addTaskFunc (
            CtUInt32 nslots,
            const std::string & id,
            F && func,
            FArgs &&...  fargs )
```

Definition at line 155 of file CtServicePool.hpp.

5.20.4.4 getSlotTime()

```
CtUInt32 CtServicePool::getSlotTime ( )
```

Get slot time.

Definition at line 76 of file CtServicePool.cpp.

5.20.4.5 loop()

```
void CtServicePool::loop ( )  [override], [private], [virtual]
```

Overridden loop function from CtThread, representing the main thread logic.

Implements CtThread.

Definition at line 84 of file CtServicePool.cpp.

5.20.4.6 removeTask()

```
void CtServicePool::removeTask (
            const std::string & id )
```

Remove a task from the service pool based on its ID.

Parameters

id	The ID of the task to be removed.

Definition at line 55 of file CtServicePool.cpp.

5.20.4.7 setSlotTime()

```
void CtServicePool::setSlotTime (
            CtUInt32 nslots )
```

Set slot time.

Definition at line 80 of file CtServicePool.cpp.

5.20.4.8 shutdownServices()

```
void CtServicePool::shutdownServices ( )
```

Shutdown the services provided by the service pool.

Definition at line 71 of file CtServicePool.cpp.

5.20.4.9 startServices()

```
void CtServicePool::startServices ( )
```

Start the services provided by the service pool.

Definition at line 64 of file CtServicePool.cpp.

5.20.5 Member Data Documentation

5.20.5.1 m_exec_time

```
uint64_t CtServicePool::m_exec_time  [private]
```

Variable used for time tracking during a loop.

Definition at line 151 of file CtServicePool.hpp.

5.20.5.2 m_mtx_control

```
std::mutex CtServicePool::m_mtx_control  [private]
```

Mutex for controlling access to shared resources.

Definition at line 148 of file CtServicePool.hpp.

5.20.5.3 m_nworkers

```
CtUInt32 CtServicePool::m_nworkers  [private]
```

The number of worker threads in the service pool.

Definition at line 145 of file CtServicePool.hpp.

5.20.5.4 m_slot_cnt

```
CtUInt32 CtServicePool::m_slot_cnt  [private]
```

Counter for the current slot.

Definition at line 146 of file CtServicePool.hpp.

5.20.5.5 m_tasks

```
std::vector<CtServicePack> CtServicePool::m_tasks  [private]
```

Vector of tasks in the service pool.

Definition at line 147 of file CtServicePool.hpp.

5.20.5.6 m_timer

```
CtTimer CtServicePool::m_timer  [private]
```

Timer for tracking time intervals.

Definition at line 150 of file CtServicePool.hpp.

5.20.5.7 m_worker_pool

```
CtWorkerPool CtServicePool::m_worker_pool  [private]
```

Worker pool for executing tasks.

Definition at line 149 of file CtServicePool.hpp.

The documentation for this class was generated from the following files:

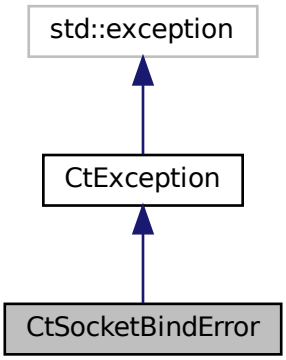
- include/threading/CtServicePool.hpp
- src/threading/CtServicePool.cpp

5.21 CtSocketBindError Class Reference

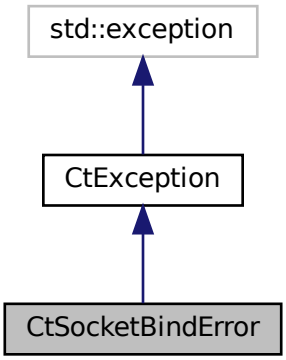
This exception is thrown when a socket bind error occurs.

```
#include <CtNetworkExceptions.hpp>
```

Inheritance diagram for CtSocketBindError:



Collaboration diagram for CtSocketBindError:



Public Member Functions

- CtSocketBindError (const std::string &msg)

Additional Inherited Members

5.21.1 Detailed Description

This exception is thrown when a socket bind error occurs.

Definition at line 50 of file CtNetworkExceptions.hpp.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 CtSocketBindError()

```
CtSocketBindError::CtSocketBindError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 52 of file CtNetworkExceptions.hpp.

The documentation for this class was generated from the following file:

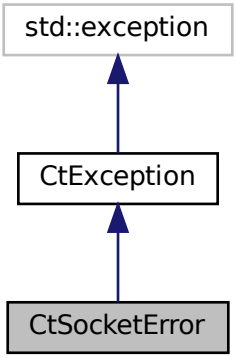
- include/exceptions/CtNetworkExceptions.hpp

5.22 CtSocketError Class Reference

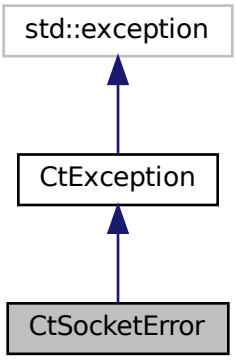
This exception is thrown when a socket error occurs.

`#include <CtNetworkExceptions.hpp>`

Inheritance diagram for CtSocketError:



Collaboration diagram for CtSocketError:



Public Member Functions

- CtSocketError (const std::string &msg)

Additional Inherited Members

5.22.1 Detailed Description

This exception is thrown when a socket error occurs.

Definition at line 41 of file CtNetworkExceptions.hpp.

5.22.2 Constructor & Destructor Documentation

5.22.2.1 CtSocketError()

```
CtSocketError::CtSocketError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 43 of file CtNetworkExceptions.hpp.

The documentation for this class was generated from the following file:

- include/exceptions/CtNetworkExceptions.hpp

5.23 CtSocketHelpers Class Reference

A class contaning helpers for various sockets utilities.

```
#include <CtSocketHelpers.hpp>
```

Static Public Member Functions

- static EXPORTED_API void setSocketTimeout (int32_t socketTimeout)
  Set the Socket Timeout object.
- static EXPORTED_API std::vector< std::string > getInterfaces ()
  Get all available interfaces the device.
- static EXPORTED_API std::string interfaceToAddress (const std::string &p_ifName)
  Get address of a specific interface.
- static EXPORTED_API CtUInt32 getAddressAsUInt (const std::string &p_addr)
  Convert address to uin32_t.
- static EXPORTED_API std::string getAddressAsString (CtUInt32 p_addr)
  Convert address to std::string.

Static Private Member Functions

- static void setConnectionTimeout (timeval &timeout, CtUInt32 timeout_ms)
  Set the Connection Timeout object.

Static Private Attributes

- static int32_t socketTimeout = 0

Friends

- class CtSocketUdp

5.23.1 Detailed Description

A class contaning helpers for various sockets utilities.

Definition at line 49 of file CtSocketHelpers.hpp.

5.23.2 Member Function Documentation

5.23.2.1 getAddressAsString()

```
std::string CtSocketHelpers::getAddressAsString (
            CtUInt32 p_addr )  [static]
```

Convert address to std::string.

Parameters

p_addr	The address in form of CtUInt32

Definition at line 101 of file CtSocketHelpers.cpp.

5.23.2.2 getAddressAsUInt()

```
CtUInt32 CtSocketHelpers::getAddressAsUInt (
            const std::string & p_addr )  [static]
```

Convert address to uin32_t.

Parameters

p_addr	The address in form of std::string

Definition at line 91 of file CtSocketHelpers.cpp.

5.23.2.3 getInterfaces()

```
std::vector< std::string > CtSocketHelpers::getInterfaces ( )  [static]
```

Get all available interfaces the device.

Definition at line 44 of file CtSocketHelpers.cpp.

5.23.2.4 interfaceToAddress()

```
std::string CtSocketHelpers::interfaceToAddress (
            const std::string & p_ifName )  [static]
```

Get address of a specific interface.

Parameters

p_ifName	The name of the interface.

Definition at line 58 of file CtSocketHelpers.cpp.

5.23.2.5 setConnectionTimeout()

```
void CtSocketHelpers::setConnectionTimeout (
            timeval & timeout,
            CtUInt32 timeout_ms )  [static], [private]
```

Set the Connection Timeout object.

Parameters

timeout	
timeout_ms	

Definition at line 111 of file CtSocketHelpers.cpp.

5.23.2.6 setSocketTimeout()

```
void CtSocketHelpers::setSocketTimeout (
            int32_t socketTimeout )  [static]
```

Set the Socket Timeout object.

Parameters

socketTimeout	The target timeout for the poll request.

Definition at line 40 of file CtSocketHelpers.cpp.

5.23.3 Friends And Related Function Documentation

5.23.3.1 CtSocketUdp

```
friend class CtSocketUdp  [friend]
```

Definition at line 95 of file CtSocketHelpers.hpp.

5.23.4 Member Data Documentation

5.23.4.1 socketTimeout

```
int32_t CtSocketHelpers::socketTimeout = 0  [static], [private]
```

The timeout value for socket poll operations.

Definition at line 85 of file CtSocketHelpers.hpp.

The documentation for this class was generated from the following files:

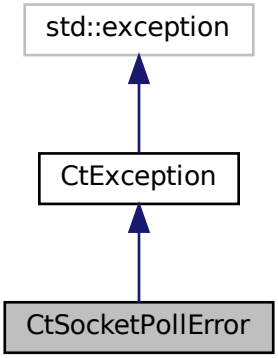
- include/networking/sockets/CtSocketHelpers.hpp
- src/networking/sockets/CtSocketHelpers.cpp

5.24 CtSocketPollError Class Reference

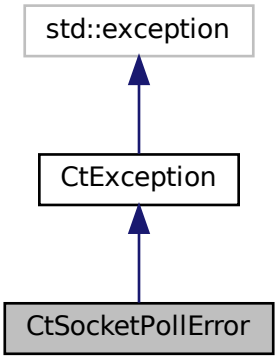
This exception is thrown when a socket listen error occurs.

`#include <CtNetworkExceptions.hpp>`

Inheritance diagram for CtSocketPollError:



Collaboration diagram for CtSocketPollError:



Public Member Functions

- CtSocketPollError (const std::string &msg)

Additional Inherited Members

5.24.1 Detailed Description

This exception is thrown when a socket listen error occurs.

Definition at line 59 of file CtNetworkExceptions.hpp.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 CtSocketPollError()

```
CtSocketPollError::CtSocketPollError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 61 of file CtNetworkExceptions.hpp.

The documentation for this class was generated from the following file:

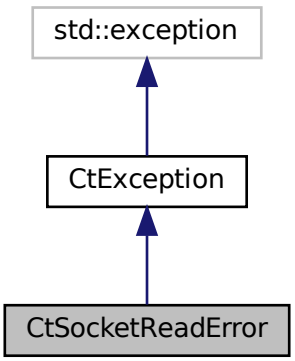
- include/exceptions/CtNetworkExceptions.hpp

5.25 CtSocketReadError Class Reference

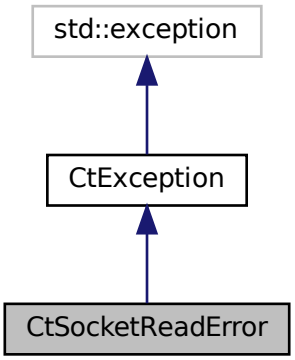
This exception is thrown when a socket accept error occurs.

```
#include <CtNetworkExceptions.hpp>
```

Inheritance diagram for CtSocketReadError:



Collaboration diagram for CtSocketReadError:



Public Member Functions

- CtSocketReadError (const std::string &msg)

Additional Inherited Members

5.25.1 Detailed Description

This exception is thrown when a socket accept error occurs.

Definition at line 68 of file CtNetworkExceptions.hpp.

5.25.2 Constructor & Destructor Documentation

5.25.2.1 CtSocketReadError()

```
CtSocketReadError::CtSocketReadError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 70 of file CtNetworkExceptions.hpp.

The documentation for this class was generated from the following file:

- include/exceptions/CtNetworkExceptions.hpp

5.26 CtSocketUdp Class Reference

A class representing a UDP socket wrapper.

`#include <CtSocketUdp.hpp>`

Public Member Functions

- EXPORTED_API CtSocketUdp ()
  Constructor for CtSocketUdp.
- EXPORTED_API ~CtSocketUdp ()
  Destructor for CtSocketUdp.
- EXPORTED_API void setSub (const std::string &p_interfaceName, uint16_t p_port)
  Set the socket for subscribing.
- EXPORTED_API void setPub (uint16_t p_port, const std::string &p_addr="0.0.0.0")
  Set the socket for publishing.
- EXPORTED_API bool pollRead ()
  Check if there is data available to read.
- EXPORTED_API bool pollWrite ()
  Check if data can be written to the fd.
- EXPORTED_API void send (uint8_t *p_data, CtUInt32 p_size)
  Send data over the socket.
- EXPORTED_API void send (CtRawData &p_message)
  Send data over the socket.
- EXPORTED_API void receive (uint8_t *p_data, CtUInt32 p_size, CtNetAddress *p_client=nullptr)
  Receive data from the socket.
- EXPORTED_API void receive (CtRawData *p_message, CtNetAddress *p_clientAddress=nullptr)
  Receive data from the socket.

Private Attributes

- int m_addrType
- int m_socket
- uint16_t m_port
- std::string m_addr
- struct pollfd m_pollin_sockets [1]
- struct pollfd m_pollout_sockets [1]
- sockaddr_in m_pubAddress
- sockaddr_in m_subAddress

5.26.1 Detailed Description

A class representing a UDP socket wrapper.

This class provides an interface for creating and managing UDP sockets. It can be used for both subscribing and publishing data. CtSocketHelpers::socketTimeout can be used to set the timeout for polling operations. By default, the timeout is set to 0 which means that the poll operation will return immediately. If the timeout is set to -1, the poll operation will block indefinitely. If set to a positive value, the poll operation will block for that amount of time.

Example subscriber:

```
// create a UDP socket
CtSocketUdp socket;
// set the socket for subscribing
socket.setSub("lo", 1234);
// run a loop to receive messages
while (true) {
     if (socket.pollRead()) {
         CtRawData message;
         socket.receive(&message);
         std::cout « "Received message: " « message.get() « std::endl;
     }
}
```

Example publisher:

```
// create a UDP socket
CtSocketUdp socket;
// set the socket for publishing
socket.setPub(1234, "127.0.0.1");
// send a message
CtRawData message("Hello, World!");
socket.send(message);
```

Definition at line 84 of file CtSocketUdp.hpp.

5.26.2 Constructor & Destructor Documentation

5.26.2.1 CtSocketUdp()

```
CtSocketUdp::CtSocketUdp ( )
```

Constructor for CtSocketUdp.

Definition at line 41 of file CtSocketUdp.cpp.

5.26.2.2 ~CtSocketUdp()

```
CtSocketUdp::~CtSocketUdp ( )
```

Destructor for CtSocketUdp.

Definition at line 55 of file CtSocketUdp.cpp.

5.26.3 Member Function Documentation

5.26.3.1 pollRead()

```
bool CtSocketUdp::pollRead ( )
```

Check if there is data available to read.

Returns

True if data is available, false otherwise.

Definition at line 77 of file CtSocketUdp.cpp.

5.26.3.2 pollWrite()

```
bool CtSocketUdp::pollWrite ( )
```

Check if data can be written to the fd.

Returns

True if there is at least one byte available, false otherwise.

Definition at line 89 of file CtSocketUdp.cpp.

5.26.3.3 receive() [1/2]

```
void CtSocketUdp::receive (
            CtRawData * p_message,
            CtNetAddress * p_clientAddress = nullptr )
```

Receive data from the socket.

Parameters

p_message	Struct to store the message received.
p_clientAddress	Pointer to a CtNetAddress object to store the client's address (output parameter).

Definition at line 128 of file CtSocketUdp.cpp.

5.26.3.4 receive() [2/2]

```
void CtSocketUdp::receive (
            uint8_t * p_data,
            CtUInt32 p_size,
            CtNetAddress * p_client = nullptr )
```

Receive data from the socket.

Parameters

p_data	Buffer containing the data to sent.
p_size	Size of the buffer.
p_client	Pointer to a CtNetAddress object to store the client's address (output parameter).

Definition at line 111 of file CtSocketUdp.cpp.

5.26.3.5 send() [1/2]

```
void CtSocketUdp::send (
            CtRawData & p_message )
```

Send data over the socket.

Parameters

p_message	Struct containing the message to sent.

Definition at line 107 of file CtSocketUdp.cpp.

5.26.3.6 send() [2/2]

```
void CtSocketUdp::send (
            uint8_t * p_data,
            CtUInt32 p_size )
```

Send data over the socket.

Parameters

p_data	Buffer containing the data to sent.
p_size	Size of the buffer.

Definition at line 101 of file CtSocketUdp.cpp.

5.26.3.7 setPub()

```
void CtSocketUdp::setPub (
            uint16_t p_port,
            const std::string & p_addr = "0.0.0.0" )
```

Set the socket for publishing.

Parameters

p_port	The port to send data to.
p_addr	The address to send data to. Default to empty string.

Definition at line 70 of file CtSocketUdp.cpp.

5.26.3.8 setSub()

```
void CtSocketUdp::setSub (
            const std::string & p_interfaceName,
            uint16_t p_port )
```

Set the socket for subscribing.

Parameters

p_interfaceName	The interface name to bind to.
p_port	The port to bind to.

Definition at line 59 of file CtSocketUdp.cpp.

5.26.4 Member Data Documentation

5.26.4.1 m_addr

```
std::string CtSocketUdp::m_addr  [private]
```

The address associated with the socket.

Definition at line 160 of file CtSocketUdp.hpp.

5.26.4.2 m_addrType

```
int CtSocketUdp::m_addrType  [private]
```

The socket domain (IPv4 or IPv6).

Definition at line 157 of file CtSocketUdp.hpp.

5.26.4.3 m_pollin_sockets

```
struct pollfd CtSocketUdp::m_pollin_sockets[1]  [private]
```

Array for polling-in file descriptors.

Definition at line 160 of file CtSocketUdp.hpp.

5.26.4.4 m_pollout_sockets

```
struct pollfd CtSocketUdp::m_pollout_sockets[1]  [private]
```

Array for polling-out file descriptors.

Definition at line 160 of file CtSocketUdp.hpp.

5.26.4.5 m_port

```
uint16_t CtSocketUdp::m_port  [private]
```

The port associated with the socket.

Definition at line 159 of file CtSocketUdp.hpp.

5.26.4.6 m_pubAddress

```
sockaddr_in CtSocketUdp::m_pubAddress  [private]
```

The address for publishing data.

Definition at line 163 of file CtSocketUdp.hpp.

5.26.4.7 m_socket

```
int CtSocketUdp::m_socket  [private]
```

The socket descriptor.

Definition at line 158 of file CtSocketUdp.hpp.

5.26.4.8 m_subAddress

```
sockaddr_in CtSocketUdp::m_subAddress  [private]
```

The address for subscribing to data.

Definition at line 164 of file CtSocketUdp.hpp.

The documentation for this class was generated from the following files:

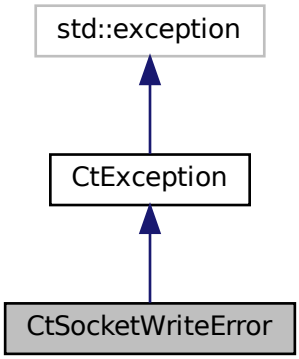
- include/networking/sockets/CtSocketUdp.hpp
- src/networking/sockets/CtSocketUdp.cpp

5.27 CtSocketWriteError Class Reference

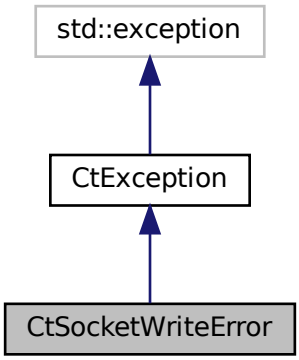
This exception is thrown when a socket connect error occurs.

```
#include <CtNetworkExceptions.hpp>
```

Inheritance diagram for CtSocketWriteError:



Collaboration diagram for CtSocketWriteError:



Public Member Functions

- CtSocketWriteError (const std::string &msg)

Additional Inherited Members

5.27.1 Detailed Description

This exception is thrown when a socket connect error occurs.

Definition at line 77 of file CtNetworkExceptions.hpp.

5.27.2 Constructor & Destructor Documentation

5.27.2.1 CtSocketWriteError()

```
CtSocketWriteError::CtSocketWriteError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 79 of file CtNetworkExceptions.hpp.

The documentation for this class was generated from the following file:

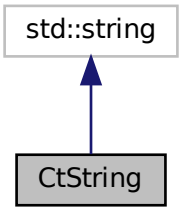
- include/exceptions/CtNetworkExceptions.hpp

5.28 CtString Class Reference

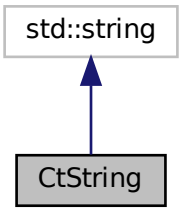
CtString class extends the std::string class. It provides additional methods for string manipulation.

`#include <CtTypes.hpp>`

Inheritance diagram for CtString:



Collaboration diagram for CtString:



Public Member Functions

- CtString (const std::string &str)

  CtString constructor.
- void split (char delimiter, std::vector< CtString > *result) const

  This method splits the string into substrings using the given delimiter.
- CtString trim (const std::string &s) const

  This method trims the string from the left and right side.

5.28.1 Detailed Description

CtString class extends the std::string class. It provides additional methods for string manipulation.

Definition at line 71 of file CtTypes.hpp.

5.28.2 Constructor & Destructor Documentation

5.28.2.1 CtString()

```
CtString::CtString (
            const std::string & str )  [inline], [explicit]
```

CtString constructor.

Parameters

str	The string to be used.

Definition at line 80 of file CtTypes.hpp.

5.28.3 Member Function Documentation

5.28.3.1 split()

```
void CtString::split (
            char delimiter,
            std::vector< CtString > * result ) const  [inline]
```

This method splits the string into substrings using the given delimiter.

Parameters

delimiter	This is the delimiter that will be used to split the string.
result	The vector that will contain the substrings.

Definition at line 88 of file CtTypes.hpp.

5.28.3.2 trim()

```
CtString CtString::trim (
            const std::string & s ) const  [inline]
```

This method trims the string from the left and right side.

Parameters

s	The string to be trimmed.

Returns

CtString The trimmed string.

Definition at line 107 of file CtTypes.hpp.

The documentation for this class was generated from the following file:

- include/CtTypes.hpp

5.29 CtTask Class Reference

Represents a task class that encapsulates a callable function (task) and a callback function.

`#include <CtTask.hpp>`

Public Member Functions

- EXPORTED_API CtTask ()
  Default constructor for CtTask. Initializes task and callback with empty lambda functions.
- EXPORTED_API CtTask (const CtTask &other)
  Copy constructor for CtTask. Copies the task and callback from another CtTask object.
- EXPORTED_API ~CtTask ()
  Destructor for CtTask.
- template<typename F , typename... FArgs>
  EXPORTED_API void setTaskFunc (const F &&func, FArgs &&... fargs)
  Set the main task function. The task function can also have arguments.
- template<typename C , typename... CArgs>
  EXPORTED_API void setCallbackFunc (const C &&callback, CArgs &&... cargs)
  Set the callback function. The callback function can also have arguments.
- EXPORTED_API std::function< void()> getTaskFunc ()
  Get the main task function.
- EXPORTED_API std::function< void()> getCallbackFunc ()
  Get the callback function.
- EXPORTED_API CtTask & operator= (const CtTask &other)
  Assignment operator for CtTask. Copies the task and callback from another CtTask object.
- template<typename F , typename... FArgs>
  void setTaskFunc (const F &&func, FArgs &&... fargs)
- template<typename C , typename... CArgs>
  void setCallbackFunc (const C &&callback, CArgs &&... cargs)

Private Attributes

- std::function< void()> m_task
- std::function< void()> m_callback

5.29.1 Detailed Description

Represents a task class that encapsulates a callable function (task) and a callback function.

The task function is the main function that will be executed. The callback function is the function that will be executed after the task function. The task and callback functions can have arguments. This method can be used to organise a specific functionality and post process of it in a single object.

```
CtTask task;
// Set the task function to a lambda function that prints the sum of two integers.
task.setTaskFunc([](int a, int b) {
  std::cout « "Task function: " « a + b « std::endl;
}, 1, 2);
task.setCallbackFunc([](int a, int b) {
  std::cout « "Callback function: " « a - b « std::endl;
}, 1, 2);
// Set the task function to a function with arguments.
task.setTaskFunc(func, arg1, arg2);
task.setCallbackFunc(callbackFunc, arg1, arg2);
```

Definition at line 62 of file CtTask.hpp.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 CtTask() [1/2]

```
CtTask::CtTask ( )  [explicit]
```

Default constructor for CtTask. Initializes task and callback with empty lambda functions.

Definition at line 34 of file CtTask.cpp.

5.29.2.2 CtTask() [2/2]

```
CtTask::CtTask (
            const CtTask & other )
```

Copy constructor for CtTask. Copies the task and callback from another CtTask object.

Parameters

other	The CtTask object to copy.

Definition at line 37 of file CtTask.cpp.

5.29.2.3 ~CtTask()

```
CtTask::~CtTask ( )
```

Destructor for CtTask.

Definition at line 40 of file CtTask.cpp.

5.29.3 Member Function Documentation

5.29.3.1 getCallbackFunc()

```
std::function< void()> CtTask::getCallbackFunc ( )
```

Get the callback function.

Returns

The callback function.

Definition at line 48 of file CtTask.cpp.

5.29.3.2 getTaskFunc()

```
std::function< void()> CtTask::getTaskFunc ( )
```

Get the main task function.

Returns

The main task function.

Definition at line 44 of file CtTask.cpp.

5.29.3.3 operator=()

```
CtTask & CtTask::operator= (
            const CtTask & other )
```

Assignment operator for CtTask. Copies the task and callback from another CtTask object.

Parameters

other	The CtTask object to copy.

Returns

CtTask& Reference to the current CtTask object.

Definition at line 52 of file CtTask.cpp.

5.29.3.4 setCallbackFunc() [1/2]

```
template<typename C , typename...  CArgs>
EXPORTED_API void CtTask::setCallbackFunc (
            const C && callback,
            CArgs &&...  cargs )
```

Set the callback function. The callback function can also have arguments.

Template Parameters

C	Type of the callable function.
CArgs	Types of the arguments for the callable function.

Parameters

callback	The callable function.
cargs	The arguments for the callable function.

Returns

void

5.29.3.5 setCallbackFunc() [2/2]

```
template<typename C , typename...  CArgs>
void CtTask::setCallbackFunc (
            const C && callback,
            CArgs &&...  cargs )
```

Definition at line 148 of file CtTask.hpp.

5.29.3.6 setTaskFunc() [1/2]

```
template<typename F , typename...  FArgs>
EXPORTED_API void CtTask::setTaskFunc (
            const F && func,
            FArgs &&...  fargs )
```

Set the main task function. The task function can also have arguments.

Template Parameters

F	Type of the callable function.
FArgs	Types of the arguments for the callable function.

Parameters

func	The callable function.
fargs	The arguments for the callable function.

Returns

void

5.29.3.7 setTaskFunc() [2/2]

```
template<typename F , typename...  FArgs>
void CtTask::setTaskFunc (
            const F && func,
            FArgs &&...  fargs )
```

Definition at line 143 of file CtTask.hpp.

5.29.4 Member Data Documentation

5.29.4.1 m_callback

```
std::function<void()> CtTask::m_callback  [private]
```

The callback function

Definition at line 139 of file CtTask.hpp.

5.29.4.2 m_task

```
std::function<void()> CtTask::m_task  [private]
```

The main task function

Definition at line 138 of file CtTask.hpp.

The documentation for this class was generated from the following files:

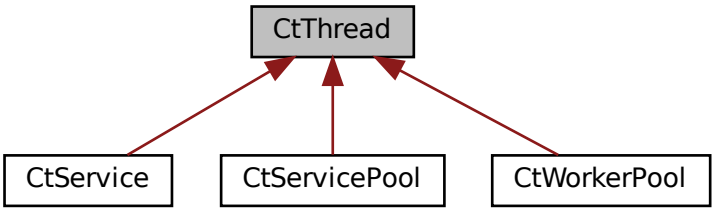
- include/utils/CtTask.hpp
- src/utils/CtTask.cpp

5.30 CtThread Class Reference

A simple C++ thread management class providing basic thread control and sleep functionality.

`#include <CtThread.hpp>`

Inheritance diagram for CtThread:



Static Public Member Functions

- static EXPORTED_API void sleepFor (uint64_t time)
  Make the thread sleep for a specified duration in milliseconds.

Protected Member Functions

- EXPORTED_API CtThread ()
  Constructor for CtThread.
- virtual EXPORTED_API ~CtThread ()
  Virtual destructor for CtThread.
- EXPORTED_API bool isRunning ()
  Check if the thread is currently running.
- EXPORTED_API void start ()
  Start the thread.
- EXPORTED_API void stop ()
  Stop the thread.
- virtual EXPORTED_API void join ()
  Join the thread, waiting for it to finish.
- virtual EXPORTED_API void loop ()=0
  Virtual function to be overridden by derived classes. Represents the main functionality of the thread.
- void setRunning (bool running)
  Set the running state of the thread.

Private Member Functions

- void run ()
  Run method executes main loop of each thread.

Private Attributes

- std::atomic< bool > m_running
- std::thread m_thread

5.30.1 Detailed Description

A simple C++ thread management class providing basic thread control and sleep functionality.

The CtThread class provides a simple interface for creating and managing threads in C++. The class is thread-safe and can be used in multi-threaded environments. It is intended to be used as a base class for creating custom thread classes.

Definition at line 51 of file CtThread.hpp.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 CtThread()

```
CtThread::CtThread ( )  [protected]
```

Constructor for CtThread.

Definition at line 38 of file CtThread.cpp.

5.30.2.2 ~CtThread()

```
CtThread::~CtThread ( )  [protected], [virtual]
```

Virtual destructor for CtThread.

Definition at line 42 of file CtThread.cpp.

5.30.3 Member Function Documentation

5.30.3.1 isRunning()

```
bool CtThread::isRunning ( )  [protected]
```

Check if the thread is currently running.

Returns

True if the thread is running, false otherwise.

Definition at line 73 of file CtThread.cpp.

5.30.3.2 join()

```
void CtThread::join ( )  [protected], [virtual]
```

Join the thread, waiting for it to finish.

Reimplemented in CtWorkerPool.

Definition at line 67 of file CtThread.cpp.

5.30.3.3 loop()

```
virtual EXPORTED_API void CtThread::loop ( )  [protected], [pure virtual]
```

Virtual function to be overridden by derived classes. Represents the main functionality of the thread.

Implemented in CtWorkerPool, CtServicePool, and CtService.

5.30.3.4 run()

```
void CtThread::run ( )  [private]
```

Run method executes main loop of each thread.

Definition at line 46 of file CtThread.cpp.

5.30.3.5 setRunning()

```
void CtThread::setRunning (
            bool running )  [protected]
```

Set the running state of the thread.

Parameters

running	The running state to set.

Definition at line 77 of file CtThread.cpp.

5.30.3.6 sleepFor()

```
void CtThread::sleepFor (
            uint64_t time )  [static]
```

Make the thread sleep for a specified duration in milliseconds.

Parameters

time	Duration to sleep in milliseconds.

Definition at line 81 of file CtThread.cpp.

5.30.3.7 start()

```
void CtThread::start ( )  [protected]
```

Start the thread.

Exceptions

CtThreadError	if the thread is already running.

Definition at line 52 of file CtThread.cpp.

5.30.3.8 stop()

```
void CtThread::stop ( )  [protected]
```

Stop the thread.

Definition at line 62 of file CtThread.cpp.

5.30.4 Member Data Documentation

5.30.4.1 m_running

```
std::atomic<bool> CtThread::m_running  [private]
```

Atomic flag indicating whether the thread is running.

Definition at line 112 of file CtThread.hpp.

5.30.4.2 m_thread

```
std::thread CtThread::m_thread  [private]
```

The underlying thread object.

Definition at line 113 of file CtThread.hpp.

The documentation for this class was generated from the following files:

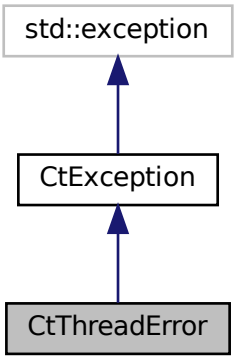
- include/threading/CtThread.hpp
- src/threading/CtThread.cpp

5.31 CtThreadError Class Reference

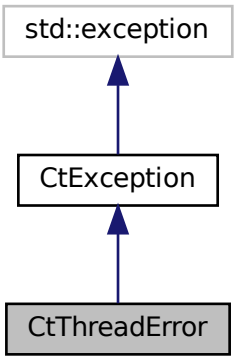
This exception is thrown when a thread error occurs.

```
#include <CtThreadExceptions.hpp>
```

Inheritance diagram for CtThreadError:



Collaboration diagram for CtThreadError:



Public Member Functions

- CtThreadError (const std::string &msg)

Additional Inherited Members

5.31.1 Detailed Description

This exception is thrown when a thread error occurs.

Definition at line 41 of file CtThreadExceptions.hpp.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 CtThreadError()

```
CtThreadError::CtThreadError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 43 of file CtThreadExceptions.hpp.

The documentation for this class was generated from the following file:

- include/exceptions/CtThreadExceptions.hpp

5.32 CtTimer Class Reference

Simple timer utility using std::chrono for high-resolution timing.

`#include <CtTimer.hpp>`

Public Member Functions

- EXPORTED_API CtTimer ()

  Constructor for CtTimer.
- EXPORTED_API ~CtTimer ()

  Destructor for CtTimer.
- EXPORTED_API void tic ()

  Record the current time as a reference point.
- EXPORTED_API uint64_t toc ()

  Measure the elapsed time since the last call to tic().

Static Public Member Functions

- static EXPORTED_API uint64_t current ()

  Get the current time in milliseconds.
- static EXPORTED_API uint64_t millisToNano (uint64_t time)

  Convert time from milliseconds to nanoseconds.

Private Attributes

- uint64_t m_reference

5.32.1 Detailed Description

Simple timer utility using std::chrono for high-resolution timing.

The CtTimer class provides a simple interface for measuring elapsed time.

```
CtTimer timer;
timer.tic();
// Do something
uint64_t elapsed = timer.toc();
std::cout « "Elapsed time: " « elapsed « " ms" « std::endl;
```

Definition at line 56 of file CtTimer.hpp.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 CtTimer()

```
CtTimer::CtTimer ( )
```

Constructor for CtTimer.

Definition at line 34 of file CtTimer.cpp.

5.32.2.2 ~CtTimer()

```
CtTimer::~CtTimer ( )
```

Destructor for CtTimer.

Definition at line 38 of file CtTimer.cpp.

5.32.3 Member Function Documentation

5.32.3.1 current()

```
uint64_t CtTimer::current ( )  [static]
```

Get the current time in milliseconds.

Returns

Current time in milliseconds.

Definition at line 50 of file CtTimer.cpp.

5.32.3.2 millisToNano()

```
static EXPORTED_API uint64_t CtTimer::millisToNano (
            uint64_t time )  [static]
```

Convert time from milliseconds to nanoseconds.

Parameters

time	Time value in milliseconds.

Returns

Time value converted to nanoseconds.

5.32.3.3 tic()

```
void CtTimer::tic ( )
```

Record the current time as a reference point.

Definition at line 42 of file CtTimer.cpp.

5.32.3.4 toc()

```
uint64_t CtTimer::toc ( )
```

Measure the elapsed time since the last call to tic().

Returns

Elapsed time in milliseconds.

Definition at line 46 of file CtTimer.cpp.

5.32.4 Member Data Documentation

5.32.4.1 m_reference

```
uint64_t CtTimer::m_reference  [private]
```

Reference time for measuring elapsed time.

Definition at line 93 of file CtTimer.hpp.

The documentation for this class was generated from the following files:

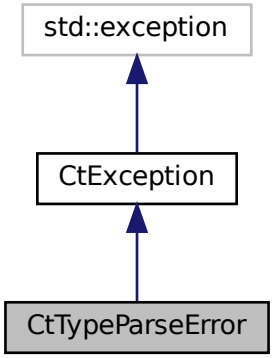
- include/time/CtTimer.hpp
- src/time/CtTimer.cpp

5.33 CtTypeParseError Class Reference

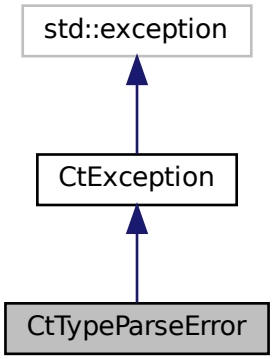
This exception is thrown when a type cannot be parsed.

`#include <CtTypeExceptions.hpp>`

Inheritance diagram for CtTypeParseError:



Collaboration diagram for CtTypeParseError:



Public Member Functions

- CtTypeParseError (const std::string &msg)

Additional Inherited Members

5.33.1 Detailed Description

This exception is thrown when a type cannot be parsed.

Definition at line 41 of file CtTypeExceptions.hpp.

5.33.2 Constructor & Destructor Documentation

5.33.2.1 CtTypeParseError()

```
CtTypeParseError::CtTypeParseError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 43 of file CtTypeExceptions.hpp.

The documentation for this class was generated from the following file:

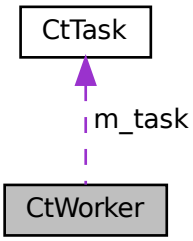
- include/exceptions/CtTypeExceptions.hpp

5.34 CtWorker Class Reference

Represents a worker thread that can execute tasks asynchronously.

```
#include <CtWorker.hpp>
```

Collaboration diagram for CtWorker:



Public Member Functions

- EXPORTED_API CtWorker ()

  Constructor for CtWorker.
- EXPORTED_API ~CtWorker ()

  Destructor for CtWorker.
- EXPORTED_API bool isRunning ()

  Returns true if the worker is currently running.
- EXPORTED_API void runTask ()

  Run the task assigned to the worker.
- EXPORTED_API void joinTask ()

  Join the worker's thread, waiting for the task to complete.
- EXPORTED_API void setTask (const CtTask &task, std::function< void()> callback=[]{})

  Set a task for the worker to execute.
- template<typename F , typename... FArgs>
  EXPORTED_API void setTaskFunc (const F &&func, FArgs &&... fargs)

  Set a task function for the worker to execute.
- template<typename F , typename... FArgs>
  void setTaskFunc (const F &&func, FArgs &&... fargs)

Private Member Functions

- void alreadyRunningCheck ()

  Helper function that checks if the CtWorker is already running and returns an exception.
- void setRunning (bool running)

  Helper function set the m_running flag.

Private Attributes

- CtTask m_task
- std::atomic< bool > m_running
- std::thread m_thread
- std::function< void()> m_callback

5.34.1 Detailed Description

Represents a worker thread that can execute tasks asynchronously.

The CtWorker class provides a mechanism for executing tasks asynchronously in a separate thread. The class is thread-safe and can be used in multi-threaded environments.

```
CtWorker worker;
// add a lambda function to the worker
worker.setTask([](){ std::cout « "Hello from worker thread!" « std::endl; });
// or add a task
worker.setTask(task);
// or add a function with arguments
worker.setTaskFunc(func, arg1, arg2);
// run the task
worker.runTask();
// wait for the task to complete
worker.joinTask();
```

Definition at line 66 of file CtWorker.hpp.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 CtWorker()

```
CtWorker::CtWorker ( )  [explicit]
```

Constructor for CtWorker.

Definition at line 36 of file CtWorker.cpp.

5.34.2.2 ~CtWorker()

```
CtWorker::~CtWorker ( )
```

Destructor for CtWorker.

Definition at line 39 of file CtWorker.cpp.

5.34.3 Member Function Documentation

5.34.3.1 alreadyRunningCheck()

```
void CtWorker::alreadyRunningCheck ( )  [private]
```

Helper function that checks if the CtWorker is already running and returns an exception.

Definition at line 73 of file CtWorker.cpp.

5.34.3.2 isRunning()

```
bool CtWorker::isRunning ( )
```

Returns true if the worker is currently running.

Returns

EXPORTED_API Worker status.

Definition at line 42 of file CtWorker.cpp.

5.34.3.3 joinTask()

```
void CtWorker::joinTask ( )
```

Join the worker's thread, waiting for the task to complete.

Definition at line 67 of file CtWorker.cpp.

5.34.3.4 runTask()

```
void CtWorker::runTask ( )
```

Run the task assigned to the worker.

Definition at line 56 of file CtWorker.cpp.

5.34.3.5 setRunning()

```
void CtWorker::setRunning (
            bool running )  [private]
```

Helper function set the m_running flag.

Definition at line 46 of file CtWorker.cpp.

5.34.3.6 setTask()

```
void CtWorker::setTask (
            const CtTask & task,
            std::function< void()> callback = [](){} )
```

Set a task for the worker to execute.

Parameters

task	The task to be executed by the worker.
callback	The callback function to be executed after the task is completed. Default is an empty lambda function.

Definition at line 50 of file CtWorker.cpp.

5.34.3.7 setTaskFunc() [1/2]

```
template<typename F , typename...  FArgs>
EXPORTED_API void CtWorker::setTaskFunc (
            const F && func,
            FArgs &&...  fargs )
```

Set a task function for the worker to execute.

Parameters

func	The task function to be executed by the worker.
fargs	The arguments of the executed task function.

5.34.3.8 setTaskFunc() [2/2]

```
template<typename F , typename...  FArgs>
void CtWorker::setTaskFunc (
            const F && func,
            FArgs &&...  fargs )
```

Definition at line 131 of file CtWorker.hpp.

5.34.4 Member Data Documentation

5.34.4.1 m_callback

```
std::function<void()> CtWorker::m_callback  [private]
```

Callback function to be executed after the task is completed.

Definition at line 127 of file CtWorker.hpp.

5.34.4.2 m_running

```
std::atomic<bool> CtWorker::m_running  [private]
```

Flag indicating if the worker is currently running.

Definition at line 125 of file CtWorker.hpp.

5.34.4.3 m_task

```
CtTask CtWorker::m_task  [private]
```

The task assigned to the worker.

Definition at line 124 of file CtWorker.hpp.

5.34.4.4 m_thread

```
std::thread CtWorker::m_thread  [private]
```

The worker's thread.

Definition at line 126 of file CtWorker.hpp.

The documentation for this class was generated from the following files:

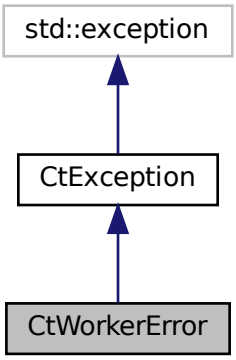
- include/threading/CtWorker.hpp
- src/threading/CtWorker.cpp

5.35 CtWorkerError Class Reference

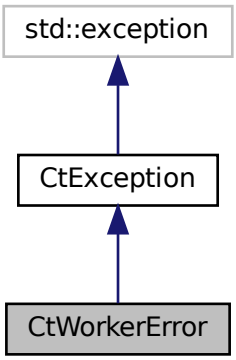
This exception is thrown when a worker error occurs.

```
#include <CtThreadExceptions.hpp>
```

Inheritance diagram for CtWorkerError:



Collaboration diagram for CtWorkerError:



Public Member Functions

- CtWorkerError (const std::string &msg)

Additional Inherited Members

5.35.1 Detailed Description

This exception is thrown when a worker error occurs.

Definition at line 59 of file CtThreadExceptions.hpp.

5.35.2 Constructor & Destructor Documentation

5.35.2.1 CtWorkerError()

```
CtWorkerError::CtWorkerError (
            const std::string & msg )  [inline], [explicit]
```

Definition at line 61 of file CtThreadExceptions.hpp.

The documentation for this class was generated from the following file:

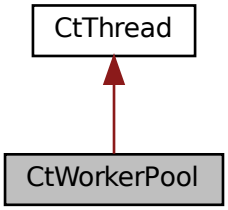
- include/exceptions/CtThreadExceptions.hpp

5.36 CtWorkerPool Class Reference

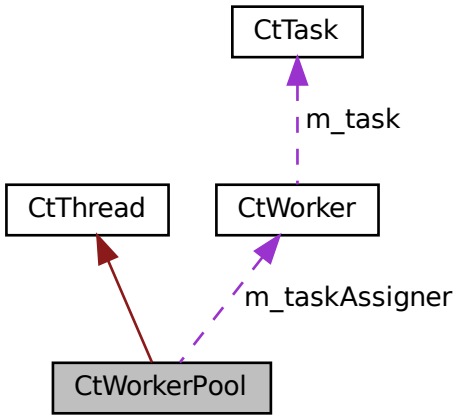
Manages a pool of worker threads for executing tasks concurrently.

`#include <CtWorkerPool.hpp>`

Inheritance diagram for CtWorkerPool:



Collaboration diagram for CtWorkerPool:



Public Member Functions

- EXPORTED_API CtWorkerPool (CtUInt32 nworkers)

  Constructor for CtWorkerPool.
- EXPORTED_API ~CtWorkerPool ()

  Destructor for CtWorkerPool.
- EXPORTED_API void addTask (const CtTask &task)

  Add a task to the worker pool.

- template<typename F , typename... FArgs>
  EXPORTED_API void addTask (const F &&func, FArgs &&... fargs)

  Add a task function to the worker pool.
- EXPORTED_API void join () override

  Wait for all worker threads to finish their tasks.
- template<typename F , typename... FArgs>
  void addTask (const F &&func, FArgs &&... fargs)

Private Member Functions

- void assignTask (CtUInt32 idx)

  Assign a task to a specified worker.
- void free ()

  Free resources and clear the worker pool.
- void loop () override

  Main loop for the worker pool.

Private Attributes

- CtUInt32 m_nworkers
- std::vector< std::unique_ptr< CtWorker > > m_workers
- std::queue< CtTask > m_tasks
- std::queue< CtUInt32 > m_available_workers_idxs
- std::mutex m_mtx_control
- std::atomic< CtUInt32 > m_active_tasks
- std::atomic< CtUInt32 > m_queued_tasks
- CtWorker m_taskAssigner

Additional Inherited Members

5.36.1 Detailed Description

Manages a pool of worker threads for executing tasks concurrently.

The CtWorkerPool class provides a mechanism for managing a pool of worker threads that can execute tasks concurrently. The class is thread-safe and can be used in multi-threaded environments.

```
// create a pool with 4 worker threads
CtWorkerPool pool(4);
// add tasks to the pool
// add a lambda function
pool.addTask([](){ std::cout << "Hello from worker thread!" << std::endl; });
// add a function with arguments
pool.addTask(func, arg1, arg2);
// add a CtTask object
pool.addTask(task);
// wait for all worker threads to finish
pool.join();
```

Definition at line 69 of file CtWorkerPool.hpp.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 CtWorkerPool()

```
CtWorkerPool::CtWorkerPool (
            CtUInt32 nworkers )  [explicit]
```

Constructor for CtWorkerPool.

Parameters

nworkers	The number of worker threads in the pool.

Definition at line 35 of file CtWorkerPool.cpp.

5.36.2.2 ~CtWorkerPool()

```
CtWorkerPool::~CtWorkerPool ( )
```

Destructor for CtWorkerPool.

Definition at line 41 of file CtWorkerPool.cpp.

5.36.3 Member Function Documentation

5.36.3.1 addTask() [1/3]

```
void CtWorkerPool::addTask (
            const CtTask & task )
```

Add a task to the worker pool.

Parameters

task	The task to be added to the pool.

Definition at line 46 of file CtWorkerPool.cpp.

5.36.3.2 addTask() [2/3]

```
template<typename F , typename...  FArgs>
EXPORTED_API void CtWorkerPool::addTask (
            const F && func,
            FArgs &&...  fargs )
```

Add a task function to the worker pool.

Parameters

func	The task function to be added to the pool.
fargs	The arguments of the task function.

5.36.3.3 addTask() [3/3]

```
template<typename F , typename...  FArgs>
void CtWorkerPool::addTask (
            const F && func,
            FArgs &&...  fargs )
```

Definition at line 133 of file CtWorkerPool.hpp.

5.36.3.4 assignTask()

```
void CtWorkerPool::assignTask (
            CtUInt32 idx )  [private]
```

Assign a task to a specified worker.

Parameters

idx	The index of the worker to which the task is assigned.

Returns

True if a task was successfully assigned, false otherwise.

Definition at line 60 of file CtWorkerPool.cpp.

5.36.3.5 free()

```
void CtWorkerPool::free ( )  [private]
```

Free resources and clear the worker pool.

Definition at line 73 of file CtWorkerPool.cpp.

5.36.3.6 join()

```
void CtWorkerPool::join ( )  [override], [virtual]
```

Wait for all worker threads to finish their tasks.

Reimplemented from CtThread.

Definition at line 56 of file CtWorkerPool.cpp.

5.36.3.7 loop()

```
void CtWorkerPool::loop ( )  [override], [private], [virtual]
```

Main loop for the worker pool.

Implements CtThread.

Definition at line 80 of file CtWorkerPool.cpp.

5.36.4 Member Data Documentation

5.36.4.1 m_active_tasks

```
std::atomic<CtUInt32> CtWorkerPool::m_active_tasks  [private]
```

Number of active tasks that are currently running.

Definition at line 127 of file CtWorkerPool.hpp.

5.36.4.2 m_available_workers_idxs

```
std::queue<CtUInt32> CtWorkerPool::m_available_workers_idxs  [private]
```

Queue of available worker indices.

Definition at line 125 of file CtWorkerPool.hpp.

5.36.4.3 m_mtx_control

```
std::mutex CtWorkerPool::m_mtx_control  [private]
```

Mutex for controlling access to shared resources.

Definition at line 126 of file CtWorkerPool.hpp.

5.36.4.4 m_nworkers

```
CtUInt32 CtWorkerPool::m_nworkers  [private]
```

Number of worker threads in the pool.

Definition at line 122 of file CtWorkerPool.hpp.

5.36.4.5 m_queued_tasks

```
std::atomic<CtUInt32> CtWorkerPool::m_queued_tasks  [private]
```

Number of queued tasks.

Definition at line 128 of file CtWorkerPool.hpp.

5.36.4.6 m_taskAssigner

```
CtWorker CtWorkerPool::m_taskAssigner  [private]
```

This worker is a task assigner, assigns active tasks to available workers.

Definition at line 129 of file CtWorkerPool.hpp.

5.36.4.7 m_tasks

```
std::queue<CtTask> CtWorkerPool::m_tasks  [private]
```

Queue of tasks to be executed.

Definition at line 124 of file CtWorkerPool.hpp.

5.36.4.8 m_workers

```
std::vector<std::unique_ptr<CtWorker> > CtWorkerPool::m_workers  [private]
```

Worker thread instances.

Definition at line 123 of file CtWorkerPool.hpp.

The documentation for this class was generated from the following files:

- include/threading/CtWorkerPool.hpp
- src/threading/CtWorkerPool.cpp

Chapter 6

File Documentation

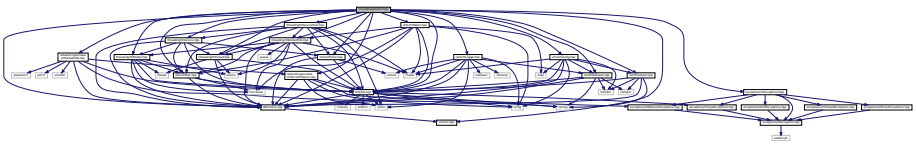
6.1 docs/mainpage.dox File Reference

6.2 include/cpptoolkit.hpp File Reference

Master header file for the C++ Toolkit library.

```
#include "version.hpp"
#include "definitions.hpp"
#include "CtTypes.hpp"
#include "exceptions/CtExceptions.hpp"
#include "io/CtFileInput.hpp"
#include "io/CtFileOutput.hpp"
#include "networking/sockets/CtSocketHelpers.hpp"
#include "networking/sockets/CtSocketUdp.hpp"
#include "threading/CtThread.hpp"
#include "threading/CtWorker.hpp"
#include "threading/CtWorkerPool.hpp"
#include "threading/CtService.hpp"
#include "threading/CtServicePool.hpp"
#include "time/CtTimer.hpp"
#include "utils/CtConfig.hpp"
#include "utils/CtLogger.hpp"
#include "utils/CtObject.hpp"
#include "utils/CtTask.hpp"
```

Include dependency graph for cpptoolkit.hpp:



6.2.1 Detailed Description

Master header file for the C++ Toolkit library.

Date

10-01-2025

Definition in file cpptoolkit.hpp.

6.3 cpptoolkit.hpp

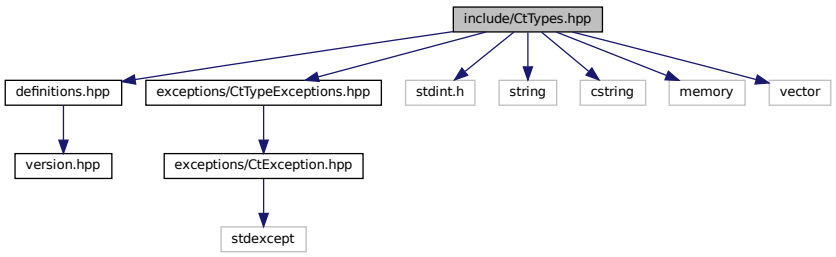
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CPPTOOLKIT_HPP_
00033 #define INCLUDE_CPPTOOLKIT_HPP_
00034
00039 #include "version.hpp"
00040 #include "definitions.hpp"
00041 #include "CtTypes.hpp"
00042 #include "exceptions/CtExceptions.hpp"
00043
00048 #include "io/CtFileInput.hpp"
00049 #include "io/CtFileOutput.hpp"
00050
00055 #include "networking/sockets/CtSocketHelpers.hpp"
00056 #include "networking/sockets/CtSocketUdp.hpp"
00057
00062 #include "threading/CtThread.hpp"
00063 #include "threading/CtWorker.hpp"
00064 #include "threading/CtWorkerPool.hpp"
00065 #include "threading/CtService.hpp"
00066 #include "threading/CtServicePool.hpp"
00067
00072 #include "time/CtTimer.hpp"
00073
00078 #include "utils/CtConfig.hpp"
00079 #include "utils/CtLogger.hpp"
00080 #include "utils/CtObject.hpp"
00081 #include "utils/CtTask.hpp"
00082
00083 #endif //INCLUDE_CPPTOOLKIT_HPP_
```

6.4 include/CtTypes.hpp File Reference

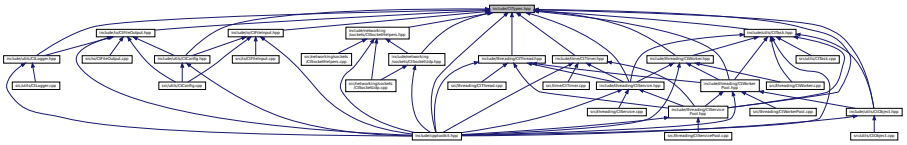
Header file for basic types and classes.

```
#include "definitions.hpp"
#include "exceptions/CtTypeExceptions.hpp"
#include <stdint.h>
#include <string>
#include <cstring>
#include <memory>
#include <vector>
```

Include dependency graph for CtTypes.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtString

  CtString class extends the std::string class. It provides additional methods for string manipulation.
- struct _CtNetAddress

  Struct describing a network address.
- class CtRawData

  Struct describing raw data buffer.

Macros

- #define CtUInt8 uint8_t

  Typedefs for basic types.
- #define CtUInt16 uint16_t
- #define CtUInt32 uint32_t
- #define CtUInt64 uint64_t
- #define CtInt8 int8_t
- #define CtInt16 int16_t
- #define CtInt32 int32_t
- #define CtInt64 int64_t
- #define CtChar char
- #define CT_BUFFER_SIZE 2048

  Default buffer size.

Typedefs

- typedef struct _CtNetAddress CtNetAddress

  Struct describing a network address.

6.4.1 Detailed Description

Header file for basic types and classes.

Date

21-01-2024

Definition in file CtTypes.hpp.

6.4.2 Macro Definition Documentation

6.4.2.1 CT_BUFFER_SIZE

```
#define CT_BUFFER_SIZE 2048
```

Default buffer size.

Definition at line 65 of file CtTypes.hpp.

6.4.2.2 CtChar

```
#define CtChar char
```

Definition at line 59 of file CtTypes.hpp.

6.4.2.3 CtInt16

```
#define CtInt16 int16_t
```

Definition at line 55 of file CtTypes.hpp.

6.4.2.4 CtInt32

```
#define CtInt32 int32_t
```

Definition at line 56 of file CtTypes.hpp.

6.4.2.5 CtInt64

```
#define CtInt64 int64_t
```

Definition at line 57 of file CtTypes.hpp.

6.4.2.6 CtInt8

```
#define CtInt8 int8_t
```

Definition at line 54 of file CtTypes.hpp.

6.4.2.7 CtUInt16

```
#define CtUInt16 uint16_t
```

Definition at line 50 of file CtTypes.hpp.

6.4.2.8 CtUInt32

```
#define CtUInt32 uint32_t
```

Definition at line 51 of file CtTypes.hpp.

6.4.2.9 CtUInt64

```
#define CtUInt64 uint64_t
```

Definition at line 52 of file CtTypes.hpp.

6.4.2.10 CtUInt8

```
#define CtUInt8 uint8_t
```

Typedefs for basic types.

Definition at line 49 of file CtTypes.hpp.

6.4.3 Typedef Documentation

6.4.3.1 CtNetAddress

```
typedef struct _CtNetAddress CtNetAddress
```

Struct describing a network address.

The network address is described by the IP address and the port number.

6.5 CtTypes.hpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTTYPES_HPP_
00033 #define INCLUDE_CTTYPES_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "exceptions/CtTypeExceptions.hpp"
00037
00038 #include <stdint.h>
00039 #include <string>
00040 #include <cstring>
00041 #include <memory>
00042 #include <string>
00043 #include <vector>
00044
00049 #define CtUInt8 uint8_t
00050 #define CtUInt16 uint16_t
00051 #define CtUInt32 uint32_t
00052 #define CtUInt64 uint64_t
00053
00054 #define CtInt8 int8_t
00055 #define CtInt16 int16_t
00056 #define CtInt32 int32_t
00057 #define CtInt64 int64_t
00058
00059 #define CtChar char
00060
00065 #define CT_BUFFER_SIZE 2048
00066
00071 class CtString : public std::string {
00072 public:
00073     using std::string::string;
00074
00080     explicit CtString(const std::string& str) : std::string(str) {}
00081
00088     void split(char delimiter, std::vector<CtString> *result) const {
00089         std::string::size_type start = 0;
00090         auto end = find(delimiter);
00091
```

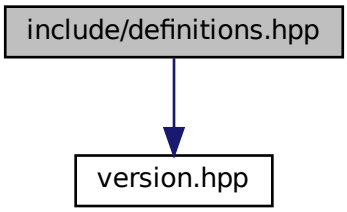
```
00092         while (end != std::string::npos) {
00093             result->push_back((CtString)trim(substr(start, end - start)));
00094             start = end + 1;
00095             end = find(delimiter, start);
00096         }
00097 
00098         result->push_back((CtString)trim(substr(start)));
00099     }
00100 
00107     CtString trim(const std::string& s) const {
00108         auto start = s.find_first_not_of(" \t\n");
00109         auto end = s.find_last_not_of(" \t\n");
00110         return (CtString)((start == std::string::npos) ? "" : s.substr(start, end - start + 1));
00111     }
00112 };
00113 
00120 typedef struct _CtNetAddress {
00121     CtString addr;
00122     CtUInt16 port;
00123 } CtNetAddress;
00124 
00147 class CtRawData {
00148 public:
00149 
00155     EXPORTED_API explicit CtRawData(CtUInt32 p_size = CT_BUFFER_SIZE) : m_maxSize(p_size) {
00156         m_data = new CtUInt8[m_maxSize];
00157         m_size = 0;
00158     };
00159 
00165     EXPORTED_API CtRawData(CtRawData& p_data) : m_maxSize(p_data.maxSize()) {
00166         m_data = new CtUInt8[m_maxSize];
00167         clone(p_data);
00168     };
00169 
00174     EXPORTED_API virtual ~CtRawData() {
00175         delete[] m_data;
00176     }
00177 
00185     EXPORTED_API void nextByte(char byte) {
00186         if (m_size < m_maxSize) {
00187             m_data[m_size++] = byte;
00188         }
00189     }
00190 
00198     EXPORTED_API CtUInt8* getNLastBytes(CtUInt32 p_num) {
00199         if (p_num > m_size) {
00200             throw CtOutOfRangeError("Data size is out of range.");
00201         }
00202         return &m_data[m_size - p_num];
00203     }
00204 
00212     EXPORTED_API void removeNLastBytes(CtUInt32 p_num) {
00213         if (p_num > m_size) {
00214             throw CtOutOfRangeError("Data size is out of range.");
00215         }
00216         m_size -= p_num;
00217     }
00218 
00224     EXPORTED_API CtUInt32 size() {
00225         return m_size;
00226     }
00227 
00233     EXPORTED_API CtUInt32 maxSize() {
00234         return m_maxSize;
00235     }
00236 
00242     EXPORTED_API CtUInt8* get() {
00243         return m_data;
00244     }
00245 
00255     EXPORTED_API void clone(const CtUInt8* p_data, CtUInt32 p_size) {
00256         if (p_size > m_maxSize) {
00257             throw CtOutOfRangeError("Data size is out of range.");
00258         }
00259         m_size = p_size;
00260         memcpy(m_data, p_data, p_size);
00261     }
00262 
00263 
00272     EXPORTED_API void clone(CtRawData& p_data) {
00273         if (p_data.size() > m_maxSize) {
00274             throw CtOutOfRangeError("Data size is out of range.");
00275         }
00276         m_size = p_data.size();
00277         memcpy(m_data, p_data.get(), p_data.size());
00278     }
00279 
```

```
00286     EXPORTED_API void reset() {
00287         m_size = 0;
00288     }
00289
00298     EXPORTED_API CtRawData& operator=(CtRawData& other) {
00299         if (this != &other) {
00300             clone(other);
00301         }
00302         return *this;
00303     }
00304
00305 private:
00306     CtUInt8* m_data;
00307     CtUInt32 m_size;
00308     const CtUInt32 m_maxSize;
00309 };
00310
00311 #endif //INCLUDE_CTTYPES_HPP_
```

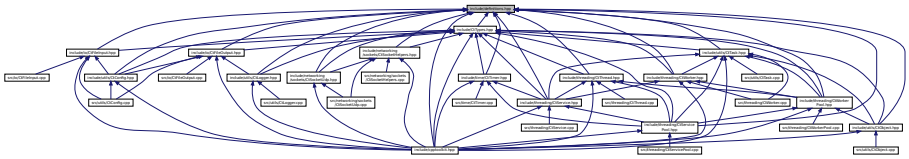
6.6 include/definitions.hpp File Reference

Header file for generic definitions used in teh project.

`#include "version.hpp"`
Include dependency graph for definitions.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define EXPORTED_API __attribute__((visibility("default")))

  EXPORTED_API macro for exporting functions in shared libraries.

6.6.1 Detailed Description

Header file for generic definitions used in teh project.

Date

18-01-2024

Definition in file definitions.hpp.

6.6.2 Macro Definition Documentation

6.6.2.1 EXPORTED_API

```
#define EXPORTED_API __attribute__((visibility("default")))
```

EXPORTED_API macro for exporting functions in shared libraries.

Definition at line 44 of file definitions.hpp.

6.7 definitions.hpp

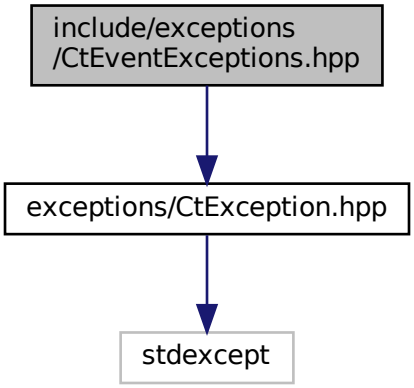
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_DEFINITIONS_HPP_
00033 #define INCLUDE_DEFINITIONS_HPP_
00034
00035 #include "version.hpp"
00036
00041 #ifdef _WIN32
00042     #define EXPORTED_API __declspec(dllexport)
00043 #else
00044     #define EXPORTED_API __attribute__((visibility("default")))
00045 #endif
00046
00047
00048 #endif //INCLUDE_DEFINITIONS_HPP_
```

6.8 include/exceptions/CtEventExceptions.hpp File Reference

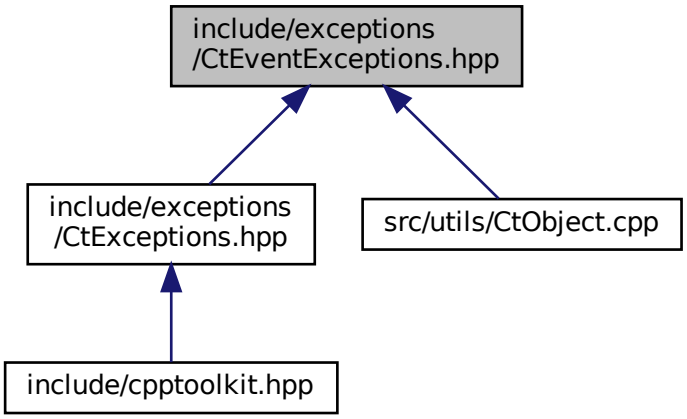
CtEventExceptions header file.

```
#include "exceptions/CtException.hpp"
```

Include dependency graph for CtEventExceptions.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtEventNotExistsError

  This exception is thrown when an event does not exist in the event manager.
- class CtEventAlreadyExistsError

  This exception is thrown when an event already exists in the event manager.

6.8.1 Detailed Description

CtEventExceptions header file.

Date

02-02-2024

Definition in file CtEventExceptions.hpp.

6.9 CtEventExceptions.hpp

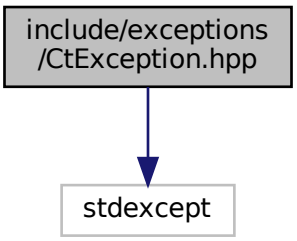
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTEVENTEXCEPTIONS_HPP_
00033 #define INCLUDE_CTEVENTEXCEPTIONS_HPP_
00034
00035 #include "exceptions/CtException.hpp"
00036
00041 class CtEventNotExistsError : public CtException {
00042 public:
00043     explicit CtEventNotExistsError(const std::string& msg): CtException(msg) {};
00044 };
00045
00050 class CtEventAlreadyExistsError : public CtException {
00051 public:
00052     explicit CtEventAlreadyExistsError(const std::string& msg): CtException(msg) {};
00053 };
00054
00055 #endif //INCLUDE_CTEVENTEXCEPTIONS_HPP_
```

6.10 include/exceptions/CtException.hpp File Reference

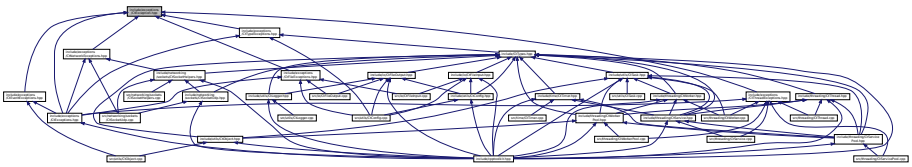
CtException header file.

```
#include <stdexcept>
```

Include dependency graph for CtException.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtException

  An exception class for the cpptoolkit library.

6.10.1 Detailed Description

CtException header file.

Date

18-01-2024

Definition in file CtException.hpp.

6.11 CtException.hpp

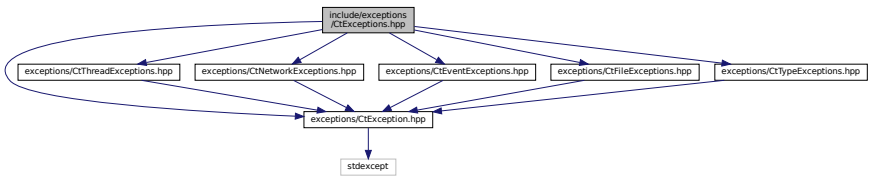
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTEXCEPTION_HPP_
00033 #define INCLUDE_CTEXCEPTION_HPP_
00034
00035 #include <stdexcept>
00036
00045 class CtException : public std::exception {
00046 protected:
00052     explicit CtException(const std::string& msg) : m_msg(msg) {};
00053
00054 public:
00060     const char* what() const noexcept override {
00061         return m_msg.c_str();
00062     };
00063
00064 private:
00065     std::string m_msg;
00066 };
00067
00068 #endif //INCLUDE_CTEXCEPTION_HPP_
```

6.12 include/exceptions/CtExceptions.hpp File Reference

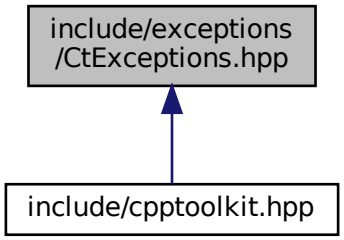
Master header file for the exceptions in the cpptoolkit library.

```
#include "exceptions/CtException.hpp"
#include "exceptions/CtThreadExceptions.hpp"
#include "exceptions/CtNetworkExceptions.hpp"
#include "exceptions/CtEventExceptions.hpp"
#include "exceptions/CtFileExceptions.hpp"
#include "exceptions/CtTypeExceptions.hpp"
```

Include dependency graph for CtExceptions.hpp:



This graph shows which files directly or indirectly include this file:



6.12.1 Detailed Description

Master header file for the exceptions in the cpptoolkit library.

Date

18-01-2024

Definition in file CtExceptions.hpp.

6.13 CtExceptions.hpp

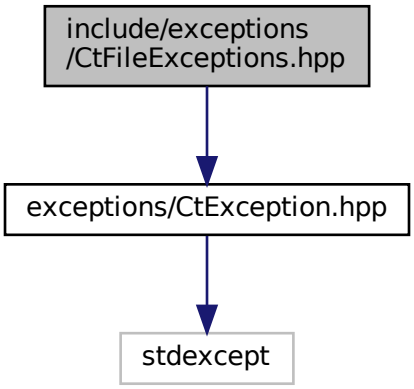
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTEXCEPTIONS_HPP_
00033 #define INCLUDE_CTEXCEPTIONS_HPP_
00034
00035 #include "exceptions/CtException.hpp"
00036 #include "exceptions/CtThreadExceptions.hpp"
00037 #include "exceptions/CtNetworkExceptions.hpp"
00038 #include "exceptions/CtEventExceptions.hpp"
00039 #include "exceptions/CtFileExceptions.hpp"
00040 #include "exceptions/CtTypeExceptions.hpp"
00041
00042 #endif //INCLUDE_CTEXCEPTIONS_HPP_
```

6.14 include/exceptions/CtFileExceptions.hpp File Reference

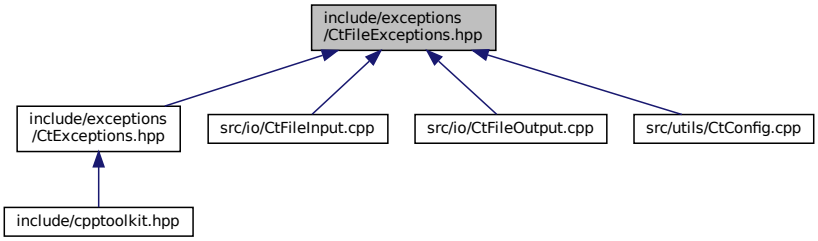
CtFileExceptions header file.

```
#include "exceptions/CtException.hpp"
```

Include dependency graph for CtFileExceptions.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtFileReadError

  This exception is thrown when a file cannot be read.
- class CtFileWriteError

  This exception is thrown when a file cannot be written.
- class CtFileParseError

  This exception is thrown when a file cannot be parsed.

6.14.1 Detailed Description

CtFileExceptions header file.

Date

10-03-2024

Definition in file CtFileExceptions.hpp.

6.15 CtFileExceptions.hpp

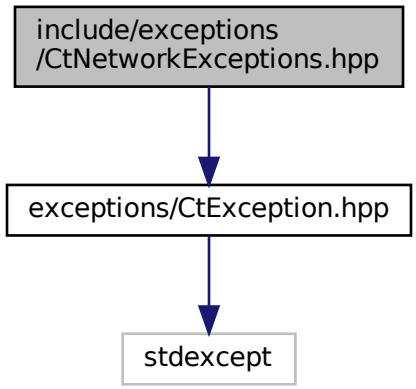
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTFILEEXCEPTIONS_HPP_
00033 #define INCLUDE_CTFILEEXCEPTIONS_HPP_
00034
00035 #include "exceptions/CtException.hpp"
00036
00041 class CtFileReadError : public CtException {
00042 public:
00043     explicit CtFileReadError(const std::string& msg): CtException(msg) {};
00044 };
00045
00050 class CtFileWriteError : public CtException {
00051 public:
00052     explicit CtFileWriteError(const std::string& msg): CtException(msg) {};
00053 };
00054
00059 class CtFileParseError : public CtException {
00060 public:
00061     explicit CtFileParseError(const std::string& msg): CtException(msg) {};
00062 };
00063
00064 #endif //INCLUDE_CTFILEEXCEPTIONS_HPP_
```

6.16 include/exceptions/CtNetworkExceptions.hpp File Reference

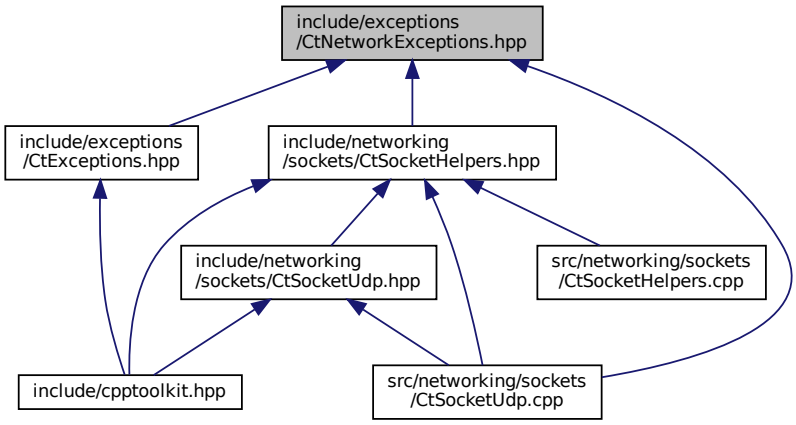
CtNetworkExceptions header file.

```
#include "exceptions/CtException.hpp"
```

Include dependency graph for CtNetworkExceptions.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtSocketError

  This exception is thrown when a socket error occurs.
- class CtSocketBindError

  This exception is thrown when a socket bind error occurs.
- class CtSocketPollError

  This exception is thrown when a socket listen error occurs.
- class CtSocketReadError

  This exception is thrown when a socket accept error occurs.
- class CtSocketWriteError

  This exception is thrown when a socket connect error occurs.

6.16.1 Detailed Description

CtNetworkExceptions header file.

Date

18-01-2024

Definition in file CtNetworkExceptions.hpp.

6.17 CtNetworkExceptions.hpp

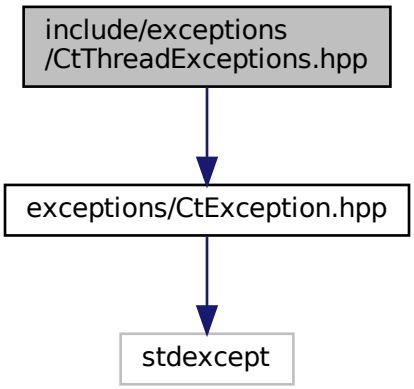
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTNETWORKEXCEPTIONS_HPP_
00033 #define INCLUDE_CTNETWORKEXCEPTIONS_HPP_
00034
00035 #include "exceptions/CtException.hpp"
00036
00041 class CtSocketError : public CtException {
00042 public:
00043     explicit CtSocketError(const std::string& msg): CtException(msg) {};
00044 };
00045
00050 class CtSocketBindError : public CtException {
00051 public:
00052     explicit CtSocketBindError(const std::string& msg): CtException(msg) {};
00053 };
00054
00059 class CtSocketPollError : public CtException {
00060 public:
00061     explicit CtSocketPollError(const std::string& msg): CtException(msg) {};
00062 };
00063
00068 class CtSocketReadError : public CtException {
00069 public:
00070     explicit CtSocketReadError(const std::string& msg): CtException(msg) {};
00071 };
00072
00077 class CtSocketWriteError : public CtException {
00078 public:
00079     explicit CtSocketWriteError(const std::string& msg): CtException(msg) {};
00080 };
00081
00082 #endif //INCLUDE_CTNETWORKEXCEPTIONS_HPP_
```

6.18 include/exceptions/CtThreadExceptions.hpp File Reference

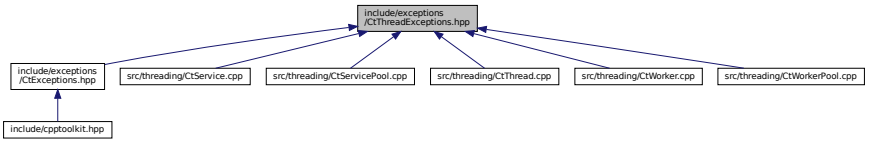
CtThreadExceptions header file.

```
#include "exceptions/CtException.hpp"
```

Include dependency graph for CtThreadExceptions.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtThreadError

  This exception is thrown when a thread error occurs.
- class CtServiceError

  This exception is thrown when a service pool error occurs.
- class CtWorkerError

  This exception is thrown when a worker error occurs.

6.18.1 Detailed Description

CtThreadExceptions header file.

Date

18-01-2024

Definition in file CtThreadExceptions.hpp.

6.19 CtThreadExceptions.hpp

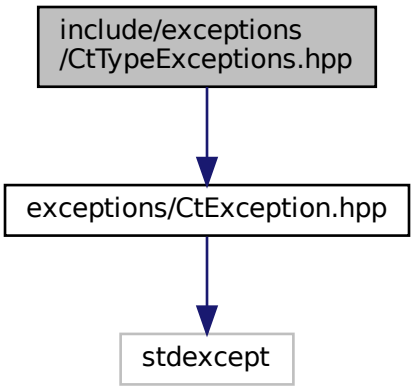
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTTHREADEXCEPTIONS_HPP_
00033 #define INCLUDE_CTTHREADEXCEPTIONS_HPP_
00034
00035 #include "exceptions/CtException.hpp"
00036
00041 class CtThreadError : public CtException {
00042 public:
00043     explicit CtThreadError(const std::string& msg): CtException(msg) {};
00044 };
00045
00050 class CtServiceError : public CtException {
00051 public:
00052     explicit CtServiceError(const std::string& msg): CtException(msg) {};
00053 };
00054
00059 class CtWorkerError : public CtException {
00060 public:
00061     explicit CtWorkerError(const std::string& msg): CtException(msg) {};
00062 };
00063
00064 #endif //INCLUDE_CTTHREADEXCEPTIONS_HPP_
```

6.20 include/exceptions/CtTypeExceptions.hpp File Reference

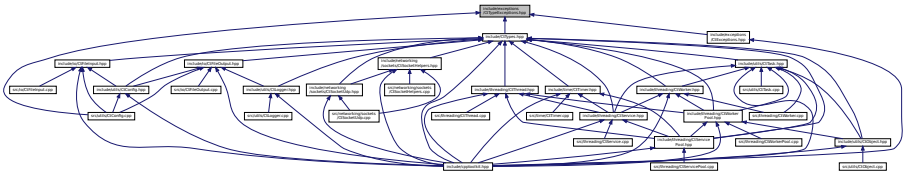
CtTypeExceptions header file.

```
#include "exceptions/CtException.hpp"
```

Include dependency graph for CtTypeExceptions.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtTypeParseError

  This exception is thrown when a type cannot be parsed.
- class CtKeyNotFoundError

  This exception is thrown when a key is not found in a container.
- class CtOutOfRangeError

  This exception is thrown when an index is out of bounds.

6.20.1 Detailed Description

CtTypeExceptions header file.

Date

10-03-2024

Definition in file CtTypeExceptions.hpp.

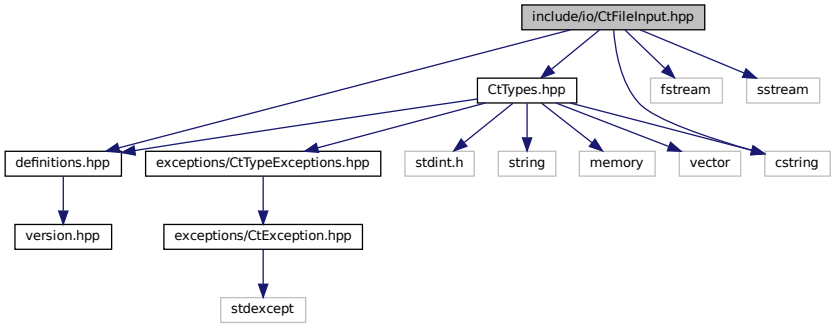
6.21 CtTypeExceptions.hpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTTYPEEXCEPTIONS_HPP_
00033 #define INCLUDE_CTTYPEEXCEPTIONS_HPP_
00034
00035 #include "exceptions/CtException.hpp"
00036
00041 class CtTypeParseError : public CtException {
00042 public:
00043     explicit CtTypeParseError(const std::string& msg): CtException(msg) {};
00044 };
00045
00050 class CtKeyNotFoundError : public CtException {
00051 public:
00052     explicit CtKeyNotFoundError(const std::string& msg): CtException(msg) {};
00053 };
00054
00059 class CtOutOfRangeError : public CtException {
00060 public:
00061     explicit CtOutOfRangeError(const std::string& msg): CtException(msg) {};
00062 };
00063
00064 #endif //INCLUDE_CTTYPEEXCEPTIONS_HPP_
```

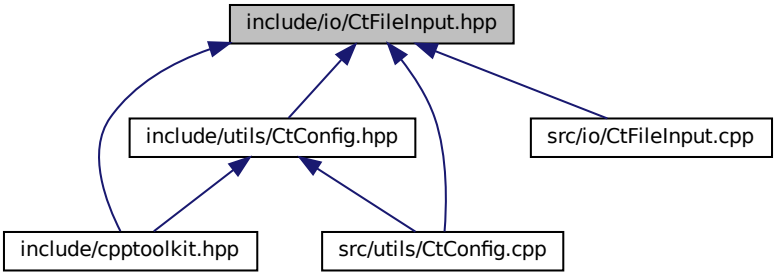
6.22 include/io/CtFileInput.hpp File Reference

```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include <fstream>
#include <sstream>
#include <cstring>
```

Include dependency graph for CtFileInput.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtFileInput

  CtFileInput class for reading data from file.

6.22.1 Detailed Description

Date

08-03-2024

Definition in file CtFileInput.hpp.

6.23 CtFileInput.hpp

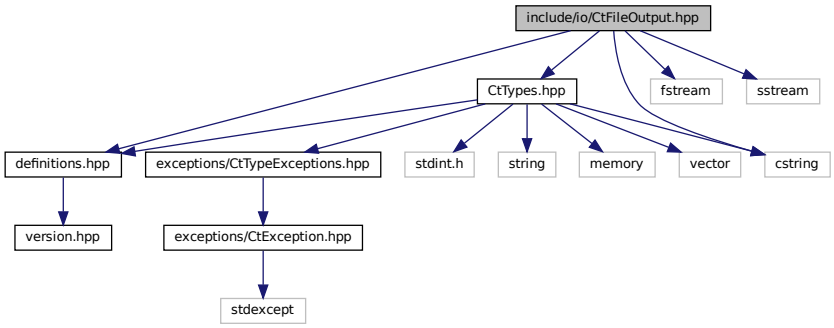
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTFILEINPUT_HPP_
00033 #define INCLUDE_CTFILEINPUT_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037
00038 #include <fstream>
```

```
00039 #include <sstream>
00040 #include <cstring>
00041
00059 class CtFileInput {
00060 public:
00066     EXPORTED_API explicit CtFileInput(const std::string& p_fileName);
00067
00073     EXPORTED_API ~CtFileInput();
00074
00081     EXPORTED_API void setDelimiter(const char* p_delim, CtUInt8 p_delim_size);
00082
00089     EXPORTED_API bool read(CtRawData* p_data);
00090
00091 private:
00092     std::ifstream m_file;
00093     char* m_delim;
00094     CtUInt8 m_delim_size;
00095 };
00096
00097 #endif //INCLUDE_CTFILEINPUT_HPP_
```

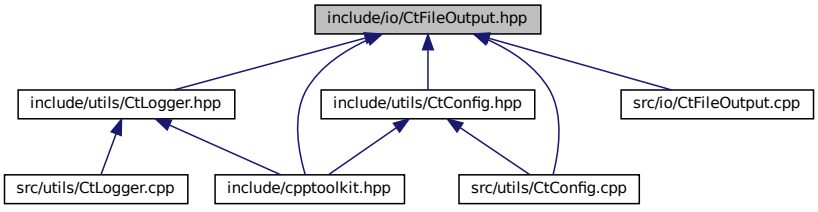
6.24 include/io/CtFileOutput.hpp File Reference

```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include <fstream>
#include <sstream>
#include <cstring>
```

Include dependency graph for CtFileOutput.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtFileOutput

  CtFileOutput class for writing data to file.

6.24.1 Detailed Description

Date

09-03-2024

Definition in file CtFileOutput.hpp.

6.25 CtFileOutput.hpp

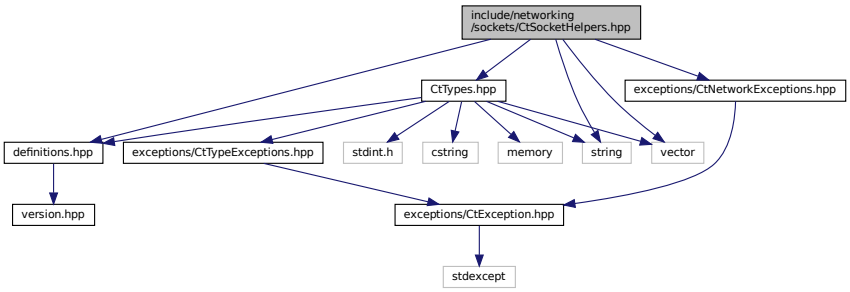
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTFILEOUTPUT_HPP_
00033 #define INCLUDE_CTFILEOUTPUT_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037
00038 #include <fstream>
00039 #include <sstream>
00040 #include <cstring>
00041
00056 class CtFileOutput {
00057 public:
00061     enum class WriteMode { Append, Truncate };
00062
00068     EXPORTED_API explicit CtFileOutput(const std::string& p_fileName, WriteMode p_mode =
      WriteMode::Append);
00069
00075     EXPORTED_API ~CtFileOutput();
00076
00083     EXPORTED_API void setDelimiter(const char* p_delim, CtUInt8 p_delim_size);
00084
00091     EXPORTED_API void write(CtRawData* p_data);
00092
00093 private:
00094     std::ofstream m_file;
00095     std::unique_ptr<char[]> m_delim;
00096     CtUInt8 m_delim_size;
00097 };
00098
00099
00100 #endif //INCLUDE_CTFILEOUTPUT_HPP_
```

6.26 include/networking/sockets/CtSocketHelpers.hpp File Reference

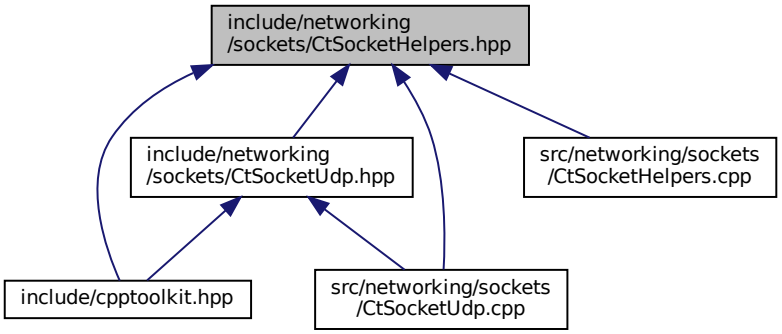
CtSocketHelpers class declaration that contains helpers for various sockets utilities.

```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include "exceptions/CtNetworkExceptions.hpp"
#include <string>
#include <vector>
```

Include dependency graph for CtSocketHelpers.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtSocketHelpers

  A class containing helpers for various sockets utilities.

6.26.1 Detailed Description

CtSocketHelpers class declaration that contains helpers for various sockets utilities.

Date

21-01-2024

Definition in file CtSocketHelpers.hpp.

6.27 CtSocketHelpers.hpp

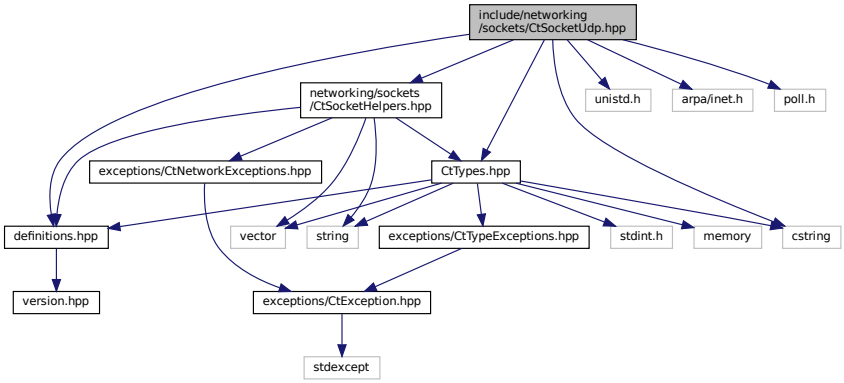
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTSOCKETHELPERS_HPP_
00033 #define INCLUDE_CTSOCKETHELPERS_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037
00038 #include "exceptions/CtNetworkExceptions.hpp"
00039
00040 #include <string>
00041 #include <vector>
00042
00043 class CtSocketUdp;
00044
00049 class CtSocketHelpers {
00050 public:
00056     EXPORTED_API static void setSocketTimeout(int32_t socketTimeout);
00057
00061     EXPORTED_API static std::vector<std::string> getInterfaces();
00062
00068     EXPORTED_API static std::string interfaceToAddress(const std::string& p_ifName);
00069
00075     EXPORTED_API static CtUInt32 getAddressAsUInt(const std::string& p_addr);
00076
00082     EXPORTED_API static std::string getAddressAsString(CtUInt32 p_addr);
00083
00084 private:
00085     static int32_t socketTimeout;
00093     static void setConnectionTimeout(timeval& timeout, CtUInt32 timeout_ms);
00094
00095 friend class CtSocketUdp;
00096 };
00097
00098 #endif //INCLUDE_CTSOCKETHELPERS_HPP_
```

6.28 include/networking/sockets/CtSocketUdp.hpp File Reference

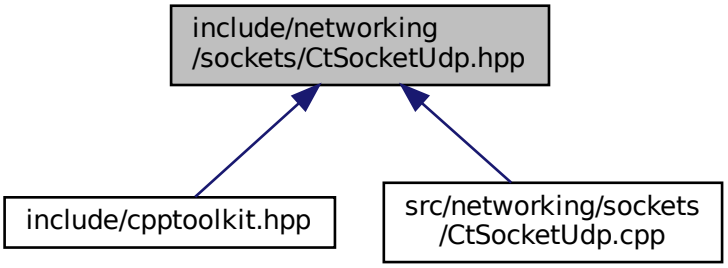
CtSocketUdp class header file.

```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include "networking/sockets/CtSocketHelpers.hpp"
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>
#include <poll.h>
```

Include dependency graph for CtSocketUdp.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtSocketUdp

  A class representing a UDP socket wrapper.

6.28.1 Detailed Description

CtSocketUdp class header file.

Date

18-01-2024

Definition in file CtSocketUdp.hpp.

6.29 CtSocketUdp.hpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTSOCKETUDP_HPP_
00033 #define INCLUDE_CTSOCKETUDP_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037 #include "networking/sockets/CtSocketHelpers.hpp"
00038
00039 #include <cstring>
00040 #include <unistd.h>
00041 #include <arpa/inet.h>
00042 #include <poll.h>
00043
00084 class CtSocketUdp {
00085 public:
00086
00091     EXPORTED_API CtSocketUdp();
00092
00097     EXPORTED_API ~CtSocketUdp();
00098
00104     EXPORTED_API void setSub(const std::string& p_interfaceName, uint16_t p_port);
00105
00111     EXPORTED_API void setPub(uint16_t p_port, const std::string& p_addr = "0.0.0.0");
00112
00118     EXPORTED_API bool pollRead();
00119
00125     EXPORTED_API bool pollWrite();
00126
00132     EXPORTED_API void send(uint8_t* p_data, CtUInt32 p_size);
00133
00138     EXPORTED_API void send(CtRawData& p_message);
00139
00147     EXPORTED_API void receive(uint8_t* p_data, CtUInt32 p_size, CtNetAddress* p_client = nullptr);
00148
00154     EXPORTED_API void receive(CtRawData* p_message, CtNetAddress* p_clientAddress = nullptr);
00155
00156 private:
00157     int m_addrType;
00158     int m_socket;
00159     uint16_t m_port;
00160     std::string m_addr;
00161     struct pollfd m_pollin_sockets[1];
00162     struct pollfd m_pollout_sockets[1];
00163     sockaddr_in m_pubAddress;
00164     sockaddr_in m_subAddress;
00165 };
00166
00167 #endif //INCLUDE_CTSOCKETUDP_HPP_
```

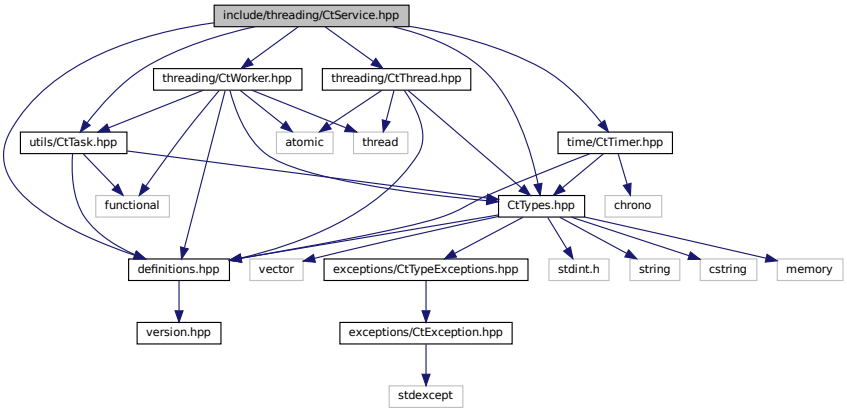
6.30 include/threading/CtService.hpp File Reference

CtService class header file.

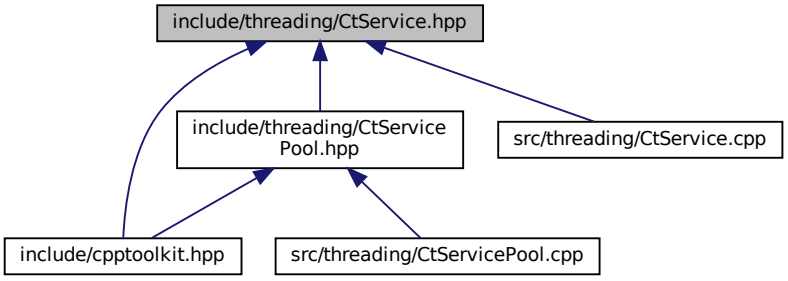
```
#include "definitions.hpp"
#include "CtTypes.hpp"
```

```
#include "threading/CtThread.hpp"
#include "threading/CtWorker.hpp"
#include "utils/CtTask.hpp"
#include "time/CtTimer.hpp"
```

Include dependency graph for CtService.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtService

  A class representing a service that runs a given task at regular intervals using a worker thread.

6.30.1 Detailed Description

CtService class header file.

Date

18-01-2024

Definition in file CtService.hpp.

6.31 CtService.hpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTSERVICE_HPP_
00033 #define INCLUDE_CTSERVICE_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037 #include "threading/CtThread.hpp"
00038 #include "threading/CtWorker.hpp"
00039 #include "utils/CtTask.hpp"
00040 #include "time/CtTimer.hpp"
00041
00060 class CtService : private CtThread {
00061 public:
00067     EXPORTED_API CtService(CtUInt64 nslots, const CtTask& task);
00068
00075     template <typename F, typename... FArgs>
00076     EXPORTED_API CtService(CtUInt64 nslots, const F&& func, FArgs&&... fargs);
00077
00081     EXPORTED_API ~CtService();
00082
00086     EXPORTED_API void runService();
00087
00091     EXPORTED_API void stopService();
00092
00093 public:
00094     static CtUInt32 m_slot_time;
00096 private:
00100     void loop() override;
00101
00102 private:
00103     CtWorker m_worker;
00104     uint64_t m_nslots;
00105 };
00106
00107 template <typename F, typename... FArgs>
00108 CtService::CtService(CtUInt64 nslots, const F&& func, FArgs&&... fargs) : m_nslots(nslots){
00109     CtTask s_task;
00110     s_task.setTaskFunc(std::bind(func, std::forward<FArgs>(fargs)...));
00111     m_worker.setTask(s_task);
00112 };
00113
00114 #endif //INCLUDE_CTSERVICE_HPP_
```

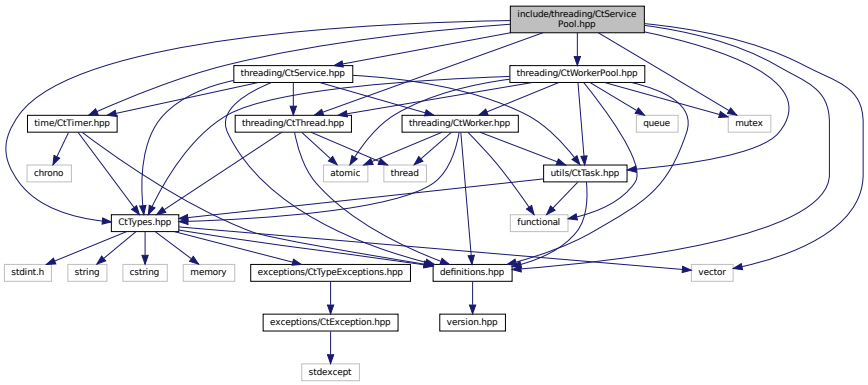
6.32 include/threading/CtServicePool.hpp File Reference

CtServicePool class header file.

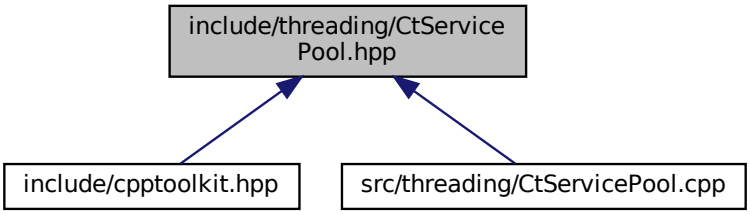
```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include "threading/CtService.hpp"
#include "threading/CtWorkerPool.hpp"
#include "threading/CtThread.hpp"
#include "time/CtTimer.hpp"
```

```
#include "utils/CtTask.hpp"
#include <vector>
#include <mutex>
```

Include dependency graph for CtServicePool.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtServicePool

  A service pool for managing and executing tasks at specified intervals using a worker pool.
- struct CtServicePool::_CtServicePack

6.32.1 Detailed Description

CtServicePool class header file.

Date

18-01-2024

Definition in file CtServicePool.hpp.

6.33 CtServicePool.hpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTSERVICEPOOL_HPP_
00033 #define INCLUDE_CTSERVICEPOOL_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037 #include "threading/CtService.hpp"
00038 #include "threading/CtWorkerPool.hpp"
00039 #include "threading/CtThread.hpp"
00040 #include "time/CtTimer.hpp"
00041 #include "utils/CtTask.hpp"
00042
00043 #include <vector>
00044 #include <mutex>
00045
00070 class CtServicePool : private CtThread {
00071 private:
00076     typedef struct _CtServicePack {
00077         CtTask task;
00078         std::string id;
00079         CtUInt32 nslots;
00080     } CtServicePack;
00081
00082 public:
00087     EXPORTED_API explicit CtServicePool(CtUInt32 nworkers);
00088
00092     EXPORTED_API ~CtServicePool();
00093
00100     EXPORTED_API void addTask(CtUInt32 nslots, const std::string& id, CtTask& task);
00101
00109     template <typename F, typename... FArgs>
00110     EXPORTED_API void addTaskFunc(CtUInt32 nslots, const std::string& id, F&& func, FArgs&&... fargs);
00111
00116     EXPORTED_API void removeTask(const std::string& id);
00117
00121     EXPORTED_API void startServices();
00122
00126     EXPORTED_API void shutdownServices();
00127
00131     EXPORTED_API CtUInt32 getSlotTime();
00132
00136     EXPORTED_API void setSlotTime(CtUInt32 nslots);
00137
00138 private:
00142     void loop() override;
00143
00144 private:
00145     CtUInt32 m_nworkers;
00146     CtUInt32 m_slot_cnt;
00147     std::vector<CtServicePack> m_tasks;
00148     std::mutex m_mtx_control;
00149     CtWorkerPool m_worker_pool;
00150     CtTimer m_timer;
00151     uint64_t m_exec_time;
00152 };
00153
00154 template <typename F, typename... FArgs>
00155 void CtServicePool::addTaskFunc(CtUInt32 nslots, const std::string& id, F&& func, FArgs&&... fargs) {
00156     CtTask s_task;
00157     s_task.setTaskFunc(std::bind(func, std::forward<FArgs>(fargs)...));
00158     addTask(nslots, id, s_task);
00159 };
```

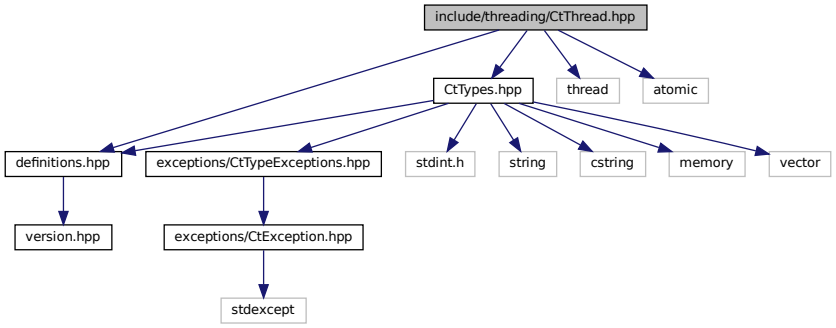
```
00160
00161 #endif //INCLUDE_CTSERVICEPOOL_HPP_
```

6.34 include/threading/CtThread.hpp File Reference

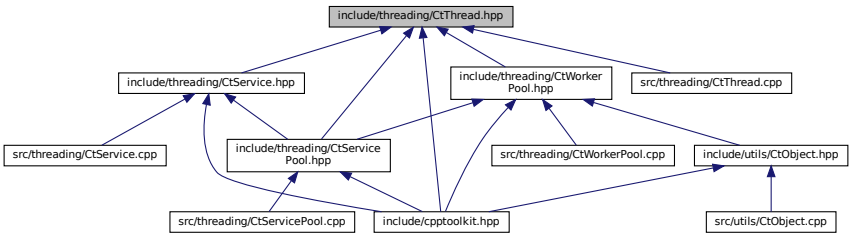
CtThread class header file.

```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include <thread>
#include <atomic>
```

Include dependency graph for CtThread.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtThread

  A simple C++ thread management class providing basic thread control and sleep functionality.

6.34.1 Detailed Description

CtThread class header file.

Date

18-01-2024

Definition in file CtThread.hpp.

6.35 CtThread.hpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTTHREAD_HPP_
00033 #define INCLUDE_CTTHREAD_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037
00038 #include <thread>
00039 #include <atomic>
00040
00051 class CtThread {
00052 public:
00057     EXPORTED_API static void sleepFor(uint64_t time);
00058
00059 protected:
00063     EXPORTED_API CtThread();
00064
00068     EXPORTED_API virtual ~CtThread();
00069
00074     EXPORTED_API bool isRunning();
00075
00080     EXPORTED_API void start();
00081
00085     EXPORTED_API void stop();
00086
00090     EXPORTED_API virtual void join();
00091
00096     EXPORTED_API virtual void loop() = 0;
00097
00098 protected:
00103     void setRunning(bool running);
00104
00105 private:
00109     void run();
00110
00111 private:
00112     std::atomic<bool> m_running;
00113     std::thread m_thread;
00114 };
00115
00116 #endif //INCLUDE_CTTHREAD_HPP_
```

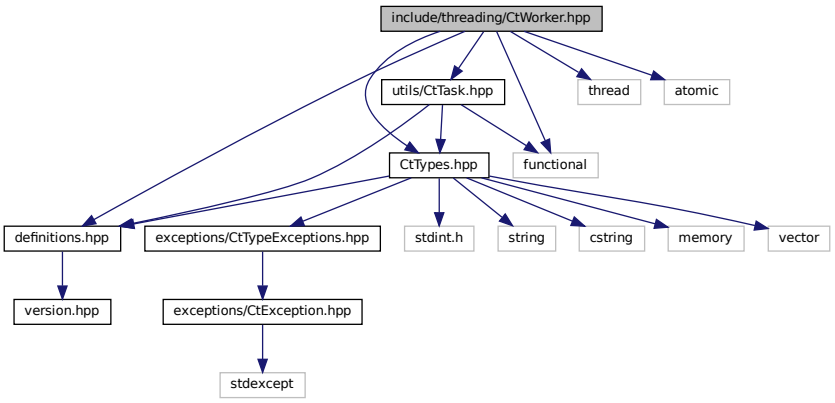
6.36 include/threading/CtWorker.hpp File Reference

CtWorker class header file.

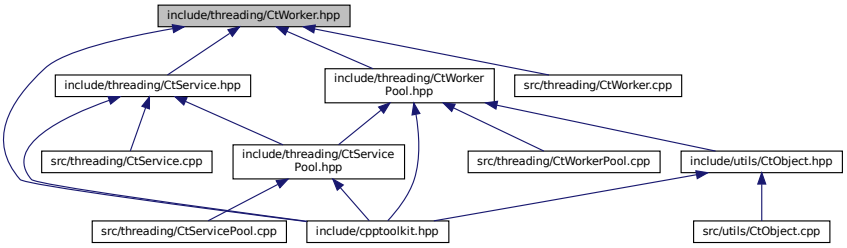
```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include "utils/CtTask.hpp"
#include <thread>
#include <atomic>
```

```
#include <functional>
```

Include dependency graph for CtWorker.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtWorker

  Represents a worker thread that can execute tasks asynchronously.

6.36.1 Detailed Description

CtWorker class header file.

Date

18-01-2024

Definition in file CtWorker.hpp.

6.37 CtWorker.hpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTWORKER_HPP_
00033 #define INCLUDE_CTWORKER_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037 #include "utils/CtTask.hpp"
00038
00039 #include <thread>
00040 #include <atomic>
00041 #include <functional>
00042
00066 class CtWorker {
00067 public:
00071     EXPORTED_API explicit CtWorker();
00072
00076     EXPORTED_API ~CtWorker();
00077
00083     EXPORTED_API bool isRunning();
00084
00088     EXPORTED_API void runTask();
00089
00093     EXPORTED_API void joinTask();
00094
00101     EXPORTED_API void setTask(const CtTask& task, std::function<void()> callback = [](){});
00102
00109     template <typename F, typename... FArgs>
00110     EXPORTED_API void setTaskFunc(const F&& func, FArgs&&... fargs);
00111
00112 private:
00116     void alreadyRunningCheck();
00117
00121     void setRunning(bool running);
00122
00123 private:
00124     CtTask m_task;
00125     std::atomic<bool> m_running;
00126     std::thread m_thread;
00127     std::function<void()> m_callback;
00128 };
00129
00130 template <typename F, typename... FArgs>
00131 void CtWorker::setTaskFunc(const F&& func, FArgs&&... fargs) {
00132     CtTask s_task;
00133     s_task.setTaskFunc(std::bind(func, std::forward<FArgs>(fargs)...));
00134     setTask(s_task);
00135 };
00136
00137 #endif //INCLUDE_CTWORKER_HPP_
```

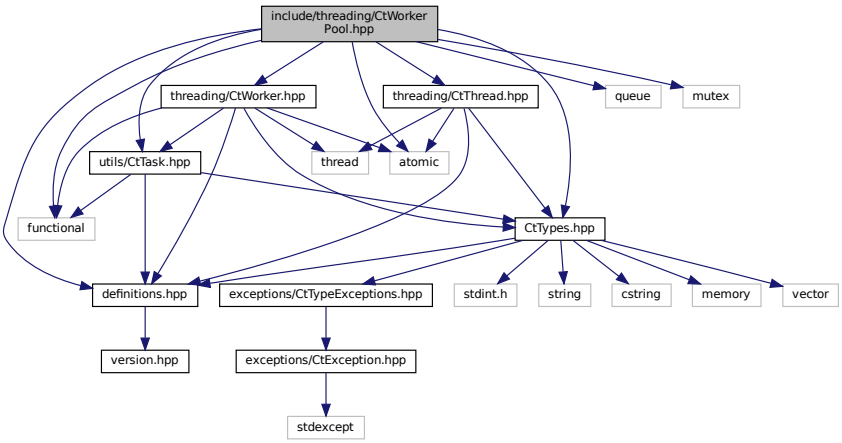
6.38 include/threading/CtWorkerPool.hpp File Reference

CtWorkerPool class header file.

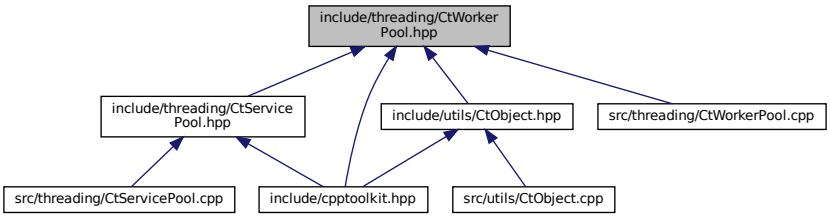
```
#include "definitions.hpp"
#include "CtTypes.hpp"
```

```
#include "threading/CtWorker.hpp"
#include "threading/CtThread.hpp"
#include "utils/CtTask.hpp"
#include <queue>
#include <atomic>
#include <mutex>
#include <functional>
```

Include dependency graph for CtWorkerPool.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtWorkerPool

  Manages a pool of worker threads for executing tasks concurrently.

6.38.1 Detailed Description

CtWorkerPool class header file.

Date

18-01-2024

Definition in file CtWorkerPool.hpp.

6.39 CtWorkerPool.hpp

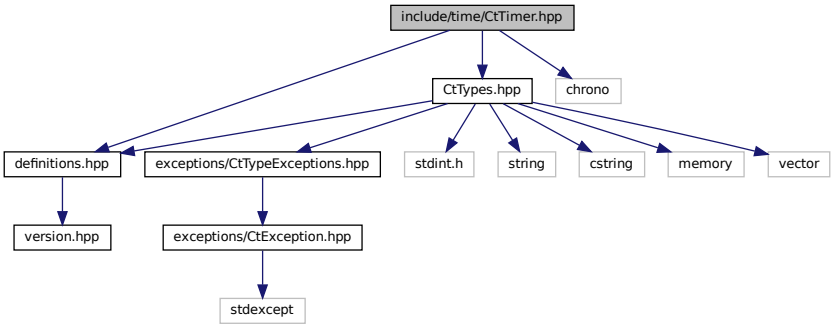
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTWORKERPOOL_HPP_
00033 #define INCLUDE_CTWORKERPOOL_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037 #include "threading/CtWorker.hpp"
00038 #include "threading/CtThread.hpp"
00039 #include "utils/CtTask.hpp"
00040
00041 #include <queue>
00042 #include <atomic>
00043 #include <mutex>
00044 #include <functional>
00045
00069 class CtWorkerPool : private CtThread {
00070 public:
00075     EXPORTED_API explicit CtWorkerPool(CtUInt32 nworkers);
00076
00080     EXPORTED_API ~CtWorkerPool();
00081
00086     EXPORTED_API void addTask(const CtTask& task);
00087
00094     template <typename F, typename... FArgs>
00095     EXPORTED_API void addTask(const F&& func, FArgs&&... fargs);
00096
00100     EXPORTED_API void join() override;
00101
00102 private:
00103
00109     void assignTask(CtUInt32 idx);
00110
00114     void free();
00115
00119     void loop() override;
00120
00121 private:
00122     CtUInt32 m_nworkers;
00123     std::vector<std::unique_ptr<CtWorker» m_workers;
00124     std::queue<CtTask> m_tasks;
00125     std::queue<CtUInt32> m_available_workers_idxs;
00126     std::mutex m_mtx_control;
00127     std::atomic<CtUInt32> m_active_tasks;
00128     std::atomic<CtUInt32> m_queued_tasks;
00129     CtWorker m_taskAssigner;
00130 };
00131
00132 template <typename F, typename... FArgs>
00133 void CtWorkerPool::addTask(const F&& func, FArgs&&... fargs) {
00134     CtTask s_task;
00135     s_task.setTaskFunc(std::bind(func, std::forward<FArgs>(fargs)...));
00136     addTask(s_task);
00137 };
00138
00139 #endif //INCLUDE_CTWORKERPOOL_HPP_
```

6.40 include/time/CtTimer.hpp File Reference

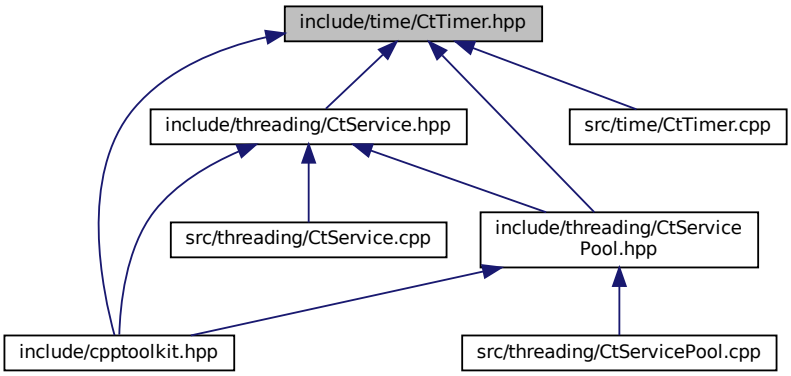
CtTimer class header file.

```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include <chrono>
```

Include dependency graph for CtTimer.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtTimer

  Simple timer utility using std::chrono for high-resolution timing.

6.40.1 Detailed Description

CtTimer class header file.

Date

18-01-2024

Definition in file CtTimer.hpp.

6.41 CtTimer.hpp

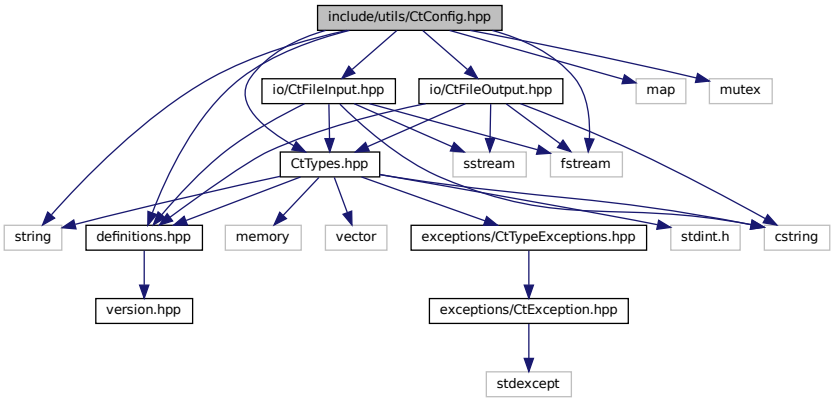
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTTIMER_HPP_
00033 #define INCLUDE_CTTIMER_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037
00038 #include <chrono>
00039
00056 class CtTimer {
00057 public:
00061     EXPORTED_API CtTimer();
00062
00066     EXPORTED_API ~CtTimer();
00067
00071     EXPORTED_API void tic();
00072
00077     EXPORTED_API uint64_t toc();
00078
00083     EXPORTED_API static uint64_t current();
00084
00090     EXPORTED_API static uint64_t millisToNano(uint64_t time);
00091
00092 private:
00093     uint64_t m_reference;
00094 };
00095
00096 #endif //INCLUDE_CTTIMER_HPP_
```

6.42 include/utils/CtConfig.hpp File Reference

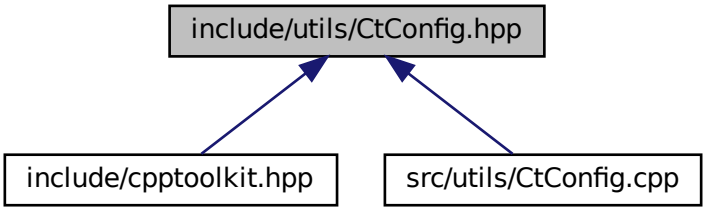
CtConfig class header file.

```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include "io/CtFileOutput.hpp"
#include "io/CtFileInput.hpp"
#include <fstream>
#include <string>
#include <map>
#include <mutex>
```

Include dependency graph for CtConfig.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtConfig

  A configuration file parser class for extracting various data types from configuration values.

6.42.1 Detailed Description

CtConfig class header file.

Date

10-03-2024

Definition in file CtConfig.hpp.

6.43 CtConfig.hpp

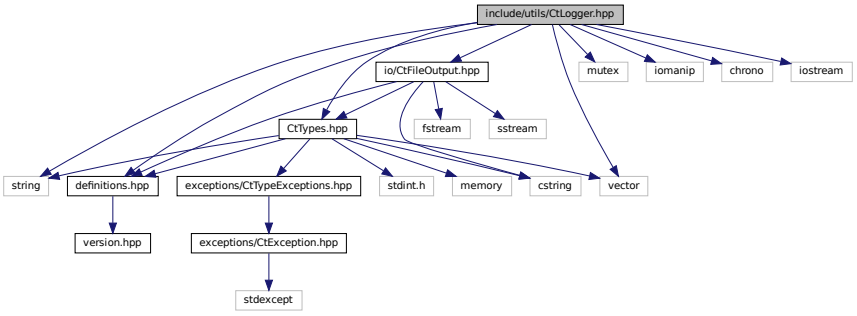
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTCONFIG_HPP_
00033 #define INCLUDE_CTCONFIG_HPP_
00034
00035
00036 #include "definitions.hpp"
00037 #include "CtTypes.hpp"
00038
00039 #include "io/CtFileOutput.hpp"
00040 #include "io/CtFileInput.hpp"
00041
00042 #include <fstream>
00043 #include <string>
00044 #include <map>
00045 #include <mutex>
00046
00056 class CtConfig {
00057 public:
00062     EXPORTED_API explicit CtConfig(const std::string& p_configFile);
00063
00067     EXPORTED_API ~CtConfig();
00068
00074     EXPORTED_API void read();
00075
00079     EXPORTED_API void write();
00080
00089     EXPORTED_API int32_t parseAsInt(const std::string& p_key);
00090
00099     EXPORTED_API uint32_t parseAsUInt(const std::string& p_key);
00100
00109     EXPORTED_API float parseAsFloat(const std::string& p_key);
00110
00119     EXPORTED_API double parseAsDouble(const std::string& p_key);
00120
00128     EXPORTED_API std::string parseAsString(const std::string& p_key);
00129
00136     EXPORTED_API void writeInt(const std::string& p_key, const int32_t& p_value);
00137
00144     EXPORTED_API void writeUInt(const std::string& p_key, const uint32_t& p_value);
00145
00152     EXPORTED_API void writeFloat(const std::string& p_key, const float& p_value);
00153
00160     EXPORTED_API void writeDouble(const std::string& p_key, const double& p_value);
00161
00168     EXPORTED_API void writeString(const std::string& p_key, const std::string& p_value);
00169
00170 private:
00178     std::string getValue(const std::string& p_key);
00179
00187     void parseLine(const std::string& p_line);
00188
00189 private:
00190     std::mutex m_mtx_control;
00191     CtFileInput* m_source;
00192     CtFileOutput* m_sink;
00193     std::string m_configFile;
00194     std::map<std::string, std::string> m_configValues;
00195 };
00196
00197 #endif //INCLUDE_CTCONFIG_HPP_
```

6.44 include/utils/CtLogger.hpp File Reference

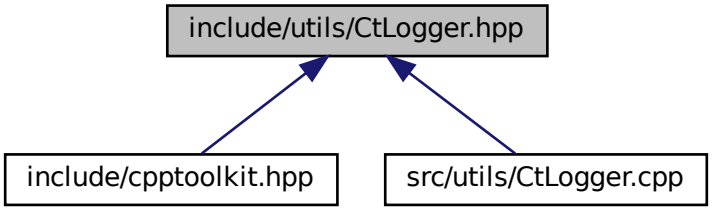
CtLogger class header file.

```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include "io/CtFileOutput.hpp"
#include <mutex>
#include <string>
#include <iomanip>
#include <chrono>
#include <vector>
#include <iostream>
```

Include dependency graph for CtLogger.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtLogger

  A simple logger with log levels and timestamp.

6.44.1 Detailed Description

CtLogger class header file.

Date

10-03-2024

Definition in file CtLogger.hpp.

6.45 CtLogger.hpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTLOGGER_HPP_
00033 #define INCLUDE_CTLOGGER_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037 #include "io/CtFileOutput.hpp"
00038
00039 #include <mutex>
00040 #include <string>
00041 #include <iomanip>
00042 #include <chrono>
00043 #include <vector>
00044 #include <iostream>
00045
00054 class CtLogger {
00055 public:
00059     enum class Level { DEBUG, INFO, WARNING, ERROR, CRITICAL };
00060
00067     EXPORTED_API explicit CtLogger(CtLogger::Level level = CtLogger::Level::DEBUG, const std::string&
      componentName = "");
00068
00072     EXPORTED_API ~CtLogger();
00073
00079     EXPORTED_API void log_debug(const std::string& message);
00080
00086     EXPORTED_API void log_info(const std::string& message);
00087
00093     EXPORTED_API void log_warning(const std::string& message);
00094
00100     EXPORTED_API void log_error(const std::string& message);
00101
00107     EXPORTED_API void log_critical(const std::string& message);
00108
00115     EXPORTED_API static CtLogger::Level stringToLevel(const std::string& level_str);
00116
00117 private:
00124     void log(CtLogger::Level level, const std::string& message);
00125
00132     static const std::string levelToString(CtLogger::Level level);
00133
```

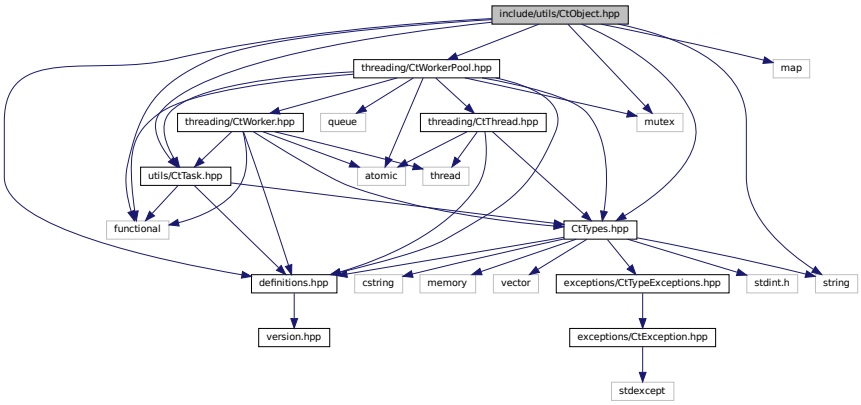
```
00142     static const std::string generateLoggerMsg(CtLogger::Level level, const std::string&
      component_name, const std::string& message);
00143
00144 private:
00145     std::mutex m_mtx_control;
00146     CtLogger::Level m_level;
00147     std::string m_componentName;
00148 };
00149
00150 #endif //INCLUDE_CTLOGGER_HPP_
```

6.46 include/utils/CtObject.hpp File Reference

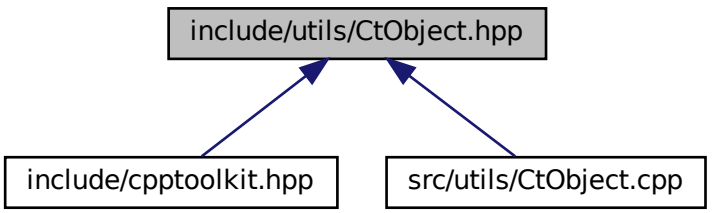
CtObject class header file.

```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include "utils/CtTask.hpp"
#include "threading/CtWorkerPool.hpp"
#include <string>
#include <map>
#include <mutex>
#include <functional>
```

Include dependency graph for CtObject.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtObject

  This abstract class can be used as a base class for objects that can trigger events.

6.46.1 Detailed Description

CtObject class header file.

Date

02-02-2024

Definition in file CtObject.hpp.

6.47 CtObject.hpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTOBJECT_HPP_
00033 #define INCLUDE_CTOBJECT_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037
00038 #include "utils/CtTask.hpp"
00039 #include "threading/CtWorkerPool.hpp"
00040
00041 #include <string>
00042 #include <map>
00043 #include <mutex>
00044 #include <functional>
00045
00061 class CtObject {
00062 public:
00076     template <typename F, typename... FArgs>
00077     EXPORTED_API static void connectEvent(CtObject* p_obj, CtUInt32 p_eventCode, F&& func, FArgs&&...
       fargs);
00078
00088     EXPORTED_API static void connectEvent(CtObject* p_obj, CtUInt32 p_eventCode, CtTask& p_task);
00089
00102     template <typename F, typename... FArgs>
00103     EXPORTED_API void connectEvent(CtUInt32 p_eventCode, F&& func, FArgs&&... fargs);
00104
00113     EXPORTED_API void connectEvent(CtUInt32 p_eventCode, CtTask& p_task);
00114
00121     EXPORTED_API void waitPendingEvents();
00122
00123 protected:
00128     EXPORTED_API CtObject();
```

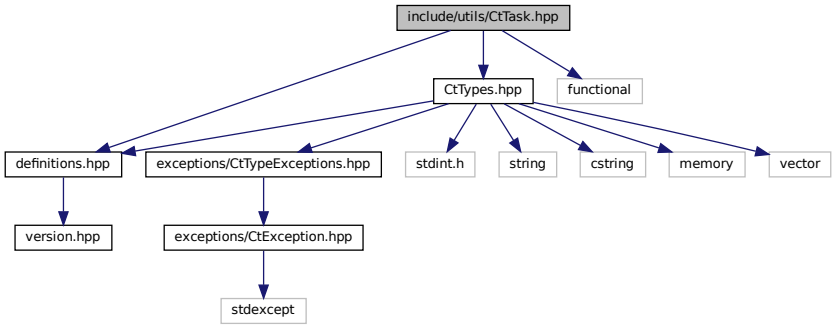
```
00129
00134     EXPORTED_API ~CtObject();
00135
00143     EXPORTED_API void triggerEvent(CtUInt32 p_eventCode);
00144
00152     EXPORTED_API void registerEvent(CtUInt32 p_eventCode);
00153
00154 private:
00162     EXPORTED_API bool hasEvent(CtUInt32 p_eventCode);
00163
00164 private:
00165     std::mutex m_mtx_control;
00166     std::vector<CtUInt32> m_events;
00167     std::multimap<CtUInt32, CtTask> m_triggers;
00168     CtWorkerPool m_pool;
00169 };
00170
00171 template <typename F, typename... FArgs>
00172 void CtObject::connectEvent(CtObject* p_obj, CtUInt32 p_eventCode, F&& func, FArgs&&... fargs) {
00173     CtTask s_task;
00174     s_task.setTaskFunc(std::bind(func, std::forward<FArgs>(fargs)...));
00175     p_obj->connectEvent(p_eventCode, s_task);
00176 };
00177
00178 template <typename F, typename... FArgs>
00179 void CtObject::connectEvent(CtUInt32 p_eventCode, F&& func, FArgs&&... fargs) {
00180     CtTask s_task;
00181     s_task.setTaskFunc(std::bind(func, std::forward<FArgs>(fargs)...));
00182     connectEvent(p_eventCode, s_task);
00183 };
00184
00185 #endif //INCLUDE_CTOBJECT_HPP_
```

6.48 include/utils/CtTask.hpp File Reference

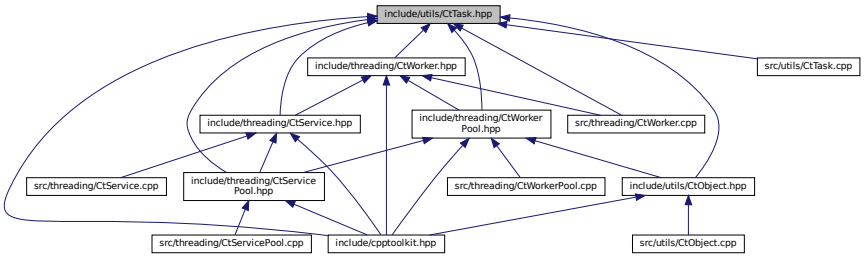
CtTask class header file.

```
#include "definitions.hpp"
#include "CtTypes.hpp"
#include <functional>
```

Include dependency graph for CtTask.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CtTask

  Represents a task class that encapsulates a callable function (task) and a callback function.

6.48.1 Detailed Description

CtTask class header file.

Date

18-01-2024

Definition in file CtTask.hpp.

6.49 CtTask.hpp

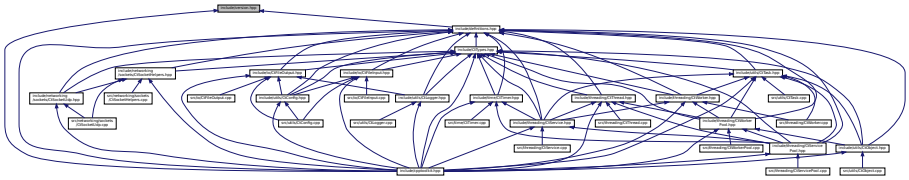
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_CTTASK_HPP_
00033 #define INCLUDE_CTTASK_HPP_
00034
00035 #include "definitions.hpp"
00036 #include "CtTypes.hpp"
00037
00038 #include <functional>
00039
```

```
00062 class CtTask {
00063 public:
00069     EXPORTED_API explicit CtTask();
00070
00078     EXPORTED_API CtTask(const CtTask& other);
00079
00083     EXPORTED_API ~CtTask();
00084
00096     template <typename F, typename... FArgs>
00097     EXPORTED_API void setTaskFunc(const F&& func, FArgs&&... fargs);
00098
00110     template <typename C, typename... CArgs>
00111     EXPORTED_API void setCallbackFunc(const C&& callback, CArgs&&... cargs);
00112
00118     EXPORTED_API std::function<void()> getTaskFunc();
00119
00125     EXPORTED_API std::function<void()> getCallbackFunc();
00126
00135     EXPORTED_API CtTask& operator=(const CtTask& other);
00136
00137 private:
00138     std::function<void()> m_task;
00139     std::function<void()> m_callback;
00140 };
00141
00142 template <typename F, typename... FArgs>
00143 void CtTask::setTaskFunc(const F&& func, FArgs&&... fargs) {
00144     m_task = std::bind(func, std::forward<FArgs>(fargs)...);
00145 };
00146
00147 template <typename C, typename... CArgs>
00148 void CtTask::setCallbackFunc(const C&& callback, CArgs&&... cargs) {
00149     m_callback = std::bind(callback, std::forward<CArgs>(cargs)...);
00150 };
00151
00152 #endif //INCLUDE_CTTASK_HPP_
```

6.50 include/version.hpp File Reference

Version information for the project.

This graph shows which files directly or indirectly include this file:



Macros

- #define CPPTOOLKIT_VERSION_MAJOR 0

  Version information for the project.
- #define CPPTOOLKIT_VERSION_MINOR 1
- #define CPPTOOLKIT_VERSION_PATCH 0
- #define CPPTOOLKIT_VERSION (CPPTOOLKIT_VERSION_MAJOR ## "." ## CPPTOOLKIT_VERSION_MINOR ## "." ## CPPTOOLKIT_VERSION_PATCH)

6.50.1 Detailed Description

Version information for the project.

Date

18-01-2024

Definition in file version.hpp.

6.50.2 Macro Definition Documentation

6.50.2.1 CPPTOOLKIT_VERSION

```
#define CPPTOOLKIT_VERSION (CPPTOOLKIT_VERSION_MAJOR ## "." ## CPPTOOLKIT_VERSION_MINOR ## "."
## CPPTOOLKIT_VERSION_PATCH)
```

Definition at line 50 of file version.hpp.

6.50.2.2 CPPTOOLKIT_VERSION_MAJOR

```
#define CPPTOOLKIT_VERSION_MAJOR 0
```

Version information for the project.

The version information is defined by three macros:

- CPPTOOLKIT_VERSION_MAJOR
- CPPTOOLKIT_VERSION_MINOR
- CPPTOOLKIT_VERSION_PATCH These macros has to be modified after a new release.

Definition at line 46 of file version.hpp.

6.50.2.3 CPPTOOLKIT_VERSION_MINOR

```
#define CPPTOOLKIT_VERSION_MINOR 1
```

Definition at line 47 of file version.hpp.

6.50.2.4 CPPTOOLKIT_VERSION_PATCH

```
#define CPPTOOLKIT_VERSION_PATCH 0
```

Definition at line 48 of file version.hpp.

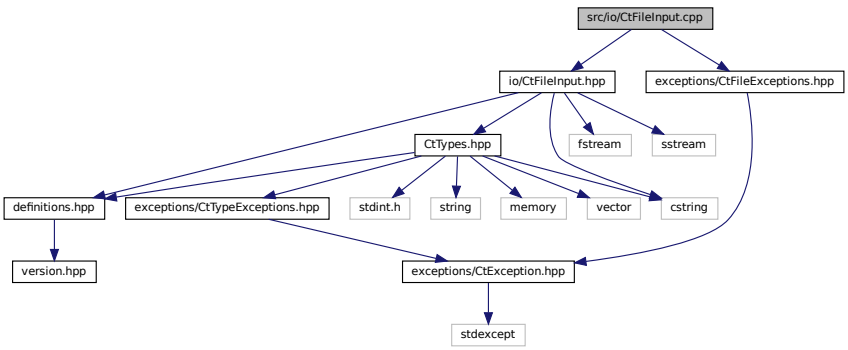
6.51 version.hpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #ifndef INCLUDE_VERSION_HPP_
00033 #define INCLUDE_VERSION_HPP_
00034
00046 #define CPPTOOLKIT_VERSION_MAJOR 0
00047 #define CPPTOOLKIT_VERSION_MINOR 1
00048 #define CPPTOOLKIT_VERSION_PATCH 0
00049
00050 #define CPPTOOLKIT_VERSION       (CPPTOOLKIT_VERSION_MAJOR ## "." ## CPPTOOLKIT_VERSION_MINOR ## "."
       ## CPPTOOLKIT_VERSION_PATCH)
00051
00052 #endif //INCLUDE_VERSION_HPP_
```

6.52 src/io/CtFileInput.cpp File Reference

```
#include "io/CtFileInput.hpp"
#include "exceptions/CtFileExceptions.hpp"
```

Include dependency graph for CtFileInput.cpp:



6.52.1 Detailed Description

Date

08-03-2024

Definition in file CtFileInput.cpp.

6.53 CtFileInput.cpp

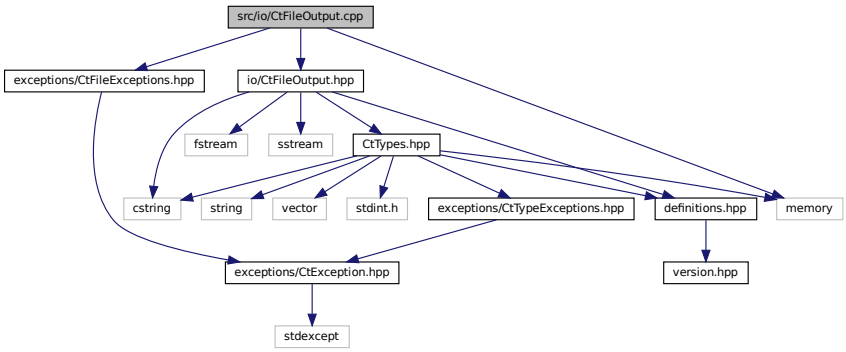
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "io/CtFileInput.hpp"
00033
00034 #include "exceptions/CtFileExceptions.hpp"
00035
00036 CtFileInput::CtFileInput(const std::string& p_fileName) {
00037     m_delim = nullptr;
00038     m_delim_size = 0;
00039     m_file.open(p_fileName, std::ofstream::in);
00040     if (!m_file.is_open()) {
00041         throw CtFileReadError("File cannot open.");
00042     }
00043 }
00044
00045 CtFileInput::~CtFileInput() {
00046     if (m_file.is_open()) {
00047         m_file.close();
00048     }
00049     if (m_delim != nullptr) {
00050         delete[] m_delim;
00051     }
00052 }
00053
00054 void CtFileInput::setDelimiter(const char* p_delim, CtUInt8 p_delim_size) {
00055     if (p_delim_size > 0 && p_delim != nullptr) {
00056         m_delim_size = p_delim_size;
00057         m_delim = new char[m_delim_size];
00058         memcpy(m_delim, p_delim, m_delim_size);
00059     }
00060 }
00061
00062 bool CtFileInput::read(CtRawData* p_data) {
00063     bool s_res = false;
00064
00065     if (m_file.is_open()) {
00066         char next_char;
00067         CtUInt8* delim_ptr = nullptr;
00068
00069         while (m_file.get(next_char)) {
00070             p_data->nextByte(next_char);
00071
00072             if (m_delim != nullptr && p_data->size() >= m_delim_size) {
00073                 delim_ptr = p_data->getNLastBytes(m_delim_size);
00074
00075                 if (memcmp(delim_ptr, m_delim, m_delim_size) == 0) {
00076                     p_data->removeNLastBytes(m_delim_size);
00077                     break;
00078                 }
00079             }
00080
00081             if (p_data->size() == p_data->maxSize()) {
00082                 break;
00083             }
00084         }
00085
00086         if (p_data->size() > 0) {
00087             s_res = true;
00088         } else if (m_file.eof()) {
00089             s_res = false;
00090         }
00091     } else {
00092         throw CtFileReadError("File is not open.");
```

```
00093     }
00094
00095     return s_res;
00096 }
```

6.54 src/io/CtFileOutput.cpp File Reference

```
#include "io/CtFileOutput.hpp"
#include "exceptions/CtFileExceptions.hpp"
#include <memory>
```

Include dependency graph for CtFileOutput.cpp:



6.54.1 Detailed Description

Date

09-03-2024

Definition in file CtFileOutput.cpp.

6.55 CtFileOutput.cpp

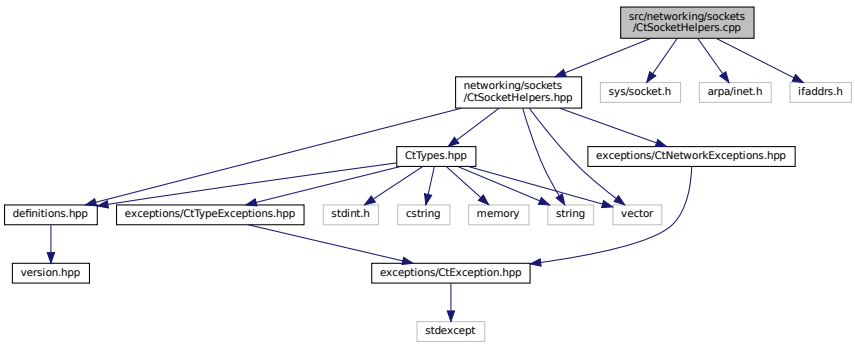
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
```

```
00024
00032 #include "io/CtFileOutput.hpp"
00033
00034 #include "exceptions/CtFileExceptions.hpp"
00035
00036 #include <memory>
00037
00038 CtFileOutput::CtFileOutput(const std::string& p_fileName, WriteMode p_mode) {
00039     m_delim_size = 0;
00040     switch (p_mode) {
00041         case WriteMode::Append:
00042             m_file.open(p_fileName, std::ios::out | std::ios::app);
00043             break;
00044             default:
00045         case WriteMode::Truncate:
00046             m_file.open(p_fileName, std::ios::out | std::ios::trunc);
00047             break;
00048     }
00049     if (!m_file.is_open()) {
00050         throw CtFileWriteError("File cannot open.");
00051     }
00052 }
00053
00054 CtFileOutput::~CtFileOutput() {
00055     if (m_file.is_open()) {
00056         m_file.close();
00057     }
00058 }
00059
00060 void CtFileOutput::setDelimiter(const char* p_delim, CtUInt8 p_delim_size) {
00061     if (p_delim_size > 0 && p_delim != nullptr) {
00062         m_delim_size = p_delim_size;
00063         m_delim.reset();
00064         m_delim = std::make_unique<char[]>(m_delim_size);
00065         memcpy(m_delim.get(), p_delim, m_delim_size);
00066     }
00067 }
00068
00069 void CtFileOutput::write(CtRawData* p_data) {
00070     if (m_file.is_open()) {
00071         m_file.write((char*)p_data->get(), p_data->size());
00072         m_file.write(m_delim.get(), m_delim_size);
00073     } else {
00074         throw CtFileWriteError("File is not open.");
00075     }
00076 }
```

6.56 src/networking/sockets/CtSocketHelpers.cpp File Reference

```
#include "networking/sockets/CtSocketHelpers.hpp"
#include <sys/socket.h>
#include <arpa/inet.h>
#include <ifaddrs.h>
```

Include dependency graph for CtSocketHelpers.cpp:



6.56.1 Detailed Description

Date

21-01-2024

Definition in file CtSocketHelpers.cpp.

6.57 CtSocketHelpers.cpp

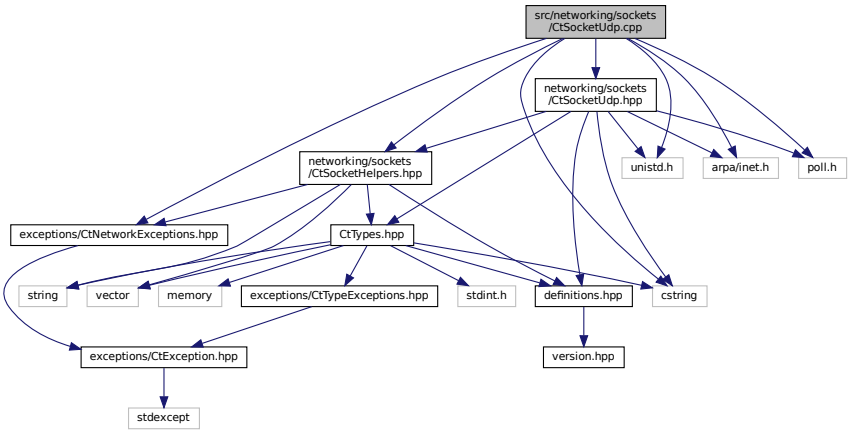
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "networking/sockets/CtSocketHelpers.hpp"
00033
00034 #include <sys/socket.h>
00035 #include <arpa/inet.h>
00036 #include <ifaddrs.h>
00037
00038 int32_t CtSocketHelpers::socketTimeout = 0;
00039
00040 void CtSocketHelpers::setSocketTimeout(int32_t p_socketTimeout) {
00041     CtSocketHelpers::socketTimeout = p_socketTimeout;
00042 }
00043
00044 std::vector<std::string> CtSocketHelpers::getInterfaces() {
00045     struct ifaddrs *s_ifaddr, *s_ifa;
00046     std::vector<std::string> s_interfaces;
00047
00048     if (getifaddrs(&s_ifaddr) == -1) {
00049         throw CtSocketError("Cannot get interfaces.");
00050     }
00051
00052     for (s_ifa = s_ifaddr; s_ifa != nullptr; s_ifa = s_ifa->ifa_next) {
00053         s_interfaces.push_back(std::string(s_ifa->ifa_name));
00054     }
00055     return s_interfaces;
00056 }
00057
00058 std::string CtSocketHelpers::interfaceToAddress(const std::string& p_ifName) {
00059     struct ifaddrs *ifaddr, *ifa;
00060
00061     if (getifaddrs(&ifaddr) == -1) {
00062         throw CtSocketError("Cannot get interfaces.");
00063     }
00064
00065     for (ifa = ifaddr; ifa != nullptr; ifa = ifa->ifa_next) {
00066         if (std::string(ifa->ifa_name) == p_ifName) {
00067             if (ifa->ifa_addr == nullptr) {
00068                 freeifaddrs(ifaddr);
00069                 throw CtSocketError("Not valid interface entered.");
00070             }
00071
00072             if (ifa->ifa_addr->sa_family == AF_INET) {
00073                 char ipBuffer[INET_ADDRSTRLEN];
00074                 sockaddr_in* sockAddr = reinterpret_cast<sockaddr_in*>(ifa->ifa_addr);
00075
00076                 if (inet_ntop(AF_INET, &(sockAddr->sin_addr), ipBuffer, INET_ADDRSTRLEN) == nullptr) {
```

```
00077                     freeifaddrs(ifaddr);
00078                     throw CtSocketError("Failed to convert IPv4 address.");
00079                 }
00080
00081                 freeifaddrs(ifaddr);
00082                 return ipBuffer;
00083             }
00084         }
00085     }
00086
00087     freeifaddrs(ifaddr);
00088     throw CtSocketError("Not valid interface found.");
00089 }
00090
00091 CtUInt32 CtSocketHelpers::getAddressAsUInt(const std::string& p_addr) {
00092     CtUInt32 result = inet_addr(p_addr.c_str());
00093
00094     if (result == INADDR_NONE) {
00095         throw CtSocketError("Invalid address given.");
00096     }
00097
00098     return result;
00099 };
00100
00101 std::string CtSocketHelpers::getAddressAsString(CtUInt32 p_addr) {
00102     char result[INET_ADDRSTRLEN];
00103
00104     if (inet_ntop(AF_INET, &p_addr, result, INET_ADDRSTRLEN) == nullptr) {
00105         throw CtSocketError("Failed to convert IPv4 address.");
00106     }
00107
00108     return std::string(result);
00109 };
00110
00111 void CtSocketHelpers::setConnectionTimeout(timeval& timeout, CtUInt32 timeout_ms) {
00112     timeout.tv_sec = timeout_ms/1000;
00113     timeout.tv_usec = 1000*(timeout_ms%1000);
00114 };
```

6.58 src/networking/sockets/CtSocketUdp.cpp File Reference

```
#include "networking/sockets/CtSocketUdp.hpp"
#include "networking/sockets/CtSocketHelpers.hpp"
#include "exceptions/CtNetworkExceptions.hpp"
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>
#include <poll.h>
```

Include dependency graph for CtSocketUdp.cpp:



6.58.1 Detailed Description

Date

18-01-2024

Definition in file CtSocketUdp.cpp.

6.59 CtSocketUdp.cpp

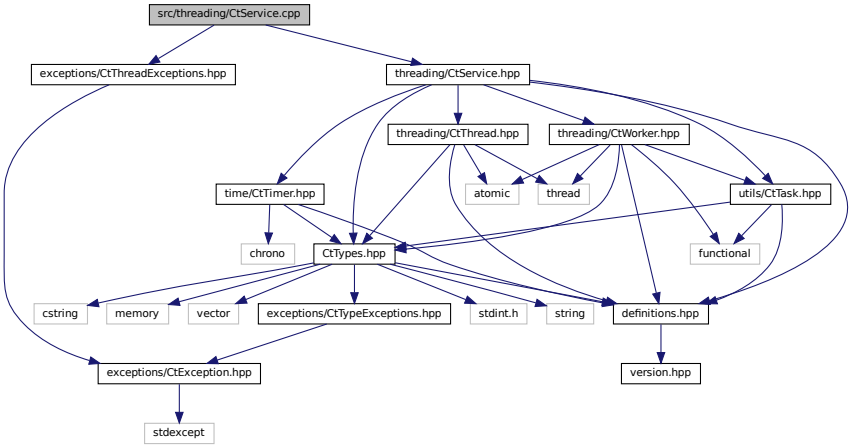
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "networking/sockets/CtSocketUdp.hpp"
00033 #include "networking/sockets/CtSocketHelpers.hpp"
00034 #include "exceptions/CtNetworkExceptions.hpp"
00035
00036 #include <cstring>
00037 #include <unistd.h>
00038 #include <arpa/inet.h>
00039 #include <poll.h>
00040
00041 CtSocketUdp::CtSocketUdp() {
00042     m_addrType = AF_INET;
00043     m_port = 0;
00044     m_socket = socket(m_addrType, SOCK_DGRAM, IPPROTO_UDP);
00045     if (m_socket == -1) {
00046         throw CtSocketError("Socket cannot be assigned.");
00047     }
00048     m_pollin_sockets[0].fd = m_socket;
00049     m_pollin_sockets[0].events = POLLIN;
00050
00051     m_pollout_sockets[0].fd = m_socket;
00052     m_pollout_sockets[0].events = POLLOUT;
00053 }
00054
00055 CtSocketUdp::~CtSocketUdp() {
00056     close(m_socket);
00057 }
00058
00059 void CtSocketUdp::setSub(const std::string& p_interfaceName, uint16_t p_port) {
00060     memset(&m_subAddress, 0, sizeof(m_subAddress));
00061     m_subAddress.sin_family = m_addrType;
00062     m_subAddress.sin_addr.s_addr =
       CtSocketHelpers::getAddressAsUInt(CtSocketHelpers::interfaceToAddress(p_interfaceName));
00063     m_subAddress.sin_port = htons(p_port);
00064
00065     if (bind(m_socket, (struct sockaddr*)&m_subAddress, sizeof(m_subAddress)) == -1) {
00066         throw CtSocketBindError(std::string("Socket bind to port ") + std::to_string(p_port) +
       std::string(" failed."));
00067     }
00068 }
00069
00070 void CtSocketUdp::setPub(uint16_t p_port, const std::string& p_addr) {
00071     memset(&m_pubAddress, 0, sizeof(m_pubAddress));
00072     m_pubAddress.sin_family = m_addrType;
00073     m_pubAddress.sin_addr.s_addr = CtSocketHelpers::getAddressAsUInt(p_addr);
00074     m_pubAddress.sin_port = htons(p_port);
```

```
00075 }
00076
00077 bool CtSocketUdp::pollRead() {
00078     int pollResult = poll(m_pollin_sockets, 1, CtSocketHelpers::socketTimeout);
00079
00080     if (pollResult < 0) {
00081         throw CtSocketPollError("Socket polling-in failed.");
00082     } else if (pollResult == 0) {
00083         return false;
00084     } else {
00085         return true;
00086     }
00087 }
00088
00089 bool CtSocketUdp::pollWrite() {
00090     int pollResult = poll(m_pollout_sockets, 1, CtSocketHelpers::socketTimeout);
00091
00092     if (pollResult < 0) {
00093         throw CtSocketPollError("Socket polling-out failed.");
00094     } else if (pollResult == 0) {
00095         return false;
00096     } else {
00097         return true;
00098     }
00099 }
00100
00101 void CtSocketUdp::send(uint8_t* p_data, CtUInt32 p_size) {
00102     if (sendto(m_socket, p_data, p_size, MSG_DONTWAIT, (struct sockaddr*)&m_pubAddress,
       sizeof(m_pubAddress)) == -1) {
00103         throw CtSocketWriteError("Sending data via socket failed.");
00104     }
00105 }
00106
00107 void CtSocketUdp::send(CtRawData& p_message) {
00108     send(p_message.get(), p_message.size());
00109 }
00110
00111 void CtSocketUdp::receive(uint8_t* p_data, CtUInt32 p_size, CtNetAddress* p_client) {
00112     sockaddr_in s_clientAddress_in;
00113     socklen_t s_clientAddressLength = sizeof(s_clientAddress_in);
00114     int bytesRead = recvfrom(m_socket, p_data, p_size, MSG_DONTWAIT, (struct
       sockaddr*)&s_clientAddress_in, &s_clientAddressLength);
00115
00116     if (bytesRead == -1) {
00117         throw CtSocketReadError("Receiving data via socket failed.");
00118     }
00119
00120     if (p_client != nullptr) {
00121         p_client->addr =
       (CtString)CtSocketHelpers::getAddressAsString(*(CtUInt32*)(&s_clientAddress_in.sin_addr));
00122         p_client->port = s_clientAddress_in.sin_port;
00123     }
00124
00125     p_data[bytesRead] = '\0';
00126 }
00127
00128 void CtSocketUdp::receive(CtRawData* p_message, CtNetAddress* p_client) {
00129     uint8_t* s_buffer = new uint8_t[p_message->maxSize()];
00130     receive(s_buffer, p_message->maxSize(), p_client);
00131     p_message->clone(s_buffer, p_message->maxSize());
00132     delete[] s_buffer;
00133 }
```

6.60 src/threading/CtService.cpp File Reference

```
#include "threading/CtService.hpp"
#include "exceptions/CtThreadExceptions.hpp"
```

Include dependency graph for CtService.cpp:



6.60.1 Detailed Description

Date

18-01-2024

Definition in file CtService.cpp.

6.61 CtService.cpp

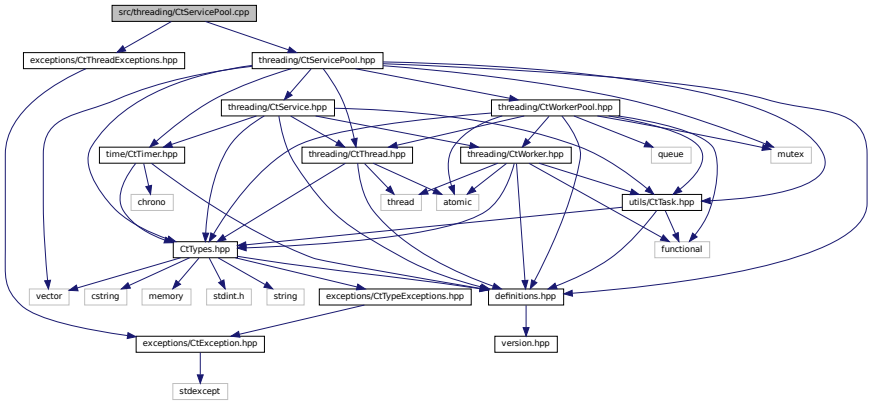
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "threading/CtService.hpp"
00033 #include "exceptions/CtThreadExceptions.hpp"
00034
00035 CtUInt32 CtService::m_slot_time = 10;
00036
00037 CtService::CtService(CtUInt64 nslots, const CtTask& task) : m_nslots(nslots){
00038     m_worker.setTask(task);
00039     runService();
00040 }
00041
00042 CtService::~CtService() {
00043     stopService();
```

```
00044 }
00045
00046 void CtService::runService() {
00047     try {
00048         start();
00049     } catch(const CtThreadError& e) {
00050
00051     }
00052 }
00053
00054 void CtService::stopService() {
00055     stop();
00056     m_worker.joinTask();
00057 }
00058
00059 void CtService::loop() {
00060     try {
00061         m_worker.runTask();
00062     } catch(const CtWorkerError& e) {
00063
00064     }
00065     CtThread::sleepFor(m_nslots*m_slot_time);
00066 }
```

6.62 src/threading/CtServicePool.cpp File Reference

```
#include "threading/CtServicePool.hpp"
#include "exceptions/CtThreadExceptions.hpp"
```

Include dependency graph for CtServicePool.cpp:



6.62.1 Detailed Description

Date

18-01-2024

Definition in file CtServicePool.cpp.

6.63 CtServicePool.cpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
```

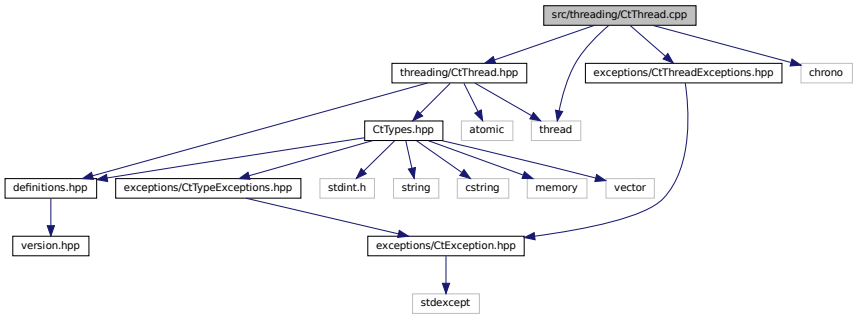
```
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "threading/CtServicePool.hpp"
00033 #include "exceptions/CtThreadExceptions.hpp"
00034
00035 CtServicePool::CtServicePool(CtUInt32 nworkers) : m_nworkers(nworkers), m_worker_pool(m_nworkers) {
00036     m_slot_cnt = 0;
00037     m_exec_time = 0;
00038     start();
00039 }
00040
00041 CtServicePool::~CtServicePool() {
00042     stop();
00043     m_worker_pool.join();
00044 }
00045
00046 void CtServicePool::addTask(CtUInt32 nslots, const std::string& id, CtTask& task) {
00047     std::scoped_lock lock(m_mtx_control);
00048     m_tasks.push_back({task, id, nslots});
00049     try {
00050         start();
00051     } catch(const CtThreadError& e) {
00052     }
00053 }
00054
00055 void CtServicePool::removeTask(const std::string& id) {
00056     std::scoped_lock lock(m_mtx_control);
00057     m_tasks.erase(std::remove_if(m_tasks.begin(), m_tasks.end(),
00058                                  [id](CtServicePack pack) {
00059                                      return pack.id.compare(id) == 0;
00060                                  }
00061                                  ), m_tasks.end());
00062 }
00063
00064 void CtServicePool::startServices() {
00065     try {
00066         start();
00067     } catch(const CtThreadError& e) {
00068     }
00069 }
00070
00071 void CtServicePool::shutdownServices() {
00072     stop();
00073     m_worker_pool.join();
00074 }
00075
00076 CtUInt32 CtServicePool::getSlotTime() {
00077     return CtService::m_slot_time;
00078 }
00079
00080 void CtServicePool::setSlotTime(CtUInt32 slot_time) {
00081     CtService::m_slot_time = slot_time;
00082 }
00083
00084 void CtServicePool::loop() {
00085     m_timer.tic();
00086     {
00087         std::scoped_lock lock(m_mtx_control);
00088         if (m_tasks.size() == 0) {
00089             return;
00090         } else {
00091             for (const CtServicePack& pack : m_tasks) {
00092                 if (m_slot_cnt % pack.nslots == 0) {
00093                     m_worker_pool.addTask(pack.task);
00094                 }
00095             }
00096         }
00097     }
00098     m_slot_cnt++;
```

```
00099     if (!isRunning()) return;
00100     m_exec_time = CtService::m_slot_time - m_timer.toc();
00101     if (m_exec_time > 0) {
00102         CtThread::sleepFor(m_exec_time);
00103     }
00104 }
```

6.64 src/threading/CtThread.cpp File Reference

```
#include "threading/CtThread.hpp"
#include "exceptions/CtThreadExceptions.hpp"
#include <chrono>
#include <thread>
```

Include dependency graph for CtThread.cpp:



6.64.1 Detailed Description

Date

18-01-2024

Definition in file CtThread.cpp.

6.65 CtThread.cpp

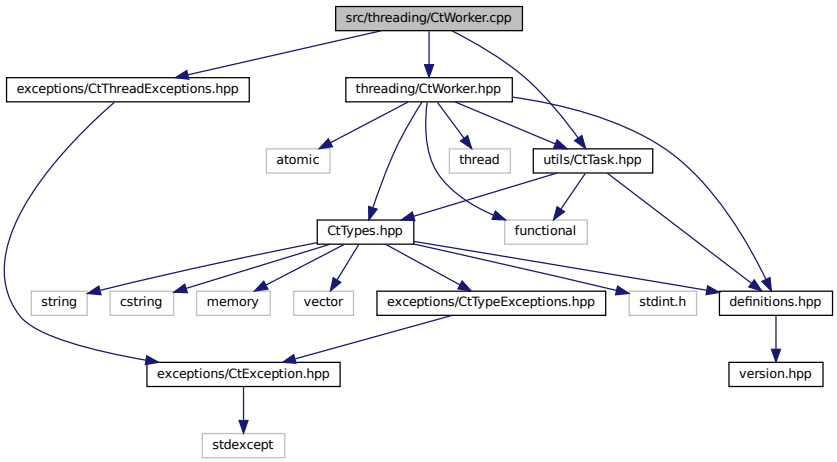
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
```

```
00023 */
00024
00032 #include "threading/CtThread.hpp"
00033 #include "exceptions/CtThreadExceptions.hpp"
00034
00035 #include <chrono>
00036 #include <thread>
00037
00038 CtThread::CtThread() : m_running(false) {
00039
00040 }
00041
00042 CtThread::~CtThread() {
00043     stop();
00044 }
00045
00046 void CtThread::run() {
00047     while(isRunning()) {
00048         loop();
00049     }
00050 }
00051
00052 void CtThread::start() {
00053     if (!isRunning()) {
00054         join();
00055         setRunning(true);
00056         m_thread = std::thread(&CtThread::run, this);
00057     } else {
00058         throw CtThreadError("Thread already running.");
00059     }
00060 }
00061
00062 void CtThread::stop() {
00063     setRunning(false);
00064     CtThread::join();
00065 }
00066
00067 void CtThread::join() {
00068     if (m_thread.joinable()) {
00069         m_thread.join();
00070     }
00071 }
00072
00073 bool CtThread::isRunning() {
00074     return m_running.load();
00075 }
00076
00077 void CtThread::setRunning(bool running) {
00078     m_running.store(running);
00079 }
00080
00081 void CtThread::sleepFor(uint64_t time) {
00082     std::this_thread::sleep_for(std::chrono::milliseconds(time));
00083 }
```

6.66 src/threading/CtWorker.cpp File Reference

```
#include "threading/CtWorker.hpp"
#include "utils/CtTask.hpp"
#include "exceptions/CtThreadExceptions.hpp"
```

Include dependency graph for CtWorker.cpp:



6.66.1 Detailed Description

Date

18-01-2024

Definition in file CtWorker.cpp.

6.67 CtWorker.cpp

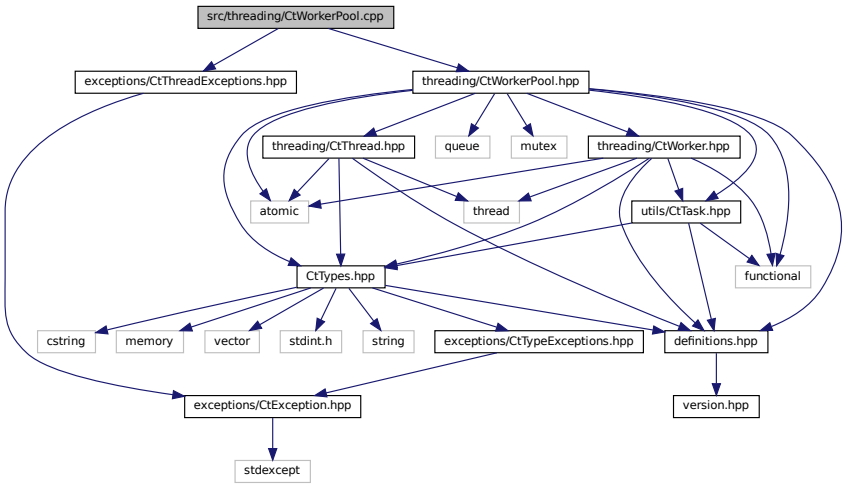
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "threading/CtWorker.hpp"
00033 #include "utils/CtTask.hpp"
00034 #include "exceptions/CtThreadExceptions.hpp"
00035
00036 CtWorker::CtWorker() : m_running(false) {
00037 }
00038
00039 CtWorker::~CtWorker() {
00040 }
00041
00042 bool CtWorker::isRunning() {
```

```
00043     return m_running.load();
00044 }
00045
00046 void CtWorker::setRunning(bool running) {
00047     return m_running.store(running);
00048 }
00049
00050 void CtWorker::setTask(const CtTask& task, std::function<void()> callback) {
00051     alreadyRunningCheck();
00052     m_task = task;
00053     m_callback = callback;
00054 }
00055
00056 void CtWorker::runTask() {
00057     alreadyRunningCheck();
00058     setRunning(true);
00059     m_thread = std::thread([this]{
00060         m_task.getTaskFunc()();
00061         m_task.getCallbackFunc()();
00062         m_callback();
00063         setRunning(false);
00064     });
00065 }
00066
00067 void CtWorker::joinTask() {
00068     if (m_thread.joinable()) {
00069         m_thread.join();
00070     }
00071 }
00072
00073 void CtWorker::alreadyRunningCheck() {
00074     if (isRunning()) {
00075         throw CtWorkerError("CtWorker already running.");
00076     }
00077     joinTask();
00078 }
```

6.68 src/threading/CtWorkerPool.cpp File Reference

```
#include "threading/CtWorkerPool.hpp"
#include "exceptions/CtThreadExceptions.hpp"
```

Include dependency graph for CtWorkerPool.cpp:



6.68.1 Detailed Description

Date

18-01-2024

Definition in file CtWorkerPool.cpp.

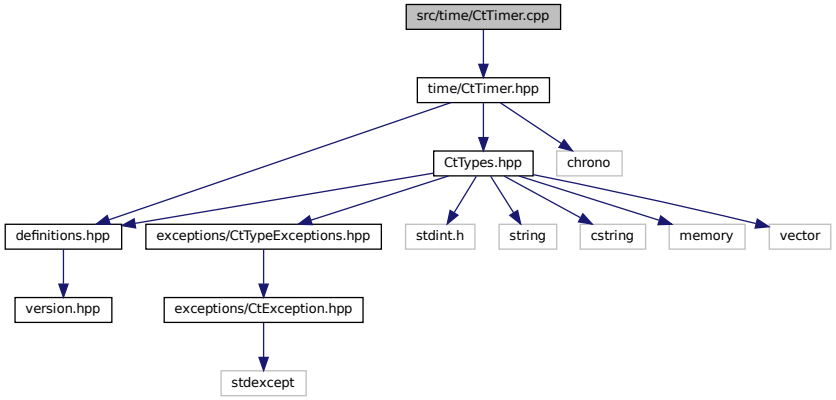
6.69 CtWorkerPool.cpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "threading/CtWorkerPool.hpp"
00033 #include "exceptions/CtThreadExceptions.hpp"
00034
00035 CtWorkerPool::CtWorkerPool(CtUInt32 nworkers) : m_nworkers(nworkers), m_active_tasks(0),
       m_queued_tasks(0) {
00036     for (int idx = 0; idx < m_nworkers; idx++) {
00037         m_workers.push_back(std::make_unique<CtWorker>());
00038     }
00039 }
00040
00041 CtWorkerPool::~CtWorkerPool() {
00042     CtThread::stop();
00043     free();
00044 }
00045
00046 void CtWorkerPool::addTask(const CtTask& task) {
00047     std::scoped_lock lock(m_mtx_control);
00048     m_tasks.push(task);
00049     m_queued_tasks++;
00050     try {
00051         start();
00052     } catch (const CtThreadError& e) {
00053     }
00054 }
00055
00056 void CtWorkerPool::join() {
00057     CtThread::join();
00058 }
00059
00060 void CtWorkerPool::assignTask(CtUInt32 idx) {
00061     std::scoped_lock lock(m_mtx_control);
00062     try {
00063         m_workers.at(idx).get()->setTask(m_tasks.front(), [this](){m_active_tasks--;});
00064         m_workers.at(idx).get()->runTask();
00065     } catch (const CtWorkerError& e) {
00066         return;
00067     }
00068     m_active_tasks++;
00069     m_tasks.pop();
00070     m_queued_tasks--;
00071 }
00072
00073 void CtWorkerPool::free() {
00074     for (int idx = 0; idx < m_nworkers; idx++) {
00075         m_workers.back().get()->joinTask();
00076         m_workers.pop_back();
00077     }
00078 }
00079
```

```
00080 void CtWorkerPool::loop() {
00081     bool s_breakFlag = false;
00082     while (!s_breakFlag) {
00083         for (int idx = 0; idx < m_nworkers; idx++) {
00084             if (!m_workers.at(idx).get()->isRunning() && m_queued_tasks.load() != 0) {
00085                 assignTask(idx);
00086             }
00087         }
00088         s_breakFlag = (m_queued_tasks.load() == 0) && (m_active_tasks.load() == 0);
00089     }
00090     setRunning(false);
00091 }
```

6.70 src/time/CtTimer.cpp File Reference

`#include "time/CtTimer.hpp"`
Include dependency graph for CtTimer.cpp:



6.70.1 Detailed Description

Date

18-01-2024

Definition in file CtTimer.cpp.

6.71 CtTimer.cpp

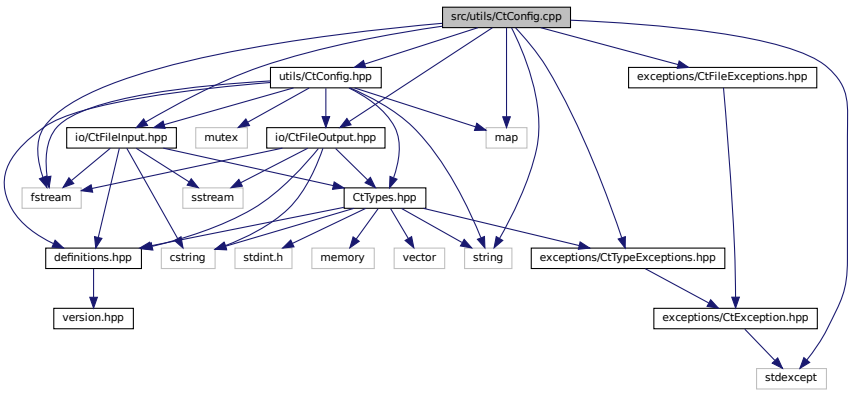
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
```

```
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "time/CtTimer.hpp"
00033
00034 CtTimer::CtTimer() {
00035     m_reference = 0;
00036 }
00037
00038 CtTimer::~CtTimer() {
00039
00040 }
00041
00042 void CtTimer::tic() {
00043     m_reference = current();
00044 }
00045
00046 uint64_t CtTimer::toc() {
00047     return current() - m_reference;
00048 }
00049
00050 uint64_t CtTimer::current() {
00051     return std::chrono::duration_cast<std::chrono::milliseconds>(
00052         std::chrono::high_resolution_clock::now().time_since_epoch()
00053     ).count();
00054 }
```

6.72 src/utils/CtConfig.cpp File Reference

```
#include "utils/CtConfig.hpp"
#include "io/CtFileInput.hpp"
#include "io/CtFileOutput.hpp"
#include "exceptions/CtFileExceptions.hpp"
#include "exceptions/CtTypeExceptions.hpp"
#include <string>
#include <map>
#include <fstream>
#include <stdexcept>
```

Include dependency graph for CtConfig.cpp:



6.72.1 Detailed Description

Date

10-03-2024

Definition in file CtConfig.cpp.

6.73 CtConfig.cpp

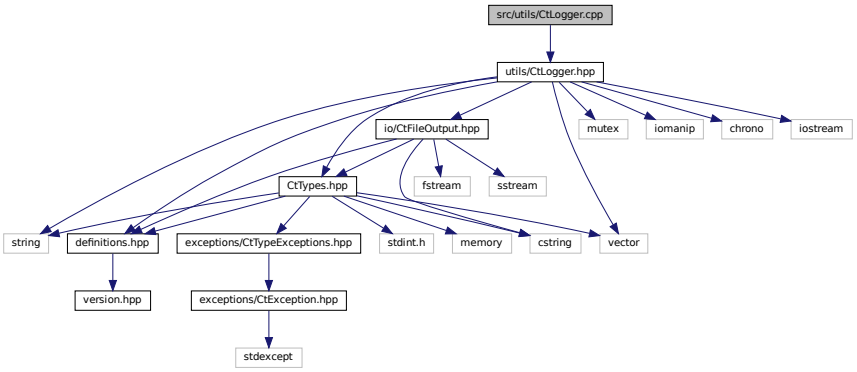
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "utils/CtConfig.hpp"
00033
00034 #include "io/CtFileInput.hpp"
00035 #include "io/CtFileOutput.hpp"
00036
00037 #include "exceptions/CtFileExceptions.hpp"
00038 #include "exceptions/CtTypeExceptions.hpp"
00039
00040 #include <string>
00041 #include <map>
00042 #include <fstream>
00043 #include <stdexcept>
00044
00045 CtConfig::CtConfig(const std::string& configFile) : m_configFile(configFile) {
00046     m_source = nullptr;
00047     m_sink = nullptr;
00048 }
00049
00050 CtConfig::~CtConfig() {
00051     if (m_source != nullptr) {
00052         delete m_source;
00053     }
00054     if (m_sink != nullptr) {
00055         delete m_sink;
00056     }
00057 }
00058
00059 void CtConfig::read() {
00060     std::scoped_lock lock(m_mtx_control);
00061     m_source = new CtFileInput(m_configFile);
00062     m_source->setDelimiter("\n", 1);
00063
00064     CtRawData data(512);
00065
00066     while(m_source->read(&data)) {
00067         parseLine(std::string((char*)data.get(), data.size()));
00068         data.reset();
00069     }
00070
00071     delete m_source;
00072     m_source = nullptr;
00073 }
00074
00075 void CtConfig::write() {
00076     std::scoped_lock lock(m_mtx_control);
00077     m_sink = new CtFileOutput(m_configFile, CtFileOutput::WriteMode::Truncate);
00078     m_sink->setDelimiter("\n", 1);
00079     std::map<std::string, std::string>::iterator iter;
00080
```

```
00081     CtRawData data(512);
00082     for (iter = m_configValues.begin(); iter != m_configValues.end(); ++iter) {
00083         std::string line = iter->first + std::string(" = ") + iter->second;
00084         data.clone((CtUInt8*)line.c_str(), line.size());
00085         m_sink->write(&data);
00086         data.reset();
00087     }
00088 
00089     delete m_sink;
00090     m_sink = nullptr;
00091 }
00092 
00093 void CtConfig::parseLine(const std::string& line) {
00094     size_t separatorPos = line.find('=');
00095     size_t commentPos = line.find('#');
00096     size_t eol = line.size();
00097     bool hasComment = commentPos != std::string::npos;
00098     bool hasSeparator = separatorPos != std::string::npos;
00099 
00100     if (hasSeparator && hasComment) {
00101         if (separatorPos > commentPos) {
00102             throw CtFileParseError("Invalid comment.");
00103         } else {
00104             eol = commentPos;
00105         }
00106     } else if (!hasSeparator && !hasComment) {
00107         throw CtFileParseError("Invalid line entry.");
00108     } else if (!hasSeparator && hasComment) {
00109         return;
00110     }
00111 
00112     std::string key = line.substr(0, separatorPos);
00113     std::string value = line.substr(separatorPos + 1, eol - (separatorPos + 1));
00114 
00115     key.erase(0, key.find_first_not_of(" \t\r\n"));
00116     key.erase(key.find_last_not_of(" \t\r\n") + 1);
00117     value.erase(0, value.find_first_not_of(" \t\r\n"));
00118     value.erase(value.find_last_not_of(" \t\r\n") + 1);
00119 
00120     m_configValues[key] = value;
00121 }
00122 
00123 int32_t CtConfig::parseAsInt(const std::string& key) {
00124     int32_t parsed_value;
00125     std::string str_value = getValue(key);
00126     try {
00127         parsed_value = stoi(str_value);
00128     } catch (...) {
00129         throw CtTypeParseError(std::string("Value of <") + key + std::string("> can not be parsed as
      int."));
00130     }
00131     return parsed_value;
00132 }
00133 
00134 uint32_t CtConfig::parseAsUInt(const std::string& key) {
00135     uint32_t parsed_value;
00136     std::string str_value = getValue(key);
00137     try {
00138         parsed_value = stoul(str_value);
00139     } catch (...) {
00140         throw CtTypeParseError(std::string("Value of <") + key + std::string("> can not be parsed as
      uint."));
00141     }
00142     return parsed_value;
00143 }
00144 
00145 float CtConfig::parseAsFloat(const std::string& key) {
00146     float parsed_value;
00147     std::string str_value = getValue(key);
00148     try {
00149         parsed_value = stof(str_value);
00150     } catch (...) {
00151         throw CtTypeParseError(std::string("Value of <") + key + std::string("> can not be parsed as
      float."));
00152     }
00153     return parsed_value;
00154 }
00155 
00156 double CtConfig::parseAsDouble(const std::string& key) {
00157     double parsed_value;
00158     std::string str_value = getValue(key);
00159     try {
00160         parsed_value = stod(str_value);
00161     } catch (...) {
00162         throw CtTypeParseError(std::string("Value of <") + key + std::string("> can not be parsed as
      double."));
00163     }
```

```
00164     return parsed_value;
00165 }
00166
00167 std::string CtConfig::parseAsString(const std::string& key) {
00168     return getValue(key);
00169 }
00170
00171 std::string CtConfig::getValue(const std::string& key) {
00172     if (m_configValues.find(key) != m_configValues.end()) {
00173         return m_configValues[key];
00174     } else {
00175         throw CtKeyNotFoundError(std::string("Key <") + key + std::string("> not found."));
00176     }
00177 }
00178
00179 void CtConfig::writeInt(const std::string& p_key, const int32_t& p_value) {
00180     writeString(p_key, std::to_string(p_value));
00181 }
00182
00183 void CtConfig::writeUInt(const std::string& p_key, const uint32_t& p_value) {
00184     writeString(p_key, std::to_string(p_value));
00185 }
00186
00187 void CtConfig::writeFloat(const std::string& p_key, const float& p_value) {
00188     writeString(p_key, std::to_string(p_value));
00189 }
00190
00191 void CtConfig::writeDouble(const std::string& p_key, const double& p_value) {
00192     writeString(p_key, std::to_string(p_value));
00193 }
00194
00195 void CtConfig::writeString(const std::string& p_key, const std::string& p_value) {
00196     m_configValues[p_key] = p_value;
00197 }
```

6.74 src/utils/CtLogger.cpp File Reference

`#include "utils/CtLogger.hpp"`
Include dependency graph for CtLogger.cpp:



6.74.1 Detailed Description

Date

10-03-2024

Definition in file CtLogger.cpp.

6.75 CtLogger.cpp

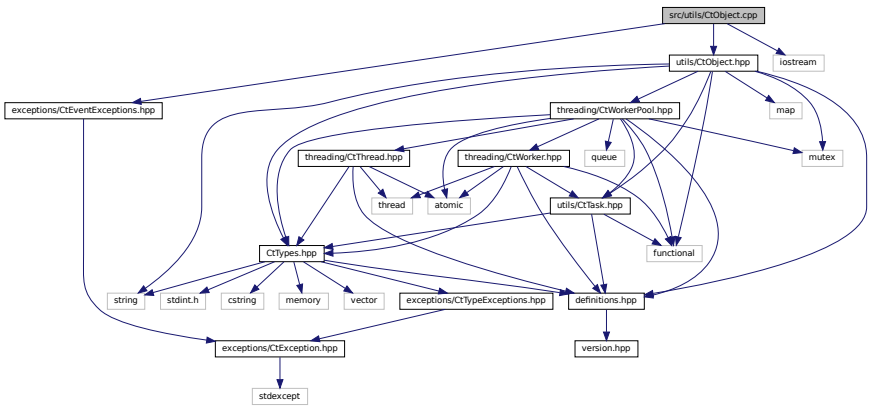
```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "utils/CtLogger.hpp"
00033
00034 CtLogger::CtLogger(CtLogger::Level level, const std::string& componentName) : m_level(level),
       m_componentName(componentName) {
00035 }
00036
00037 CtLogger::~CtLogger() {
00038 }
00039
00040 void CtLogger::log_debug(const std::string& message) {
00041     log(CtLogger::Level::DEBUG, message);
00042 }
00043
00044 void CtLogger::log_info(const std::string& message) {
00045     log(CtLogger::Level::INFO, message);
00046 }
00047
00048 void CtLogger::log_warning(const std::string& message) {
00049     log(CtLogger::Level::WARNING, message);
00050 }
00051
00052 void CtLogger::log_error(const std::string& message) {
00053     log(CtLogger::Level::ERROR, message);
00054 }
00055
00056 void CtLogger::log_critical(const std::string& message) {
00057     log(CtLogger::Level::CRITICAL, message);
00058 }
00059
00060 void CtLogger::log(CtLogger::Level level, const std::string& message) {
00061     std::scoped_lock lock(m_mtx_control);
00062     if (level >= m_level) {
00063         std::string logEntry = generateLoggerMsg(level, m_componentName, message);
00064         std::cout « logEntry « std::endl;
00065     }
00066 }
00067
00068 const std::string CtLogger::generateLoggerMsg(CtLogger::Level level, const std::string& componentName,
       const std::string& message) {
00069     std::chrono::system_clock::time_point now = std::chrono::system_clock::now();
00070     std::time_t now_c = std::chrono::system_clock::to_time_t(now);
00071     std::stringstream timestamp;
00072     timestamp « std::put_time(std::localtime(&now_c), "%Y-%m-%d %X");
00073
00074     return std::string("[" + timestamp.str() + "] [" + levelToString(level) + "] " + componentName +
       ": " + message);
00075 }
00076
00077 const std::string CtLogger::levelToString(CtLogger::Level level) {
00078     std::string levelStr;
00079     switch (level) {
00080         case CtLogger::Level::DEBUG:
00081             levelStr = "DEBUG";
00082             break;
00083         case CtLogger::Level::INFO:
00084             levelStr = "INFO";
00085             break;
00086         case CtLogger::Level::WARNING:
00087             levelStr = "WARNING";
00088             break;
00089         case CtLogger::Level::ERROR:
```

```
00090             levelStr = "ERROR";
00091             break;
00092         case CtLogger::Level::CRITICAL:
00093             levelStr = "CRITICAL";
00094             break;
00095         default:
00096             levelStr = "DEBUG";
00097             break;
00098     }
00099
00100     return levelStr;
00101 }
00102
00103 CtLogger::Level CtLogger::stringToLevel(const std::string& levelStr) {
00104     Level level;
00105     if (levelStr.compare("DEBUG") == 0) {
00106         level = CtLogger::Level::DEBUG;
00107     } else if (levelStr.compare("INFO") == 0) {
00108         level = CtLogger::Level::INFO;
00109     } else if (levelStr.compare("WARNING") == 0) {
00110         level = CtLogger::Level::WARNING;
00111     } else if (levelStr.compare("ERROR") == 0) {
00112         level = CtLogger::Level::ERROR;
00113     } else if (levelStr.compare("CRITICAL") == 0) {
00114         level = CtLogger::Level::CRITICAL;
00115     } else {
00116         level = CtLogger::Level::DEBUG;
00117     }
00118
00119     return level;
00120 }
```

6.76 src/utils/CtObject.cpp File Reference

```
#include "utils/CtObject.hpp"
#include "exceptions/CtEventExceptions.hpp"
#include <iostream>
```

Include dependency graph for CtObject.cpp:



6.76.1 Detailed Description

Date

02-02-2024

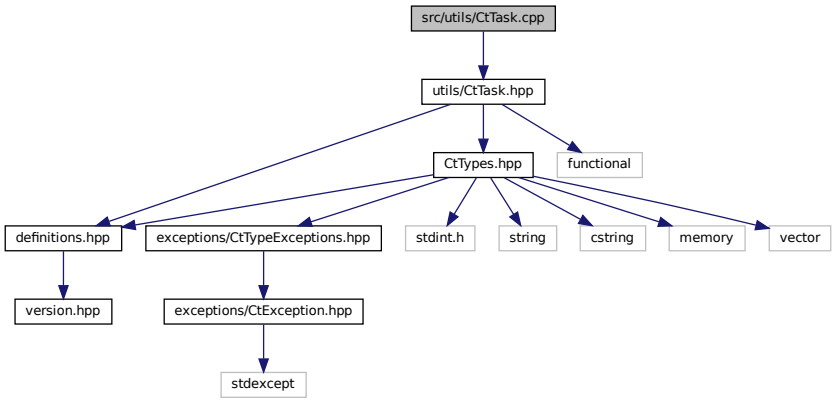
Definition in file CtObject.cpp.

6.77 CtObject.cpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "utils/CtObject.hpp"
00033
00034 #include "exceptions/CtEventExceptions.hpp"
00035
00036 #include <iostream>
00037
00038 CtObject::CtObject() : m_pool(1) {
00039
00040 }
00041
00042 CtObject::~CtObject() {
00043     m_pool.join();
00044 }
00045
00046 void CtObject::connectEvent(CtObject* p_obj, CtUInt32 p_eventCode, CtTask& p_task) {
00047     p_obj->connectEvent(p_eventCode, p_task);
00048 }
00049
00050 void CtObject::waitPendingEvents() {
00051     m_pool.join();
00052 }
00053
00054 void CtObject::connectEvent(CtUInt32 p_eventCode, CtTask& p_task) {
00055     std::scoped_lock lock(m_mtx_control);
00056     if (!hasEvent(p_eventCode)) {
00057         throw CtEventNotExistsError("Event is not registed. " + std::to_string(p_eventCode));
00058     }
00059     m_triggers.insert({p_eventCode, p_task});
00060 }
00061
00062 void CtObject::triggerEvent(CtUInt32 p_eventCode) {
00063     std::scoped_lock lock(m_mtx_control);
00064     if (!hasEvent(p_eventCode)) {
00065         throw CtEventNotExistsError("Event is not registed. " + std::to_string(p_eventCode));
00066     }
00067     std::pair<std::multimap<CtUInt32, CtTask>::iterator, std::multimap<CtUInt32, CtTask>::iterator>
       s_iterRange;
00068     s_iterRange = m_triggers.equal_range(p_eventCode);
00069
00070     std::multimap<CtUInt32, CtTask>::iterator s_iter;
00071     for (s_iter = s_iterRange.first; s_iter != s_iterRange.second; ++s_iter) {
00072         m_pool.addTask(s_iter->second);
00073     }
00074 }
00075
00076 void CtObject::registerEvent(CtUInt32 p_eventCode) {
00077     std::scoped_lock lock(m_mtx_control);
00078     if (hasEvent(p_eventCode)) {
00079         throw CtEventAlreadyExistsError("Event is already registed.");
00080     }
00081     m_events.push_back(p_eventCode);
00082 }
00083
00084 bool CtObject::hasEvent(CtUInt32 p_eventCode) {
00085     return std::any_of(m_events.begin(), m_events.end(), [&p_eventCode](CtUInt8 s_event) {
00086         return (s_event == p_eventCode);
00087     });
00088 }
```

6.78 src/utils/CtTask.cpp File Reference

`#include "utils/CtTask.hpp"`
Include dependency graph for CtTask.cpp:



6.78.1 Detailed Description

Date

18-01-2024

Definition in file CtTask.cpp.

6.79 CtTask.cpp

```
00001 /*
00002 MIT License
00003
00004 Copyright (c) 2024 Mouzenidis Panagiotis
00005
00006 Permission is hereby granted, free of charge, to any person obtaining a copy
00007 of this software and associated documentation files (the "Software"), to deal
00008 in the Software without restriction, including without limitation the rights
00009 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010 copies of the Software, and to permit persons to whom the Software is
00011 furnished to do so, subject to the following conditions:
00012
00013 The above copyright notice and this permission notice shall be included in all
00014 copies or substantial portions of the Software.
00015
00016 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022 SOFTWARE.
00023 */
00024
00032 #include "utils/CtTask.hpp"
00033
00034 CtTask::CtTask() : m_task([]{}), m_callback([]{}) {
00035 }
00036
00037 CtTask::CtTask(const CtTask& other) : m_task(other.m_task), m_callback(other.m_callback) {
00038 }
00039
```

```
00040 CtTask::~CtTask() {
00041
00042 }
00043
00044 std::function<void()> CtTask::getTaskFunc() {
00045     return m_task;
00046 }
00047
00048 std::function<void()> CtTask::getCallbackFunc() {
00049     return m_callback;
00050 }
00051
00052 CtTask& CtTask::operator=(const CtTask& other) {
00053     if (this != &other) {
00054         m_task = other.m_task;
00055         m_callback = other.m_callback;
00056     }
00057     return *this;
00058 }
```

Index