lrc - Linux Resource Compiler

1.2.1

Generated by Doxygen 1.8.13

Contents

Chapter 1

lrc - Linux Resource Compiler

1.1 Introduction

This documentation covers the classes and data structures of the `lrc` compiler and the library `liblrc`.

1.2 Compiler Options

The compiler needs at least one input file. It is called as follows:
`lrc -h | [-d] [-m] [-o <rdfFile>] [-c <compression type>]<RCFile>|<rif↩File>`

The options have the following meaning:
`-h`: Show the usage of the compiler and quit.
`-d`: Deny overwrite an existing .rdf file. This parameter is optional and the default is overwriting allowed. If `-d` is set and the file already exists, the compiler exits with an error message.
`-m`: Prepare the data for direct linking. During the compilation another file with the extension .o is created. This file can be linked directly to the executable. The labels `_binary_file_start` and `_binary_file_end` are pointers for the start and the end within the executable. See `ShowEmbeddedImage.cxx` for more information.
`-o <file>`: Specify the resource output file (.rdf). This parameter is optional. If it is omitted the output file will be the name of the input file, but with .rdf as file extension.
`-c <compression type>`: Specify the compression type for the whole .rdf file. Allowed are all compression types that are allowed for a single resource file. These are at the moment:

- zLib: zLib compression
- bzip2: bzip2 compression

1.3 The library liblrc

The library consists mainly of two classes: the ResourceManager and the Resource class. One ResourceManager instance is used for one resource data file (.rdf).
The resource manager provides two main methods: one lists all resource entries from the .rdf file, the other returns an instance of a Resource class, identified by its ID or index.

1.4 Example

A few examples that demonstrates the `lrc` and `liblrc` can be found in the `src/example` directory.

Author

Andreas Tscharner

Date

2014-08-22

Chapter 2

Todo List

Member Irc::CompressionType

Add more possibilities to compress resource

Member Irc::EncryptionType

Add more possibilities to encrypt resource

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

Chapter 7

Namespace Documentation

7.1 lrc Namespace Reference

Namespace lrc for classes in the library and for extending `lrc`.

Classes

- class CompressDecompress

  Compression and Decompression base class.
- class EncryptDecrypt

  Encryption and Decryption base class.
- class Resource

  Resource class for use with library.
- class ResourceManager

  Manager class handling all resources of a resource file.

Enumerations

- enum CompressionType { NoneCompression, zLibCompression, bz2LibCompression, lastCompression }
- enum EncryptionType { NoneEncryption, SerpentEncryption, lastEncryption }

7.1.1 Detailed Description

Namespace lrc for classes in the library and for extending `lrc`.

The namespace lrc was introduced and used for classes and functions that will be used in the `liblrc` library to avoid confusion with other classes that may have the same name.

It is also used for two abstract classes: CompressDecompress and EncryptDecrypt. These two classes are the base classes for compression and decompression and for encryption and decryption of the resource data. They can be found in the CompressDecompress.hxx and the EncryptDecrypt.hxx file respectively.

7.1.2 Enumeration Type Documentation

7.1.2.1 CompressionType

`enum lrc::CompressionType`

Possible compression types for resource

Todo Add more possibilities to compress resource

Enumerator

NoneCompression	No compression at all.
zLibCompression	zlib compression
bz2LibCompression	bzip2 compression
lastCompression	Marker for last entry.

Definition at line 44 of file CompressDecompress.hxx.

7.1.2.2 EncryptionType

`enum lrc::EncryptionType`

Possible encryption types for resource

Todo Add more possibilities to encrypt resource

Enumerator

NoneEncryption	No encryption at all.
SerpentEncryption	Serpent algorithm for encryption.
lastEncryption	Marker for last entry.

Definition at line 44 of file EncryptDecrypt.hxx.

Chapter 8

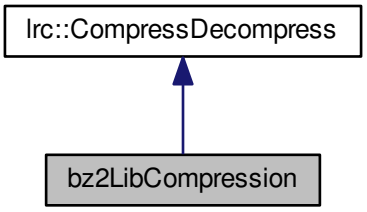
Class Documentation

8.1 bz2LibCompression Class Reference

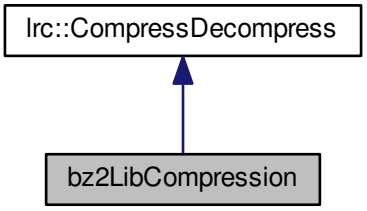
Compression class that uses the *bzip2* algorithm.

`#include <bz2LibCompression.hxx>`

Inheritance diagram for bz2LibCompression:



Collaboration diagram for bz2LibCompression:



Public Member Functions

- int compress (const unsigned char *, size_t, unsigned char **, size_t &)
  bzip2 compression
- int decompress (const unsigned char *, size_t, unsigned char **, size_t &)
  bzip2 decompression

8.1.1 Detailed Description

Compression class that uses the *bzip2* algorithm.

This class uses the bzip2 algorithm for compression and decompression of the resource data.

Definition at line 45 of file bz2LibCompression.hxx.

8.1.2 Member Function Documentation

8.1.2.1 compress()

```
int bz2LibCompression::compress (
            const unsigned char * ,
            size_t ,
            unsigned char ** ,
            size_t &  )  [virtual]
```

bzip2 compression

This method compresses the given data using the bzip2 algorithm

Implements lrc::CompressDecompress.

8.1.2.2 decompress()

```
int bz2LibCompression::decompress (
            const unsigned char * ,
            size_t ,
            unsigned char ** ,
            size_t &  )  [virtual]
```

bzip2 decompression

This method decompresses the given data using the bzip2 algorithm

Implements lrc::CompressDecompress.

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

- /home/andy/Programming/Projects/lrc/src/strategies/bz2LibCompression.hxx

8.2 Collector Class Reference

Class to collect all resource data.

`#include <Collector.hxx>`

Public Member Functions

- Collector (char *p_rcName, char *p_rdfName, bool p_overwriteAllow) throw (lrcFileExistsException)
  Constructor.
- ~Collector (void)
  Destructor.
- int collect (std::vector< ResourceData *> *p_resEntries, lrc::CompressionType p_compress, lrc::EncryptionType p_encrypt, const unsigned char *p_key)
  Collect, compress and encrypt data.

Private Member Functions

- int are_resIDs_unique (std::vector< ResourceData *> p_entries, inFilePosition &p_doubleIDPos, char **p_doubleID)
  Check for unique resource IDs.
- void show_resource_data_error (int p_errorCode, ResourceData *p_resData)
  Show error message from resource data.

Private Attributes

- char * m_rcName
  Filename of .rc file.
- char * m_rdfName
  Filename for .rdf file.

8.2.1 Detailed Description

Class to collect all resource data.

This class collects all files that are defined as resource data and generates one big file from them

Definition at line 47 of file Collector.hxx.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 Collector()

```
Collector::Collector (
            char * p_rcName,
            char * p_rdfName,
            bool p_overwriteAllow ) throw lrcFileExistsException)
```

Constructor.

Ths constructor is responsible to set up the class for collecting the resource data. It expects a filename for the .rdf file and a flag indicating whether or not overwriting is allowed

Parameters

in	*p_rcName*	Filename of rc file
in	*p_rdfName*	Filename for .rdf file
in	*p_overwriteAllow*	Flag indicating that overwriting is allowed (or not)

Exceptions

IrcFileExistsException	Exception that is thrown if the given .rdf file exists and the overwrite flag indicates that overwriting is forbidden

8.2.2.2 ~Collector()

```
Collector::~Collector (
            void  )
```

Destructor.

Clean up the memory that was needed by the class

8.2.3 Member Function Documentation

8.2.3.1 are_resIDs_unique()

```
int Collector::are_resIDs_unique (
            std::vector< ResourceData *> p_entries,
            inFilePosition & p_doubleIDPos,
            char ** p_doubleID )  [private]
```

Check for unique resource IDs.

This method checks if there are alle resource IDs unique and returns a warning if they are not

Parameters

in	*p_entries*	List of resource data (of type resEntry)
out	*p_doubleIDPos*	Position of double ID
out	*p_doubleID*	Resource ID that is not unique

Return values

NO_ERROR	All IDs are unique
WARNING_DOUBLE_RESOURCE_ID	A resource ID appears more than once

Return values

ERROR_INVALID_PARAMETER	One or more parameters are invalid

Remarks

The caller is responsible to free the string containing the name of the resource ID that appears more than once

8.2.3.2 collect()

```
int Collector::collect (
            std::vector< ResourceData *> * p_resEntries,
            lrc::CompressionType p_compress,
            lrc::EncryptionType p_encrypt,
            const unsigned char * p_key )
```

Collect, compress and encrypt data.

This method collects all resource data files, reads them into memory, compresses and encrypts it if desired and finally generates the one big resource file.

Parameters

in	*p_resEntries*	Collection of data resource entries
in	*p_compress*	Compression type for complete file
in	*p_encrypt*	Encryption type for complete file
in	*p_key*	Password if encryption is requested

Return values

NO_ERROR	Resource Data File successfully compiled
ERROR_FILE_OPEN	An error occurred opening the rdf file
ERROR_FILE_NOT_FOUND	The desired resource data file could not be found
ERROR_INVALID_PARAMETER	The provided parameter was nullptr
ERROR_FILE_NOT_FOUND	The resource file could not be found
ERROR_COMPRESSION_NOT_AVAILABLE	The selected compression is not available
ERROR_COMPRESSION_COMPRESS	An error occurred while compressing the complete file
ERROR_ENCRYPTION_NOT_AVAILABLE	The selected encryption is not availale
ERROR_ENCRYPTION_ENCRYPT	An error occurred while encrypting the complete file
ERROR_FILE_READ	An error occurred while reading the file
ERROR_FILE_WRITE	An error occurred while writing the file

8.2.3.3 show_resource_data_error()

```
void Collector::show_resource_data_error (
            int p_errorCode,
            ResourceData * p_resData )  [private]
```

Show error message from resource data.

This method shows an appropriate error message if the return value from the ResourceData class indicates an error

Parameters

in	*p_errorCode*	Error code
in	*p_resData*	ResourceData instance that caused the error

8.2.4 Member Data Documentation

8.2.4.1 m_rcName

```
char* Collector::m_rcName  [private]
```

Filename of .rc file.

Definition at line 50 of file Collector.hxx.

8.2.4.2 m_rdfName

```
char* Collector::m_rdfName  [private]
```

Filename for .rdf file.

Definition at line 51 of file Collector.hxx.

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

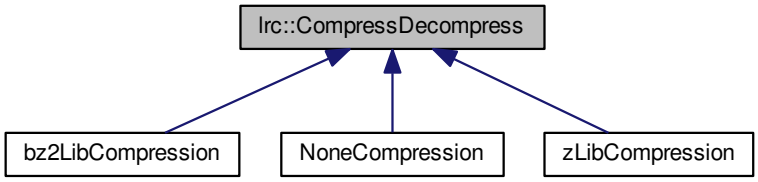
- /home/andy/Programming/Projects/lrc/src/compiler/Collector.hxx

8.3 lrc::CompressDecompress Class Reference

Compression and Decompression base class.

`#include <CompressDecompress.hxx>`

Inheritance diagram for lrc::CompressDecompress:



Public Member Functions

- virtual int compress (const unsigned char *p_decompData, size_t p_decompSize, unsigned char **p_↩ compData, size_t &p_compSize)=0

  Abstract method for compression.
- virtual int decompress (const unsigned char *p_compData, size_t p_compSize, unsigned char **p_↩ decompData, size_t &p_decompSize)=0

  Abstract method for decompression.

8.3.1 Detailed Description

Compression and Decompression base class.

The class CompressDecompress is the base class for all compression and decompression in `lrc` and `liblrc`.

Definition at line 62 of file CompressDecompress.hxx.

8.3.2 Member Function Documentation

8.3.2.1 compress()

```
virtual int lrc::CompressDecompress::compress (
            const unsigned char * p_decompData,
            size_t p_decompSize,
            unsigned char ** p_compData,
            size_t & p_compSize )  [pure virtual]
```

Abstract method for compression.

This abstract method has to be implemented by a dervied class for compression

Parameters

in	*p_decompData*	Not yet compressed resource data
in	*p_decompSize*	Size of the not yet compressed resource data
out	*p_compData*	Compressed resource data
out	*p_compSize*	Size of the compressed resource data

Return values

NO_ERROR	Data successfully compressed

Remarks

The caller is responsible to free the used memory

Implemented in NoneCompression, bz2LibCompression, and zLibCompression.

8.3.2.2 decompress()

```
virtual int lrc::CompressDecompress::decompress (
            const unsigned char * p_compData,
            size_t p_compSize,
            unsigned char ** p_decompData,
            size_t & p_decompSize )  [pure virtual]
```

Abstract method for decompression.

This abstract method has to be implemented by a derived class for decompression

Parameters

in	*p_compData*	Compressed resource data
in	*p_compSize*	Size of compressed resource data
out	*p_decompData*	Decompressed resource data
out	*p_decompSize*	Size of decompressed resource data

Return values

NO_ERROR	Data successfully decompressed

Remarks

The caller is responsible to free the used memory

Implemented in NoneCompression, bz2LibCompression, and zLibCompression.

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

- /home/andy/Programming/Projects/lrc/src/include/CompressDecompress.hxx

8.4 CompressionFactory Class Reference

Factory to create compression class instances.

`#include <Factories.hxx>`

Static Public Member Functions

- static lrc::CompressDecompress * get_compression_class (lrc::CompressionType p_compType)

  Creates an instance of the desired class.

8.4.1 Detailed Description

Factory to create compression class instances.

This class is used to create an instance of a lrc::CompressDecompress class. The created class instance compresses and decompresses the data, depending on the given lrc::CompressionType

Definition at line 48 of file Factories.hxx.

8.4.2 Member Function Documentation

8.4.2.1 get_compression_class()

```
static lrc::CompressDecompress* CompressionFactory::get_compression_class (
            lrc::CompressionType p_compType )  [static]
```

Creates an instance of the desired class.

This method returns an instance of the desired compression/decompression class (if possible) or `nullptr`

Parameters

in	*p_compType*	Compression type

Returns

Instance of desired compression class

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

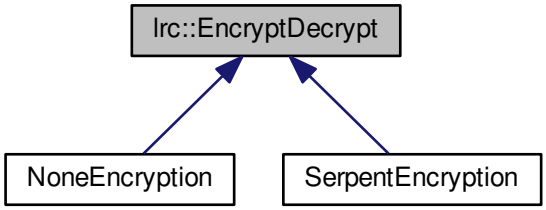
- /home/andy/Programming/Projects/lrc/src/Factories.hxx

8.5 lrc::EncryptDecrypt Class Reference

Encryption and Decryption base class.

`#include <EncryptDecrypt.hxx>`

Inheritance diagram for lrc::EncryptDecrypt:



Public Member Functions

- virtual int encrypt (const unsigned char *p_key, const unsigned char *p_clearData, size_t p_clearSize, unsigned char **p_encData, size_t &p_encSize)=0

  Abstract method for encryption.
- virtual int decrypt (const unsigned char *p_key, const unsigned char *p_encData, size_t p_encSize, unsigned char **p_clearData, size_t &p_clearSize)=0

  Abstract method for decryption.

8.5.1 Detailed Description

Encryption and Decryption base class.

This class is the base class for all encryption and decryption in `lrc` and `liblrc`

Definition at line 60 of file EncryptDecrypt.hxx.

8.5.2 Member Function Documentation

8.5.2.1 decrypt()

```
virtual int lrc::EncryptDecrypt::decrypt (
            const unsigned char * p_key,
            const unsigned char * p_encData,
            size_t p_encSize,
            unsigned char ** p_clearData,
            size_t & p_clearSize )  [pure virtual]
```

Abstract method for decryption.

This abstract method has to be implemented by a derived class for decryption

Parameters

in	*p_key*	Key for decryption
in	*p_encData*	Encrypted resource data for decryption
in	*p_encSize*	Size of encrypted resource data
out	*p_clearData*	Decrypted/clear resource data
out	*p_clearSize*	Size of clear resource data

Return values

NO_ERROR	Resource data successfully decrypted
ERROR_INVALID_PARAMETER	The return buffer p_clearData was `nullptr`

Remarks

The caller is responsible to free the used memory

Implemented in SerpentEncryption, and NoneEncryption.

8.5.2.2 encrypt()

```
virtual int lrc::EncryptDecrypt::encrypt (
            const unsigned char * p_key,
            const unsigned char * p_clearData,
            size_t p_clearSize,
            unsigned char ** p_encData,
            size_t & p_encSize )  [pure virtual]
```

Abstract method for encryption.

This abstract method has to be implemented by a derived class for encryption

Parameters

in	*p_key*	Key for encryption
in	*p_clearData*	Clear resource data for encryption
in	*p_clearSize*	Size of clear resource data
out	*p_encData*	Encrypted resource data
out	*p_encSize*	Size of encrypted data

Return values

NO_ERROR	Resource data successfully encrypted
ERROR_INVALID_PARAMETER	The return buffer p_encData was `nullptr`

Remarks

The caller is responsible to free the used memory

Implemented in SerpentEncryption, and NoneEncryption.

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

- /home/andy/Programming/Projects/lrc/src/include/EncryptDecrypt.hxx

8.6 EncryptionFactory Class Reference

Factory to create encryption class instances.

`#include <Factories.hxx>`

Static Public Member Functions

- static lrc::EncryptDecrypt * get_encryption_class (lrc::EncryptionType p_encType, char *p_resID) throw (lrc↩ EncryptionDisabledException)

  Creates an instance of the desired class.

8.6.1 Detailed Description

Factory to create encryption class instances.

This class is used to create an instance of a lrc::EncryptDecrypt class. The created class instance encrypts and decrypts the data, depending on the given lrc::EncryptionType

Definition at line 70 of file Factories.hxx.

8.6.2 Member Function Documentation

8.6.2.1 get_encryption_class()

```
static lrc::EncryptDecrypt* EncryptionFactory::get_encryption_class (
            lrc::EncryptionType p_encType,
            char * p_resID ) throw lrcEncryptionDisabledException)   [static]
```

Creates an instance of the desired class.

This method returns an instance of the desired encryption/decryption class (if possible or `nullptr` otherwise)

Parameters

in	*p_encType*	Encryption type
in	*p_resID*	ID of resource that requests encryption

Returns

Instance of desired encryption class

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

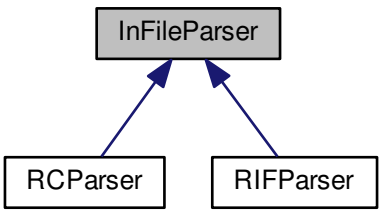
- /home/andy/Programming/Projects/lrc/src/Factories.hxx

8.7 InFileParser Class Reference

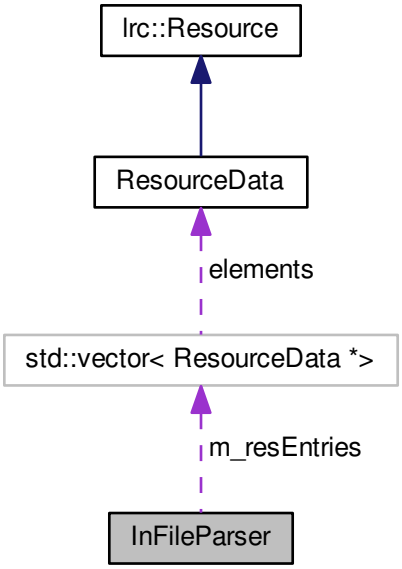
Base class for all `lrc` input files.

```
#include <InFileParser.hxx>
```

Inheritance diagram for InFileParser:



Collaboration diagram for InFileParser:



Public Member Functions

- InFileParser (char *p_filename) throw (IrcFileNotFoundException)

  Constructor.
- ~InFileParser (void)

  Destructor.
- virtual int parse (void)=0

  Parses the file.
- virtual int get_internal_error (inFilePosition &p_errPos, char **p_errMsg)

  Returns internal error.
- std::vector< ResourceData * > * get_resource_entries (void)

  Return all resource entries.

Static Public Member Functions

- static Irc::CompressionType eval_compression_type (const char *p_compStr)

  Evaluate compression type from string.
- static Irc::EncryptionType eval_encryption_type (const char *p_encStr)

  Evaluate encryption type from string.
- static unsigned char * get_password (const char *p_passwdStr) throw (IrcFileNotFoundException)

  Get password.

Protected Types

- enum internalErrorType {
  ieNone, ieInvalidElement, ieIdentNotFound, ieMissingPassword,
  ieFilenameNotFound, iePasswordFileNotFound, ieUnknownCompression, ieUnknownEncryption }

  Possible internal errors.

Protected Member Functions

- virtual void clear_internal_error (void)

  Clear all internal errors.

Protected Attributes

- char * m_filename

  Filename of the RC file.
- std::vector< ResourceData * > * m_resEntries

  List of resource data entries.
- inFilePosition m_errorPosition

  Line and column of error.
- int m_lastError

  Error code of last error.
- internalErrorType m_internalError

  Internal error.

8.7.1 Detailed Description

Base class for all `lrc` input files.

This class is the base class for all input files for the Linux Resource Compiler. All input file parser must derive from it.

Definition at line 47 of file InFileParser.hxx.

8.7.2 Member Enumeration Documentation

8.7.2.1 internalErrorType

```
enum InFileParser::internalErrorType  [protected]
```

Possible internal errors.

An enumeration of all possible internal errors

Enumerator

ieNone	No internal error.
ieInvalidElement	The root or any other element has a wrong name.
ieIdentNotFound	The identifier could not be found.
ieMissingPassword	An encryption is used, but no password is given.
ieFilenameNotFound	The actual resource filename is missing.
iePasswordFileNotFound	The password file could not be found.
ieUnknownCompression	The compression is incorrectly spelled or not available.
ieUnknownEncryption	The encryption is incorrectly spelled or not available.

Definition at line 54 of file InFileParser.hxx.

8.7.3 Constructor & Destructor Documentation

8.7.3.1 InFileParser()

```
InFileParser::InFileParser (
            char * p_filename ) throw lrcFileNotFoundException)
```

Constructor.

This is the constructor. It expects the name of the file to parse and initializes all the members of the class

Parameters

in	*p_filename*	Filename of input file

Exceptions

lrcFileNotFoundException	Exception that is thrown if the given file could not be found

8.7.3.2 ~InFileParser()

```
InFileParser::~InFileParser (
            void  )
```

Destructor.

Cleans up the memory and resources the parser class needed

Exceptions

lrcFileNotFoundException	This exception will be thrown if the given file for parsing could not be found

8.7.4 Member Function Documentation

8.7.4.1 clear_internal_error()

```
virtual void InFileParser::clear_internal_error (
            void  )  [protected], [virtual]
```

Clear all internal errors.

This method is used to clear all internal errors. It is virtual and should be overwritten by derived classes

8.7.4.2 eval_compression_type()

```
static lrc::CompressionType InFileParser::eval_compression_type (
            const char * p_compStr )  [static]
```

Evaluate compression type from string.

This method evaluates which compression type should be used, defined by the given string

Parameters

in	*p_compStr*	Compression type as string

Return values

lrc::NoneComression	No compression
lrc::zLibCompression	zlib compression

Remarks

lrc::NoneCompression is the default if the given string defines no other compression type

8.7.4.3 eval_encryption_type()

```
static lrc::EncryptionType InFileParser::eval_encryption_type (
            const char * p_encStr )  [static]
```

Evaluate encryption type from string.

This method evaluated which encryption type should be used defined by the given string

Parameters

in	*p_encStr*	Encryption type as string

Return values

lrc::NoneEncryption	No encryption
lrc::SerpentEncryption	Serpent encryption

Remarks

lrc::NoneEncryption is the default if the ginve string defines no other encryption type

8.7.4.4 get_internal_error()

```
virtual int InFileParser::get_internal_error (
            inFilePosition & p_errPos,
            char ** p_errMsg )  [virtual]
```

Returns internal error.

This method returns the state of the last internal error together with the line and column position where it happened and an error message

Remarks

The internal error state will be cleared after that call

Parameters

out	*p_errPos*	Line and column of error
out	*p_errMsg*	Error message

Returns

Error code of last error

8.7.4.5 get_password()

```
static unsigned char* InFileParser::get_password (
            const char * p_passwdStr ) throw lrcFileNotFoundException)   [static]
```

Get password.

This method gets the password, either directly from the .rc or .rif file or it is getting the password from a different file if the "@" notation (re-direct to a file) is used.

Parameters

in	*p_passwdStr*	Password string

Returns

Password directly from input file or from re-directed file

Exceptions

IrcFileNotFoundException	This exception will be thrown if the password should be read from a file and the given file could not be found

Remarks

The caller is responsible to free the used memory that the returned password needs

8.7.4.6 get_resource_entries()

```
std::vector<ResourceData *>* InFileParser::get_resource_entries (
            void  )
```

Return all resource entries.

This method returns all parsed resource entries in a vector of pointers to a ResourceData class

Returns

Resource entries

8.7.4.7 parse()

```
virtual int InFileParser::parse (
            void  )  [pure virtual]
```

Parses the file.

This method parses the file, creates a ResourceData class for each entry and adds them to the internal structure

Remarks

If the method returns ERROR_PARSE, get_internal_error will provide more information

Return values

NO_ERROR	File successfully parsed
ERROR_FILE_OPEN	An error occurred while trying to open the file
ERROR_PARSE	An error occurred while trying to parse the file

Implemented in RCParser, and RIFParser.

8.7.5 Member Data Documentation

8.7.5.1 m_errorPosition

```
inFilePosition InFileParser::m_errorPosition  [protected]
```

Line and column of error.

Definition at line 67 of file InFileParser.hxx.

8.7.5.2 m_filename

```
char* InFileParser::m_filename  [protected]
```

Filename of the RC file.

Definition at line 65 of file InFileParser.hxx.

8.7.5.3 m_internalError

```
internalErrorType InFileParser::m_internalError  [protected]
```

Internal error.

Definition at line 69 of file InFileParser.hxx.

8.7.5.4 m_lastError

```
int InFileParser::m_lastError  [protected]
```

Error code of last error.

Definition at line 68 of file InFileParser.hxx.

8.7.5.5 m_resEntries

```
std::vector<ResourceData *>* InFileParser::m_resEntries  [protected]
```

List of resource data entries.

Definition at line 66 of file InFileParser.hxx.

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

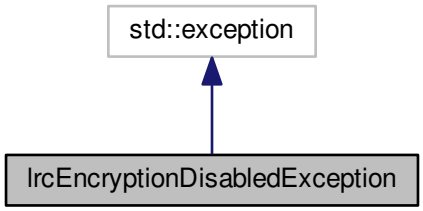
- /home/andy/Programming/Projects/Irc/src/compiler/InFileParser.hxx

8.8 IrcEncryptionDisabledException Class Reference

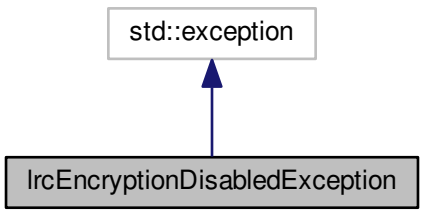
Exception if encryption is disabled but required.

```
#include <IrcExceptions.hxx>
```

Inheritance diagram for IrcEncryptionDisabledException:



Collaboration diagram for IrcEncryptionDisabledException:



Public Member Functions

- lrcEncryptionDisabledException (char *p_resourceID)

  Constructor.
- ~lrcEncryptionDisabledException (void) throw ()

  Destructor.
- virtual const char * what (void) const throw ()

  Gives a reason for the exception.

Private Attributes

- char * m_resourceID

  Resource that requires encryption.

8.8.1 Detailed Description

Exception if encryption is disabled but required.

This exception is thrown if the compiler is compiled without encryption support, but the defined input file (.rc or .rif) defines to encrypt a resource

Definition at line 111 of file lrcExceptions.hxx.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 lrcEncryptionDisabledException()

```
lrcEncryptionDisabledException::lrcEncryptionDisabledException (
            char * p_resourceID )
```

Constructor.

The constructor expects the name of the resource that requires encryption

Parameters

in	*p_resourceID*	Resource ID

8.8.2.2 ~lrcEncryptionDisabledException()

```
lrcEncryptionDisabledException::~lrcEncryptionDisabledException (
            void  ) throw )
```

Destructor.

Frees the memory of the exception

8.8.3 Member Function Documentation

8.8.3.1 what()

```
virtual const char* lrcEncryptionDisabledException::what (
            void  ) const throw )   [virtual]
```

Gives a reason for the exception.

Returns a message explaining that the user disabled encryption at compile time and therefore encryption of the resource is not possible

Returns

Reason for exception

8.8.4 Member Data Documentation

8.8.4.1 m_resourceID

```
char* lrcEncryptionDisabledException::m_resourceID  [private]
```

Resource that requires encryption.

Definition at line 114 of file lrcExceptions.hxx.

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

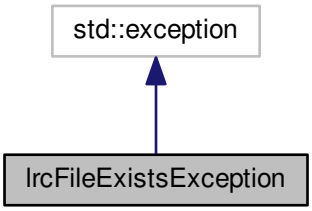
- /home/andy/Programming/Projects/lrc/src/lrcExceptions.hxx

8.9 lrcFileExistsException Class Reference

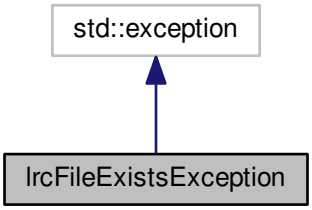
Exception if an existing file should be overwritten.

`#include <lrcExceptions.hxx>`

Inheritance diagram for lrcFileExistsException:



Collaboration diagram for lrcFileExistsException:



Public Member Functions

- lrcFileExistsException (char *p_filename)
  Constructor.
- ~lrcFileExistsException (void) throw ()
  Destructor.
- virtual const char * what (void) const throw ()
  Gives a reason for the exception.

Private Attributes

- char * m_fileOverwrite
  Filename of file that should have been overwritten.

8.9.1 Detailed Description

Exception if an existing file should be overwritten.

This exception is thrown if an existing file should be overwritten, but overwriting is not allowed

Definition at line 75 of file lrcExceptions.hxx.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 lrcFileExistsException()

```
lrcFileExistsException::lrcFileExistsException (
            char * p_filename )
```

Constructor.

The constructor expects the filename of the file that should be overwritten

Parameters

in	*p_filename*	File of file to be overwritten

8.9.2.2 ~lrcFileExistsException()

```
lrcFileExistsException::~lrcFileExistsException (
            void  ) throw )
```

Destructor.

Frees up memory needed by the class

8.9.3 Member Function Documentation

8.9.3.1 what()

```
virtual const char* lrcFileExistsException::what (
            void  ) const throw )   [virtual]
```

Gives a reason for the exception.

Returns the reason the exception

Returns

Reason for exception

8.9.4 Member Data Documentation

8.9.4.1 m_fileOverwrite

```
char* lrcFileExistsException::m_fileOverwrite  [private]
```

Filename of file that should have been overwritten.

Definition at line 78 of file lrcExceptions.hxx.

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

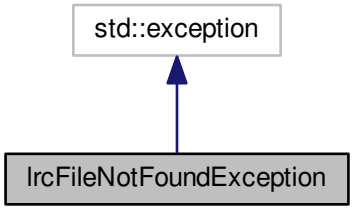
- /home/andy/Programming/Projects/lrc/src/lrcExceptions.hxx

8.10 lrcFileNotFoundException Class Reference

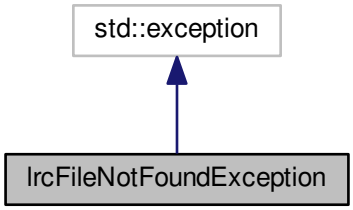
Exception class if a file could not be found.

```
#include <lrcExceptions.hxx>
```

Inheritance diagram for lrcFileNotFoundException:



Collaboration diagram for lrcFileNotFoundException:



Public Member Functions

- lrcFileNotFoundException (char *p_fileNotFoundName)
  Constructor.
- ~lrcFileNotFoundException (void) throw ()
  Destructor.
- virtual const char * what (void) const throw ()
  Method to return reason.

Private Attributes

- char * m_fileNotFound
  Filename of file that could not be found.

8.10.1 Detailed Description

Exception class if a file could not be found.

This is the exception that gets thrown if a file could not be found

Definition at line 41 of file lrcExceptions.hxx.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 lrcFileNotFoundException()

```
lrcFileNotFoundException::lrcFileNotFoundException (
            char * p_fileNotFoundName )
```

Constructor.

The constructor expects the filename of the file that could not be found

Parameters

in	*p_fileNotFoundName*	Name of file that could not be found

8.10.2.2 ~lrcFileNotFoundException()

```
lrcFileNotFoundException::~lrcFileNotFoundException (
            void  ) throw )
```

Destructor.

8.10.3 Member Function Documentation

8.10.3.1 what()

```
virtual const char* lrcFileNotFoundException::what (
            void  ) const throw )   [virtual]
```

Method to return reason.

The overwritten method what is used to return the reason of the exception

Returns

Reason of exception

8.10.4 Member Data Documentation

8.10.4.1 m_fileNotFound

```
char* lrcFileNotFoundException::m_fileNotFound  [private]
```

Filename of file that could not be found.

Definition at line 44 of file lrcExceptions.hxx.

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

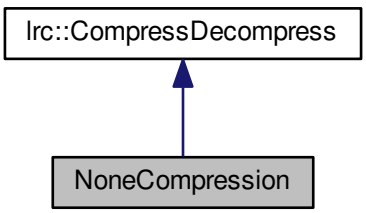
- /home/andy/Programming/Projects/lrc/src/lrcExceptions.hxx

8.11 NoneCompression Class Reference

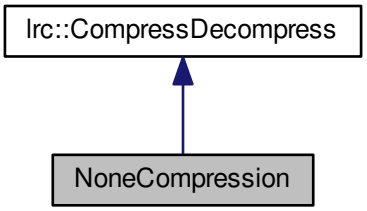
Compression class that does *NO* compression.

```
#include <NoneCompression.hxx>
```

Inheritance diagram for NoneCompression:



Collaboration diagram for NoneCompression:



Public Member Functions

- int compress (const unsigned char *, size_t, unsigned char **, size_t &)
  No compression.
- int decompress (const unsigned char *, size_t, unsigned char **, size_t &)
  No decompression.

8.11.1 Detailed Description

Compression class that does *NO* compression.

This class does no compression at all. It is used for the simplest case of no compression. It is created and implemented nonetheless to fit in the Strategy Pattern

Definition at line 46 of file NoneCompression.hxx.

8.11.2 Member Function Documentation

8.11.2.1 compress()

```
int NoneCompression::compress (
            const unsigned char * ,
            size_t ,
            unsigned char ** ,
            size_t &  )  [virtual]
```

No compression.

This method does no compression and returns a copy of the given data.

Implements Irc::CompressDecompress.

8.11.2.2 decompress()

```
int NoneCompression::decompress (
            const unsigned char * ,
            size_t ,
            unsigned char ** ,
            size_t &  )  [virtual]
```

No decompression.

Thos method does no decompression and returns a copy of the given data

Implements lrc::CompressDecompress.

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

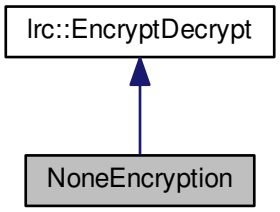
- /home/andy/Programming/Projects/lrc/src/strategies/NoneCompression.hxx

8.12 NoneEncryption Class Reference

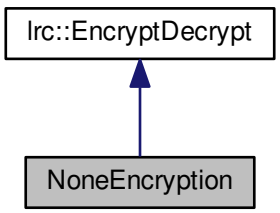
Encryption class that does *NO* encryption.

`#include <NoneEncryption.hxx>`

Inheritance diagram for NoneEncryption:



Collaboration diagram for NoneEncryption:



Public Member Functions

- int encrypt (const unsigned char *, const unsigned char *, size_t, unsigned char **, size_t &)
  No encryption.
- int decrypt (const unsigned char *, const unsigned char *, size_t, unsigned char **, size_t &)
  No decryption.

8.12.1 Detailed Description

Encryption class that does *NO* encryption.

This class does no encryption at all. It is used for the simplest case of no encryption. It is created and implemented nonetheless to fit in the Strategy Pattern

Definition at line 45 of file NoneEncryption.hxx.

8.12.2 Member Function Documentation

8.12.2.1 decrypt()

```
int NoneEncryption::decrypt (
            const unsigned char * ,
            const unsigned char * ,
            size_t ,
            unsigned char ** ,
            size_t &  )  [virtual]
```

No decryption.

This method does not decryption and returns a copy of the given data

Implements lrc::EncryptDecrypt.

8.12.2.2 encrypt()

```
int NoneEncryption::encrypt (
            const unsigned char * ,
            const unsigned char * ,
            size_t ,
            unsigned char ** ,
            size_t &  )  [virtual]
```

No encryption.

This method does no encryption and returns a copy of the given data

Implements lrc::EncryptDecrypt.

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

- /home/andy/Programming/Projects/lrc/src/strategies/NoneEncryption.hxx

8.13 ParserFactory Class Reference

Factory class to create an appropriate parser class.

`#include <ParserFactory.hxx>`

Static Public Member Functions

- static InFileParser * create_input_parser (const char *p_filename)

  Creates an appropiate parser class.

8.13.1 Detailed Description

Factory class to create an appropriate parser class.

This factory class is used to create a parser class that is able to parse a given input file. There are two possibilities: Parsing a .rc file or parsing a .rif file (xml).

Definition at line 46 of file ParserFactory.hxx.

8.13.2 Member Function Documentation

8.13.2.1 create_input_parser()

```
static InFileParser* ParserFactory::create_input_parser (
            const char * p_filename )  [static]
```

Creates an appropiate parser class.

This method creates a parser class depending on the given file. The file can be a .rc or a .rif (xml) file

Parameters

in	*p_filename*	Name of the file to parse

Returns

Instance of a macthing parser class

Remarks

The caller is responsible to free/delete the returned class
If the file does not exist `nullptr` will be returned

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

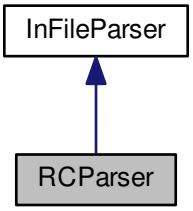
- /home/andy/Programming/Projects/lrc/src/compiler/ParserFactory.hxx

8.14 RCParser Class Reference

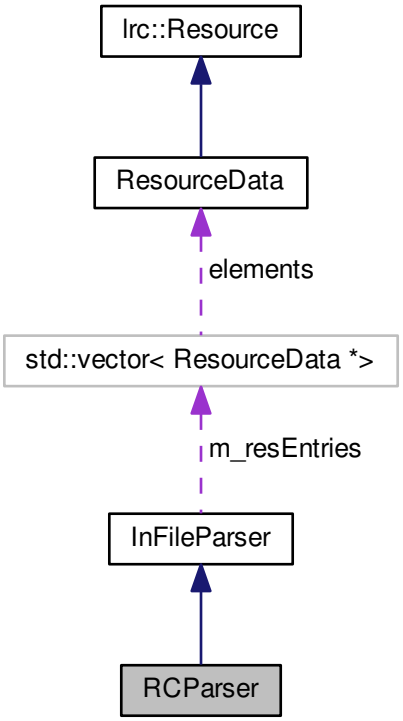
Class to parse an .rc file.

```
#include <RCParser.hxx>
```

Inheritance diagram for RCParser:



Collaboration diagram for RCParser:



Public Member Functions

- RCParser (char *p_filename) throw (lrcFileNotFoundException)

  Constructor.
- int parse (void)

  Parses the file.

Private Member Functions

- bool is_comment (char *p_line)

  Checks if the line is a comment line.
- bool is_windows_line (char *p_line)

  Checks if the line would be valid for Windows.
- void copy_mandatory_data (std::cmatch p_mandatoryMatch, bool p_winData, char **p_resIdent, char **p↩ _resFilename)

  Copy mandatory data in existing variables.

Additional Inherited Members

8.14.1 Detailed Description

Class to parse an .rc file.

Ths RCParser class is responsible to parse a RC file. The RC file format is quite simple and somewhat similar to the Windows rc files.

Definition at line 44 of file RCParser.hxx.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 RCParser()

```
RCParser::RCParser (
            char * p_filename ) throw lrcFileNotFoundException)
```

Constructor.

This is the only constructor of the class and it expects a filename as parameter

Parameters

in	*p_filename*	Filename of RC file

Exceptions

lrcFileNotFoundException	Exception that is thrown if the given .rc file could not be found

8.14.3 Member Function Documentation

8.14.3.1 copy_mandatory_data()

```
void RCParser::copy_mandatory_data (
            std::cmatch p_mandatoryMatch,
            bool p_winData,
            char ** p_resIdent,
            char ** p_resFilename )  [private]
```

Copy mandatory data in existing variables.

This method copies the mandatory data from a C++ regex match into the existing variables. It differs between a Windows and a normal lrc match

Parameters

in	*p_mandatoryMatch*	C++ match containing the data
in	*p_winData*	Flag indicating windows
out	*p_resIdent*	Resource identifier
out	*p_resFilename*	Resource filename

8.14.3.2 is_comment()

```
bool RCParser::is_comment (
            char * p_line )  [private]
```

Checks if the line is a comment line.

This method reads the given text as a text line from left to right abd checks if there are any characters at all and if there are, if they start with a '#'. An empty line or a line starting with '#' is treated as comment line

Parameters

in	*p_line*	Line of text

Return values

true	Line is comment or empty
false	Line contains information

8.14.3.3 is_windows_line()

```
bool RCParser::is_windows_line (
            char * p_line )  [private]
```

Checks if the line would be valid for Windows.

This method checks if the given line is a valid line for a Windows RC file. It will be called whenever the check for the mandatory lrc entries fail.

Parameters

in	*p_line*	Line of RC file

Return values

true	It is a valid line for Windows RC
false	It would also fail on Windows

8.14.3.4 parse()

```
int RCParser::parse (
            void  )  [virtual]
```

Parses the file.

This method parses the .rc file, creates a ResourceData class for each entry and adds them to the internal structure

Remarks

If the method returns ERROR_PARSE, get_internal_error will provide more information

Return values

NO_ERROR	File successfully parsed
ERROR_FILE_OPEN	An error occurred while trying to open the file
ERROR_PARSE	An error occurred while trying to parse the file

Implements InFileParser.

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

- /home/andy/Programming/Projects/lrc/src/compiler/RCParser.hxx

8.15 resEntry_ Struct Reference

Data for one resource entry.

`#include <Resource.hxx>`

Public Attributes

- char resID [MAX_ID_LEN]
  Resource ID.
- unsigned int startOffset
  Offset from the start.
- unsigned int resSize
  Size of resource.
- lrc::EncryptionType encType
  Type of encryption of this entry.
- lrc::CompressionType compType
  Type of compression of this entry.

8.15.1 Detailed Description

Data for one resource entry.

This struct contains the data for one resource entry in a form to save to the .rdf file

Definition at line 93 of file Resource.hxx.

8.15.2 Member Data Documentation

8.15.2.1 compType

`lrc::CompressionType resEntry_::compType`

Type of compression of this entry.

Definition at line 98 of file Resource.hxx.

8.15.2.2 encType

`lrc::EncryptionType resEntry_::encType`

Type of encryption of this entry.

Definition at line 97 of file Resource.hxx.

8.15.2.3 resID

```
char resEntry_::resID[MAX_ID_LEN]
```

Resource ID.

Definition at line 94 of file Resource.hxx.

8.15.2.4 resSize

```
unsigned int resEntry_::resSize
```

Size of resource.

Definition at line 96 of file Resource.hxx.

8.15.2.5 startOffset

```
unsigned int resEntry_::startOffset
```

Offset from the start.

Definition at line 95 of file Resource.hxx.

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

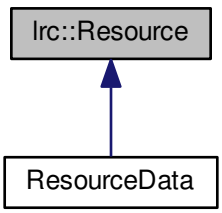
- /home/andy/Programming/Projects/lrc/src/include/Resource.hxx

8.16 lrc::Resource Class Reference

Resource class for use with library.

```
#include <Resource.hxx>
```

Inheritance diagram for lrc::Resource:



Public Member Functions

- Resource (void)
  Constructor.
- ~Resource (void)
  Destructor.
- char * get_ID (void)
  Returns the ID of the resource.
- unsigned char * get_res_data (void)
  Actual resource data.
- size_t get_res_size (void)
  Actual resource size.

Protected Attributes

- char * m_resID
  Resource ID.
- unsigned char * m_resData
  Pointer to resource data.
- size_t m_resSize
  Size of resource data.

8.16.1 Detailed Description

Resource class for use with library.

This class is used to define one resource for the developer that uses lrc. It contains a pointer to the data and the size of the data. It also provides its ID.

Definition at line 127 of file Resource.hxx.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 Resource()

```
lrc::Resource::Resource (
            void  )
```

Constructor.

This is a standard constructor. It initializes all members to default values

Remarks

The values will be filled by the derived class

8.16.2.2 ~Resource()

```
lrc::Resource::~Resource (
            void  )
```

Destructor.

Cleans up the used memory of the class

8.16.3 Member Function Documentation

8.16.3.1 get_ID()

```
char* lrc::Resource::get_ID (
            void  )
```

Returns the ID of the resource.

Method to return the ID of the resource

Returns

ID of resource

8.16.3.2 get_res_data()

```
unsigned char* lrc::Resource::get_res_data (
            void  )
```

Actual resource data.

This method returns a pointer to the actual resource data as bytes

Returns

Pointer to resource data

8.16.3.3 get_res_size()

```
size_t lrc::Resource::get_res_size (
            void  )
```

Actual resource size.

Method to return the actual size of the resource (in bytes)

Returns

Size of resource

8.16.4 Member Data Documentation

8.16.4.1 m_resData

```
unsigned char* lrc::Resource::m_resData  [protected]
```

Pointer to resource data.

Definition at line 131 of file Resource.hxx.

8.16.4.2 m_resID

```
char* lrc::Resource::m_resID  [protected]
```

Resource ID.

Definition at line 130 of file Resource.hxx.

8.16.4.3 m_resSize

```
size_t lrc::Resource::m_resSize  [protected]
```

Size of resource data.

Definition at line 132 of file Resource.hxx.

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

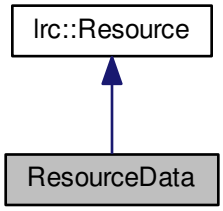
- /home/andy/Programming/Projects/lrc/src/include/Resource.hxx

8.17 ResourceData Class Reference

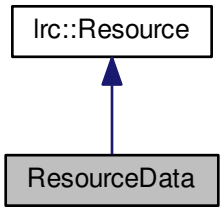
Internal class for resource with more information.

`#include <ResourceData.hxx>`

Inheritance diagram for ResourceData:



Collaboration diagram for ResourceData:



Public Member Functions

- ResourceData (void)
  Constructor.
- ~ResourceData (void)
  Destructor.
- void set_ident (const char *p_resIdent)
  Set resource identifier.
- void set_file (const char *p_resFilename)
  Set file containing resource.
- char * get_file (void)
  Return filename.
- void set_encryption (Irc::EncryptionType p_resEncryption, const unsigned char *p_password)

Set encryption type.

- lrc::EncryptionType get_encryption (void)

  Return encryption type.

- void set_compression (lrc::CompressionType p_resCompression)

  Set compression type.

- lrc::CompressionType get_compression (void)

  Return compression type.

- void set_rc_position (int p_line, int p_col)

  Set position of resource description.

- inFilePosition get_rc_position (void)

  Return resource description position.

- int prepare_resource_from_file (unsigned char **p_resData, size_t &p_resSize)

  Prepare resource for collecting.

- int get_data_from_memory (unsigned char *p_dataStart, resEntry p_resEntry, const unsigned char *p_↩ password=nullptr)

  Get resource data from compressed and encrypted chunk of memory.

- char * get_error_msg (void)

  Return error message.

Protected Member Functions

- void set_error_msg (char *p_newErrMsg)

  Set new error message.

Protected Attributes

- char * m_filename

  Filename of file containing resource.

- lrc::EncryptionType m_encryption

  Type of encryption for this resource.

- lrc::CompressionType m_compression

  Type of compression for this resource.

- inFilePosition m_inFilePosition

  Position of resource in RC file.

- char * m_errorMsg

  Error message (if any)

Private Attributes

- unsigned char * m_password

  Password for encryption.

8.17.1 Detailed Description

Internal class for resource with more information.

This class contains all needed data of a resource. It is derived from Resource

Definition at line 53 of file ResourceData.hxx.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 ResourceData()

```
ResourceData::ResourceData (
            void  )
```

Constructor.

Standard constructor to initialize the class

8.17.2.2 ~ResourceData()

```
ResourceData::~ResourceData (
            void  )
```

Destructor.

Cleans up memory used from the class

8.17.3 Member Function Documentation

8.17.3.1 get_compression()

```
lrc::CompressionType ResourceData::get_compression (
            void  )
```

Return compression type.

Method to return the compression type of this very resource

Returns

Compression type of resource

8.17.3.2 get_data_from_memory()

```
int ResourceData::get_data_from_memory (
            unsigned char * p_dataStart,
            resEntry p_resEntry,
            const unsigned char * p_password = nullptr )
```

Get resource data from compressed and encrypted chunk of memory.

This method extracts the information of the class from a compressed and encrypted chunk of memory. It is usually used after the data is loaded from file

Parameters

in	*p_dataStart*	Start adress of memory block
in	*p_resEntry*	Entry block of resource
in	*p_password*	Password if resource is encrypted

Return values

NO_ERROR	Resource data successfully extracted
ERROR_INVALID_PARAMETER	One or more provided parameters
ERROR_COMPRESSION_NOT_AVAILABLE	The required compression class is not available
ERROR_ENCRYPTION_NOT_AVAILABLE	The required encryption class is not available
ERROR_ENCRYPTION_DECRYPT	An error occurred while decrypting the data
ERROR_COMPRESSION_DECOMPRESS	An error occurred while decompressing the data

Remarks

The caller is responsible to free the returned class

8.17.3.3 get_encryption()

```
lrc::EncryptionType ResourceData::get_encryption (
            void  )
```

Return encryption type.

Method to return the encryption type of this very resource

Returns

Encryption type of resource

8.17.3.4 get_error_msg()

```
char* ResourceData::get_error_msg (
            void  )
```

Return error message.

This method returns the internal error message (if there is any) and clears it

Returns

Internal error message

Remarks

The caller is responsible to free the used memory for the message

8.17.3.5 get_file()

```
char* ResourceData::get_file (
            void  )
```

Return filename.

This method returns the filename of the resource data file of this very resource

Returns

Filename of resource data file

8.17.3.6 get_rc_position()

```
inFilePosition ResourceData::get_rc_position (
            void  )
```

Return resource description position.

Method to return the resource description in the RC file of this very resource

Returns

Resource position in RC file

8.17.3.7 prepare_resource_from_file()

```
int ResourceData::prepare_resource_from_file (
            unsigned char ** p_resData,
            size_t & p_resSize )
```

Prepare resource for collecting.

This method loads the file containing the resource, compresses and encrypts the data and provides it for the collectors

Parameters

out	*p_resData*	Resource data for collector
out	*p_resSize*	Size of resource data

Return values

NO_ERROR	Resource successfully prepared
ERROR_INVALID_PARAMETER	The provided parameter was nullptr

Return values

ERROR_FILE_NOT_FOUND	The resource file could not be found
ERROR_COMPRESSION_NOT_AVAILABLE	The selected compression is not available
ERROR_ENCRYPTION_NOT_AVAILABLE	The selected encryption is not availale
ERROR_FILE_OPEN	The file could not be opened
ERROR_FILE_READ	An error occurred while reading the file

Remarks

The caller is responsible to free the used memory

8.17.3.8 set_compression()

```
void ResourceData::set_compression (
            lrc::CompressionType p_resCompression )
```

Set compression type.

Method to set compression type of resource

Parameters

in	*p_resCompression*	Compression type

8.17.3.9 set_encryption()

```
void ResourceData::set_encryption (
            lrc::EncryptionType p_resEncryption,
            const unsigned char * p_password )
```

Set encryption type.

Method to set encryption type of resource

Parameters

in	*p_resEncryption*	Encryption type
in	*p_password*	Password for encryption

8.17.3.10 set_error_msg()

```
void ResourceData::set_error_msg (
            char * p_newErrMsg )  [protected]
```

Set new error message.

This method clears an old error message (if any) and sets the new given one

Parameters

in	*p_newErrMsg*	New error message

8.17.3.11 set_file()

```
void ResourceData::set_file (
            const char * p_resFilename )
```

Set file containing resource.

Method to set the filename containing the resource data

Parameters

in	*p_resFilename*	Filename containing resource

8.17.3.12 set_ident()

```
void ResourceData::set_ident (
            const char * p_resIdent )
```

Set resource identifier.

Method to set the internal resource identifier

Parameters

in	*p_resIdent*	Resource identifier

8.17.3.13 set_rc_position()

```
void ResourceData::set_rc_position (
```

```
int p_line,
int p_col )
```

Set position of resource description.

Method to set the position of the resource description in the RC file. It is used for a sane error message in the collecting process

Parameters

in	*p_line*	Line number in RC file
in	*p_col*	Column number in RC file

See also

Collector

8.17.4 Member Data Documentation

8.17.4.1 m_compression

```
lrc::CompressionType ResourceData::m_compression  [protected]
```

Type of compression for this resource.

Definition at line 61 of file ResourceData.hxx.

8.17.4.2 m_encryption

```
lrc::EncryptionType ResourceData::m_encryption  [protected]
```

Type of encryption for this resource.

Definition at line 60 of file ResourceData.hxx.

8.17.4.3 m_errorMsg

```
char* ResourceData::m_errorMsg  [protected]
```

Error message (if any)

Definition at line 64 of file ResourceData.hxx.

8.17.4.4 m_filename

```
char* ResourceData::m_filename  [protected]
```

Filename of file containing resource.

Definition at line 59 of file ResourceData.hxx.

8.17.4.5 m_inFilePosition

```
inFilePosition ResourceData::m_inFilePosition  [protected]
```

Position of resource in RC file.

Definition at line 62 of file ResourceData.hxx.

8.17.4.6 m_password

```
unsigned char* ResourceData::m_password  [private]
```

Password for encryption.

Definition at line 56 of file ResourceData.hxx.

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

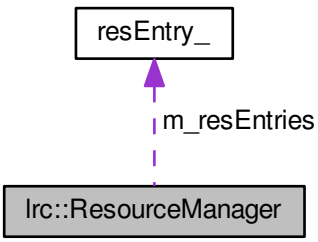
- /home/andy/Programming/Projects/lrc/src/ResourceData.hxx

8.18 lrc::ResourceManager Class Reference

Manager class handling all resources of a resource file.

```
#include <ResourceManager.hxx>
```

Collaboration diagram for lrc::ResourceManager:



Public Member Functions

- ResourceManager (const char *p_resFilename, const CompressionType p_compress, const EncryptionType p_encrypt, const unsigned char *p_key) throw (IrcFileNotFoundException)

  Constructor expecting resource file name.
- ResourceManager (const unsigned char *p_startAddr, const unsigned char *p_endAddr, const CompressionType p_compress, const EncryptionType p_encrypt, const unsigned char *p_key)

  Constructor for embedded resources.
- ~ResourceManager (void)

  Destructor.
- char ** get_resource_ids (int &p_numRes)

  Returns a list of all resources by ID.
- Resource * get_resource (const char *p_resID, const unsigned char *p_password=nullptr)

  Returns the requested resource.
- Resource * get_resource (unsigned int p_resIdx, const unsigned char *p_password=nullptr)

  Returns the resource at the index.

Private Member Functions

- int decrypt_data (const unsigned char *p_encData, size_t p_encSize, unsigned char **p_clearData, size_t &p_clearSize)

  Decrypt resource data.
- int decompress_data (const unsigned char *p_compData, size_t p_compSize, unsigned char **p_decomp↩ Data, size_t &p_decompSize)

  Decompress resource data.
- int setup_resources (const unsigned char *p_resData, size_t p_resSize)

  Setup new resources.
- int load_from_file (void)

  Method to load resource file.
- int load_embedded (const unsigned char *p_startAddr, const unsigned char *p_endAddr)

  Method to load embedded resource.

Private Attributes

- char * m_resourceFile

  Filename of resource file.
- resEntry * m_resEntries

  Entries of the resource file.
- int m_numResEntries

  Number of entries in resource file.
- unsigned char * m_resData

  Pointer to data of resource file.
- size_t m_resDataSize

  Size of resource data.
- CompressionType m_compType

  Compression type for complete file.
- EncryptionType m_encType

  Encryption type for complete file.
- unsigned char * m_password

  Password is complete file is encrypted.

8.18.1 Detailed Description

Manager class handling all resources of a resource file.

This is the main class of the `liblrc` library. It expects a filename of a resource file in the constructor and handles all resources in that given file. If the constructor with no filename is used, it expects the resource data is directly compiled into the executable file.

Definition at line 52 of file ResourceManager.hxx.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 ResourceManager() [1/2]

```
lrc::ResourceManager::ResourceManager (
            const char * p_resFilename,
            const CompressionType p_compress,
            const EncryptionType p_encrypt,
            const unsigned char * p_key ) throw lrcFileNotFoundException)
```

Constructor expecting resource file name.

The constructor expects the name of a resource file. The file will be loaded at first use, but an exception will be thrown if it does not exist

Parameters

in	*p_resFilename*	Filename of resource file
in	*p_compress*	Compression type if the complete file is compressed
in	*p_encrypt*	Encryption type if the complete file is encrypted
in	*p_key*	Password if whole file is encrypted

Exceptions

lrcFileNotFoundException	Will be thrown if the file does not exist

8.18.2.2 ResourceManager() [2/2]

```
lrc::ResourceManager::ResourceManager (
            const unsigned char * p_startAddr,
            const unsigned char * p_endAddr,
            const CompressionType p_compress,
            const EncryptionType p_encrypt,
            const unsigned char * p_key )
```

Constructor for embedded resources.

The constructor expects the start and end address of the resource data in memory and the compression and encryption type if the complete resource is compressed and encrypted respectively. If it is encrypted a password is expected.

Parameters

in	*p_startAddr*	Start address of resource data in memory
in	*p_endAddr*	End address of resource data in memory
in	*p_compress*	Compression type if the complete resource data is compressed
in	*p_encrypt*	Encryption type if the complete resource data is encrypted
in	*p_key*	Password if the complete resource is encrypted

Remarks

How to get to the start and end address in the linked file, see the Wiki or the man page

8.18.2.3 ~ResourceManager()

```
lrc::ResourceManager::~ResourceManager (
            void  )
```

Destructor.

The destructor cleans up all the used memory of the class

8.18.3 Member Function Documentation

8.18.3.1 decompress_data()

```
int lrc::ResourceManager::decompress_data (
            const unsigned char * p_compData,
            size_t p_compSize,
            unsigned char ** p_decompData,
            size_t & p_decompSize )  [private]
```

Decompress resource data.

This method decompresses the resource data in the memory

Parameters

in	*p_compData*	Compressed resource data
in	*p_compSize*	Size of compressed resource data
out	*p_decompData*	Decompressed resource data
out	*p_decompSize*	Size of decompressed resource data

Return values

NO_ERROR	Data successfully decompressed
ERROR_COMPRESSION_DECOMPRESS	An error occurred while decompressing the data

8.18.3.2 decrypt_data()

```
int lrc::ResourceManager::decrypt_data (
            const unsigned char * p_encData,
            size_t p_encSize,
            unsigned char ** p_clearData,
            size_t & p_clearSize )  [private]
```

Decrypt resource data.

This methods decrypts the resource data in the memory

Parameters

in	*p_encData*	Encrypted resource data for decryption
in	*p_encSize*	Size of encrypted resource data
out	*p_clearData*	Decrypted/clear resource data
out	*p_clearSize*	Size of clear resource data

Return values

NO_ERROR	Data successfully decrypted
ERROR_ENCRYPTION_NOT_AVAILABLE	Encryption/Decryption support is not compiled in
ERROR_ENCRYPTION_DECRYPT	An error occurred while decrypting the data

8.18.3.3 get_resource() [1/2]

```
Resource* lrc::ResourceManager::get_resource (
            const char * p_resID,
            const unsigned char * p_password = nullptr )
```

Returns the requested resource.

This method returns the requested resource defined by the given resource ID or `nullptr` if it does not exist

Parameters

in	*p_resID*	Resource ID
in	*p_password*	Password if resource is encrypted

Returns

Instance of Resource class of the requested resource (or `nullptr` if it does not exist)

Remarks

The caller is responsible to free the used memory

8.18.3.4 get_resource() [2/2]

```
Resource* lrc::ResourceManager::get_resource (
            unsigned int p_resIdx,
            const unsigned char * p_password = nullptr )
```

Returns the resource at the index.

This method returns the requested resource defined by the given index or `nullptr` if the index is out of range

Parameters

in	*p_resIdx*	Index of resource
in	*p_password*	Password if resource is encrypted

Returns

Instance of Resource class of the resource at the index (or `nullptr` if the index is out of range)

Remarks

Indices are zero based, i.e. valid indices are between 0 and n-1 (n means the number of all resources)
The caller is responsible to free the used memory

8.18.3.5 get_resource_ids()

```
char** lrc::ResourceManager::get_resource_ids (
            int & p_numRes )
```

Returns a list of all resources by ID.

This method returns a list of all resources. The list contains all IDs of the resources

Parameters

out	*p_numRes*	Number of resources in list

Remarks

The caller is resposible to free the used memory

8.18.3.6 load_embedded()

```
int lrc::ResourceManager::load_embedded (
            const unsigned char * p_startAddr,
            const unsigned char * p_endAddr )  [private]
```

Method to load embedded resource.

This method loads all embedded resource data

Parameters

in	*p_startAddr*	Start address of resource data in memory
in	*p_endAddr*	End address of resource data in memory

Return values

NO_ERROR	Embedded resources successfully accessed
ERROR_ENCRYPTION_NOT_AVAILABLE	Encryption/Decryption support is not compiled in
ERROR_ENCRYPTION_DECRYPT	An error occurred while decrypting the data
ERROR_COMPRESSION_DECOMPRESS	An error occurred while decompressing the data

8.18.3.7 load_from_file()

```
int lrc::ResourceManager::load_from_file (
            void  )  [private]
```

Method to load resource file.

This method loads all data from the resource file

Return values

NO_ERROR	Resource file successfully loaded
ERROR_FILE_READ	An error occurred while reading the file
ERROR_ENCRYPTION_NOT_AVAILABLE	Encryption/Decryption support is not compiled in
ERROR_ENCRYPTION_DECRYPT	An error occurred while decrypting the data
ERROR_COMPRESSION_DECOMPRESS	An error occurred while decompressing the data

8.18.3.8 setup_resources()

```
int lrc::ResourceManager::setup_resources (
            const unsigned char * p_resData,
            size_t p_resSize )  [private]
```

Setup new resources.

This method sets up internal data to provide the resource data in the memory

Parameters

in	*p_resData*	Decrypted and decompressed resource data
in	*p_resSize*	Size of resource data

Return values

NO_ERROR	Internal data successfully set up

8.18.4 Member Data Documentation

8.18.4.1 m_compType

```
CompressionType lrc::ResourceManager::m_compType  [private]
```

Compression type for complete file.

Definition at line 60 of file ResourceManager.hxx.

8.18.4.2 m_encType

```
EncryptionType lrc::ResourceManager::m_encType  [private]
```

Encryption type for complete file.

Definition at line 61 of file ResourceManager.hxx.

8.18.4.3 m_numResEntries

```
int lrc::ResourceManager::m_numResEntries  [private]
```

Number of entries in resource file.

Definition at line 57 of file ResourceManager.hxx.

8.18.4.4 m_password

```
unsigned char* lrc::ResourceManager::m_password  [private]
```

Password is complete file is encrypted.

Definition at line 62 of file ResourceManager.hxx.

8.18.4.5 m_resData

```
unsigned char* lrc::ResourceManager::m_resData  [private]
```

Pointer to data of resource file.

Definition at line 58 of file ResourceManager.hxx.

8.18.4.6 m_resDataSize

```
size_t lrc::ResourceManager::m_resDataSize  [private]
```

Size of resource data.

Definition at line 59 of file ResourceManager.hxx.

8.18.4.7 m_resEntries

```
resEntry* lrc::ResourceManager::m_resEntries  [private]
```

Entries of the resource file.

Definition at line 56 of file ResourceManager.hxx.

8.18.4.8 m_resourceFile

```
char* lrc::ResourceManager::m_resourceFile  [private]
```

Filename of resource file.

Definition at line 55 of file ResourceManager.hxx.

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

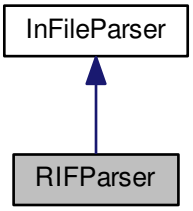
- /home/andy/Programming/Projects/lrc/src/include/ResourceManager.hxx

8.19 RIFParser Class Reference

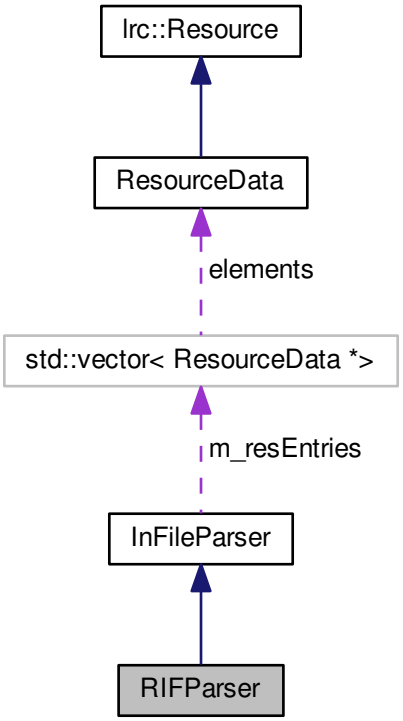
Class to parse a .rif (XML) file.

`#include <RIFParser.hxx>`

Inheritance diagram for RIFParser:



Collaboration diagram for RIFParser:



Public Member Functions

- RIFParser (char *p_filename) throw (lrcFileNotFoundException)

  Constructor.
- int parse (void)

  Parses the file.

Additional Inherited Members

8.19.1 Detailed Description

Class to parse a .rif (XML) file.

This class is used to parse a .rif file. The .rif file is actually a XML file

Definition at line 44 of file RIFParser.hxx.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 RIFParser()

```
RIFParser::RIFParser (
            char * p_filename ) throw lrcFileNotFoundException)
```

Constructor.

This is the only constructor of the class and it expects a filename as parameter

Parameters

in	*p_filename*	Filename of .rif file

Exceptions

lrcFileNotFoundException	Exception that is thrown if the given .rif file could not be found

8.19.3 Member Function Documentation

8.19.3.1 parse()

```
int RIFParser::parse (
            void  )  [virtual]
```

Parses the file.

This method parses the .rif file, creates a ResourceData class for each entry and adds them to the internal structure

Remarks

If the method returns ERROR_PARSE, get_internal_error will provide more information

Return values

NO_ERROR	File successfully parsed
ERROR_FILE_OPEN	An error occurred while trying to open the file
ERROR_PARSE	An error occurred while trying to parse the file

Implements InFileParser.

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

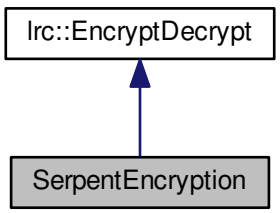
- /home/andy/Programming/Projects/lrc/src/compiler/RIFParser.hxx

8.20 SerpentEncryption Class Reference

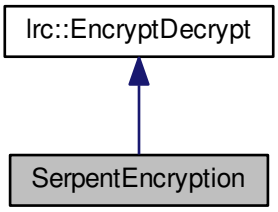
Class to encrypt/decrypt using the *Serpent* algorithm.

```
#include <SerpentEncryption.hxx>
```

Inheritance diagram for SerpentEncryption:



Collaboration diagram for SerpentEncryption:



Public Member Functions

- int encrypt (const unsigned char *p_key, const unsigned char *p_clearData, size_t p_clearSize, unsigned char **p_encData, size_t &p_encSize)

  Serpent encryption.
- int decrypt (const unsigned char *p_key, const unsigned char *p_encData, size_t p_encSize, unsigned char **p_clearData, size_t &p_clearSize)

  Serpent decryption.

Protected Member Functions

- int create_initialization_vector (unsigned char *p_ivArray, size_t p_ivSize)

  Create initialization vector.

8.20.1 Detailed Description

Class to encrypt/decrypt using the *Serpent* algorithm.

This class encrypts and decrypts the given data using the *Serpent* algorithm. The algorithm is not implemented here, instead the algorithm from the crypto++ library is used

Definition at line 47 of file SerpentEncryption.hxx.

8.20.2 Member Function Documentation

8.20.2.1 create_initialization_vector()

```
int SerpentEncryption::create_initialization_vector (
            unsigned char * p_ivArray,
            size_t p_ivSize )  [protected]
```

Create initialization vector.

This method creates a random initialization vector for the Serpent encryption

Parameters

in,out	*p_ivArray*	Array for initialization vector
in	*p_ivSize*	Size of the array

Return values

NO_ERROR	Initialization vector successfully created
ERROR_INVALID_PARAMETER	The array was not created

8.20.2.2 decrypt()

```
int SerpentEncryption::decrypt (
            const unsigned char * p_key,
            const unsigned char * p_encData,
            size_t p_encSize,
            unsigned char ** p_clearData,
            size_t & p_clearSize )  [virtual]
```

Serpent decryption.

Method to decrypt the given data using the Serpent algorithm

Implements Irc::EncryptDecrypt.

8.20.2.3 encrypt()

```
int SerpentEncryption::encrypt (
            const unsigned char * p_key,
            const unsigned char * p_clearData,
            size_t p_clearSize,
            unsigned char ** p_encData,
            size_t & p_encSize )  [virtual]
```

Serpent encryption.

Method to encrypt the given data using the Serpent algorithm

Implements Irc::EncryptDecrypt.

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

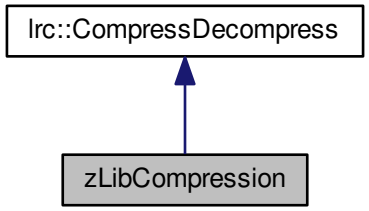
- /home/andy/Programming/Projects/Irc/src/strategies/SerpentEncryption.hxx

8.21 zLibCompression Class Reference

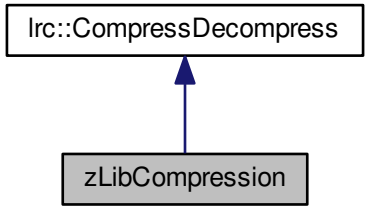
Compression class that uses the *zLib* algorithm.

`#include <zLibCompression.hxx>`

Inheritance diagram for zLibCompression:



Collaboration diagram for zLibCompression:



Public Member Functions

- int compress (const unsigned char *, size_t, unsigned char **, size_t &)
  zLib compression
- int decompress (const unsigned char *, size_t, unsigned char **, size_t &)
  zLib decompression

8.21.1 Detailed Description

Compression class that uses the *zLib* algorithm.

This class uses the zLib algorithm for compression and decompression of the resource data.

Definition at line 45 of file zLibCompression.hxx.

8.21.2 Member Function Documentation

8.21.2.1 compress()

```
int zLibCompression::compress (
            const unsigned char * ,
            size_t ,
            unsigned char ** ,
            size_t &  )  [virtual]
```

zLib compression

This method compresses the given data using the zLib algorithm

Implements lrc::CompressDecompress.

8.21.2.2 decompress()

```
int zLibCompression::decompress (
            const unsigned char * ,
            size_t ,
            unsigned char ** ,
            size_t &  )  [virtual]
```

zLib decompression

This method decompresses the given data using the zLib algorithm

Implements lrc::CompressDecompress.

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

- /home/andy/Programming/Projects/lrc/src/strategies/zLibCompression.hxx

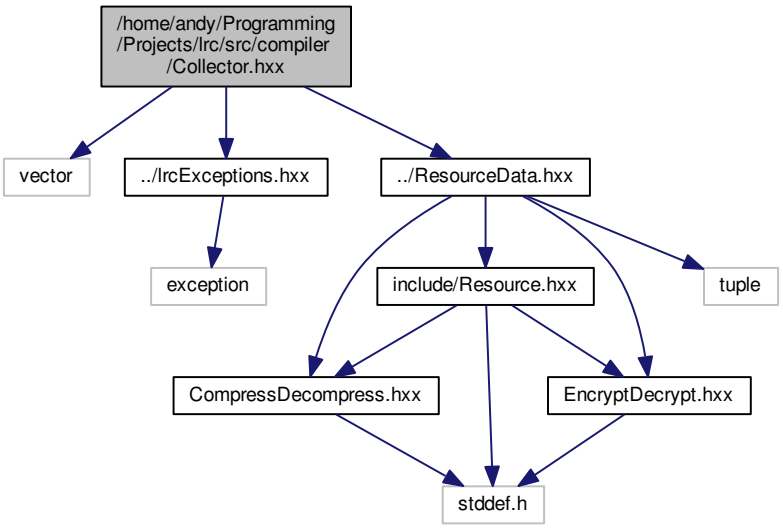
Chapter 9

File Documentation

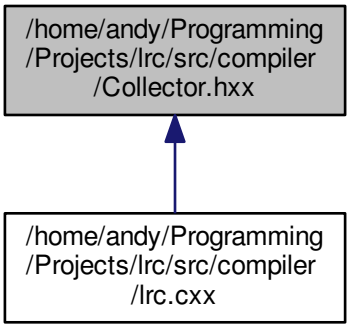
9.1 /home/andy/Programming/Projects/lrc/src/compiler/Collector.hxx File Reference

```
#include <vector>
#include "../lrcExceptions.hxx"
#include "../ResourceData.hxx"
```

Include dependency graph for Collector.hxx:



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



Classes

- class Collector

  Class to collect all resource data.

9.1.1 Detailed Description

This file contains the class definition of the collector class that is responsible to collect all resource data

Author

Andreas Tscharner

Date

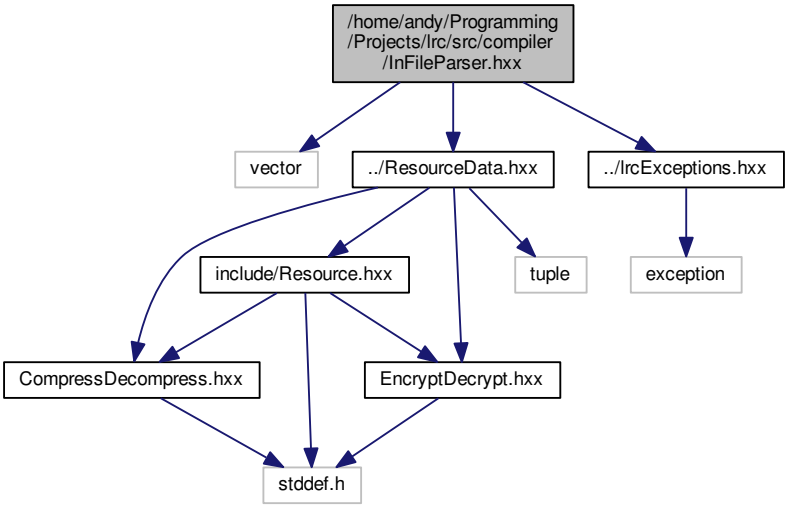
2014-08-23

9.2 /home/andy/Programming/Projects/lrc/src/compiler/lnFileParser.hxx File Reference

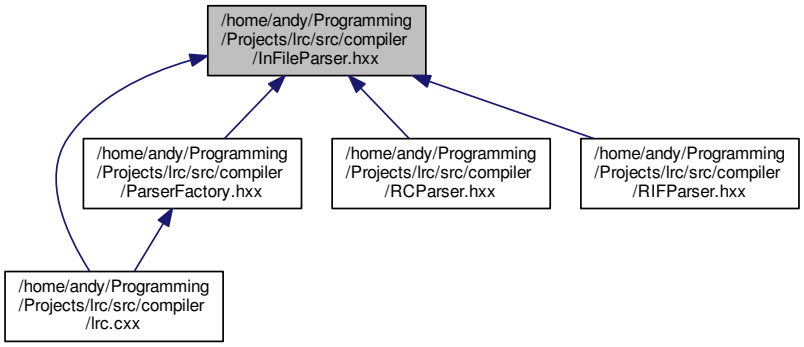
```
#include <vector>
#include "../ResourceData.hxx"
```

```
#include "../lrcExceptions.hxx"
```

Include dependency graph for InFileParser.hxx:



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



Classes

- class InFileParser

  Base class for all `lrc` input files.

9.2.1 Detailed Description

This file contains the base class for all parsers that read and parse an input file for the Linux Resource Compiler

Author

Andreas Tscharner

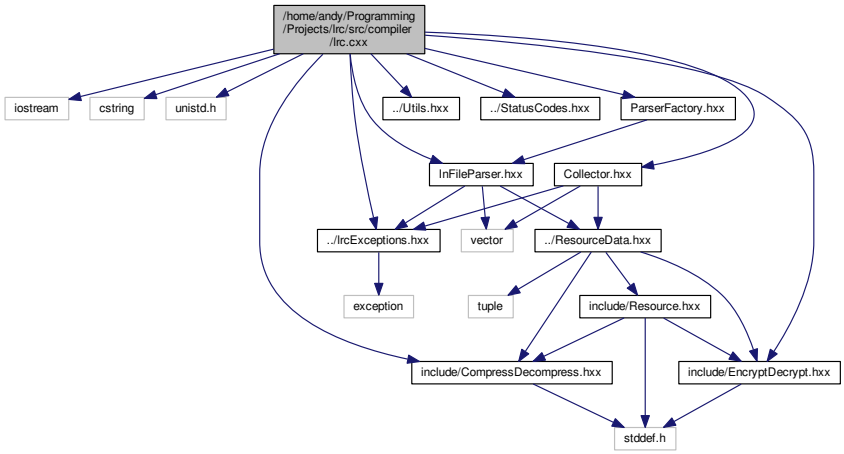
Date

2014-10-26

9.3 /home/andy/Programming/Projects/lrc/src/compiler/lrc.cxx File Reference

```
#include <iostream>
#include <cstring>
#include <unistd.h>
#include "include/CompressDecompress.hxx"
#include "include/EncryptDecrypt.hxx"
#include "../lrcExceptions.hxx"
#include "../Utils.hxx"
#include "../StatusCodes.hxx"
#include "InFileParser.hxx"
#include "Collector.hxx"
#include "ParserFactory.hxx"
```

Include dependency graph for lrc.cxx:



Macros

- #define VERSION "<undefined>"

  Define for version number if generated header file is missing.

Functions

- void usage (int argc, char **argv)

  Show the usage of the compiler.
- int convert_to_elf (const char *p_rdfFilename, const char *p_elfFilename)

  Convert data to ELF binary format.
- int main (int argc, char **argv)

  Main program.

9.3.1 Detailed Description

This is the main file for the compiler. It calls the appropiate parser and then the collector which creates the .rdf file

Author

Andreas Tscharner

Date

2014-08-23

9.3.2 Macro Definition Documentation

9.3.2.1 VERSION

```
#define VERSION "<undefined>"
```

Define for version number if generated header file is missing.

Definition at line 46 of file lrc.cxx.

9.3.3 Function Documentation

9.3.3.1 convert_to_elf()

```
int convert_to_elf (
            const char * p_rdfFilename,
            const char * p_elfFilename )
```

Convert data to ELF binary format.

Definition at line 72 of file lrc.cxx.

9.3.3.2 main()

```
int main (
            int argc,
            char ** argv )
```

Main program.

Definition at line 103 of file lrc.cxx.

9.3.3.3 usage()

```
void usage (
             int argc,
             char ** argv )
```

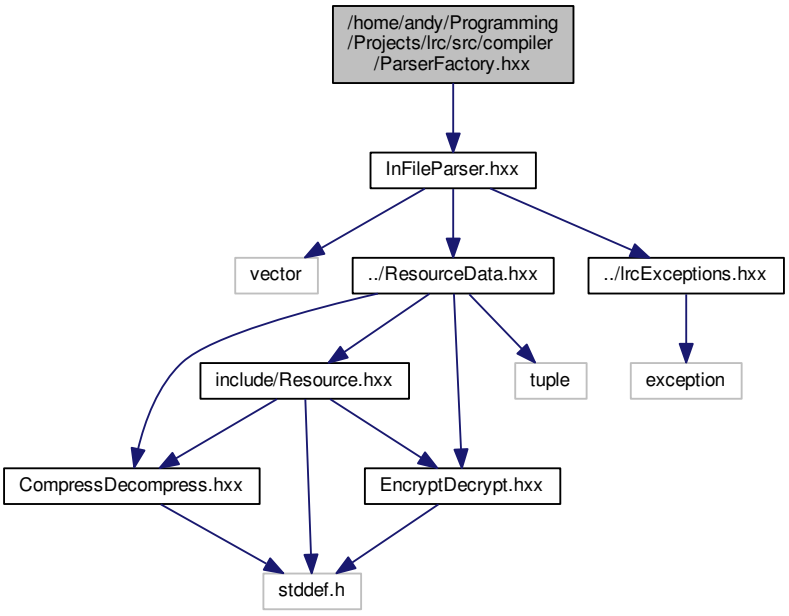
Show the usage of the compiler.

Definition at line 55 of file lrc.cxx.

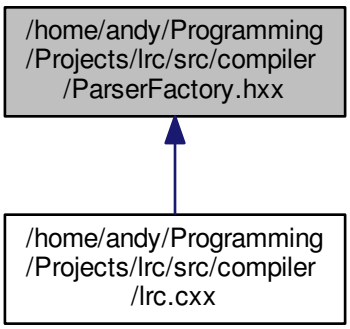
9.4 /home/andy/Programming/Projects/lrc/src/compiler/ParserFactory.hxx File Reference

```
#include "InFileParser.hxx"
```

Include dependency graph for ParserFactory.hxx:



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



Classes

- class ParserFactory

  Factory class to create an appropriate parser class.

9.4.1 Detailed Description

This file contains the declaration of the ParserFactory class that is used to create a matching parser class depending on the file to parse.

Author

Andreas Tscharner

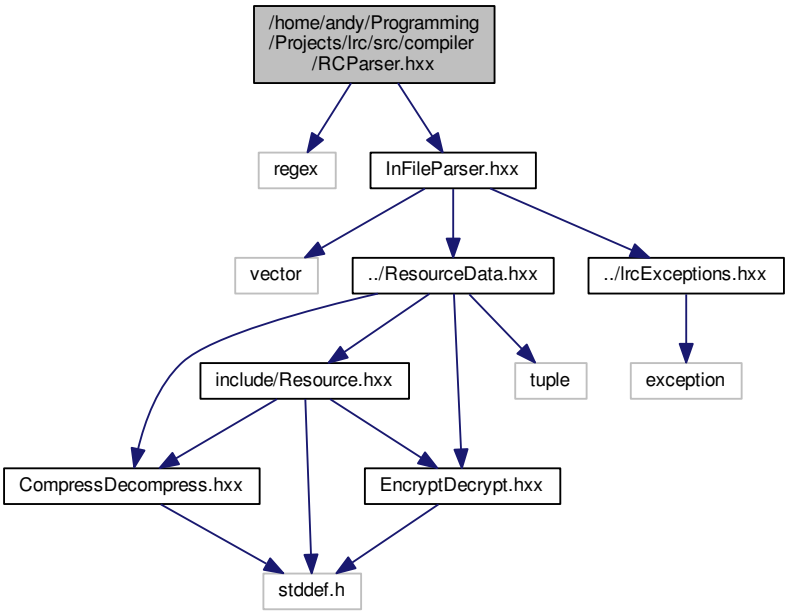
Date

2014-08-23

9.5 /home/andy/Programming/Projects/lrc/src/compiler/RCParser.hxx File Reference

```
#include <regex>
#include "InFileParser.hxx"
```

Include dependency graph for RCParser.hxx:



Classes

- class RCParser

  Class to parse an .rc file.

9.5.1 Detailed Description

This file contains the class definition for the parser for the simple RC format

Author

Andreas Tscharner

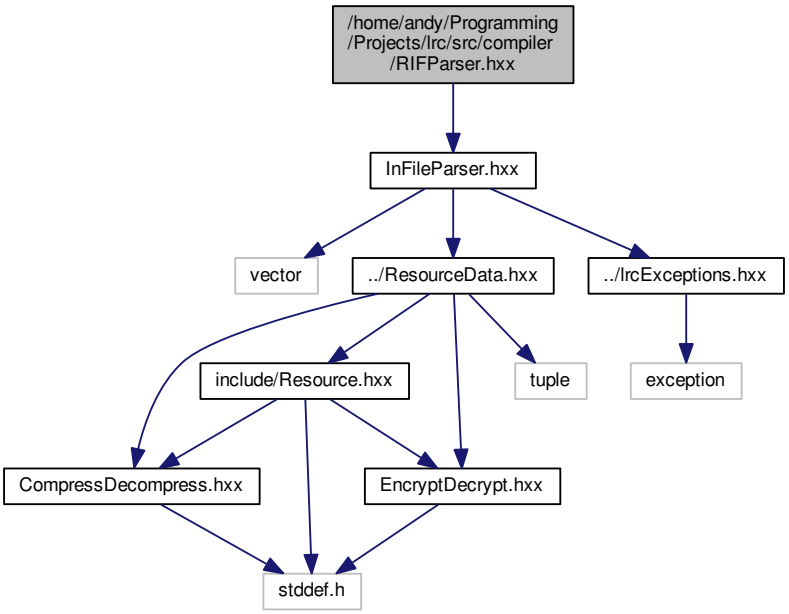
Date

2014-10-26

9.6 /home/andy/Programming/Projects/lrc/src/compiler/RIFParser.hxx File Reference

```
#include "InFileParser.hxx"
```

Include dependency graph for RIFParser.hxx:



Classes

- class RIFParser

  Class to parse a .rif (XML) file.

9.6.1 Detailed Description

This file contains the declaration of the RIFParser class, a class to parse the RIF input file for `lrc`

Author

Andreas Tscharner

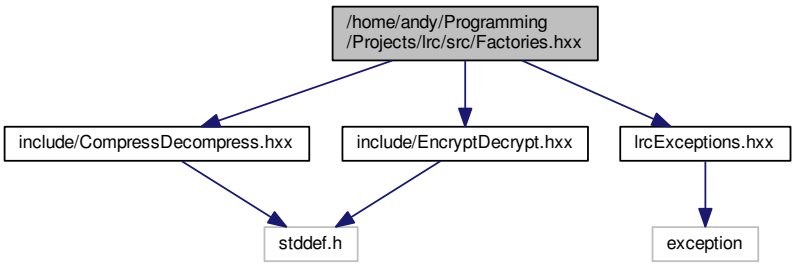
Date

2014-08-23

9.7 /home/andy/Programming/Projects/lrc/src/Factories.hxx File Reference

```
#include "include/CompressDecompress.hxx"
#include "include/EncryptDecrypt.hxx"
#include "lrcExceptions.hxx"
```

Include dependency graph for Factories.hxx:



Classes

- class CompressionFactory

  Factory to create compression class instances.
- class EncryptionFactory

  Factory to create encryption class instances.

9.7.1 Detailed Description

This file contains the declarations of the compression factory class and the encryption factory class

Author

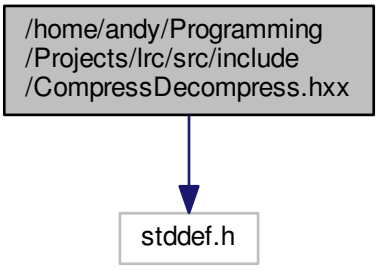
Andreas Tscharner

Date

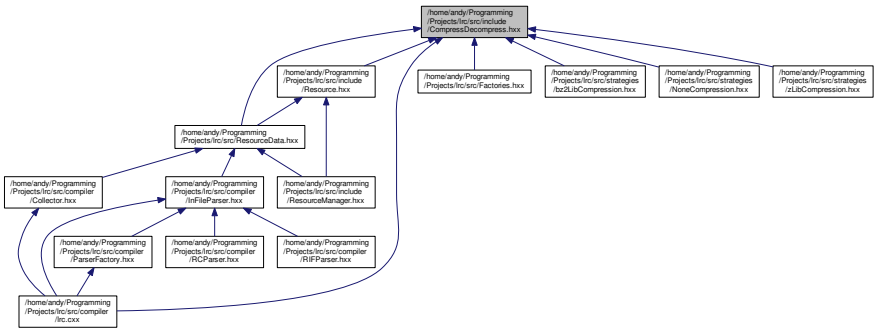
2014-08-23

9.8 /home/andy/Programming/Projects/lrc/src/include/CompressDecompress.hxx File Reference

`#include <stddef.h>`
Include dependency graph for CompressDecompress.hxx:



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



Classes

- class lrc::CompressDecompress

  Compression and Decompression base class.

Namespaces

- lrc

  Namespace lrc for classes in the library and for extending `lrc`.

Enumerations

- enum lrc::CompressionType { lrc::NoneCompression, lrc::zLibCompression, lrc::bz2LibCompression, lrc::lastCompression }

9.8.1 Detailed Description

File that contains the CompressDecompress class, the base class for all compression/decompression algorithms used in `lrc`

Author

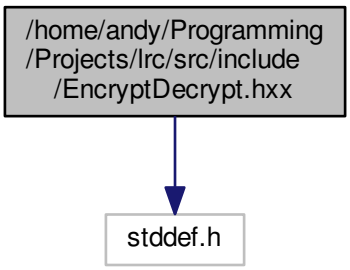
Andreas Tscharner

Date

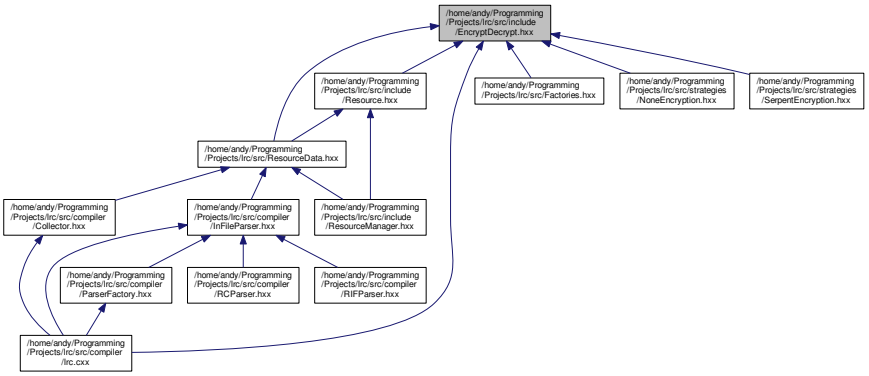
2014-08-23

9.9 /home/andy/Programming/Projects/lrc/src/include/EncryptDecrypt.hxx File Reference

`#include <stddef.h>`
Include dependency graph for EncryptDecrypt.hxx:



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



Classes

- class lrc::EncryptDecrypt

  Encryption and Decryption base class.

Namespaces

- lrc

  Namespace lrc for classes in the library and for extending `lrc`.

Enumerations

- enum lrc::EncryptionType { lrc::NoneEncryption, lrc::SerpentEncryption, lrc::lastEncryption }

9.9.1 Detailed Description

This file contains the abstract class EncryptDecrypt. It is the base class for all encryption and decryption used in `lrc`.

Author

Andreas Tscharner

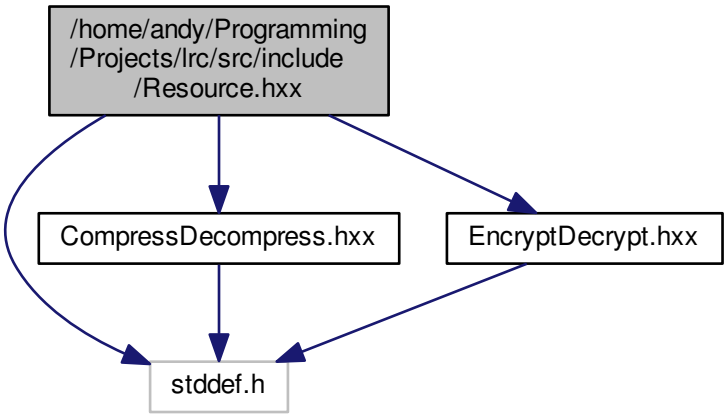
Date

2014-08-23

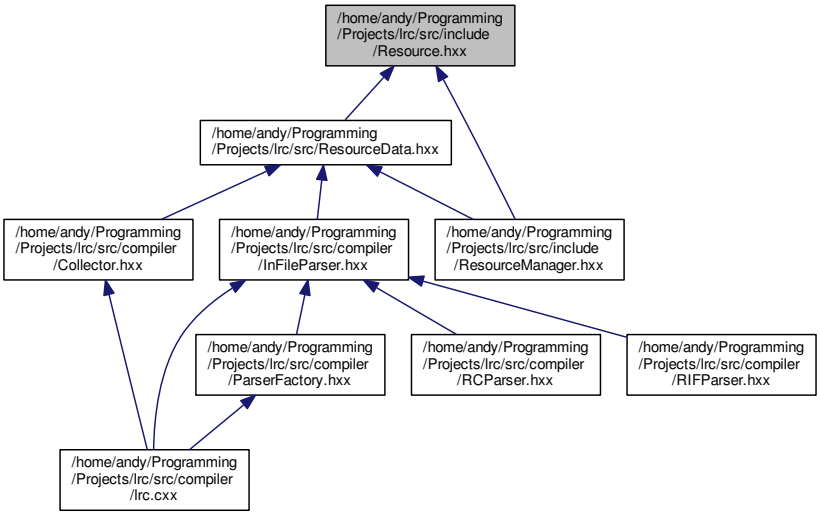
9.10 /home/andy/Programming/Projects/lrc/src/include/Resource.hxx File Reference

```
#include <stddef.h>
#include "CompressDecompress.hxx"
#include "EncryptDecrypt.hxx"
```

Include dependency graph for Resource.hxx:



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



Classes

- struct resEntry_

  Data for one resource entry.
- class lrc::Resource

  Resource class for use with library.

Namespaces

- lrc

  Namespace lrc for classes in the library and for extending `lrc`.

Macros

- #define MAX_ID_LEN 80

  Maximum length of resource ID.

Typedefs

- typedef struct resEntry_ resEntry

  Define a better name for struct resEntry_.

9.10.1 Detailed Description

This file contains one of the main classes: Resource

Author

Andreas Tscharner

Date

2014-08-22

9.10.2 Macro Definition Documentation

9.10.2.1 MAX_ID_LEN

```
#define MAX_ID_LEN 80
```

Maximum length of resource ID.

Definition at line 85 of file Resource.hxx.

9.10.3 Typedef Documentation

9.10.3.1 resEntry

```
typedef struct resEntry_ resEntry
```

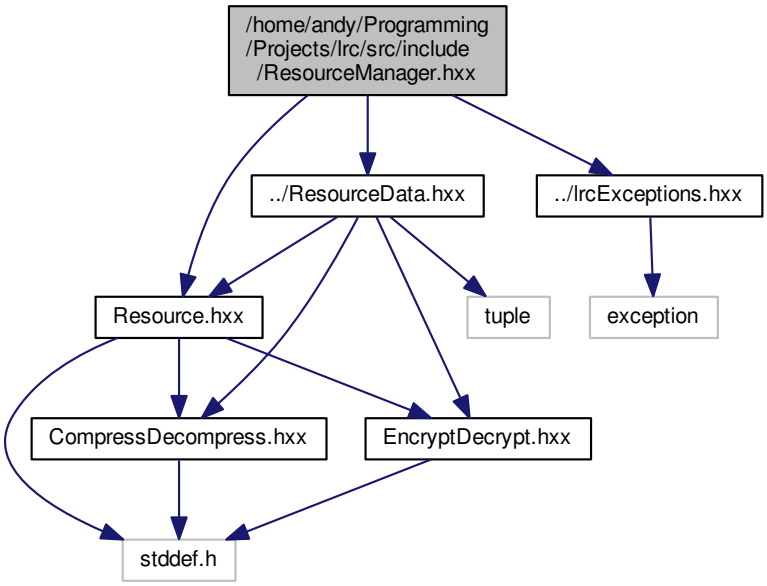
Define a better name for struct resEntry_.

Definition at line 102 of file Resource.hxx.

9.11 /home/andy/Programming/Projects/lrc/src/include/ResourceManager.hxx File Reference

```
#include "Resource.hxx"
#include "../ResourceData.hxx"
#include "../lrcExceptions.hxx"
```

Include dependency graph for ResourceManager.hxx:



Classes

- class lrc::ResourceManager

  Manager class handling all resources of a resource file.

Namespaces

- lrc

  Namespace lrc for classes in the library and for extending `lrc`.

9.11.1 Detailed Description

This file contains the declaration of the ResourceManager, the main class for `liblrc`

Author

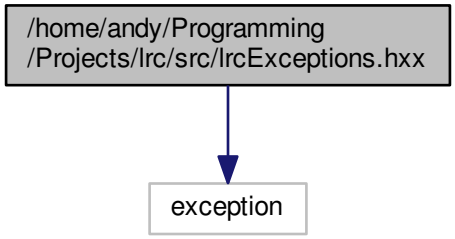
Andreas Tscharner

Date

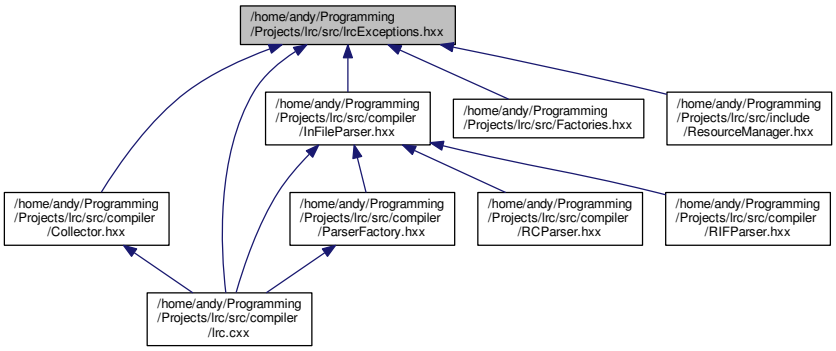
2014-08-23

9.12 /home/andy/Programming/Projects/lrc/src/lrcExceptions.hxx File Reference

`#include <exception>`
Include dependency graph for lrcExceptions.hxx:



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



Classes

- class lrcFileNotFoundException

  Exception class if a file could not be found.
- class lrcFileExistsException

  Exception if an existing file should be overwritten.
- class lrcEncryptionDisabledException

  Exception if encryption is disabled but required.

9.12.1 Detailed Description

This file contains all exception classes for lrc project

Author

Andreas Tscharner

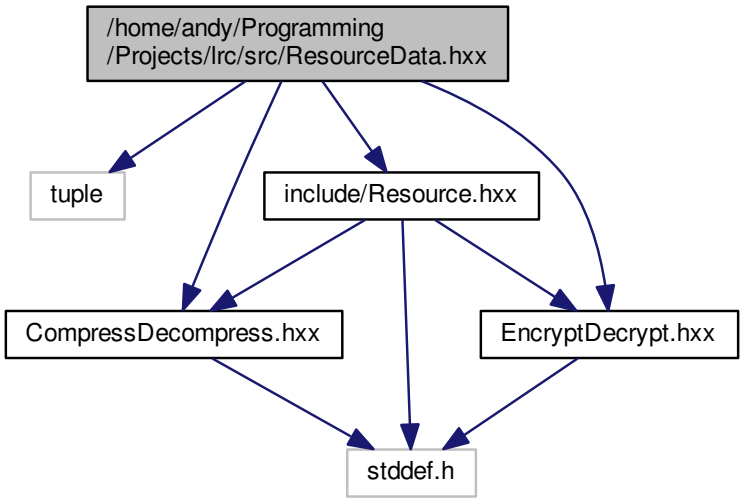
Date

2014-08-23

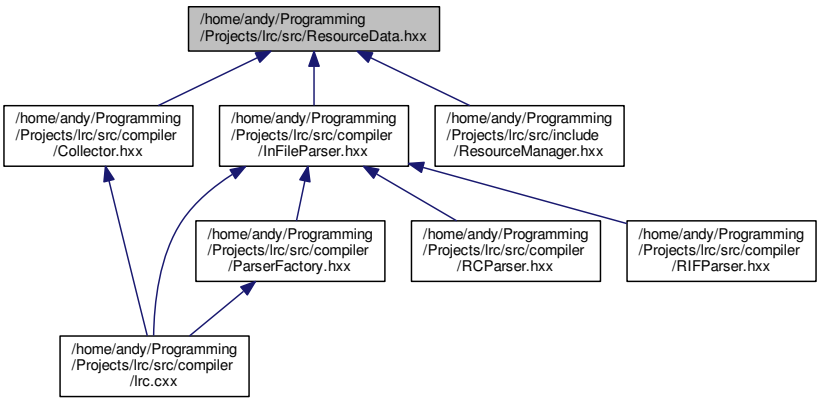
9.13 /home/andy/Programming/Projects/lrc/src/ResourceData.hxx File Reference

```
#include <tuple>
#include "include/Resource.hxx"
#include "include/CompressDecompress.hxx"
#include "include/EncryptDecrypt.hxx"
```

Include dependency graph for ResourceData.hxx:



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



Classes

- class ResourceData

  Internal class for resource with more information.

Typedefs

- typedef std::tuple< int, int > inFilePosition

  Define a datatype of its own for the file position in the input file.

9.13.1 Detailed Description

This file contains the class definition of the ResourceData class. ResourceData is derived from Resource, but contains a lot more information

Author

Andreas Tscharner

Date

2014-08-023

9.13.2 Typedef Documentation

9.13.2.1 inFilePosition

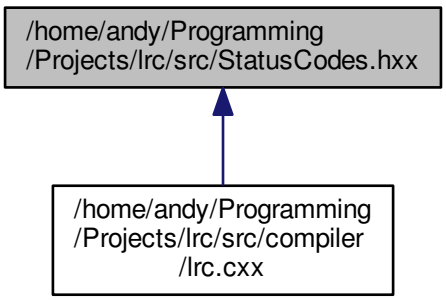
```
typedef std::tuple<int, int> inFilePosition
```

Define a datatype of its own for the file position in the input file.

Definition at line 44 of file ResourceData.hxx.

9.14 /home/andy/Programming/Projects/lrc/src/StatusCodes.hxx File Reference

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



Macros

- #define NO_ERROR 0

  No error, everything OK.
- #define WARNING_BASE 0

  Base value for all warning codes.
- #define WARNING_DOUBLE_RESOURCE_ID WARNING_BASE-1

  The resource ID appears more than once.
- #define ERROR_BASE -1000

  Base value for all error codes.
- #define ERROR_FILE_OPEN ERROR_BASE-1

  An error occurred while opening the file.
- #define ERROR_FILE_NOT_FOUND ERROR_BASE-2

  The desired file could not be found.
- #define ERROR_FILE_READ ERROR_BASE-3

  An error occurred while reading the file.
- #define ERROR_FILE_WRITE ERROR_BASE-4

  An error occurred while writing the file.
- #define ERROR_PARSE ERROR_BASE-10

  An error occurred while parsing the file.
- #define ERROR_COMPRESSION_NOT_AVAILABLE ERROR_BASE-20

  The desired compression is not available.
- #define ERROR_ENCRYPTION_NOT_AVAILABLE ERROR_BASE-21

  The desired encryption is not available.
- #define ERROR_COMPRESSION_COMPRESS ERROR_BASE-22

  An error occurred while compressing.
- #define ERROR_COMPRESSION_DECOMPRESS ERROR_BASE-23

  An error occurred while decompressing.
- #define ERROR_ENCRYPTION_ENCRYPT ERROR_BASE-24

  An error occurred while encrypting.
- #define ERROR_ENCRYPTION_DECRYPT ERROR_BASE-25

  An error occurred while decrypting.
- #define ERROR_UNKNOWN_ARCH ERROR_BASE-40

  The architecture is unknown.
- #define ERROR_INVALID_PARAMETER ERROR_BASE-100

  The provided parameter was illegal.
- #define ERROR_NOT_ENOUGH_MEMORY ERROR_BASE-101

  Not enough memory available for this action.

Functions

- bool no_error (int c)

  Check for NO_ERROR.
- bool success (int c)

  Check for success.
- bool is_warning (int c)

  Check for warning.
- bool is_error (int c)

  Check for error.

9.14.1 Detailed Description

This file contains a number of defines that are used as status (success, warning or error) codes and a few inline functions for better handling these codes

Author

Andreas Tscharner

Date

2014-08-10

9.14.2 Macro Definition Documentation

9.14.2.1 ERROR_BASE

```
#define ERROR_BASE -1000
```

Base value for all error codes.

Definition at line 44 of file StatusCodes.hxx.

9.14.2.2 ERROR_COMPRESSION_COMPRESS

```
#define ERROR_COMPRESSION_COMPRESS ERROR_BASE-22
```

An error occurred while compressing.

Definition at line 58 of file StatusCodes.hxx.

9.14.2.3 ERROR_COMPRESSION_DECOMPRESS

```
#define ERROR_COMPRESSION_DECOMPRESS ERROR_BASE-23
```

An error occurred while decompressing.

Definition at line 59 of file StatusCodes.hxx.

9.14.2.4 ERROR_COMPRESSION_NOT_AVAILABLE

```
#define ERROR_COMPRESSION_NOT_AVAILABLE ERROR_BASE-20
```

The desired compression is not available.

Definition at line 56 of file StatusCodes.hxx.

9.14.2.5 ERROR_ENCRYPTION_DECRYPT

```
#define ERROR_ENCRYPTION_DECRYPT ERROR_BASE-25
```

An error occurred while decrypting.

Definition at line 61 of file StatusCodes.hxx.

9.14.2.6 ERROR_ENCRYPTION_ENCRYPT

```
#define ERROR_ENCRYPTION_ENCRYPT ERROR_BASE-24
```

An error occurred while encrypting.

Definition at line 60 of file StatusCodes.hxx.

9.14.2.7 ERROR_ENCRYPTION_NOT_AVAILABLE

```
#define ERROR_ENCRYPTION_NOT_AVAILABLE ERROR_BASE-21
```

The desired encryption is not available.

Definition at line 57 of file StatusCodes.hxx.

9.14.2.8 ERROR_FILE_NOT_FOUND

```
#define ERROR_FILE_NOT_FOUND ERROR_BASE-2
```

The desired file could not be found.

Definition at line 48 of file StatusCodes.hxx.

9.14.2.9 ERROR_FILE_OPEN

```
#define ERROR_FILE_OPEN ERROR_BASE-1
```

An error occurred while opening the file.

Definition at line 47 of file StatusCodes.hxx.

9.14.2.10 ERROR_FILE_READ

```
#define ERROR_FILE_READ ERROR_BASE-3
```

An error occurred while reading the file.

Definition at line 49 of file StatusCodes.hxx.

9.14.2.11 ERROR_FILE_WRITE

```
#define ERROR_FILE_WRITE ERROR_BASE-4
```

An error occurred while writing the file.

Definition at line 50 of file StatusCodes.hxx.

9.14.2.12 ERROR_INVALID_PARAMETER

```
#define ERROR_INVALID_PARAMETER ERROR_BASE-100
```

The provided parameter was illegal.

Definition at line 67 of file StatusCodes.hxx.

9.14.2.13 ERROR_NOT_ENOUGH_MEMORY

```
#define ERROR_NOT_ENOUGH_MEMORY ERROR_BASE-101
```

Not enough memory available for this action.

Definition at line 68 of file StatusCodes.hxx.

9.14.2.14 ERROR_PARSE

```
#define ERROR_PARSE ERROR_BASE-10
```

An error occurred while parsing the file.

Definition at line 53 of file StatusCodes.hxx.

9.14.2.15 ERROR_UNKNOWN_ARCH

```
#define ERROR_UNKNOWN_ARCH ERROR_BASE-40
```

The architecture is unknown.

Definition at line 64 of file StatusCodes.hxx.

9.14.2.16 NO_ERROR

```
#define NO_ERROR 0
```

No error, everything OK.

Definition at line 37 of file StatusCodes.hxx.

9.14.2.17 WARNING_BASE

```
#define WARNING_BASE 0
```

Base value for all warning codes.

Definition at line 40 of file StatusCodes.hxx.

9.14.2.18 WARNING_DOUBLE_RESOURCE_ID

```
#define WARNING_DOUBLE_RESOURCE_ID WARNING_BASE-1
```

The resource ID appears more than once.

Definition at line 41 of file StatusCodes.hxx.

9.14.3 Function Documentation

9.14.3.1 is_error()

```
bool is_error (
            int c )  [inline]
```

Check for error.

Small inline function to check if the given parameter stands for an error

Parameters

in	*c*	Value to check

Return values

true	Value is an error code
false	Value stands for warning or success

Definition at line 115 of file StatusCodes.hxx.

9.14.3.2 is_warning()

```
bool is_warning (
            int c )  [inline]
```

Check for warning.

Small inline function to check if the given parameter stands for a warning

Parameters

in	*c*	Value to check

Return values

true	Value is a warning code
false	Value stands for success or error

Definition at line 104 of file StatusCodes.hxx.

9.14.3.3 no_error()

```
bool no_error (
            int c )  [inline]
```

Check for NO_ERROR.

Small inline function to check if the given parameter is NO_ERROR

Parameters

in	*c*	Value to check

Return values

true	Value is NO_ERROR
false	Value is not NO_ERROR

Definition at line 81 of file StatusCodes.hxx.

9.14.3.4 success()

```
bool success (
             int c )  [inline]
```

Check for success.

Small inline function to check if the given parameter means success, e.g. is greater than NO_ERROR

Parameters

in	*c*	Value to check

Return values

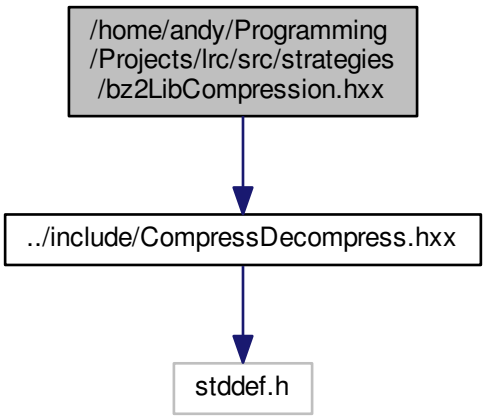
true	Value means success
false	Value means warning or error

Definition at line 92 of file StatusCodes.hxx.

9.15 /home/andy/Programming/Projects/Irc/src/strategies/bz2LibCompression.hxx File Reference

```
#include "../include/CompressDecompress.hxx"
```

Include dependency graph for bz2LibCompression.hxx:



Classes

- class bz2LibCompression

  Compression class that uses the bzip2 algorithm.

9.15.1 Detailed Description

This file contains the declaration of the bz2Lib compression class. This class is used to compress and decompress teh resource data using the bzip2 algorithm.

Author

Andreas Tscharner

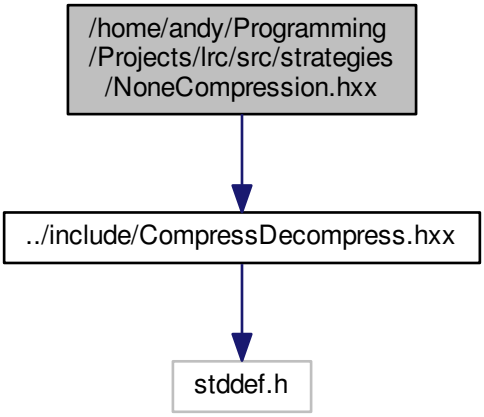
Date

2012-01-18

9.16 /home/andy/Programming/Projects/lrc/src/strategies/NoneCompression.hxx File Reference

```
#include "../include/CompressDecompress.hxx"
```

Include dependency graph for NoneCompression.hxx:



Classes

- class NoneCompression

  Compression class that does NO compression.

9.16.1 Detailed Description

This file contains the class declaration of the NoneCompression class. This class is a dummy class that does no compression at all, but is used to fit in the Strategy Pattern

Author

Andreas Tscharner

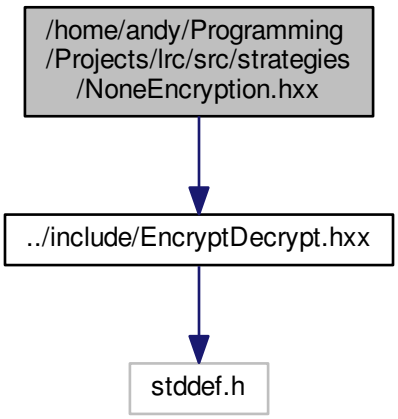
Date

2011-06-25

9.17 /home/andy/Programming/Projects/lrc/src/strategies/NoneEncryption.hxx File Reference

```
#include "../include/EncryptDecrypt.hxx"
```

Include dependency graph for NoneEncryption.hxx:



Classes

- class NoneEncryption

  Encryption class that does NO encryption.

9.17.1 Detailed Description

This file contains the NoneEncryption class. This class does no encryption at all, but is used to fit in the Strategy Pattern

Author

Andreas Tscharner

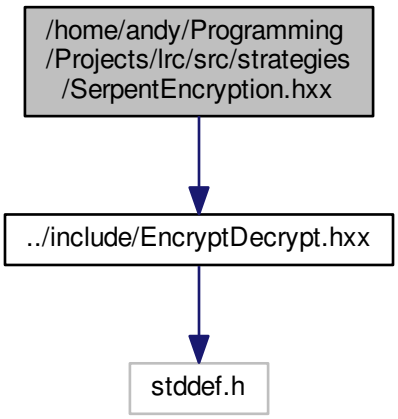
Date

0211-07-21

9.18 /home/andy/Programming/Projects/lrc/src/strategies/SerpentEncryption.hxx File Reference

```
#include "../include/EncryptDecrypt.hxx"
```

Include dependency graph for SerpentEncryption.hxx:



Classes

- class SerpentEncryption

  Class to encrypt/decrypt using the Serpent algorithm.

9.18.1 Detailed Description

This file contains the declaration of the SerpenEncryption class. This class is used to encrypt and decrypt the data using the *Serpent* algorithm

Author

Andreas Tscharner

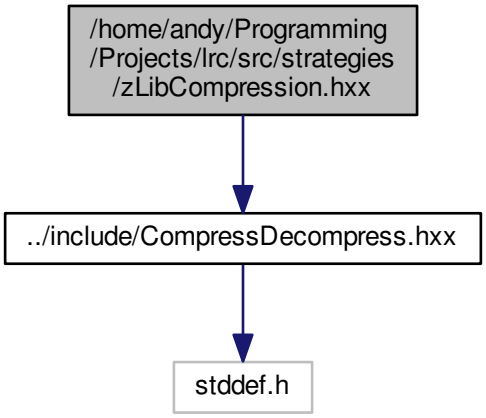
Date

2014-08-23

9.19 /home/andy/Programming/Projects/lrc/src/strategies/zLibCompression.hxx File Reference

```
#include "../include/CompressDecompress.hxx"
```

Include dependency graph for zLibCompression.hxx:



Classes

- class zLibCompression

  Compression class that uses the zLib algorithm.

9.19.1 Detailed Description

This file contains the declaration of the zLib compression class. This class is used to compress and decompress teh resource data using the zLib algorithm.

Author

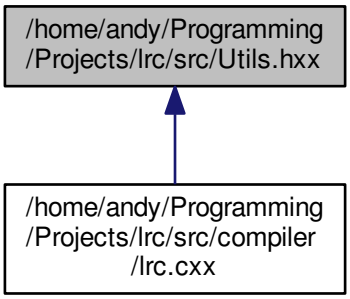
Andreas Tscharner

Date

2011-08-14

9.20 /home/andy/Programming/Projects/lrc/src/Utils.hxx File Reference

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



Macros

- #define DEBUG_PRINT(x)

  Macro for debugging.

Functions

- bool file_exists (const char *p_filename)

  Check if a file exists.
- int file_size (const char *p_filename)

  Returns the size of a file.
- char * get_extension (char *p_filename)

  Returns the file extension.
- char * replace_extension (char *p_filename, const char *p_newExt)

  Replace the file extension.
- void delete_list (unsigned char **p_dataList, unsigned int p_dataSize)

  Delete list of data.
- int password_len (const unsigned char *p_password)

  Gets length of a password.

9.20.1 Detailed Description

This file contains the declaration of a few utility functions that are used in the whole project

Author

Andreas Tscharner

Date

2014-08-23

9.20.2 Macro Definition Documentation

9.20.2.1 DEBUG_PRINT

```
#define DEBUG_PRINT(
              x )
```

Macro for debugging.

This macro expands to a printf if the **DEBUG** define is set or to nothing otherwise

Definition at line 43 of file Utils.hxx.

9.20.3 Function Documentation

9.20.3.1 delete_list()

```
void delete_list (
             unsigned char ** p_dataList,
             unsigned int p_dataSize )
```

Delete list of data.

This function deletes a list of data (double pointers to unsigned chars)

Parameters

in	*p_dataList*	List of data to delete
in	*p_dataSize*	Size of data list

9.20.3.2 file_exists()

```
bool file_exists (
            const char * p_filename )
```

Check if a file exists.

This function checks if a given file exists and returns true if it does and false otherwise

Parameters

in	*p_filename*	Full or relative path to file

Return values

true	File exists
false	File does not exist

9.20.3.3 file_size()

```
int file_size (
            const char * p_filename )
```

Returns the size of a file.

This function returns the size of a file

Parameters

in	*p_filename*	Filename

Returns

Size of file

Remarks

If an error occurres, the function returns -1

9.20.3.4 get_extension()

```
char* get_extension (
            char * p_filename )
```

Returns the file extension.

This function returns the file extension of the given file

Parameters

in	*p_filename*	Filename

Returns

File extension (if any)

Remarks

The returned string is a pointer to the file extension within the given string! It is NULL, if the given filename has no extension.

9.20.3.5 password_len()

```
int password_len (
            const unsigned char * p_password )
```

Gets length of a password.

This function is used to get the length of a zero-terminated password. It counts all *unsigned* characters until the first zero

Parameters

in	*p_password*	Zero-terminated password

Returns

Length of zero-terminated password

Remarks

-1 will be returned if the given password was `nullptr`

9.20.3.6 replace_extension()

```
char* replace_extension (
            char * p_filename,
            const char * p_newExt )
```

Replace the file extension.

This function replaces the extension of the given file with the given new extension. The extention is expected to have a dot ('.') as first character. If the given filename has no extension, the new extension is appended.

Parameters

in	*p_filename*	Filename
in	*p_newExt*	New file extension

Returns

Filename with new extension

Remarks

The caller is responsible to free the allocated memory of the returned string

Index