ESA JPIP server

0.1

Generated by Doxygen 1.8.11

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 data Namespace Reference

Contains a set of classes to easy the handling of data and files, as well as the serialization.

Classes

- class BaseFile

  This is a wrapper class for the `FILE` functions that provides all the functionality to handle files safely.
- class BaseStream

  This template is used as the base for the input/output stream classes.
- class FileSegment

  Identifies a data segment of a file.
- struct InputOperator

  This struct identifies a basic input operator to be applied to a `File` object.
- class InputStream

  Specialization of the `BaseStream` for input serializations.
- struct LockedAccess

  Struct for wrapping the basic `FILE` locked functions for reading and writing defined in `stdio.h`.
- struct OutputOperator

  This struct identifies a basic output operator to be applied to a `File` object.
- class OutputStream

  Specialization of the `BaseStream` for output serializations.
- struct Serializer

  This template class allows to define a "serializer".
- struct Serializer< bool >

  Serializer for the `bool` type.
- struct Serializer< int >

  Serializer for the `int` type.
- struct Serializer< multimap< string, int > >

  Serializer for the `multimap<string,int>` class.
- struct Serializer< string >

  Serializer for the `string` class.
- struct Serializer< uint64_t >

  Serializer for the `uint64_t` type.

- struct Serializer< vector< T > >

  Serializer for the `vector` class.
- struct UnlockedAccess

  Struct for wrapping the basic `FILE` unlocked functions for reading and writing defined in `stdio_exts.h`.
- class vint_vector

  This class has been implemented with the same philosophy that the class STL vector, but specifically designed to store integers with a length in bytes that can be not multiple from 2 (e.g.

Typedefs

- typedef BaseFile< LockedAccess > File

  Specialization of the class `BaseFile` with locked access.
- typedef BaseFile< UnlockedAccess > FastFile

  Specialization of the class `BaseFile` with unlocked access.

5.1.1 Detailed Description

Contains a set of classes to easy the handling of data and files, as well as the serialization.

5.1.2 Typedef Documentation

5.1.2.1 typedef BaseFile<UnlockedAccess> data::FastFile

Specialization of the class `BaseFile` with unlocked access.

See also

BaseFile
UnlockedAccess

5.1.2.2 typedef BaseFile<LockedAccess> data::File

Specialization of the class `BaseFile` with locked access.

See also

BaseFile
LockedAccess

5.2 http Namespace Reference

Contains the definition of a set of classes for working easily with the protocol HTTP.

Classes

- class Header

  Class used to handle a HTTP header.
- class HeaderBase

  Template class used to identify a HTTP header.
- class HeaderBase< HeaderName::UNDEFINED >

  Specialization of the `HeaderBase` template class with the `HeaderName::UNDEFINED` value.
- class HeaderName

  Container for the strings associated to the most common HTTP headers, used for the specialization of the class `HeaderBase`.
- class Protocol

  Class used to identify the HTTP protocol.
- class Request

  Class used to identify a HTTP request (GET or POST).
- class Response

  Class used to identify a HTTP response.

Functions

- istream & operator>> (istream &in, Request &request)
- ostream & operator<< (ostream &out, const Request &request)

5.2.1 Detailed Description

Contains the definition of a set of classes for working easily with the protocol HTTP.

5.2.2 Function Documentation

5.2.2.1 ostream& http::operator<< (ostream & *out,* const Request & *request*)

Here is the caller graph for this function:

5.2.2.2 istream& http::operator>> (istream & *in,* Request & *request*)

Here is the call graph for this function:

Here is the caller graph for this function:

5.3 ipc Namespace Reference

Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

Classes

- class Event

  IPC object that offers the functionality of an event (Windows IPC object), implemented by means of a combination of the pthread mutex and conditional variables API.
- class IPCObject

  Class base of all the IPC classes that has the basic operations (`Init`, `Wait` and `Dispose`) to be overloaded.
- class Mutex

  IPC object that offers the functionality of a mutex, implemented by means of the pthread mutex API.
- class RdWrLock

  IPC object that offers the functionality of a read/write lock, implemented by means of the pthread rwlock API.

Enumerations

- enum WaitResult { WAIT_OBJECT = 0, WAIT_TIMEOUT, WAIT_ERROR }

  Enumeration of the possible values returned when a wait operation is performed for an IPC object.

5.3.1 Detailed Description

Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

These classes have been implemented to offer an object-oriented mechanism similar to the one offered by Windows, because of its simplicity and flexibility.

5.3.2 Enumeration Type Documentation

5.3.2.1 enum ipc::WaitResult

Enumeration of the possible values returned when a wait operation is performed for an IPC object.

Enumerator

WAIT_OBJECT Wait successful (object got)

WAIT_TIMEOUT Time out.

WAIT_ERROR Error.

5.4 jpeg2000 Namespace Reference

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

Classes

- class CodestreamIndex

  Class used for indexing the information of a JPEG2000 codestream.
- class CodingParameters

  Contains the coding parameters of a JPEG2000 image codestream.
- class FileManager

  Manages the image files of a repository, allowing read their indexing information, with a caching mechanism for efficiency.
- class ImageIndex

  Contains the indexing information of a JPEG2000 image file that is managed by the index manager.
- class ImageInfo

  Contains the indexing information of a JPEG2000 image.
- class IndexManager

  Manages the indexing information of a repository fo images.
- class Metadata

  Contains the indexing information associated to the meta-data of a JPEG2000 image file.
- class Packet

  Contains the information of a packet.
- class PacketIndex

  Class used for indexing the packets of a codestream image.
- class PlaceHolder

  Contains the information of a place-holder.
- class Point

  Represents a couple of integer values that can be used to identify a coordinate as well as a size.
- class Range

  Represents a range of integer values, defined by two values, first and last, which are assumed to be included in the range.

Typedefs

- typedef Point Size

  It is a synonymous of the class `Point`.

5.4.1 Detailed Description

Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

5.4.2 Typedef Documentation

5.4.2.1 typedef Point jpeg2000::Size

It is a synonymous of the class `Point`.

See also

Point

5.5 jpip Namespace Reference

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

Classes

- class CacheModel

  The cache model of a JPIP client is handled using this class.
- class DataBinClass

  Class that contains the definitions of all the data-bin classes defined for the JPIP protocol.
- struct DataBinSelector

  Template class that is specialized for allowing basic operations (add and get) with cache models depending on the data-bin classes.
- struct DataBinSelector< DataBinClass::MAIN_HEADER >
- struct DataBinSelector< DataBinClass::META_DATA >
- struct DataBinSelector< DataBinClass::PRECINCT >
- struct DataBinSelector< DataBinClass::TILE_HEADER >
- class DataBinServer

  Contains the core functionality of a (JPIP) data-bin server, which maintains a cache model and is capable of generating data chunks of variable length;.
- class DataBinWriter

  Class used to generate data-bin segments and write them into a memory buffer.
- class EOR

  Class that contains all the definitions of the EOF messages defined for the JPIP protocol.
- class Request

  Class derived from the HTTP `Request` class that contains the required code for properly analyzing a JPIP request, when this protocol is used over the HTTP.
- class WOI

  Class that identifies a WOI (Window Of Interest).
- class WOIComposer

  By means of this class it is possible to find out the which packets of an image are associated to a WOI.

5.5.1 Detailed Description

Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

5.6 net Namespace Reference

Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

Classes

- class Address

  Abstract base class to wrap the `sockaddr` derived structures.
- class InetAddress

  Class to identify and handle an Internet address.
- struct PollFD

  Wrapper structure for the structure `pollfd` used by the kernel `poll` functions.
- class PollTable

  This class allows to perfom polls easily over a vector of descriptors.
- class Socket

  This class has been designed to work with UNIX sockets in an easy and object oriented way.
- class SocketBuffer

  Class derived from the STL `std::streambuf` to allow streaming with sockets.
- class SocketStream

  Class derived from `std::iostream` and `SocketBuffer` that represents a socket stream.
- class UnixAddress

  Class to identify and handle an UNIX address.

5.6.1 Detailed Description

Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

Chapter 6

Class Documentation

6.1 net::Address Class Reference

Abstract base class to wrap the `sockaddr` derived structures.

`#include <address.h>`

Inheritance diagram for net::Address:

Collaboration diagram for net::Address:

Public Member Functions

- Address ()

  Empty constructor.
- virtual sockaddr * GetSockAddr () const =0

  Returns a pointer to a `sockaddr` structure.
- virtual int GetSize () const =0

  Returns the size in bytes of the `sockaddr` structure returned by the previous method.
- virtual ~Address ()

  Empty destructor.

6.1.1 Detailed Description

Abstract base class to wrap the `sockaddr` derived structures.

This class is the base of the address classes.

See also

InetAddress
UnixAddress

6.1.2 Constructor & Destructor Documentation

6.1.2.1 net::Address::Address ( ) [inline]

Empty constructor.

6.1.2.2 virtual net::Address::~Address ( ) [inline], [virtual]

Empty destructor.

6.1.3 Member Function Documentation

6.1.3.1 virtual int net::Address::GetSize ( ) const [pure virtual]

Returns the size in bytes of the `sockaddr` structure returned by the previous method.

Implemented in net::UnixAddress, and net::InetAddress.

Here is the caller graph for this function:

6.1.3.2 virtual sockaddr* net::Address::GetSockAddr ( ) const [pure virtual]

Returns a pointer to a `sockaddr` structure.

Implemented in net::UnixAddress, and net::InetAddress.

Here is the caller graph for this function:

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

- net/address.h

6.2 AppConfig Class Reference

Contains the configuration parameters of the application.

`#include <app_config.h>`

Collaboration diagram for AppConfig:

Public Member Functions

- AppConfig ()
  Initializes the object with zero and empty values.
- bool Load (const char *file_name)
  Loads the parameters from a configuration file.
- int port () const
  Returns the listening port.
- string address () const
  Returns the listening address.
- string images_folder () const
  Returns the folder of the images.
- string caching_folder () const
  Returns the folder used for caching.
- string logging_folder () const
  Returns the folder used for the logging files.
- int max_chunk_size () const
  Returns the maximum chunk size.
- int max_connections () const
  Returns the maximum number of connections.
- bool logging () const
  Returns `true` if the logging messages are allowed.
- bool log_requests () const
  Returns `true` if the client requests are logged.
- int com_time_out () const
  Returns the connection time-out.
- int cache_max_time () const
  Returns the maximum time for the cache files in seconds.
- virtual ~AppConfig ()

Private Attributes

- int port_
  Listening port.
- int logging_
  `true` if logs messages are allowed
- int log_requests_
  `true` if the client requests are logged
- string address_
  Listening address.
- string images_folder_
  Directory for the images.
- string caching_folder_
  Directory for the caching files.
- string logging_folder_
  Directory for the logging files.
- int max_chunk_size_
  Maximum chunk size.
- int max_connections_
  Maximum number of connections.
- int com_time_out_
  Connection time-out.
- int cache_max_time_
  Maximum time for the cache files.

Friends

- ostream & operator<< (ostream &out, const AppConfig &cfg)

6.2.1 Detailed Description

Contains the configuration parameters of the application.

It is possible to load these parameters from a configuration file. This class can be printed.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 AppConfig::AppConfig ( ) [inline]

Initializes the object with zero and empty values.

6.2.2.2 virtual AppConfig::~AppConfig ( ) [inline], [virtual]

6.2.3 Member Function Documentation

6.2.3.1 string AppConfig::address ( ) const [inline]

Returns the listening address.

Here is the caller graph for this function:

6.2.3.2 int AppConfig::cache_max_time ( ) const [inline]

Returns the maximum time for the cache files in seconds.

Here is the caller graph for this function:

6.2.3.3 string AppConfig::caching_folder ( ) const [inline]

Returns the folder used for caching.

Here is the caller graph for this function:

6.2.3.4 int AppConfig::com_time_out ( ) const [inline]

Returns the connection time-out.

Here is the caller graph for this function:

6.2.3.5 string AppConfig::images_folder ( ) const [inline]

Returns the folder of the images.

Here is the caller graph for this function:

6.2.3.6 bool AppConfig::Load (const char * *file_name*)

Loads the parameters from a configuration file.

Parameters

file_name	Configuration file.

Returns

`true` if successful.

Here is the caller graph for this function:

6.2.3.7 bool AppConfig::log_requests ( ) const [inline]

Returns `true` if the client requests are logged.

Here is the caller graph for this function:

6.2.3.8 bool AppConfig::logging ( ) const [inline]

Returns `true` if the logging messages are allowed.

Here is the caller graph for this function:

6.2.3.9 string AppConfig::logging_folder ( ) const [inline]

Returns the folder used for the logging files.

Here is the caller graph for this function:

6.2.3.10 int AppConfig::max_chunk_size ( ) const [inline]

Returns the maximum chunk size.

Here is the caller graph for this function:

6.2.3.11 int AppConfig::max_connections ( ) const [inline]

Returns the maximum number of connections.

Here is the caller graph for this function:

6.2.3.12 int AppConfig::port ( ) const [inline]

Returns the listening port.

Here is the caller graph for this function:

6.2.4 Friends And Related Function Documentation

6.2.4.1 ostream& operator<< (ostream & *out,* const AppConfig & *cfg*) [friend]

6.2.5 Member Data Documentation

6.2.5.1 string AppConfig::address_ [private]

Listening address.

6.2.5.2 int AppConfig::cache_max_time_ [private]

Maximum time for the cache files.

6.2.5.3 string AppConfig::caching_folder_ [private]

Directory for the caching files.

6.2.5.4 int AppConfig::com_time_out_ [private]

Connection time-out.

6.2.5.5 string AppConfig::images_folder_ [private]

Directory for the images.

6.2.5.6 int AppConfig::log_requests_ [private]

`true` if the client requests are logged

6.2.5.7 int AppConfig::logging_ [private]

`true` if logs messages are allowed

6.2.5.8 string AppConfig::logging_folder_ [private]

Directory for the logging files.

6.2.5.9 int AppConfig::max_chunk_size_ [private]

Maximum chunk size.

6.2.5.10 int AppConfig::max_connections_ [private]

Maximum number of connections.

6.2.5.11 int AppConfig::port_ [private]

Listening port.

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

- app_config.h
- app_config.cc

6.3 AppInfo Class Reference

Contains the run-time information of the application.

`#include <app_info.h>`

Collaboration diagram for AppInfo:

Classes

- struct Data

  Contains the data block that is maintained in shared memory.

Public Member Functions

- AppInfo ()

  Initializes the object.
- bool Init ()

  Initializes the object and the handling of the application run-time information.
- bool is_running () const

  Returns `true` if the application is running.
- AppInfo & Update ()

  Updates the run-time information of the application.
- double available_memory () const

  Returns the available memory of the system.
- double father_memory () const

  Returns the memory used by the father process.
- double child_memory () const

  Returns the memory used by the child process.
- int num_threads () const

  Returns the number of active threads.
- unsigned long child_time () const

  Returns the time spent by the child process.
- unsigned long time () const

  Returns the time spent by the father process.
- Data * operator-> () const
- ~AppInfo ()

Private Member Functions

- string GetProcStat_ (int pid, int field) const

  Returns a specific field of /proc/<pid>/stat as a string.
- template<typename TYPE >
  TYPE GetProcStat (int pid, int field) const

  Returns a specific field of /proc/<pid>/stat as a defined type.

Private Attributes

- int shmid

  Identifier of the shared memory block.
- int lock_file

  Lock file.
- Data * data_ptr

  Pointer to the shared memory block.
- bool is_running_

  `true` *if the application is running*
- int num_threads_

  Number of active threads.
- double child_memory_

  Memory used by the child process.
- unsigned long time_

  Time spent by the father.
- double father_memory_

  Memory used by the father process.
- double available_memory_

  Available memory in the system.
- unsigned long child_time_

  Time spend by the child.

Friends

- ostream & operator<< (ostream &out, const AppInfo &app)

6.3.1 Detailed Description

Contains the run-time information of the application.

This class can be printed.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 AppInfo::AppInfo ( ) [inline]

Initializes the object.

6.3.2.2 AppInfo::~AppInfo ( )

6.3.3 Member Function Documentation

6.3.3.1 double AppInfo::available_memory ( ) const [inline]

Returns the available memory of the system.

Here is the caller graph for this function:

6.3.3.2 double AppInfo::child_memory ( ) const [inline]

Returns the memory used by the child process.

Here is the caller graph for this function:

6.3.3.3 unsigned long AppInfo::child_time ( ) const [inline]

Returns the time spent by the child process.

Here is the caller graph for this function:

6.3.3.4 double AppInfo::father_memory ( ) const [inline]

Returns the memory used by the father process.

Here is the caller graph for this function:

6.3.3.5 template<typename TYPE > TYPE AppInfo::GetProcStat (int *pid,* int *field*) const [inline], [private]

Returns a specific field of /proc/<pid>/stat as a defined type.

6.3.3.6 string AppInfo::GetProcStat_ (int *pid,* int *field*) const [private]

Returns a specific field of /proc/<pid>/stat as a string.

6.3.3.7 bool AppInfo::Init ( )

Initializes the object and the handling of the application run-time information.

Returns

`true` if successful.

Here is the caller graph for this function:

6.3.3.8 bool AppInfo::is_running ( ) const [inline]

Returns `true` if the application is running.

Here is the caller graph for this function:

6.3.3.9 int AppInfo::num_threads ( ) const [inline]

Returns the number of active threads.

Here is the caller graph for this function:

6.3.3.10 Data* AppInfo::operator-> ( ) const [inline]

6.3.3.11 unsigned long AppInfo::time ( ) const [inline]

Returns the time spent by the father process.

6.3.3.12 AppInfo & AppInfo::Update ( )

Updates the run-time information of the application.

Here is the caller graph for this function:

6.3.4 Friends And Related Function Documentation

6.3.4.1 ostream& operator<< (ostream & *out,* const AppInfo & *app*) [friend]

6.3.5 Member Data Documentation

6.3.5.1 double AppInfo::available_memory_ [private]

Available memory in the system.

6.3.5.2 double AppInfo::child_memory_ [private]

Memory used by the child process.

6.3.5.3 unsigned long AppInfo::child_time_ [private]

Time spend by the child.

6.3.5.4 Data* AppInfo::data_ptr [private]

Pointer to the shared memory block.

6.3.5.5 double AppInfo::father_memory_ [private]

Memory used by the father process.

6.3.5.6 bool AppInfo::is_running_ [private]

`true` if the application is running

6.3.5.7 int AppInfo::lock_file [private]

Lock file.

6.3.5.8 int AppInfo::num_threads_ [private]

Number of active threads.

6.3.5.9 int AppInfo::shmid [private]

Identifier of the shared memory block.

6.3.5.10 unsigned long AppInfo::time_ [private]

Time spent by the father.

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

- app_info.h
- app_info.cc

6.4 ArgsParser Class Reference

Class that allows to parse and handle the application command line parameters.

`#include <args_parser.h>`

Collaboration diagram for ArgsParser:

Public Member Functions

- ArgsParser (AppConfig &_cfg, AppInfo &_app_info)

  Initializes the object.
- bool Parse (int argc, char **argv)

  Parses and handles the application command line parameters.

Private Attributes

- AppConfig & cfg

  Application configuration.
- AppInfo & app_info

  Application run-time information.

6.4.1 Detailed Description

Class that allows to parse and handle the application command line parameters.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 ArgsParser::ArgsParser (AppConfig & *_cfg,* AppInfo & *_app_info*) [inline]

Initializes the object.

Parameters

_cfg	Application configuration.
_app_info	Application run-time information.

Here is the call graph for this function:

6.4.3 Member Function Documentation

6.4.3.1 bool ArgsParser::Parse (int *argc,* char ** *argv*)

Parses and handles the application command line parameters.

Parameters

argc	Number of parameters.
argv	Command line parameters.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.4.4 Member Data Documentation

6.4.4.1 AppInfo& ArgsParser::app_info [private]

Application run-time information.

6.4.4.2 AppConfig& ArgsParser::cfg [private]

Application configuration.

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

- args_parser.h
- args_parser.cc

6.5 base Struct Reference

Contains a set of useful static methods used by the application.

`#include <base.h>`

Collaboration diagram for base:

Static Public Member Functions

- template<typename TYPE >
  static std::string to_string (TYPE val)
  Converts a value to a string.
- template<typename T >
  static void copy (std::vector< T > &dest, const std::vector< T > &src)
  Copies a vector.
- template<typename T >
  static void copy (std::vector< std::vector< T > > &dest, const std::vector< std::vector< T > > &src)
  Copies a vector of vectors.
- template<typename T1 , typename T2 >
  static void copy (std::multimap< T1, T2 > &dest, const std::multimap< T1, T2 > &src)
  Copies a multimap.

6.5.1 Detailed Description

Contains a set of useful static methods used by the application.

6.5.2 Member Function Documentation

6.5.2.1 template<typename T > static void base::copy (std::vector< T > & *dest,* const std::vector< T > & *src*) [inline], [static]

Copies a vector.

Here is the caller graph for this function:

6.5.2.2 template<typename T > static void base::copy (std::vector< std::vector< T > > & *dest,* const std::vector< std::vector< T > > & *src*) [inline], [static]

Copies a vector of vectors.

Here is the call graph for this function:

6.5.2.3 template<typename T1 , typename T2 > static void base::copy (std::multimap< T1, T2 > & *dest,* const std::multimap< T1, T2 > & *src*) [inline], [static]

Copies a multimap.

6.5.2.4 template<typename TYPE > static std::string base::to_string (TYPE *val*) [inline], [static]

Converts a value to a string.

Parameters

val	Value to convert.

Here is the caller graph for this function:

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

- base.h

6.6 data::BaseFile< IO > Class Template Reference

This is a wrapper class for the `FILE` functions that provides all the functionality to handle files safely.

`#include <file.h>`

Collaboration diagram for data::BaseFile< IO >:

Public Types

- typedef SHARED_PTR< BaseFile< IO > > Ptr

  Safe pointer to this class.

Public Member Functions

- BaseFile ()

  Initialized the internal file pointer to `NULL`.
- bool Open (const char *file_name, const char *access)

  Opens a file with a specific access mode.
- bool Open (const string &file_name, const char *access)

  Opens a file with a specific access mode.
- template<class IO2 >
  bool Open (const BaseFile< IO2 > &file, const char *access)

  Opens a file with a specific access mode given an already opened `File` object.
- bool OpenForReading (const char *file_name)
- bool OpenForReading (const string &file_name)
- template<class IO2 >
  bool OpenForReading (const BaseFile< IO2 > &file)
- bool OpenForWriting (const char *file_name)
- bool OpenForWriting (const string &file_name)
- template<class IO2 >
  bool OpenForWriting (const BaseFile< IO2 > &file)
- bool Seek (int offset, int origin=SEEK_SET) const

  Changes the current position of the file.
- void Close ()

  Closes the file.
- uint64_t GetOffset () const

  Returns the current file position.
- int IsEOF () const

  Returns the EOF status (`feof`) of the file.
- int GetDescriptor () const

  Returns the file descriptor.
- uint64_t GetSize () const

  Return the current size of the file, without modifying the file position.
- int ReadByte () const

  Reads a byte from the file.
- template<typename T >
  bool Read (T *value, int num_bytes=sizeof(T)) const

  Reads a value from the file.
- template<typename T >
  bool ReadReverse (T *value, int num_bytes=sizeof(T)) const

  Reads a value from the file in reverse order.
- int WriteByte (int c) const

  Writes a byte to the file.
- template<typename T >
  bool Write (T *value, int num_bytes=sizeof(T)) const

  Writes a value to the file.
- template<typename T >
  bool WriteReverse (T *value, int num_bytes=sizeof(T)) const

  Writes a value to the file in reverse order.
- bool IsValid () const

  Returns `true` if the file pointer is not `NULL`.
- operator bool () const

  Returns `true` if the file pointer is not `NULL`.
- virtual ~BaseFile ()

  The destructor closes the file.

Static Public Member Functions

- static bool Exists (const char *file_name)

  Returns `true` if the given file exists.

Private Attributes

- FILE * file_ptr

  File pointer.

6.6.1 Detailed Description

template<class IO>
class data::BaseFile< IO >

This is a wrapper class for the `FILE` functions that provides all the functionality to handle files safely.

It is defined as a template in order to allow to use either the locked or the unlocked API, by means of the structs `LockedAccess` and `UnlockedAccess`. The unlocked API is not thread-safe, but it provides faster file operations.

See also

LockedAccess
UnlockedAccess

6.6.2 Member Typedef Documentation

6.6.2.1 template<class IO> typedef SHARED_PTR< BaseFile<IO> > data::BaseFile< IO >::Ptr

Safe pointer to this class.

6.6.3 Constructor & Destructor Documentation

6.6.3.1 template<class IO> data::BaseFile< IO >::BaseFile ( ) [inline]

Initialized the internal file pointer to `NULL`.

6.6.3.2 template<class IO> virtual data::BaseFile< IO >::~BaseFile ( ) [inline], [virtual]

The destructor closes the file.

6.6.4 Member Function Documentation

6.6.4.1 template<class IO> void data::BaseFile< IO >::Close ( ) [inline]

Closes the file.

Here is the caller graph for this function:

6.6.4.2 template<class IO> static bool data::BaseFile< IO >::Exists (const char * *file_name*) [inline], [static]

Returns `true` if the given file exists.

This is a wrapper of the system funcion `stat`.

6.6.4.3 template<class IO> int data::BaseFile< IO >::GetDescriptor ( ) const [inline]

Returns the file descriptor.

Here is the caller graph for this function:

6.6.4.4 template<class IO> uint64_t data::BaseFile< IO >::GetOffset ( ) const [inline]

Returns the current file position.

Here is the caller graph for this function:

6.6.4.5 template<class IO> uint64_t data::BaseFile< IO >::GetSize ( ) const [inline]

Return the current size of the file, without modifying the file position.

Here is the caller graph for this function:

6.6.4.6 template<class IO> int data::BaseFile< IO >::IsEOF ( ) const [inline]

Returns the EOF status (`feof`) of the file.

6.6.4.7 template<class IO> bool data::BaseFile< IO >::IsValid ( ) const [inline]

Returns `true` if the file pointer is not `NULL`.

Here is the caller graph for this function:

6.6.4.8 template<class IO> bool data::BaseFile< IO >::Open (const char * *file_name*, const char * *access*) [inline]

Opens a file with a specific access mode.

Parameters

file_name	Path name of the file to open.
access	Access mode as a `fopen` compatible format string.

Returns

`true` if successful.

Here is the caller graph for this function:

6.6.4.9 template<class IO> bool data::BaseFile< IO >::Open (const string & *file_name,* const char * *access*) [inline]

Opens a file with a specific access mode.

Parameters

file_name	Path name of the file to open.
access	Access mode as a `fopen` compatible format string.

Returns

`true` if successful.

6.6.4.10 template<class IO> template<class IO2 > bool data::BaseFile< IO >::Open (const BaseFile< IO2 > & *file,* const char * *access*) [inline]

Opens a file with a specific access mode given an already opened `File` object.

The descriptor of the opened file is duplicated and re-opened with the access mode given.

Parameters

file	Opened file.
access	Access mode as a `fopen` compatible format string.

Returns

`true` if successful.

6.6.4.11 template<class IO> bool data::BaseFile< IO >::OpenForReading (const char * *file_name*) [inline]

Here is the caller graph for this function:

6.6.4.12 template<class IO> bool data::BaseFile< IO >::OpenForReading (const string & *file_name*) [inline]

6.6.4.13 template<class IO> template<class IO2 > bool data::BaseFile< IO >::OpenForReading (const BaseFile< IO2 > & *file*) [inline]

6.6.4.14 template<class IO> bool data::BaseFile< IO >::OpenForWriting (const char * *file_name*) [inline]

6.6.4.15 template<class IO> bool data::BaseFile< IO >::OpenForWriting (const string & *file_name*) [inline]

6.6.4.16 template<class IO> template<class IO2 > bool data::BaseFile< IO >::OpenForWriting (const BaseFile< IO2 > & *file*) [inline]

6.6.4.17 template<class IO> data::BaseFile< IO >::operator bool ( ) const [inline]

Returns `true` if the file pointer is not `NULL`.

6.6.4.18 template<class IO> template<typename T > bool data::BaseFile< IO >::Read (T * *value*, int *num_bytes* = `sizeof(T)`) const [inline]

Reads a value from the file.

Parameters

value	Pointer to the value where to store.
num_bytes	Number of bytes to read (by default, the size of the value).

Returns

`true` if successful.

Here is the caller graph for this function:

6.6.4.19 template<class IO> int data::BaseFile< IO >::ReadByte ( ) const [inline]

Reads a byte from the file.

6.6.4.20 template<class IO> template<typename T > bool data::BaseFile< IO >::ReadReverse (T * *value*, int *num_bytes* = `sizeof(T)`) const [inline]

Reads a value from the file in reverse order.

Parameters

value	Pointer to the value where to store.
num_bytes	Number of bytes to read (by default, the size of the value).

Returns

`true` if successful.

Here is the caller graph for this function:

6.6.4.21 template<class IO> bool data::BaseFile< IO >::Seek (int *offset*, int *origin* = `SEEK_SET`) const [inline]

Changes the current position of the file.

Parameters

offset	Offset to add to the current position.
origin	Origin to use for the change (`SEEK_SET` by default).

Returns

`true` if successful.

Here is the caller graph for this function:

6.6.4.22 template<class IO> template<typename T > bool data::BaseFile< IO >::Write (T * *value*, int *num_bytes* = `sizeof(T)`) const [inline]

Writes a value to the file.

Parameters

value	Pointer to the value.
num_bytes	Number of bytes to write (by default, the size of the value).

Returns

`true` if successful.

Here is the caller graph for this function:

6.6.4.23 template<class IO> int data::BaseFile< IO >::WriteByte (int *c*) const [inline]

Writes a byte to the file.

6.6.4.24 template<class IO> template<typename T > bool data::BaseFile< IO >::WriteReverse (T * *value*, int *num_bytes* = `sizeof(T)`) const [inline]

Writes a value to the file in reverse order.

Parameters

value	Pointer to the value.
num_bytes	Number of bytes to write (by default, the size of the value).

Returns

`true` if successful.

6.6.5 Member Data Documentation

6.6.5.1 template<class IO> FILE* data::BaseFile< IO >::file_ptr [private]

File pointer.

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

- data/file.h

6.7 data::BaseStream< StreamClass, StreamOperator > Class Template Reference

This template is used as the base for the input/output stream classes.

`#include <serialize.h>`

Collaboration diagram for data::BaseStream< StreamClass, StreamOperator >:

Public Member Functions

- BaseStream ()
  Initializes the status to `false`.
- StreamClass & Open (const char *file_name)
  Opens a file for serialization.
- StreamClass & Open (const char *file_name, const char *access)
  Opens a file for serialization.
- StreamClass & Close ()
  Closes the file of the serialization and finish the serialization.
- StreamClass & SerializeBytes (void *ptr, int num_bytes)
  Serializes a number of bytes.
- template<typename T >
  StreamClass & operator& (T &var)
  This operator overloading is the key of the serialization mechanism.
- bool result () const
  Returns the internal serialization status.
- operator bool () const
  Return the internal serialization status.
- virtual ~BaseStream ()
  The destructor automatically closes the file-.

Protected Attributes

- File file_

  File used for the serialization.
- bool result_

  Internal current status of the serialization.

6.7.1 Detailed Description

template<typename StreamClass, typename StreamOperator>
class data::BaseStream< StreamClass, StreamOperator >

This template is used as the base for the input/output stream classes.

Contains the basic functionality for the serialization with files, composed by a file object and an internal status. This status is updated with each operation and, in a sequence of serialization, this is stopped just after this status is set to `false`.

In order to have a type serializable, it must comply with one of these requirements: i) to have implemented a "serializer" with the class `Serializer`, or ii) to have defined a member method called `SerializeWith`.

The first option is useful for the basic types (int, float, etc.) and for those classes already defined and that can not be modified. The second option is more elegant for those classes that can be modified specifically for serialization.

The `SerializeWith` method must be defined as follows:

```
template<typename T> T& SerializeWith(T& stream) { return (stream & member1
& member2 & ...); }
```

See also

Serializer
InputStream
OutputStream

6.7.2 Constructor & Destructor Documentation

6.7.2.1 template<typename StreamClass, typename StreamOperator> data::BaseStream< StreamClass, StreamOperator >::BaseStream ( ) [inline]

Initializes the status to `false`.

6.7.2.2 template<typename StreamClass, typename StreamOperator> virtual data::BaseStream< StreamClass, StreamOperator >::~BaseStream ( ) [inline], [virtual]

The destructor automatically closes the file-.

6.7.3 Member Function Documentation

6.7.3.1 template<typename StreamClass, typename StreamOperator> StreamClass& data::BaseStream< StreamClass, StreamOperator >::Close ( ) [inline]

Closes the file of the serialization and finish the serialization.

6.7.3.2 template<typename StreamClass, typename StreamOperator> StreamClass& data::BaseStream< StreamClass, StreamOperator >::Open (const char * *file_name*) [inline]

Opens a file for serialization.

Parameters

file_name	Path name of the file to open.

Returns

*this.

Here is the caller graph for this function:

6.7.3.3 template<typename StreamClass, typename StreamOperator> StreamClass& data::BaseStream< StreamClass, StreamOperator >::Open (const char * *file_name,* const char * *access*) [inline]

Opens a file for serialization.

Parameters

file_name	Path name of the file to open.
access	Access mode to use to open the file.

Returns

*this.

6.7.3.4 template<typename StreamClass, typename StreamOperator> data::BaseStream< StreamClass, StreamOperator >::operator bool ( ) const [inline]

Return the internal serialization status.

6.7.3.5 template<typename StreamClass, typename StreamOperator> template<typename T > StreamClass& data::BaseStream< StreamClass, StreamOperator >::operator& (T & *var*) [inline]

This operator overloading is the key of the serialization mechanism.

6.7.3.6 template<typename StreamClass, typename StreamOperator> bool data::BaseStream< StreamClass, StreamOperator >::result ( ) const [inline]

Returns the internal serialization status.

6.7.3.7 template<typename StreamClass, typename StreamOperator> StreamClass& data::BaseStream< StreamClass, StreamOperator >::SerializeBytes (void * *ptr,* int *num_bytes*) [inline]

Serializes a number of bytes.

Depending on the stream operator in the template, this serialization is either a read or a write operation.

Parameters

ptr	Pointer to the buffer.
num_bytes	Number of bytes.

Returns

`*this`.

Here is the caller graph for this function:

6.7.4 Member Data Documentation

6.7.4.1 template<typename StreamClass, typename StreamOperator> File data::BaseStream< StreamClass, StreamOperator >::file_ [protected]

File used for the serialization.

6.7.4.2 template<typename StreamClass, typename StreamOperator> bool data::BaseStream< StreamClass, StreamOperator >::result_ [protected]

Internal current status of the serialization.

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

- data/serialize.h

6.8 jpip::CacheModel Class Reference

The cache model of a JPIP client is handled using this class.

`#include <cache_model.h>`

Collaboration diagram for jpip::CacheModel:

Classes

- class Codestream

  Sub-class of the cache model class used to identify a codestream.

Public Member Functions

- CacheModel ()

  Empty constructor.
- CacheModel (const CacheModel &model)

  Copy constructor.
- CacheModel & operator= (const CacheModel &model)

  Copy assignment.
- CacheModel & operator+= (const CacheModel &model)

  Add the content of the given cache model.
- template<typename T >
  T & SerializeWith (T &stream)
- Codestream & GetCodestream (int num_codestream)

  Returns the reference of a codestream.
- int GetMetadata (int id)

  Returns the amount of a meta-data.
- int AddToMetadata (int id, int amount, bool complete=false)

  Increases the amount of a meta-data.
- template<int BIN_CLASS>
  int GetDataBin (int num_codestream, int id)

  Returns the amount of a data-bin item using the class `DataBinSelector`.
- template<int BIN_CLASS>
  int AddToDataBin (int num_codestream, int id, int amount, bool complete=false)

  Increases the amount of a data-bin item using the class `DataBinSelector`.
- bool IsFullMetadata () const

  Returns the full flag of the meta-datas.
- void SetFullMetadata ()

  Sets the full flag for the meta-datas to true.
- void Pack (int min_sum=1)

  Calls the `Pack` method of all the codestreams.
- void Clear ()

  Clear all the amounts.
- virtual ~CacheModel ()

Private Attributes

- bool full_meta

  Says if the meta-data has been totally sent.
- vector< int > meta_data

  Amounts for the meta-datas.
- vector< Codestream > codestreams

  Amounts for the codestreams.

6.8.1 Detailed Description

The cache model of a JPIP client is handled using this class.

It allows to maintain a cache model recording the amount of data sent by the server regarding the meta-datas, headers, tile-headers and precincts. This implementation only allows to record incremental amounts, from the beginning of each entity. The value `INT_MAX` is used to specify that an item is complete. This class is serializable.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 jpip::CacheModel::CacheModel ( ) [inline]

Empty constructor.

6.8.2.2 jpip::CacheModel::CacheModel (const CacheModel & *model*) [inline]

Copy constructor.

6.8.2.3 virtual jpip::CacheModel::~CacheModel ( ) [inline], [virtual]

6.8.3 Member Function Documentation

6.8.3.1 template<int BIN_CLASS> int jpip::CacheModel::AddToDataBin (int *num_codestream,* int *id,* int *amount,* bool *complete* = false) [inline]

Increases the amount of a data-bin item using the class DataBinSelector.

Parameters

num_codestream	Index number of the associated codestream.
id	Index number of the data-bin.
amount	Amount increment.
complete	true if the data-bin is complete after the increment.

Returns

the new amount value.

Here is the caller graph for this function:

6.8.3.2 int jpip::CacheModel::AddToMetadata (int *id,* int *amount,* bool *complete* = false) [inline]

Increases the amount of a meta-data.

Parameters

id	Index number of the meta-data.
amount	Amount increment.
complete	true if the meta-data is complete after the increment.

Returns

the new amount value.

Here is the caller graph for this function:

6.8.3.3 void jpip::CacheModel::Clear ( ) [inline]

Clear all the amounts.

Here is the caller graph for this function:

6.8.3.4 Codestream& jpip::CacheModel::GetCodestream (int *num_codestream*) [inline]

Returns the reference of a codestream.

Parameters

num_codestream	Index number of the codestream.

Here is the caller graph for this function:

6.8.3.5 template<int BIN_CLASS> int jpip::CacheModel::GetDataBin (int *num_codestream,* int *id*) [inline]

Returns the amount of a data-bin item using the class DataBinSelector.

Parameters

num_codestream	Index number of the associated codestream.
id	Index number of the data-bin.

Here is the caller graph for this function:

6.8.3.6 int jpip::CacheModel::GetMetadata (int *id*) [inline]

Returns the amount of a meta-data.

Parameters

id	Index number of the meta-data.

Here is the caller graph for this function:

6.8.3.7 bool jpip::CacheModel::IsFullMetadata ( ) const [inline]

Returns the full flag of the meta-datas.

Here is the caller graph for this function:

6.8.3.8 CacheModel& jpip::CacheModel::operator+= (const CacheModel & *model*) [inline]

Add the content of the given cache model.

6.8.3.9 CacheModel& jpip::CacheModel::operator= (const CacheModel & *model*) [inline]

Copy assignment.

Here is the call graph for this function:

6.8.3.10 void jpip::CacheModel::Pack (int *min_sum* = 1) [inline]

Calls the `Pack` method of all the codestreams.

Here is the caller graph for this function:

6.8.3.11 template<typename T > T& jpip::CacheModel::SerializeWith (T & *stream*) [inline]

6.8.3.12 void jpip::CacheModel::SetFullMetadata ( ) [inline]

Sets the full flag for the meta-datas to true.

Here is the caller graph for this function:

6.8.4 Member Data Documentation

6.8.4.1 vector<Codestream> jpip::CacheModel::codestreams [private]

Amounts for the codestreams.

6.8.4.2 bool jpip::CacheModel::full_meta [private]

Says if the meta-data has been totally sent.

6.8.4.3 vector<int> jpip::CacheModel::meta_data [private]

Amounts for the meta-datas.

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

- jpip/cache_model.h

6.9 ClientInfo Class Reference

Contains information of a connected client.

`#include <client_info.h>`

Collaboration diagram for ClientInfo:

Public Member Functions

- ClientInfo (int base_id, int sock, int father_sock)
  Initializes the object.
- int sock () const
  Returns the client socket.
- int base_id () const
  Returns the base identifier.
- int father_sock () const
  Returns the father socket.
- long bytes_sent () const
  Returns the total bytes sent.
- long time () const
  Returns the time spent from the starting of the connection.
- virtual ~ClientInfo ()

Private Attributes

- int sock_
  Client socket.
- int base_id_
  Base identifier.
- time_t tm_start
  When the connection started.
- int father_sock_
  Father socket.
- long bytes_sent_
  Total bytes sent.

6.9.1 Detailed Description

Contains information of a connected client.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 ClientInfo::ClientInfo (int *base_id*, int *sock*, int *father_sock*) [inline]

Initializes the object.

Parameters

base_id	Base identifier.
sock	Client socket.
father_sock	Father socket.

Here is the call graph for this function:

6.9.2.2 virtual ClientInfo::~ClientInfo ( ) [inline], [virtual]

6.9.3 Member Function Documentation

6.9.3.1 int ClientInfo::base_id ( ) const [inline]

Returns the base identifier.

Here is the caller graph for this function:

6.9.3.2 long ClientInfo::bytes_sent ( ) const [inline]

Returns the total bytes sent.

6.9.3.3 int ClientInfo::father_sock ( ) const [inline]

Returns the father socket.

Here is the caller graph for this function:

6.9.3.4 int ClientInfo::sock ( ) const [inline]

Returns the client socket.

Here is the caller graph for this function:

6.9.3.5 long ClientInfo::time ( ) const [inline]

Returns the time spent from the starting of the connection.

Here is the caller graph for this function:

6.9.4 Member Data Documentation

6.9.4.1 int ClientInfo::base_id_ [private]

Base identifier.

6.9.4.2 long ClientInfo::bytes_sent_ [private]

Total bytes sent.

6.9.4.3 int ClientInfo::father_sock_ [private]

Father socket.

6.9.4.4 int ClientInfo::sock_ [private]

Client socket.

6.9.4.5 time_t ClientInfo::tm_start [private]

When the connection started.

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

- client_info.h

6.10 ClientManager Class Reference

Handles a client connection with a dedicated thread.

`#include <client_manager.h>`

Collaboration diagram for ClientManager:

Public Member Functions

- ClientManager (AppConfig &_cfg, AppInfo &_app_info, IndexManager &_index_manager)
  Initializes the object.
- void Run (ClientInfo *client_info)
  Starts the handling of a client connection.
- void RunBasic (ClientInfo *client_info)
  Starts the handling of a client connection but it does not do anything.
- virtual ~ClientManager ()

Private Attributes

- AppConfig & cfg
  Application configuration.
- AppInfo & app_info
  Application run-time information.
- IndexManager & index_manager
  Index manager.

6.10.1 Detailed Description

Handles a client connection with a dedicated thread.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 ClientManager::ClientManager (AppConfig & *_cfg,* AppInfo & *_app_info,* IndexManager & *_index_manager*) [inline]

Initializes the object.

Parameters

_cfg	Application configuration.
_app_info	Application run-time information.
_index_manager	Index manager.

6.10.2.2 virtual ClientManager::~ClientManager ( ) [inline], [virtual]

6.10.3 Member Function Documentation

6.10.3.1 void ClientManager::Run (ClientInfo * *client_info*)

Starts the handling of a client connection.

Parameters

client_info	Client information.

Here is the call graph for this function:

Here is the caller graph for this function:

6.10.3.2 void ClientManager::RunBasic (ClientInfo * *client_info*)

Starts the handling of a client connection but it does not do anything.

This method is used for testing the architecture of the server.

Parameters

client_info	Client information.

Here is the call graph for this function:

Here is the caller graph for this function:

6.10.4 Member Data Documentation

6.10.4.1 AppInfo& ClientManager::app_info [private]

Application run-time information.

6.10.4.2 AppConfig& ClientManager::cfg [private]

Application configuration.

6.10.4.3 IndexManager& ClientManager::index_manager [private]

Index manager.

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

- client_manager.h
- client_manager.cc

6.11 jpip::CacheModel::Codestream Class Reference

Sub-class of the cache model class used to identify a codestream.

`#include <cache_model.h>`

Collaboration diagram for jpip::CacheModel::Codestream:

Public Member Functions

- Codestream ()

  Initializes all the members to zero.
- Codestream (const Codestream &model)

  Copy constructor.
- Codestream & operator= (const Codestream &model)

  Copy assignment.
- Codestream & operator+= (const Codestream &model)

  Add the content of the given codestream cache model.
- template<typename T >
  T & SerializeWith (T &stream)
- int GetMainHeader () const

  Returns the amount of the main header.
- int GetTileHeader () const

  Returns the amount of the tile header.
- int AddToMainHeader (int amount, bool complete=false)

  Increases the amount of the main header.
- int AddToTileHeader (int amount, bool complete=false)

  Increases the amount of the tile header.
- int GetPrecinct (int num_precinct)

  Returns the amount of a precinct.
- int AddToPrecinct (int num_precinct, int amount, bool complete=false)

  Increases the amount of a precinct.
- void Pack (int min_sum=1)

  Packs the information stored regarding the precincts, removing those initial elements that are consecutive and completes.

Private Attributes

- int header

  Amount for the header.
- int tile_header

  Amount for the tile-header.
- vector< int > precincts

  Amount for the precincts.
- int min_precinct

  Minimum identifier of the non-consecutive precinct completely sent.

6.11.1 Detailed Description

Sub-class of the cache model class used to identify a codestream.

This class is serializable.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 jpip::CacheModel::Codestream::Codestream ( ) [inline]

Initializes all the members to zero.

6.11.2.2 jpip::CacheModel::Codestream::Codestream (const Codestream & *model*) [inline]

Copy constructor.

6.11.3 Member Function Documentation

6.11.3.1 int jpip::CacheModel::Codestream::AddToMainHeader (int *amount*, bool *complete* = false) [inline]

Increases the amount of the main header.

Parameters

amount	Amount increment.
complete	`true` if the main header is complete after the increment.

Returns

the new amount value.

Here is the caller graph for this function:

6.11.3.2 int jpip::CacheModel::Codestream::AddToPrecinct (int *num_precinct,* int *amount,* bool *complete* = `false`) [inline]

Increases the amount of a precinct.

Parameters

num_precinct	Index number of the precinct.
amount	Amount increment.
complete	`true` if the precinct is complete after the increment.

Returns

the new amount value.

Here is the caller graph for this function:

6.11.3.3 int jpip::CacheModel::Codestream::AddToTileHeader (int *amount,* bool *complete* = `false`) [inline]

Increases the amount of the tile header.

Parameters

amount	Amount increment.
complete	`true` if the tile header is complete after the increment.

Returns

the new amount value.

Here is the caller graph for this function:

6.11.3.4 int jpip::CacheModel::Codestream::GetMainHeader ( ) const [inline]

Returns the amount of the main header.

Here is the caller graph for this function:

6.11.3.5 int jpip::CacheModel::Codestream::GetPrecinct (int *num_precinct*) [inline]

Returns the amount of a precinct.

Parameters

num_precinct	Index number of the precinct.

Here is the caller graph for this function:

6.11.3.6 int jpip::CacheModel::Codestream::GetTileHeader ( ) const [inline]

Returns the amount of the tile header.

Here is the caller graph for this function:

6.11.3.7 Codestream& jpip::CacheModel::Codestream::operator+= (const Codestream & *model*) [inline]

Add the content of the given codestream cache model.

6.11.3.8 Codestream& jpip::CacheModel::Codestream::operator= (const Codestream & *model*) [inline]

Copy assignment.

Here is the call graph for this function:

6.11.3.9 void jpip::CacheModel::Codestream::Pack (int *min_sum* = 1) [inline]

Packs the information stored regarding the precincts, removing those initial elements that are consecutive and completes.

Parameters

min_sum	Only the packing is performed if there are a number of items equal or greater than this value (1 by default).

6.11.3.10 template<typename T > T& jpip::CacheModel::Codestream::SerializeWith (T & *stream*) [inline]

6.11.4 Member Data Documentation

6.11.4.1 int jpip::CacheModel::Codestream::header [private]

Amount for the header.

6.11.4.2 int jpip::CacheModel::Codestream::min_precinct [private]

Minimum identifier of the non-consecutive precinct completely sent.

All the initial precincts already sent completely to the client are removed, so this value contains the next precinct. The vector `precincts` is related to the precincts starting from this index.

6.11.4.3 vector<int> jpip::CacheModel::Codestream::precincts [private]

Amount for the precincts.

6.11.4.4 int jpip::CacheModel::Codestream::tile_header [private]

Amount for the tile-header.

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

- jpip/cache_model.h

6.12 jpeg2000::CodestreamIndex Class Reference

Class used for indexing the information of a JPEG2000 codestream.

`#include <codestream_index.h>`

Collaboration diagram for jpeg2000::CodestreamIndex:

Public Member Functions

- CodestreamIndex ()
  Empty constructor.
- CodestreamIndex (const CodestreamIndex &index)
  Copy constructor.
- void Clear ()
  Clears the information.
- const CodestreamIndex & operator= (const CodestreamIndex &index)
  Copy assignment.
- template<typename T >
  T & SerializeWith (T &stream)
- virtual ~CodestreamIndex ()

Public Attributes

- FileSegment header
  Main header segment.
- vector< FileSegment > packets
  Tile-part packets segments.
- vector< FileSegment > PLT_markers
  PLT markers segments.

Friends

- ostream & operator<< (ostream &out, const CodestreamIndex &index)

6.12.1 Detailed Description

Class used for indexing the information of a JPEG2000 codestream.

The indexed information is the segment of the main header, the contiguous segments of packets (usually the data of each tile-part) and the segments of the existing PLT markers. This class can be printed and serialized.

See also

data::FileSegment

6.12.2 Constructor & Destructor Documentation

6.12.2.1 jpeg2000::CodestreamIndex::CodestreamIndex ( ) [inline]

Empty constructor.

6.12.2.2 jpeg2000::CodestreamIndex::CodestreamIndex (const CodestreamIndex & *index*) [inline]

Copy constructor.

6.12.2.3 virtual jpeg2000::CodestreamIndex::~CodestreamIndex ( ) [inline], [virtual]

6.12.3 Member Function Documentation

6.12.3.1 void jpeg2000::CodestreamIndex::Clear ( ) [inline]

Clears the information.

6.12.3.2 const CodestreamIndex& jpeg2000::CodestreamIndex::operator= (const CodestreamIndex & *index*) [inline]

Copy assignment.

Here is the call graph for this function:

6.12.3.3 template<typename T > T& jpeg2000::CodestreamIndex::SerializeWith (T & *stream*) [inline]

6.12.4 Friends And Related Function Documentation

6.12.4.1 ostream& operator<< (ostream & *out,* const CodestreamIndex & *index*) [friend]

6.12.5 Member Data Documentation

6.12.5.1 FileSegment jpeg2000::CodestreamIndex::header

Main header segment.

6.12.5.2 vector<FileSegment> jpeg2000::CodestreamIndex::packets

Tile-part packets segments.

6.12.5.3 vector<FileSegment> jpeg2000::CodestreamIndex::PLT_markers

PLT markers segments.

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

- jpeg2000/codestream_index.h

6.13 jpeg2000::CodingParameters Class Reference

Contains the coding parameters of a JPEG2000 image codestream.

`#include <coding_parameters.h>`

Collaboration diagram for jpeg2000::CodingParameters:

Public Types

- enum {
  LRCP_PROGRESSION = 0, RLCP_PROGRESSION = 1, RPCL_PROGRESSION = 2, PCRL_PROGRESSION = 3,
  CPRL_PROGRESSION = 4 }
  All the progression orders defined in the JPEG2000 standard (Part 1).
- typedef SHARED_PTR< CodingParameters > Ptr
  Pointer to an object of this class.

Public Member Functions

- CodingParameters ()
  Initializes the object.
- CodingParameters (const CodingParameters &cod_params)
  Copy constructor.
- const CodingParameters & operator= (const CodingParameters &cod_params)
  Copy assignment.
- template<typename T >
  T & SerializeWith (T &stream)
- bool IsResolutionProgression () const
  Returns `true` if the progression is RLCP or RPCL.
- Size GetPrecincts (int r, const Size &point)
  Returns a precinct coordinate adjusted to a given resolution level.
- int GetProgressionIndex (const Packet &packet)
  Returns the index of a packet according to the progression order.
- int GetPrecinctDataBinId (const Packet &packet)
  Returns the data-bin identifier associated to the given packet.
- int GetClosestResolution (const Size &res_size, Size *res_image_size)
  Returns the resolution level according to the given size and the closest round policy.
- int GetRoundUpResolution (const Size &res_size, Size *res_image_size)
  Returns the resolution level according to the given size and the round-up round policy.
- int GetRoundDownResolution (const Size &res_size, Size *res_image_size)
  Returns the resolution level according to the given size and the round-down round policy.
- virtual ~CodingParameters ()

Public Attributes

- Size size
  Image size.
- int num_levels
  Number of resolution levels.
- int num_layers
  Number of quality layers.
- int progression
  Progression order.
- int num_components
  Number of components.
- vector< Size > precinct_size
  Precinct sizes of each resolution level.

Private Member Functions

- void FillTotalPrecinctsVector ()
  Fills the vector `total_precincts`.
- int GetProgressionIndexRPCL (int l, int r, int c, int px, int py)
  Returns the index of a packet according to the RPCL progression.
- int GetProgressionIndexRLCP (int l, int r, int c, int px, int py)
  Returns the index of a packet according to the RLCP progression.
- int GetProgressionIndexLRCP (int l, int r, int c, int px, int py)
  Returns the index of a packet according to the LRCP progression.

Private Attributes

- vector< int > total_precincts
  Contains the number of precincts of each resolution level.

Friends

- ostream & operator<< (ostream &out, const CodingParameters &params)

6.13.1 Detailed Description

Contains the coding parameters of a JPEG2000 image codestream.

This class can be serialized and printed.

6.13.2 Member Typedef Documentation

6.13.2.1 typedef SHARED_PTR<CodingParameters> jpeg2000::CodingParameters::Ptr

Pointer to an object of this class.

6.13.3 Member Enumeration Documentation

6.13.3.1 anonymous enum

All the progression orders defined in the JPEG2000 standard (Part 1).

Enumerator

LRCP_PROGRESSION LRCP.
RLCP_PROGRESSION RLCP.
RPCL_PROGRESSION RPCL.
PCRL_PROGRESSION PCRL.
CPRL_PROGRESSION CPRL.

6.13.4 Constructor & Destructor Documentation

6.13.4.1 jpeg2000::CodingParameters::CodingParameters ( ) [inline]

Initializes the object.

6.13.4.2 jpeg2000::CodingParameters::CodingParameters (const CodingParameters & *cod_params*) [inline]

Copy constructor.

6.13.4.3 virtual jpeg2000::CodingParameters::~CodingParameters ( ) [inline], [virtual]

6.13.5 Member Function Documentation

6.13.5.1 void jpeg2000::CodingParameters::FillTotalPrecinctsVector ( ) [private]

Fills the vector `total_precincts`.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.5.2 int jpeg2000::CodingParameters::GetClosestResolution (const Size & *res_size*, Size * *res_image_size*)

Returns the resolution level according to the given size and the closest round policy.

Parameters

res_size	Resolution size.
res_image_size	Image size associated to the resolution level returned.

Here is the caller graph for this function:

6.13.5.3 int jpeg2000::CodingParameters::GetPrecinctDataBinId (const Packet & *packet*) [inline]

Returns the data-bin identifier associated to the given packet.

Parameters

packet	Packet information.

Here is the call graph for this function:

6.13.5.4 Size jpeg2000::CodingParameters::GetPrecincts (int *r*, const Size & *point*) [inline]

Returns a precinct coordinate adjusted to a given resolution level.

Parameters

r	Resolution level.
point	Precinct coordinate.

Here is the caller graph for this function:

6.13.5.5 int jpeg2000::CodingParameters::GetProgressionIndex (const Packet & *packet*) [inline]

Returns the index of a packet according to the progression order.

Parameters

packet	Packet information.

Here is the call graph for this function:

6.13.5.6 int jpeg2000::CodingParameters::GetProgressionIndexLRCP (int *l*, int *r*, int *c*, int *px*, int *py*) [inline], [private]

Returns the index of a packet according to the LRCP progression.

Parameters

l	Quality layer.
r	Resolution level.
c	Component.
px	Precinct position X.
py	Precinct position Y.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.5.7 int jpeg2000::CodingParameters::GetProgressionIndexRLCP (int *l*, int *r*, int *c*, int *px*, int *py*) [inline], [private]

Returns the index of a packet according to the RLCP progression.

Parameters

l	Quality layer.
r	Resolution level.
c	Component.
px	Precinct position X.
py	Precinct position Y.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.5.8 int jpeg2000::CodingParameters::GetProgressionIndexRPCL (int *l*, int *r*, int *c*, int *px*, int *py*) [inline], [private]

Returns the index of a packet according to the RPCL progression.

Parameters

l	Quality layer.
r	Resolution level.
c	Component.
px	Precinct position X.
py	Precinct position Y.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.5.9 int jpeg2000::CodingParameters::GetRoundDownResolution (const Size & *res_size*, Size * *res_image_size*)

Returns the resolution level according to the given size and the round-down round policy.

Parameters

res_size	Resolution size.
res_image_size	Image size associated to the resolution level returned.

Here is the caller graph for this function:

6.13.5.10 int jpeg2000::CodingParameters::GetRoundUpResolution (const Size & *res_size,* Size * *res_image_size*)

Returns the resolution level according to the given size and the round-up round policy.

Parameters

res_size	Resolution size.
res_image_size	Image size associated to the resolution level returned.

Here is the caller graph for this function:

6.13.5.11 bool jpeg2000::CodingParameters::IsResolutionProgression ( ) const [inline]

Returns `true` if the progression is RLCP or RPCL.

6.13.5.12 const CodingParameters& jpeg2000::CodingParameters::operator= (const CodingParameters & *cod_params*) [inline]

Copy assignment.

Here is the call graph for this function:

6.13.5.13 template<typename T > T& jpeg2000::CodingParameters::SerializeWith (T & *stream*) [inline]

6.13.6 Friends And Related Function Documentation

6.13.6.1 ostream& operator<< (ostream & *out,* const CodingParameters & *params*) [friend]

6.13.7 Member Data Documentation

6.13.7.1 int jpeg2000::CodingParameters::num_components

Number of components.

6.13.7.2 int jpeg2000::CodingParameters::num_layers

Number of quality layers.

6.13.7.3 int jpeg2000::CodingParameters::num_levels

Number of resolution levels.

6.13.7.4 vector<Size> jpeg2000::CodingParameters::precinct_size

Precinct sizes of each resolution level.

6.13.7.5 int jpeg2000::CodingParameters::progression

Progression order.

6.13.7.6 Size jpeg2000::CodingParameters::size

Image size.

6.13.7.7 vector<int> jpeg2000::CodingParameters::total_precincts [private]

Contains the number of precincts of each resolution level.

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

- jpeg2000/coding_parameters.h
- jpeg2000/coding_parameters.cc

6.14 AppInfo::Data Struct Reference

Contains the data block that is maintained in shared memory.

Collaboration diagram for AppInfo::Data:

Public Member Functions

- void Reset ()

  Clears the values.

Public Attributes

- int father_pid

  PID of the father process.
- int child_pid

  PID of the child process.
- int num_connections

  Number of open connections.
- int child_iterations

  Number of iterations done by the child.

6.14.1 Detailed Description

Contains the data block that is maintained in shared memory.

6.14.2 Member Function Documentation

6.14.2.1 void AppInfo::Data::Reset ( ) [inline]

Clears the values.

6.14.3 Member Data Documentation

6.14.3.1 int AppInfo::Data::child_iterations

Number of iterations done by the child.

6.14.3.2 int AppInfo::Data::child_pid

PID of the child process.

6.14.3.3 int AppInfo::Data::father_pid

PID of the father process.

6.14.3.4 int AppInfo::Data::num_connections

Number of open connections.

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

- app_info.h

6.15 jpip::DataBinClass Class Reference

Class that contains the definitions of all the data-bin classes defined for the JPIP protocol.

`#include <jpip.h>`

Collaboration diagram for jpip::DataBinClass:

Public Types

- enum {
  PRECINCT = 0, EXTENDED_PRECINCT = 1, TILE_HEADER = 2, TILE_DATA = 4,
  EXTENDED_TILE = 5, MAIN_HEADER = 6, META_DATA = 8 }

Static Public Member Functions

- static const char * GetName (int class_name)

  Returns a string with the name of the databin class name given,.

Private Member Functions

- DataBinClass ()

6.15.1 Detailed Description

Class that contains the definitions of all the data-bin classes defined for the JPIP protocol.

It is not possible to create an object of this class.

6.15.2 Member Enumeration Documentation

6.15.2.1 anonymous enum

Enumerator

PRECINCT Class identifier for precinct data-bins.

EXTENDED_PRECINCT Class identifier for extended precinct data-bins.

TILE_HEADER Class identifier for tile header data-bins.

TILE_DATA Class identifier for tile data-bins.

EXTENDED_TILE Class identifier for extended tile data-bins.

MAIN_HEADER Class identifier for main header data-bins.

META_DATA Class identifier for meta-data data-bins.

6.15.3 Constructor & Destructor Documentation

6.15.3.1 jpip::DataBinClass::DataBinClass ( ) [inline], [private]

6.15.4 Member Function Documentation

6.15.4.1 const char * jpip::DataBinClass::GetName (int *class_name*) [static]

Returns a string with the name of the databin class name given,.

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

- jpip/jpip.h
- jpip/jpip.cc

6.16 jpip::DataBinSelector< BIN_CLASS > Struct Template Reference

Template class that is specialized for allowing basic operations (add and get) with cache models depending on the data-bin classes.

`#include <cache_model.h>`

Collaboration diagram for jpip::DataBinSelector< BIN_CLASS >:

6.16.1 Detailed Description

template<int BIN_CLASS>
struct jpip::DataBinSelector< BIN_CLASS >

Template class that is specialized for allowing basic operations (add and get) with cache models depending on the data-bin classes.

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

- jpip/cache_model.h

6.17 jpip::DataBinSelector< DataBinClass::MAIN_HEADER > Struct Template Reference

`#include <cache_model.h>`

Collaboration diagram for jpip::DataBinSelector< DataBinClass::MAIN_HEADER >:

Static Public Member Functions

- static int Get (CacheModel &model, int num_codestream, int id)
- static int AddTo (CacheModel &model, int num_codestream, int id, int amount, bool complete)

6.17.1 Member Function Documentation

6.17.1.1 static int jpip::DataBinSelector< DataBinClass::MAIN_HEADER >::AddTo (CacheModel & *model,* int *num_codestream,* int *id,* int *amount,* bool *complete*) [inline], [static]

Here is the call graph for this function:

6.17.1.2 static int jpip::DataBinSelector< DataBinClass::MAIN_HEADER >::Get (CacheModel & *model,* int *num_codestream,* int *id*) [inline], [static]

Here is the call graph for this function:

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

- jpip/cache_model.h

6.18 jpip::DataBinSelector< DataBinClass::META_DATA > Struct Template Reference

`#include <cache_model.h>`

Collaboration diagram for jpip::DataBinSelector< DataBinClass::META_DATA >:

Static Public Member Functions

- static int Get (CacheModel &model, int num_codestream, int id)
- static int AddTo (CacheModel &model, int num_codestream, int id, int amount, bool complete)

6.18.1 Member Function Documentation

6.18.1.1 static int jpip::DataBinSelector< DataBinClass::META_DATA >::AddTo (CacheModel & *model,* int *num_codestream,* int *id,* int *amount,* bool *complete*) [inline], [static]

Here is the call graph for this function:

6.18.1.2 static int jpip::DataBinSelector< DataBinClass::META_DATA >::Get (CacheModel & *model,* int *num_codestream,* int *id*) [inline], [static]

Here is the call graph for this function:

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

- jpip/cache_model.h

6.19 jpip::DataBinSelector< DataBinClass::PRECINCT > Struct Template Reference

`#include <cache_model.h>`

Collaboration diagram for jpip::DataBinSelector< DataBinClass::PRECINCT >:

Static Public Member Functions

- static int Get (CacheModel &model, int num_codestream, int id)
- static int AddTo (CacheModel &model, int num_codestream, int id, int amount, bool complete)

6.19.1 Member Function Documentation

6.19.1.1 static int jpip::DataBinSelector< DataBinClass::PRECINCT >::AddTo (CacheModel & *model,* int *num_codestream,* int *id,* int *amount,* bool *complete*) [inline], [static]

Here is the call graph for this function:

6.19.1.2 static int jpip::DataBinSelector< DataBinClass::PRECINCT >::Get (CacheModel & *model,* int *num_codestream,* int *id*) [inline], [static]

Here is the call graph for this function:

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

- jpip/cache_model.h

6.20 jpip::DataBinSelector< DataBinClass::TILE_HEADER > Struct Template Reference

`#include <cache_model.h>`

Collaboration diagram for jpip::DataBinSelector< DataBinClass::TILE_HEADER >:

Static Public Member Functions

- static int Get (CacheModel &model, int num_codestream, int id)
- static int AddTo (CacheModel &model, int num_codestream, int id, int amount, bool complete)

6.20.1 Member Function Documentation

6.20.1.1 static int jpip::DataBinSelector< DataBinClass::TILE_HEADER >::AddTo (CacheModel & *model,* int *num_codestream,* int *id,* int *amount,* bool *complete*) [inline], [static]

Here is the call graph for this function:

6.20.1.2 static int jpip::DataBinSelector< DataBinClass::TILE_HEADER >::Get (CacheModel & *model,* int *num_codestream,* int *id*) [inline], [static]

Here is the call graph for this function:

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

- jpip/cache_model.h

6.21 jpip::DataBinServer Class Reference

Contains the core functionality of a (JPIP) data-bin server, which maintains a cache model and is capable of generating data chunks of variable length;.

`#include <databin_server.h>`

Collaboration diagram for jpip::DataBinServer:

Public Member Functions

- DataBinServer ()

  Initializes the obect.
- bool end_woi () const

  Returns `true` if the end of the WOI has been reached, that is, there is not more associated packets to send.
- bool Reset (const ImageIndex::Ptr image_index)

  Resets the server assigning a new image to serve.
- bool SetRequest (const Request &req)

  Sets the new current request to take into account for generating the chunks of data.
- bool GenerateChunk (char *buff, int *len, bool *last)

  Generates a new chunk of data for the current image and WOI, according to the last indicated request.
- virtual ~DataBinServer ()

Private Types

- enum { MINIMUM_SPACE = 60 }

Private Member Functions

- template<int BIN_CLASS>
  int WriteSegment (int num_codestream, int id, FileSegment segment, int offset=0, bool last=true)

  Writes a new data-bin segment or a part of it that is not already cached.
- int WritePlaceHolder (int num_codestream, int id, const PlaceHolder &place_holder, int offset=0, bool last=false)

  Writes a new place-holder segment, only if it is possible to write it completely.

Private Attributes

- WOI woi

  Current WOI.
- int pending

  Number of pending bytes.
- Range range

  Range of codestreams.
- bool has_woi

  `true` if the last request contained a WOI
- bool metareq

  `true` if the last request contained a "metareq"
- bool end_woi_

  `true` if the WOI has been completely sent
- File::Ptr file

  Pointer to the associated image file.
- int current_idx

  Current codestream index.
- bool eof

  `true` if the end has been reached and the last write operation could not be completed.
- CacheModel cache_model

  Cache model of the client.

- vector< File::Ptr > files

  List of files (for hyperlinked JPX files)
- WOIComposer woi_composer

  WOI composer for determining the packets.
- ImageIndex::Ptr im_index

  Pointer to the associated image index.
- DataBinWriter data_writer

  Data-bin writer for generating the chunks.

6.21.1 Detailed Description

Contains the core functionality of a (JPIP) data-bin server, which maintains a cache model and is capable of generating data chunks of variable length;.

6.21.2 Member Enumeration Documentation

6.21.2.1 anonymous enum [private]

Enumerator

MINIMUM_SPACE Minimum space in the chunk.

6.21.3 Constructor & Destructor Documentation

6.21.3.1 jpip::DataBinServer::DataBinServer ( ) [inline]

Initializes the obect.

6.21.3.2 virtual jpip::DataBinServer::~DataBinServer ( ) [inline], [virtual]

6.21.4 Member Function Documentation

6.21.4.1 bool jpip::DataBinServer::end_woi ( ) const [inline]

Returns `true` if the end of the WOI has been reached, that is, there is not more associated packets to send.

Here is the caller graph for this function:

6.21.4.2 bool jpip::DataBinServer::GenerateChunk (char * *buff*, int * *len*, bool * *last*)

Generates a new chunk of data for the current image and WOI, according to the last indicated request.

Parameters

buff	Pointer to the memory buffer.
len	Length of the memory buffer. It is modified by the method to indicate how many bytes have been written to the buffer.
last	Output parameter to indicates if this is the last chunk of data associated to the last request.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.21.4.3 bool jpip::DataBinServer::Reset (const ImageIndex::Ptr *image_index*)

Resets the server assigning a new image to serve.

It also resets the maintained cache model.

Parameters

image_index	Pointer to the new image index to use.

Returns

`true` if successful.

Here is the caller graph for this function:

6.21.4.4 bool jpip::DataBinServer::SetRequest (const Request & *req*)

Sets the new current request to take into account for generating the chunks of data.

Parameters

req	Request.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.21.4.5 int jpip::DataBinServer::WritePlaceHolder (int *num_codestream,* int *id,* const PlaceHolder & *place_holder,* int *offset* = `0`, bool *last* = `false`) [inline],[private]

Writes a new place-holder segment, only if it is possible to write it completely.

Parameters

num_codestream	Index number of the codestream.
id	Data-bin identifier.
place_holder	Place-holder information.
offset	Data-bin offset of the data (0 by default).
last	`true` if this is the last data of the data-bin.

Returns

1 if the segment content was completely written and/or cached, 0 if it was incompletely written (or not at all, if EOF flag is set), or -1 if an error was generated.

Here is the call graph for this function:

Here is the caller graph for this function:

6.21.4.6 template<int BIN_CLASS> int jpip::DataBinServer::WriteSegment (int *num_codestream,* int *id,* FileSegment *segment,* int *offset* = 0, bool *last* = true) [inline], [private]

Writes a new data-bin segment or a part of it that is not already cached.

Parameters

num_codestream	Index number of the codestream.
id	Data-bin identifier.
segment	File segment associated.
offset	Data-bin offset of the data (0 by default).
last	true if this is the last data of the data-bin.

Returns

1 if the segment content was completely written and/or cached, 0 if it was incompletely written (or not at all, if EOF flag is set), or -1 if an error was generated.

Here is the call graph for this function:

6.21.5 Member Data Documentation

6.21.5.1 CacheModel jpip::DataBinServer::cache_model [private]

Cache model of the client.

6.21.5.2 int jpip::DataBinServer::current_idx [private]

Current codestream index.

6.21.5.3 DataBinWriter jpip::DataBinServer::data_writer [private]

Data-bin writer for generating the chunks.

6.21.5.4 bool jpip::DataBinServer::end_woi_ [private]

true if the WOI has been completely sent

6.21.5.5 bool jpip::DataBinServer::eof [private]

`true` if the end has been reached and the last write operation could not be completed.

6.21.5.6 File::Ptr jpip::DataBinServer::file [private]

Pointer to the associated image file.

6.21.5.7 vector<File::Ptr> jpip::DataBinServer::files [private]

List of files (for hyperlinked JPX files)

6.21.5.8 bool jpip::DataBinServer::has_woi [private]

`true` if the last request contained a WOI

6.21.5.9 ImageIndex::Ptr jpip::DataBinServer::im_index [private]

Pointer to the associated image index.

6.21.5.10 bool jpip::DataBinServer::metareq [private]

`true` if the last request contained a "metareq"

6.21.5.11 int jpip::DataBinServer::pending [private]

Number of pending bytes.

6.21.5.12 Range jpip::DataBinServer::range [private]

Range of codestreams.

6.21.5.13 WOI jpip::DataBinServer::woi [private]

Current WOI.

6.21.5.14 WOIComposer jpip::DataBinServer::woi_composer [private]

WOI composer for determining the packets.

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

- jpip/databin_server.h
- jpip/databin_server.cc

6.22 jpip::DataBinWriter Class Reference

Class used to generate data-bin segments and write them into a memory buffer.

`#include <databin_writer.h>`

Collaboration diagram for jpip::DataBinWriter:

Public Member Functions

- DataBinWriter ()

  Initializes the object.
- DataBinWriter & SetBuffer (char *buf, int buf_len)

  Sets the associated memory buffer.
- DataBinWriter & ClearPreviousIds ()

  Clears the previous identifiers of data-bin class and codestream index numbers.
- DataBinWriter & SetCodestream (int value)

  Sets the current codestream.
- DataBinWriter & SetDataBinClass (int databin_class)

  Sets the current data-bin class.
- DataBinWriter & Write (uint64_t bin_id, uint64_t bin_offset, const File &file, const FileSegment &segment, bool last_byte=false)

  Writes a data-bin segment into the buffer.
- DataBinWriter & WritePlaceHolder (uint64_t bin_id, uint64_t bin_offset, const File &file, const PlaceHolder &place_holder, bool last_byte=false)

  Writes a place-holder segment into the buffer.
- DataBinWriter & WriteEmpty (uint64_t bin_id=0)

  Writes an empty segment.
- int GetCount () const

  Returns the number of bytes written.
- int GetFree () const

  Returns the number of bytes available.
- DataBinWriter & WriteEOR (int reason)

  Writes a EOR message into the buffer.
- operator bool () const

  Returns the `EOF` status of the object.
- virtual ~DataBinWriter ()

Private Member Functions

- template<typename T >
  DataBinWriter & WriteValue (T value)

  Writes a value into the buffer.
- DataBinWriter & WriteVBAS (uint64_t value)

  Writes a new integer value into the buffer coded as VBAS.
- DataBinWriter & WriteHeader (uint64_t bin_id, uint64_t bin_offset, uint64_t bin_length, bool last_byte=false)

  Writes a data-bin header into the buffer.

Private Attributes

- bool eof

  `true` if the end of the buffer has been reached and the last value could not be written.
- char * ini

  Pointer to the beginning of the buffer.
- char * ptr

  Current position of the buffer.
- char * end

  Pointer to the end of the buffer.
- int databin_class

  Current data-bin class.
- int codestream_idx

  Current codestream index number.
- int prev_databin_class

  Previous data-bin class.
- int prev_codestream_idx

  Previous codestream index number.

6.22.1 Detailed Description

Class used to generate data-bin segments and write them into a memory buffer.

See also

DataBinServer
DataBinClass
EOR

6.22.2 Constructor & Destructor Documentation

6.22.2.1 jpip::DataBinWriter::DataBinWriter ( ) [inline]

Initializes the object.

6.22.2.2 virtual jpip::DataBinWriter::~DataBinWriter ( ) [inline], [virtual]

6.22.3 Member Function Documentation

6.22.3.1 DataBinWriter& jpip::DataBinWriter::ClearPreviousIds ( ) [inline]

Clears the previous identifiers of data-bin class and codestream index numbers.

Returns

The object itself.

Here is the caller graph for this function:

6.22.3.2 int jpip::DataBinWriter::GetCount ( ) const [inline]

Returns the number of bytes written.

Here is the caller graph for this function:

6.22.3.3 int jpip::DataBinWriter::GetFree ( ) const [inline]

Returns the number of bytes available.

Here is the caller graph for this function:

6.22.3.4 jpip::DataBinWriter::operator bool ( ) const [inline]

Returns the EOF status of the object.

6.22.3.5 DataBinWriter& jpip::DataBinWriter::SetBuffer (char * *buf*, int *buf_len*) [inline]

Sets the associated memory buffer.

Parameters

buf	Memory buffer.
buf_len	Length of the memory buffer.

Returns

The object itself.

Here is the caller graph for this function:

6.22.3.6 DataBinWriter& jpip::DataBinWriter::SetCodestream (int *value*) [inline]

Sets the current codestream.

Parameters

value	Index number of the codestream.

Returns

The object itself.

Here is the caller graph for this function:

6.22.3.7 DataBinWriter& jpip::DataBinWriter::SetDataBinClass (int *databin_class*) [inline]

Sets the current data-bin class.

Parameters

databin_class	Data-bin class.

Returns

The object itself.

Here is the caller graph for this function:

6.22.3.8 DataBinWriter & jpip::DataBinWriter::Write (uint64_t *bin_id,* uint64_t *bin_offset,* const File & *file,* const FileSegment & *segment,* bool *last_byte* = false)

Writes a data-bin segment into the buffer.

Parameters

bin_id	Data-bin identifier.
bin_offset	Data-bin offset.
file	File from where to read the data.
segment	File segment of the data.
last_byte	true if the data contains the last byte of the data-bin.

Returns

The object itself.

Here is the call graph for this function:

Here is the caller graph for this function:

6.22.3.9 DataBinWriter & jpip::DataBinWriter::WriteEmpty (uint64_t *bin_id* = 0)

Writes an empty segment.

Parameters

bin_id	Data-bin identifier.

Returns

The object itself.

Here is the call graph for this function:

6.22.3.10 DataBinWriter& jpip::DataBinWriter::WriteEOR (int *reason*) [inline]

Writes a EOR message into the buffer.

Parameters

reason	Reason of the message.

Returns

The object itself.

Here is the caller graph for this function:

6.22.3.11 DataBinWriter & jpip::DataBinWriter::WriteHeader (uint64_t *bin_id,* uint64_t *bin_offset,* uint64_t *bin_length,* bool *last_byte* = `false`) [private]

Writes a data-bin header into the buffer.

Parameters

bin_id	Data-bin identifier.
bin_offset	Data-bin offset.
bin_length	Data-bin length.
last_byte	`true` if the data related to this header contains the last byte of the data-bin.

Returns

The object itself.

Here is the call graph for this function:

Here is the caller graph for this function:

6.22.3.12 DataBinWriter & jpip::DataBinWriter::WritePlaceHolder (uint64_t *bin_id,* uint64_t *bin_offset,* const File & *file,* const PlaceHolder & *place_holder,* bool *last_byte* = `false`)

Writes a place-holder segment into the buffer.

Parameters

bin_id	Data-bin identifier.
bin_offset	Data-bin offset.
file	File from where to read the data.
place_holder	Place-holder information.
last_byte	`true` if the data contains the last byte of the data-bin.

Returns

The object itself.

Here is the call graph for this function:

Here is the caller graph for this function:

6.22.3.13 template<typename T > DataBinWriter& jpip::DataBinWriter::WriteValue (T *value*) [inline], [private]

Writes a value into the buffer.

Parameters

value	Value to write.

Returns

The object itself.

6.22.3.14 DataBinWriter & jpip::DataBinWriter::WriteVBAS (uint64_t *value*) [private]

Writes a new integer value into the buffer coded as VBAS.

Parameters

value	Value to write.

Returns

The object itself.

Here is the caller graph for this function:

6.22.4 Member Data Documentation

6.22.4.1 int jpip::DataBinWriter::codestream_idx [private]

Current codestream index number.

6.22.4.2 int jpip::DataBinWriter::databin_class [private]

Current data-bin class.

6.22.4.3 char* jpip::DataBinWriter::end [private]

Pointer to the end of the buffer.

6.22.4.4 bool jpip::DataBinWriter::eof [private]

`true` if the end of the buffer has been reached and the last value could not be written.

6.22.4.5 char* jpip::DataBinWriter::ini [private]

Pointer to the beginning of the buffer.

6.22.4.6 int jpip::DataBinWriter::prev_codestream_idx [private]

Previous codestream index number.

6.22.4.7 int jpip::DataBinWriter::prev_databin_class [private]

Previous data-bin class.

6.22.4.8 char* jpip::DataBinWriter::ptr [private]

Current position of the buffer.

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

- jpip/databin_writer.h
- jpip/databin_writer.cc

6.23 jpip::EOR Class Reference

Class that contains all the definitions of the EOF messages defined for the JPIP protocol.

`#include <jpip.h>`

Collaboration diagram for jpip::EOR:

Public Types

- enum {
  IMAGE_DONE = 1, WINDOW_DONE = 2, WINDOW_CHANGE = 3, BYTE_LIMIT_REACHED = 4,
  QUALITY_LIMIT_REACHED = 5, SESSION_LIMIT_REACHED = 6, RESPONSE_LIMIT_REACHED = 7,
  NON_SPECIFIED = 0xFF }

Private Member Functions

- EOR ()

6.23.1 Detailed Description

Class that contains all the definitions of the EOF messages defined for the JPIP protocol.

It is not possible to create an object of this class.

6.23.2 Member Enumeration Documentation

6.23.2.1 anonymous enum

Enumerator

IMAGE_DONE EOR code sent when the server has transferred all available image information (not just information relevant to the requested view-window) to the client.

WINDOW_DONE EOR code sent when the server has transferred all available information that is relevant to the requested view-window.

WINDOW_CHANGE EOR code sent when the server is terminating its response in order to service a new request.

BYTE_LIMIT_REACHED EOR code sent when the server is terminating its response because the byte limit specified in a byte limit specified in a max length request field has been reached.

QUALITY_LIMIT_REACHED EOR code sent when the server is terminating its response because the quality limit specified in a quality request field has been reached.

SESSION_LIMIT_REACHED EOR code sent when the server is terminating its response because some limit on the session resources, e.g. a time limit, has been reached. No further request should be issued using a channel ID assigned in that session.

RESPONSE_LIMIT_REACHED EOR code sent when the server is terminating its response because some limit, e.g., a time limit, has been reached. If the request is issued in a session, further requests can still be issued using a channel ID assigned in that session.

NON_SPECIFIED EOR code sent when there is not any specific EOR reason.

6.23.3 Constructor & Destructor Documentation

6.23.3.1 jpip::EOR::EOR ( ) [inline],[private]

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

- jpip/jpip.h

6.24 ipc::Event Class Reference

IPC object that offers the functionality of an event (Windows IPC object), implemented by means of a combination of the pthread mutex and conditional variables API.

`#include <event.h>`

Inheritance diagram for ipc::Event:

Collaboration diagram for ipc::Event:

Public Types

- typedef SHARED_PTR< Event > Ptr

  Pointer to a Event object.

Public Member Functions

- virtual bool Init ()

  Initializes the object desactivated and with automatic reset.
- bool Init (bool manual_reset, bool initial_state=false)

  Initializes the object.
- virtual WaitResult Wait (int time_out=-1)

  Performs a wait operation with the object to get it.
- virtual bool Dispose ()

  Release the resources associated to the IPC object and sets the internal status to `false`.
- bool Set (bool new_state=true)

  Sets the state of the object.
- bool Get () const

  Returns the current activation state of the object.
- bool Pulse ()

  Generates the same result as if the event has automatic reset and the `Set` method is called with `true`, independently of the real reset type.
- bool Reset ()

  Desactivates the object.

Private Attributes

- bool state

  Current activation state of the event.
- bool manual_reset

  Indicates if the event reset is manual.
- pthread_cond_t condv

  Conditional variable information.
- pthread_mutex_t mutex

  Mutex information.

6.24.1 Detailed Description

IPC object that offers the functionality of an event (Windows IPC object), implemented by means of a combination of the pthread mutex and conditional variables API.

See also

IPCObject

6.24.2 Member Typedef Documentation

6.24.2.1 typedef SHARED_PTR<Event> ipc::Event::Ptr

Pointer to a Event object.

6.24.3 Member Function Documentation

6.24.3.1 bool ipc::Event::Dispose ( ) [virtual]

Release the resources associated to the IPC object and sets the internal status to `false`.

Returns

`true` if successful.

Reimplemented from ipc::IPCObject.

Here is the call graph for this function:

Here is the caller graph for this function:

6.24.3.2 bool ipc::Event::Get ( ) const [inline]

Returns the current activation state of the object.

Here is the call graph for this function:

6.24.3.3 virtual bool ipc::Event::Init ( ) [inline], [virtual]

Initializes the object desactivated and with automatic reset.

Returns

`true` if successful.

Reimplemented from ipc::IPCObject.

Here is the call graph for this function:

6.24.3.4 bool ipc::Event::Init (bool *manual_reset*, bool *initial_state* = `false`)

Initializes the object.

Parameters

manual_reset	`true` if the reset is manual.
initial_state	`true` if the initial state is activated.

Returns

`true` if successful.

Here is the call graph for this function:

6.24.3.5 bool ipc::Event::Pulse ( )

Generates the same result as if the event has automatic reset and the `Set` method is called with `true`, independently of the real reset type.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.24.3.6 bool ipc::Event::Reset ( ) [inline]

Desactivates the object.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.24.3.7 bool ipc::Event::Set (bool *new_state* = `true`)

Sets the state of the object.

If it is activated (with `true`) and the reset is manual, all the threads waiting for the object will be resumed. If the reset is not manual (automatic), only one thread will be resumed and the state will be set to `false` again.

Parameters

new_state	New state of the object.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.24.3.8 WaitResult ipc::Event::Wait (int *time_out* = `-1`) [virtual]

Performs a wait operation with the object to get it.

Parameters

time_out	Time out (infinite by default).

Returns

`WAIT_OBJECT` if successful, `WAIT_TIMEOUT` if time out or `WAIT_ERROR` is error.

Reimplemented from ipc::IPCObject.

Here is the call graph for this function:

Here is the caller graph for this function:

6.24.4 Member Data Documentation

6.24.4.1 pthread_cond_t ipc::Event::condv [private]

Conditional variable information.

6.24.4.2 bool ipc::Event::manual_reset [private]

Indicates if the event reset is manual.

6.24.4.3 pthread_mutex_t ipc::Event::mutex [private]

Mutex information.

6.24.4.4 bool ipc::Event::state [private]

Current activation state of the event.

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

- ipc/event.h
- ipc/event.cc

6.25 jpeg2000::FileManager Class Reference

Manages the image files of a repository, allowing read their indexing information, with a caching mechanism for efficiency.

```
#include <file_manager.h>
```

Collaboration diagram for jpeg2000::FileManager:

Public Member Functions

- string GetCacheFileName (const string &path_image_file)

  Returns the cache file name equivalent to the given image file name.
- FileManager ()

  Initializes the object.
- FileManager (string root_dir, string cache_dir)

  Initializes the object.
- bool Init (string root_dir="./", string cache_dir="./")

  Initializes the object.
- string root_dir () const

  Returns the root directory of the image repository.
- string cache_dir () const

  Returns the directory used for caching.
- bool ReadImage (const string &name_image_file, ImageInfo *image_info)

  Reads an image file and creates the associated cache file if it does not exist yet.
- virtual ~FileManager ()

Private Member Functions

- bool ExistCacheImage (const string &path_image_file, string *path_cache_file)

  Returns `true` if the cache file exists and it is updated.
- bool ReadBoxHeader (const File &fim, uint32_t *type_box, uint64_t *length_box)

  Reads the header information.
- bool ReadCodestream (const File &file, CodingParameters *params, CodestreamIndex *index)

  Reads the information of a codestream.
- bool ReadSIZMarker (const File &file, CodingParameters *params)

  Reads the information of a SIZ marker.
- bool ReadCODMarker (const File &file, CodingParameters *params)

  Reads the information of a COD marker.
- bool ReadSOTMarker (const File &file, CodestreamIndex *index)

  Reads the information of a SOT marker.
- bool ReadPLTMarker (const File &file, CodestreamIndex *index)

  Reads the information of a PLT marker.
- bool ReadSODMarker (const File &file, CodestreamIndex *index)

  Reads the information of a SOD marker.
- bool ReadJP2 (const File &file, ImageInfo *image_info)

  Reads the information of a JP2 image file.
- bool ReadJPX (const File &file, ImageInfo *image_info)

  Reads the information of a JPX image file.
- bool ReadNlstBox (const File &file, int *num_codestream, int length_box)

  Reads the information of a NLST box.
- bool ReadFlstBox (const File &file, uint64_t length_box, uint16_t *data_reference)

  Reads the information of a FLST box.
- bool ReadUrlBox (const File &file, uint64_t length_box, string *path_file)

  Reads the information of a URL box.

Private Attributes

- string root_dir_

  Root directory of the repository.
- string cache_dir_

  Caching directory.

6.25.1 Detailed Description

Manages the image files of a repository, allowing read their indexing information, with a caching mechanism for efficiency.

6.25.2 Constructor & Destructor Documentation

6.25.2.1 jpeg2000::FileManager::FileManager ( ) [inline]

Initializes the object.

6.25.2.2 jpeg2000::FileManager::FileManager (string *root_dir*, string *cache_dir*) [inline]

Initializes the object.

Parameters

root_dir	Root directory of the image repository.
cache_dir	Directory for caching.

Here is the call graph for this function:

6.25.2.3 virtual jpeg2000::FileManager::~FileManager ( ) [inline], [virtual]

6.25.3 Member Function Documentation

6.25.3.1 string jpeg2000::FileManager::cache_dir ( ) const [inline]

Returns the directory used for caching.

Here is the call graph for this function:

Here is the caller graph for this function:

6.25.3.2 bool jpeg2000::FileManager::ExistCacheImage (const string & *path_image_file*, string * *path_cache_file*) [private]

Returns `true` if the cache file exists and it is updated.

Parameters

path_image_file	Path of the image file.
path_cache_file	Receives the path of the associated cache file.

6.25.3.3 string jpeg2000::FileManager::GetCacheFileName (const string & *path_image_file*)

Returns the cache file name equivalent to the given image file name.

Here is the caller graph for this function:

6.25.3.4 bool jpeg2000::FileManager::Init (string *root_dir* = " . / ", string *cache_dir* = " . / ") [inline]

Initializes the object.

Parameters

root_dir	Root directory of the image repository.
cache_dir	Directory for caching.

Returns

`true` if successful

Here is the call graph for this function:

Here is the caller graph for this function:

6.25.3.5 bool jpeg2000::FileManager::ReadBoxHeader (const File & *fim*, uint32_t * *type_box*, uint64_t * *length_box*) [private]

Reads the header information.

of a JP2/JPX box.

Parameters

fim	Image file.
type_box	Receives the type of the box.
length_box	Receives the length of the box.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.6 bool jpeg2000::FileManager::ReadCodestream (const File & *file*, CodingParameters * *params*, CodestreamIndex * *index*) [private]

Reads the information of a codestream.

Parameters

file	Image file.
params	Receives the coding parameters.
index	Receives the indexing information.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.7 bool jpeg2000::FileManager::ReadCODMarker (const File & *file*, CodingParameters * *params*) [private]

Reads the information of a COD marker.

Parameters

file	Image file.
params	Pointer to the coding parameters to update.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.8 bool jpeg2000::FileManager::ReadFlstBox (const File & *file*, uint64_t *length_box*, uint16_t * *data_reference*) [private]

Reads the information of a FLST box.

Parameters

file	Image file.
length_box	Box length in bytes.
data_reference	Receives the data reference.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.9 bool jpeg2000::FileManager::ReadImage (const string & *name_image_file*, ImageInfo * *image_info*)

Reads an image file and creates the associated cache file if it does not exist yet.

Parameters

name_image_file	File name of the image.
image_info	Receives the information of the image.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.25.3.10 bool jpeg2000::FileManager::ReadJP2 (const File & *file*, ImageInfo * *image_info*) [private]

Reads the information of a JP2 image file.

Parameters

file	Image file.
image_info	Receives the image information.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.11 bool jpeg2000::FileManager::ReadJPX (const File & *file*, ImageInfo * *image_info*) [private]

Reads the information of a JPX image file.

Parameters

file	Image file.
image_info	Receives the image information.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.12 bool jpeg2000::FileManager::ReadNlstBox (const File & *file,* int * *num_codestream,* int *length_box*) [private]

Reads the information of a NLST box.

Parameters

file	Image file.
num_codestream	Receives the number of codestream read.
length_box	Box length in bytes.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.13 bool jpeg2000::FileManager::ReadPLTMarker (const File & *file,* CodestreamIndex * *index*) [private]

Reads the information of a PLT marker.

Parameters

file	Image file.
index	Pointer to the indexing information to update.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.14 bool jpeg2000::FileManager::ReadSIZMarker (const File & *file,* CodingParameters * *params*) [private]

Reads the information of a SIZ marker.

Parameters

file	Image file.
params	Pointer to the coding parameters to update.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.15 bool jpeg2000::FileManager::ReadSODMarker (const File & *file*, CodestreamIndex * *index*) [private]

Reads the information of a SOD marker.

Parameters

file	Image file.
index	Pointer to the indexing information to update.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.16 bool jpeg2000::FileManager::ReadSOTMarker (const File & *file*, CodestreamIndex * *index*) [private]

Reads the information of a SOT marker.

Parameters

file	Image file.
index	Pointer to the indexing information to update.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.17 bool jpeg2000::FileManager::ReadUrlBox (const File & *file*, uint64_t *length_box*, string * *path_file*) [private]

Reads the information of a URL box.

Parameters

file	Image file.
length_box	Box length in bytes.
path_file	Receives the URL path read.

Returns

`true` if successful.

Here is the call graph for this function:

6.25.3.18 string jpeg2000::FileManager::root_dir ( ) const [inline]

Returns the root directory of the image repository.

Here is the caller graph for this function:

6.25.4 Member Data Documentation

6.25.4.1 string jpeg2000::FileManager::cache_dir_ [private]

Caching directory.

6.25.4.2 string jpeg2000::FileManager::root_dir_ [private]

Root directory of the repository.

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

- jpeg2000/file_manager.h
- jpeg2000/file_manager.cc

6.26 data::FileSegment Class Reference

Identifies a data segment of a file.

`#include <file_segment.h>`

Collaboration diagram for data::FileSegment:

Public Member Functions

- FileSegment ()
  Initializes all the member variables with zero, being a null segment.
- FileSegment (uint64_t offset, uint64_t length)
  Initializes the segment with the given parameters.
- FileSegment (const FileSegment &segment)
  Copy constructor.
- FileSegment & operator= (const FileSegment &segment)
  Copy assignment.
- FileSegment & RemoveFirst (int count)
  Removes the first bytes of the segment.
- FileSegment & RemoveLast (int count)
  Removes the last bytes of the segment.
- bool IsContiguousTo (const FileSegment &segment) const
  Returns `true` if the segment is contiguous to another given segment, so the first byte of the given segment is just the next byte after the last byte of the segment.
- bool operator== (const FileSegment &segment) const
- bool operator!= (const FileSegment &segment) const
- template<typename T >
  T & SerializeWith (T &stream)
- virtual ~FileSegment ()

Public Attributes

- uint64_t offset

  Offset of the data segment.
- uint64_t length

  Length of the data segment.

Static Public Attributes

- static const FileSegment Null

  Identifies a null segment, with the offset as well as the length set to zero.

Friends

- ostream & operator<< (ostream &out, const FileSegment &segment)

6.26.1 Detailed Description

Identifies a data segment of a file.

This segment is defined by an offset and a length (number of bytes), both of them with an unsigned integer of 64 bits. This class is serializable and can be printed.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 data::FileSegment::FileSegment ( ) [inline]

Initializes all the member variables with zero, being a null segment.

6.26.2.2 data::FileSegment::FileSegment (uint64_t *offset*, uint64_t *length*) [inline]

Initializes the segment with the given parameters.

Parameters

offset	Offset of the segment.
length	Length of the segment.

6.26.2.3 data::FileSegment::FileSegment (const FileSegment & *segment*) [inline]

Copy constructor.

6.26.2.4 virtual data::FileSegment::~FileSegment ( ) [inline], [virtual]

6.26.3 Member Function Documentation

6.26.3.1 bool data::FileSegment::IsContiguousTo (const FileSegment & *segment*) const [inline]

Returns `true` if the segment is contiguous to another given segment, so the first byte of the given segment is just the next byte after the last byte of the segment.

6.26.3.2 bool data::FileSegment::operator!= (const FileSegment & *segment*) const [inline]

6.26.3.3 FileSegment& data::FileSegment::operator= (const FileSegment & *segment*) [inline]

Copy assignment.

6.26.3.4 bool data::FileSegment::operator== (const FileSegment & *segment*) const [inline]

6.26.3.5 FileSegment& data::FileSegment::RemoveFirst (int *count*) [inline]

Removes the first bytes of the segment.

Modifies the segment as if a number of bytes (specified by the parameter) was removed from the beginning of the segment.

Parameters

count	Number of bytes to remove.

Returns

`*this`.

Here is the caller graph for this function:

6.26.3.6 FileSegment& data::FileSegment::RemoveLast (int *count*) [inline]

Removes the last bytes of the segment.

Modifies the segment as if a number of bytes (specified by the parameter) was removed from the end of the segment.

Parameters

count	Number of bytes to remove.

Returns

*this.

6.26.3.7 template<typename T > T& data::FileSegment::SerializeWith (T & *stream*) [inline]

6.26.4 Friends And Related Function Documentation

6.26.4.1 ostream& operator<< (ostream & *out,* const FileSegment & *segment*) [friend]

6.26.5 Member Data Documentation

6.26.5.1 uint64_t data::FileSegment::length

Length of the data segment.

6.26.5.2 const FileSegment data::FileSegment::Null [static]

Identifies a null segment, with the offset as well as the length set to zero.

6.26.5.3 uint64_t data::FileSegment::offset

Offset of the data segment.

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

- data/file_segment.h
- data/file_segment.cc

6.27 http::Header Class Reference

Class used to handle a HTTP header.

`#include <header.h>`

Inheritance diagram for http::Header:

Collaboration diagram for http::Header:

Public Types

- typedef HeaderBase< HeaderName::CONTENT_TYPE > ContentType
  Predefined "Content-Type".
- typedef HeaderBase< HeaderName::CACHE_CONTROL > CacheControl
  Predefined "Cache-Control" header.
- typedef HeaderBase< HeaderName::CONTENT_LENGTH > ContentLength
  Predefined "Content-Length" header.
- typedef HeaderBase< HeaderName::TRANSFER_ENCODING > TransferEncoding
  Predefined "Transfer-Encoding" header.

Public Member Functions

- Header ()

  Empty constructor.
- Header (const string &name, const string &value)

  Initializes the header content (name and value).

Friends

- template<const char * NAME>
  bool operator== (const Header &a, const HeaderBase< NAME > &b)

  Returns `true` if the names of the two headers are equal.

Additional Inherited Members

6.27.1 Detailed Description

Class used to handle a HTTP header.

See also

HeaderBase
HeaderName

6.27.2 Member Typedef Documentation

6.27.2.1 typedef HeaderBase<HeaderName::CACHE_CONTROL> http::Header::CacheControl

Predefined "Cache-Control" header.

6.27.2.2 typedef HeaderBase<HeaderName::CONTENT_LENGTH> http::Header::ContentLength

Predefined "Content-Length" header.

6.27.2.3 typedef HeaderBase<HeaderName::CONTENT_TYPE> http::Header::ContentType

Predefined "Content-Type".

6.27.2.4 typedef HeaderBase<HeaderName::TRANSFER_ENCODING> http::Header::TransferEncoding

Predefined "Transfer-Encoding" header.

6.27.3 Constructor & Destructor Documentation

6.27.3.1 http::Header::Header ( ) [inline]

Empty constructor.

6.27.3.2 http::Header::Header (const string & *name*, const string & *value*) [inline]

Initializes the header content (name and value).

Parameters

name	Header name.
value	Header value.

6.27.4 Friends And Related Function Documentation

6.27.4.1 template<const char * NAME> bool operator== (const Header & *a*, const HeaderBase< NAME > & *b*) [friend]

Returns `true` if the names of the two headers are equal.

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

- http/header.h

6.28 http::HeaderBase< NAME > Class Template Reference

Template class used to identify a HTTP header.

`#include <header.h>`

Collaboration diagram for http::HeaderBase< NAME >:

Public Member Functions

- HeaderBase ()
  Empty constructor.
- HeaderBase (const string &value)
  Initializes the header value.

Static Public Member Functions

- static const char * name ()
  Returns the name of the header, used in the specialization of the class.

Private Attributes

- string value
  String value of the header.

Friends

- ostream & operator<< (ostream &out, const HeaderBase &header)
- istream & operator>> (istream &in, HeaderBase &header)

6.28.1 Detailed Description

template<const char * NAME>
class http::HeaderBase< NAME >

Template class used to identify a HTTP header.

It is possible to use this class with standard streams. This class is specialized with the header name.

See also

Header

6.28.2 Constructor & Destructor Documentation

6.28.2.1 template<const char * NAME> http::HeaderBase< NAME >::HeaderBase ( ) [inline]

Empty constructor.

6.28.2.2 template<const char * NAME> http::HeaderBase< NAME >::HeaderBase (const string & *value*) [inline]

Initializes the header value.

6.28.3 Member Function Documentation

6.28.3.1 template<const char * NAME> static const char* http::HeaderBase< NAME >::name ( ) [inline], [static]

Returns the name of the header, used in the specialization of the class.

6.28.4 Friends And Related Function Documentation

6.28.4.1 template<const char * NAME> ostream& operator<< (ostream & *out,* const HeaderBase< NAME > & *header*) [friend]

6.28.4.2 template<const char * NAME> istream& operator>> (istream & *in,* HeaderBase< NAME > & *header*) [friend]

6.28.5 Member Data Documentation

6.28.5.1 template<const char * NAME> string http::HeaderBase< NAME >::value [private]

String value of the header.

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

- http/header.h

6.29 http::HeaderBase< HeaderName::UNDEFINED > Class Template Reference

Specialization of the HeaderBase template class with the HeaderName::UNDEFINED value.

`#include <header.h>`

Inheritance diagram for http::HeaderBase< HeaderName::UNDEFINED >:

Collaboration diagram for http::HeaderBase< HeaderName::UNDEFINED >:

Public Member Functions

- HeaderBase ()
  Empty constructor.
- HeaderBase (const string &name, const string &value)
  Initializes the header content (name and value).

Public Attributes

- string name
  Header name.
- string value
  Header value.

Friends

- ostream & operator<< (ostream &out, const HeaderBase &header)
- istream & operator>> (istream &in, HeaderBase &header)

6.29.1 Detailed Description

template<>
class http::HeaderBase< HeaderName::UNDEFINED >

Specialization of the HeaderBase template class with the HeaderName::UNDEFINED value.

In this case the header name is not fixed, handled by an internal variable. This class is used as base for the class Header.

See also

Header

6.29.2 Constructor & Destructor Documentation

6.29.2.1 http::HeaderBase< HeaderName::UNDEFINED >::HeaderBase ( ) [inline]

Empty constructor.

6.29.2.2 http::HeaderBase< HeaderName::UNDEFINED >::HeaderBase (const string & *name*, const string & *value*) [inline]

Initializes the header content (name and value).

Parameters

name	Header name.
value	Header value.

6.29.3 Friends And Related Function Documentation

6.29.3.1 ostream& operator<< (ostream & *out,* const HeaderBase< HeaderName::UNDEFINED > & *header*) [friend]

6.29.3.2 istream& operator>> (istream & *in,* HeaderBase< HeaderName::UNDEFINED > & *header*) [friend]

6.29.4 Member Data Documentation

6.29.4.1 string http::HeaderBase< HeaderName::UNDEFINED >::name

Header name.

6.29.4.2 string http::HeaderBase< HeaderName::UNDEFINED >::value

Header value.

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

- http/header.h

6.30 http::HeaderName Class Reference

Container for the strings associated to the most common HTTP headers, used for the specialization of the class `HeaderBase`.

`#include <header.h>`

Collaboration diagram for http::HeaderName:

Static Public Attributes

- static const char UNDEFINED [ ] = ""
  No header name defined.
- static const char CONTENT_TYPE [ ] = "Content-Type"
  The header `Content-Type`
- static const char CACHE_CONTROL [ ] = "Cache-Control"
  The header `Cache-Control`
- static const char CONTENT_LENGTH [ ] = "Content-Length"
  The header `Content-Length`
- static const char TRANSFER_ENCODING [ ] = "Transfer-Encoding"
  The header `Transfer-Encoding`

6.30.1 Detailed Description

Container for the strings associated to the most common HTTP headers, used for the specialization of the class `HeaderBase`.

See also

HeaderBase

6.30.2 Member Data Documentation

6.30.2.1 const char http::HeaderName::CACHE_CONTROL = "Cache-Control" [static]

The header `Cache-Control`

6.30.2.2 const char http::HeaderName::CONTENT_LENGTH = "Content-Length" [static]

The header `Content-Length`

6.30.2.3 const char http::HeaderName::CONTENT_TYPE = "Content-Type" [static]

The header `Content-Type`

6.30.2.4 const char http::HeaderName::TRANSFER_ENCODING = "Transfer-Encoding" [static]

The header `Transfer-Encoding`

6.30.2.5 const char http::HeaderName::UNDEFINED = "" [static]

No header name defined.

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

- http/header.h
- http/header.cc

6.31 jpeg2000::ImageIndex Class Reference

Contains the indexing information of a JPEG2000 image file that is managed by the index manager.

`#include <image_index.h>`

Collaboration diagram for jpeg2000::ImageIndex:

Public Types

- typedef list< ImageIndex >::iterator Ptr

  Pointer of an object of this class.

Public Member Functions

- ImageIndex (const ImageIndex &image_index)

  Copy constructor.
- int GetNumCodestreams () const

  Returns the number of codestreams.
- int GetNumMetadatas () const

  Returns the number of meta-data blocks.
- bool ReadLock (const Range &range=Range(0, 0))

  Gets the lock for reading, for a specific range of codestreams.
- bool ReadUnlock (const Range &range=Range(0, 0))

  Releases the lock for reading, for a specific range of codestreams.
- string GetPathName () const

  Returns the path name of the image.
- string GetPathName (int num_codestream) const

  Returns the path name of a given codestream, if it is a hyperlinked codestream.
- FileSegment GetMainHeader (int num_codestream) const

  Returns the file segment the main header of a given codestream.
- FileSegment GetMetadata (int num_metadata) const

  Returns the file segment of a meta-data block.
- PlaceHolder GetPlaceHolder (int num_placeholder) const

  Returns the information of a place-holder.
- FileSegment GetPacket (int num_codestream, const Packet &packet, int *offset=NULL)

  Returns the file segment of a packet.
- CodingParameters::Ptr GetCodingParameters () const

  Returns a pointer to the coding parameters.
- bool IsHyperLinked (int num_codestream) const

  Returns `true` *if the image contains hyperlinks.*
- Ptr GetHyperLink (int num_codestream) const

  Returns a pointer to a hyperlink.
- int GetNumHyperLinks () const

  Returns the number of hyperlinks.
- operator CodingParameters::Ptr () const
- ImageIndex & operator= (const ImageIndex &image_index)
- virtual ~ImageIndex ()

Private Member Functions

- bool GetPLTLength (const File &file, int ind_codestream, uint64_t *length_packet)

  Gets the packet lengths from a PLT marker.
- void GetOffsetPacket (const File &file, int ind_codestream, uint64_t length_packet)

  Gets the packet offsets.
- bool BuildIndex (int ind_codestream, int max_index)

  Builds the required index for the required resolution levels.
- bool Init (const string &path_name, const ImageInfo &image_info)

Initializes the object.

- bool Init (const string &path_name, CodingParameters::Ptr coding_parameters, const ImageInfo &image_↩ info, int index)

  Initializes the object.

- ImageIndex ()

  Empty constructor.

Private Attributes

- RdWrLock::Ptr rdwr_lock

  Read/write lock.

- vector< int > last_plt
- vector< int > last_packet
- vector< uint64_t > last_offset_PLT
- vector< uint64_t > last_offset_packet
- string path_name

  Image file name.

- Metadata meta_data

  Image Metadata.

- int num_references

  Number of references.

- vector< int > max_resolution

  Maximum resolution number.

- vector< PacketIndex > packet_indexes

  Code-stream packet index.

- vector< CodestreamIndex > codestreams

  Image code-streams.

- CodingParameters::Ptr coding_parameters

  Image coding parameters.

- vector< list< ImageIndex >::iterator > hyper_links

  Image hyperlinks.

Friends

- class IndexManager
- ostream & operator<< (ostream &out, const ImageIndex &info_node)

6.31.1 Detailed Description

Contains the indexing information of a JPEG2000 image file that is managed by the index manager.

This class can be printed.

Maintains a read/write lock for controlling the multi-thread access to the indexing information. For instance, by default all the threads usually want to read the information. The packet index built on demand, so only when a thread wants to create a new level of the packet index, it needs to write.

See also

IndexManager

6.31.2 Member Typedef Documentation

6.31.2.1 typedef list<ImageIndex>::iterator jpeg2000::ImageIndex::Ptr

Pointer of an object of this class.

6.31.3 Constructor & Destructor Documentation

6.31.3.1 jpeg2000::ImageIndex::ImageIndex ( ) [inline], [private]

Empty constructor.

Only the index manager can use this constructor.

6.31.3.2 jpeg2000::ImageIndex::ImageIndex (const ImageIndex & *image_index*) [inline]

Copy constructor.

6.31.3.3 virtual jpeg2000::ImageIndex::~ImageIndex ( ) [inline], [virtual]

6.31.4 Member Function Documentation

6.31.4.1 bool jpeg2000::ImageIndex::BuildIndex (int *ind_codestream,* int *max_index*) [private]

Builds the required index for the required resolution levels.

Parameters

ind_codestream	Codestream index.
max_index	Maximum resolution level.

Returns

`true` if successful

Here is the call graph for this function:

Here is the caller graph for this function:

6.31.4.2 CodingParameters::Ptr jpeg2000::ImageIndex::GetCodingParameters ( ) const [inline]

Returns a pointer to the coding parameters.

6.31.4.3 Ptr jpeg2000::ImageIndex::GetHyperLink (int *num_codestream*) const [inline]

Returns a pointer to a hyperlink.

Parameters

num_codestream	Number of the hyperlink (codestream).

6.31.4.4 FileSegment jpeg2000::ImageIndex::GetMainHeader (int *num_codestream*) const [inline]

Returns the file segment the main header of a given codestream.

Parameters

num_codestream	Codestream number

6.31.4.5 FileSegment jpeg2000::ImageIndex::GetMetadata (int *num_metadata*) const [inline]

Returns the file segment of a meta-data block.

Parameters

num_metadata	Meta-data number.

6.31.4.6 int jpeg2000::ImageIndex::GetNumCodestreams ( ) const [inline]

Returns the number of codestreams.

6.31.4.7 int jpeg2000::ImageIndex::GetNumHyperLinks ( ) const [inline]

Returns the number of hyperlinks.

6.31.4.8 int jpeg2000::ImageIndex::GetNumMetadatas ( ) const [inline]

Returns the number of meta-data blocks.

6.31.4.9 void jpeg2000::ImageIndex::GetOffsetPacket (const File & *file*, int *ind_codestream*, uint64_t *length_packet*) [private]

Gets the packet offsets.

Parameters

file	File where to read the data from.
ind_codestream	Codestream index.
length_packet	Packet length.

Returns

`true` if successful.

Here is the caller graph for this function:

6.31.4.10 FileSegment jpeg2000::ImageIndex::GetPacket (int *num_codestream*, const Packet & *packet*, int * *offset* = NULL)

Returns the file segment of a packet.

Parameters

num_codestream	Codestream number.
packet	Packet information.
offset	If it is not `NULL` receives the offset of the packet.

Here is the call graph for this function:

6.31.4.11 string jpeg2000::ImageIndex::GetPathName ( ) const [inline]

Returns the path name of the image.

6.31.4.12 string jpeg2000::ImageIndex::GetPathName (int *num_codestream*) const [inline]

Returns the path name of a given codestream, if it is a hyperlinked codestream.

Parameters

num_codestream	Codestream number.

6.31.4.13 PlaceHolder jpeg2000::ImageIndex::GetPlaceHolder (int *num_placeholder*) const [inline]

Returns the information of a place-holder.

Parameters

num_placeholder	Place-holder number.

6.31.4.14 bool jpeg2000::ImageIndex::GetPLTLength (const File & *file*, int *ind_codestream*, uint64_t * *length_packet*) [private]

Gets the packet lengths from a PLT marker.

Parameters

file	File where to read the data from.
ind_codestream	Codestream index.
length_packet	It is returned the length of the packet.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.31.4.15 bool jpeg2000::ImageIndex::Init (const string & *path_name,* const ImageInfo & *image_info*) [private]

Initializes the object.

Parameters

path_name	Path name of the image.
image_info	Indexing image information.

Returns

`true` if successful

Here is the call graph for this function:

Here is the caller graph for this function:

6.31.4.16 bool jpeg2000::ImageIndex::Init (const string & *path_name,* CodingParameters::Ptr *coding_parameters,* const ImageInfo & *image_info,* int *index*) [private]

Initializes the object.

Parameters

path_name	Path name of the image.
coding_parameters	Coding parameters.
image_info	Indexing image information.
index	Image index.

Returns

`true` if successful

6.31.4.17 bool jpeg2000::ImageIndex::IsHyperLinked (int *num_codestream*) const [inline]

Returns `true` if the image contains hyperlinks.

6.31.4.18 jpeg2000::ImageIndex::operator CodingParameters::Ptr ( ) const [inline]

6.31.4.19 ImageIndex& jpeg2000::ImageIndex::operator= (const ImageIndex & *image_index*) [inline]

Here is the call graph for this function:

6.31.4.20 bool jpeg2000::ImageIndex::ReadLock (const Range & *range* = Range (0, 0))

Gets the lock for reading, for a specific range of codestreams.

Returns

`true` if successful

6.31.4.21 bool jpeg2000::ImageIndex::ReadUnlock (const Range & *range* = Range (0, 0))

Releases the lock for reading, for a specific range of codestreams.

Returns

`true` if successful

6.31.5 Friends And Related Function Documentation

6.31.5.1 friend class IndexManager [friend]

6.31.5.2 ostream& operator<< (ostream & *out,* const ImageIndex & *info_node*) [friend]

6.31.6 Member Data Documentation

6.31.6.1 vector<CodestreamIndex> jpeg2000::ImageIndex::codestreams [private]

Image code-streams.

6.31.6.2 CodingParameters::Ptr jpeg2000::ImageIndex::coding_parameters [private]

Image coding parameters.

6.31.6.3 vector<list<ImageIndex>::iterator> jpeg2000::ImageIndex::hyper_links [private]

Image hyperlinks.

6.31.6.4 vector<uint64_t> jpeg2000::ImageIndex::last_offset_packet [private]

6.31.6.5 vector<uint64_t> jpeg2000::ImageIndex::last_offset_PLT [private]

6.31.6.6 vector<int> jpeg2000::ImageIndex::last_packet [private]

6.31.6.7 vector<int> jpeg2000::ImageIndex::last_plt [private]

6.31.6.8 vector<int> jpeg2000::ImageIndex::max_resolution [private]

Maximum resolution number.

6.31.6.9 Metadata jpeg2000::ImageIndex::meta_data [private]

Image Metadata.

6.31.6.10 int jpeg2000::ImageIndex::num_references [private]

Number of references.

6.31.6.11 vector<PacketIndex> jpeg2000::ImageIndex::packet_indexes [private]

Code-stream packet index.

6.31.6.12 string jpeg2000::ImageIndex::path_name [private]

Image file name.

6.31.6.13 RdWrLock::Ptr jpeg2000::ImageIndex::rdwr_lock [private]

Read/write lock.

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

- jpeg2000/image_index.h
- jpeg2000/image_index.cc

6.32 jpeg2000::ImageInfo Class Reference

Contains the indexing information of a JPEG2000 image.

`#include <image_info.h>`

Collaboration diagram for jpeg2000::ImageInfo:

Public Member Functions

- ImageInfo ()
  Empty constructor.
- ImageInfo (const ImageInfo &info)
  Copy constructor.
- const ImageInfo & operator= (const ImageInfo &info)
  Copy assignment.
- template<typename T >
  T & SerializeWith (T &stream)
- virtual ~ImageInfo ()

Public Attributes

- Metadata meta_data
  Meta-data information.
- multimap< string, int > paths
  Paths of the hyperlinks (if any)
- CodingParameters coding_parameters
  Coding parameters.
- vector< CodestreamIndex > codestreams
  Codestreams information.
- vector< Metadata > meta_data_hyperlinks
  Meta-data of the hyperlinks.

Friends

- ostream & operator<< (ostream &out, const ImageInfo &info)

6.32.1 Detailed Description

Contains the indexing information of a JPEG2000 image.

This class can be serialized and printed.

See also

CodingParameters
CodestreamIndex
Metadata

6.32.2 Constructor & Destructor Documentation

6.32.2.1 jpeg2000::ImageInfo::ImageInfo ( ) [inline]

Empty constructor.

6.32.2.2 jpeg2000::ImageInfo::ImageInfo (const ImageInfo & *info*) [inline]

Copy constructor.

6.32.2.3 virtual jpeg2000::ImageInfo::~ImageInfo ( ) [inline], [virtual]

6.32.3 Member Function Documentation

6.32.3.1 const ImageInfo& jpeg2000::ImageInfo::operator= (const ImageInfo & *info*) [inline]

Copy assignment.

Here is the call graph for this function:

6.32.3.2 template<typename T > T& jpeg2000::ImageInfo::SerializeWith (T & *stream*) [inline]

6.32.4 Friends And Related Function Documentation

6.32.4.1 ostream& operator<< (ostream & *out,* const ImageInfo & *info*) [friend]

6.32.5 Member Data Documentation

6.32.5.1 vector<CodestreamIndex> jpeg2000::ImageInfo::codestreams

Codestreams information.

6.32.5.2 CodingParameters jpeg2000::ImageInfo::coding_parameters

Coding parameters.

6.32.5.3 Metadata jpeg2000::ImageInfo::meta_data

Meta-data information.

6.32.5.4 vector<Metadata> jpeg2000::ImageInfo::meta_data_hyperlinks

Meta-data of the hyperlinks.

6.32.5.5 multimap<string, int> jpeg2000::ImageInfo::paths

Paths of the hyperlinks (if any)

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

- jpeg2000/image_info.h

6.33 jpeg2000::IndexManager Class Reference

Manages the indexing information of a repository fo images.

`#include <index_manager.h>`

Collaboration diagram for jpeg2000::IndexManager:

Public Member Functions

- IndexManager ()
  Empty constructor.
- bool Init (string root_dir, string cache_dir)
  Initializes the object.
- ImageIndex::Ptr GetBegin ()
  Returns a pointer to the first image index.
- ImageIndex::Ptr GetEnd ()
  Returns a pointer to the last image index.
- FileManager & file_manager ()
  Returns a reference to the base file manager.
- bool OpenImage (string &path_image_file, ImageIndex::Ptr *image_index)
  Opens an image and adds its index to the list.
- bool CloseImage (const ImageIndex::Ptr &image_index)
  Closes an image and removes its index from the list, only if it is not used by any other one.
- int GetSize () const
  Returns the size of the list.
- virtual ~IndexManager ()

Private Member Functions

- bool UnsafeOpenImage (string &path_image_file, ImageIndex::Ptr *image_index)
  Unsafely (without mutex) opens an image and adds its index to the list.
- bool UnsafeCloseImage (const ImageIndex::Ptr &image_index)
  Unsafely (without mutex) closes an image and removes its index from the list, only if it is not used by any other one.

Private Attributes

- Mutex mutex

  Mutex for the operations with the list.
- FileManager file_manager_

  File manager.
- list< ImageIndex > index_list

  List of the indexes.

6.33.1 Detailed Description

Manages the indexing information of a repository fo images.

Maintains a list in memory of the indexes (using the class `ImageIndex` for the nodes) of all the opened images and allows a multi-thread access to the information.

See also

FileManager
ImageIndex

6.33.2 Constructor & Destructor Documentation

6.33.2.1 jpeg2000::IndexManager::IndexManager ( ) [inline]

Empty constructor.

6.33.2.2 virtual jpeg2000::IndexManager::~IndexManager ( ) [inline], [virtual]

6.33.3 Member Function Documentation

6.33.3.1 bool jpeg2000::IndexManager::CloseImage (const ImageIndex::Ptr & *image_index*)

Closes an image and removes its index from the list, only if it is not used by any other one.

Parameters

image_index	Associated image index.

Returns

`true` if successful.

Here is the caller graph for this function:

6.33.3.2 FileManager& jpeg2000::IndexManager::file_manager ( ) [inline]

Returns a reference to the base file manager.

Here is the caller graph for this function:

6.33.3.3 ImageIndex::Ptr jpeg2000::IndexManager::GetBegin ( ) [inline]

Returns a pointer to the first image index.

Here is the caller graph for this function:

6.33.3.4 ImageIndex::Ptr jpeg2000::IndexManager::GetEnd ( ) [inline]

Returns a pointer to the last image index.

Here is the caller graph for this function:

6.33.3.5 int jpeg2000::IndexManager::GetSize ( ) const [inline]

Returns the size of the list.

Here is the caller graph for this function:

6.33.3.6 bool jpeg2000::IndexManager::Init (string *root_dir,* string *cache_dir*) [inline]

Initializes the object.

Parameters

root_dir	Root directory of the image repository.
cache_dir	Directory used for caching.

Returns

`true` if successful

Here is the call graph for this function:

Here is the caller graph for this function:

6.33.3.7 bool jpeg2000::IndexManager::OpenImage (string & *path_image_file,* ImageIndex::Ptr * *image_index*)

Opens an image and adds its index to the list.

Parameters

path_image_file	Path of the image file.
image_index	Receives the pointer to the image index created.

Returns

`true` if successful.

Here is the caller graph for this function:

6.33.3.8 bool jpeg2000::IndexManager::UnsafeCloseImage (const ImageIndex::Ptr & *image_index*) [private]

Unsafely (without mutex) closes an image and removes its index from the list, only if it is not used by any other one.

Parameters

image_index	Associated image index.

Returns

`true` if successful.

6.33.3.9 bool jpeg2000::IndexManager::UnsafeOpenImage (string & *path_image_file,* ImageIndex::Ptr * *image_index*) [private]

Unsafely (without mutex) opens an image and adds its index to the list.

Parameters

path_image_file	Path of the image file.
image_index	Receives the pointer to the image index created.

Returns

`true` if successful.

Here is the call graph for this function:

6.33.4 Member Data Documentation

6.33.4.1 FileManager jpeg2000::IndexManager::file_manager_ [private]

File manager.

6.33.4.2 list<ImageIndex> jpeg2000::IndexManager::index_list [private]

List of the indexes.

6.33.4.3 Mutex jpeg2000::IndexManager::mutex [private]

Mutex for the operations with the list.

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

- jpeg2000/index_manager.h
- jpeg2000/index_manager.cc

6.34 net::InetAddress Class Reference

Class to identify and handle an Internet address.

`#include <address.h>`

Inheritance diagram for net::InetAddress:

Collaboration diagram for net::InetAddress:

Public Member Functions

- InetAddress ()
  Initializes the address to zero.
- InetAddress (const InetAddress &address)
  Copy constructor.
- InetAddress (int port)
  Initializes the address with given port.
- InetAddress (const char *path, int port)
  Initializes the address with the given path and port.
- InetAddress & operator= (const InetAddress &address)
  Copy assignment.
- virtual sockaddr * GetSockAddr () const
  Overloaded from the base class to use the internal address structure.
- virtual int GetSize () const
  Overloaded from the base class to use the internal address structure.
- string GetPath () const
  Returns the address path.
- int GetPort () const
  Returns the port number.

Private Attributes

- sockaddr_in sock_addr
  Internal address structure.

6.34.1 Detailed Description

Class to identify and handle an Internet address.

The used internal address structure is `sockaddr_in`.

See also

Address

6.34.2 Constructor & Destructor Documentation

6.34.2.1 net::InetAddress::InetAddress ( ) [inline]

Initializes the address to zero.

6.34.2.2 net::InetAddress::InetAddress (const InetAddress & *address*) [inline]

Copy constructor.

6.34.2.3 net::InetAddress::InetAddress (int *port*) [inline]

Initializes the address with given port.

The used path is `INADDR_ANY`.

Parameters

port	Port number.

6.34.2.4 net::InetAddress::InetAddress (const char * *path*, int *port*) [inline]

Initializes the address with the given path and port.

Parameters

path	Address path.
port	Port number.

6.34.3 Member Function Documentation

6.34.3.1 string net::InetAddress::GetPath ( ) const [inline]

Returns the address path.

Here is the caller graph for this function:

6.34.3.2 int net::InetAddress::GetPort ( ) const [inline]

Returns the port number.

Here is the caller graph for this function:

6.34.3.3 virtual int net::InetAddress::GetSize ( ) const [inline], [virtual]

Overloaded from the base class to use the internal address structure.

Implements net::Address.

6.34.3.4 virtual sockaddr* net::InetAddress::GetSockAddr ( ) const [inline], [virtual]

Overloaded from the base class to use the internal address structure.

Implements net::Address.

6.34.3.5 InetAddress& net::InetAddress::operator= (const InetAddress & *address*) [inline]

Copy assignment.

6.34.4 Member Data Documentation

6.34.4.1 sockaddr_in net::InetAddress::sock_addr [private]

Internal address structure.

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

- net/address.h

6.35 data::InputOperator Struct Reference

This struct identifies a basic input operator to be applied to a `File` object.

`#include <serialize.h>`

Collaboration diagram for data::InputOperator:

Static Public Member Functions

- static const char * FileAccess ()

  Returns the required file access for this operator.
- static bool SerializeBytes (File &file, void *ptr, int num_bytes)

  Performs an input (read) serialization of bytes for a file.

6.35.1 Detailed Description

This struct identifies a basic input operator to be applied to a `File` object.

See also

BaseStream
File

6.35.2 Member Function Documentation

6.35.2.1 static const char* data::InputOperator::FileAccess ( ) [inline], [static]

Returns the required file access for this operator.

6.35.2.2 static bool data::InputOperator::SerializeBytes (File & *file*, void * *ptr*, int *num_bytes*) [inline], [static]

Performs an input (read) serialization of bytes for a file.

Parameters

file	File to use for the operation.
ptr	Pointer to the buffer where to store the bytes.
num_bytes	Number of bytes to read from the file.

Returns

`true` if successful.

Here is the call graph for this function:

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

- data/serialize.h

6.36 data::InputStream Class Reference

Specialization of the `BaseStream` for input serializations.

`#include <serialize.h>`

Inheritance diagram for data::InputStream:

Collaboration diagram for data::InputStream:

Public Member Functions

- template<typename T >
  InputStream & Serialize (T &var)

Additional Inherited Members

6.36.1 Detailed Description

Specialization of the `BaseStream` for input serializations.

See also

BaseStream

6.36.2 Member Function Documentation

6.36.2.1 template<typename T > InputStream& data::InputStream::Serialize (T & *var*) [inline]

Here is the call graph for this function:

Here is the caller graph for this function:

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

- data/serialize.h

6.37 ipc::IPCObject Class Reference

Class base of all the IPC classes that has the basic operations (`Init`, `Wait` and `Dispose`) to be overloaded.

`#include <ipc_object.h>`

Inheritance diagram for ipc::IPCObject:

Collaboration diagram for ipc::IPCObject:

Public Types

- typedef SHARED_PTR< IPCObject > Ptr

  Pointer to an IPC object.

Public Member Functions

- IPCObject ()

  Initializes the internal status to `false`.
- virtual bool Init ()

  Sets the internal status to `true`
- virtual WaitResult Wait (int time_out=-1)

  Performs a wait operation with the object to get it.
- bool IsValid ()

  Returns `true` if the object is valid, that is, the internal status value is `true`.
- virtual bool Dispose ()

  Release the resources associated to the IPC object and sets the internal status to `false`.
- virtual ~IPCObject ()

  The desctructor calls the method `Dispose`.

Private Attributes

- bool valid

  Internal status of the object.

6.37.1 Detailed Description

Class base of all the IPC classes that has the basic operations (`Init`, `Wait` and `Dispose`) to be overloaded.

It has also an internal boolean value to set the object status.

For the IPC objects the Windows IPC philosophy has been adopted because of its simplicity and flexibility. Under this philosophy, the main operation that can be performed to an IPC object is `Wait`, to wait for getting the control of the object. Depending on the type of the IPC object (mutex, event, etc.), the meaning of "getting" the control of the object can be different.

6.37.2 Member Typedef Documentation

6.37.2.1 typedef SHARED_PTR<IPCObject> ipc::IPCObject::Ptr

Pointer to an IPC object.

6.37.3 Constructor & Destructor Documentation

6.37.3.1 ipc::IPCObject::IPCObject ( ) [inline]

Initializes the internal status to `false`.

6.37.3.2 virtual ipc::IPCObject::~IPCObject ( ) [inline], [virtual]

The desctructor calls the method `Dispose`.

Here is the call graph for this function:

6.37.4 Member Function Documentation

6.37.4.1 virtual bool ipc::IPCObject::Dispose ( ) [inline], [virtual]

Release the resources associated to the IPC object and sets the internal status to `false`.

Returns

`true` if successful. If it is not overloaded, it always returns `true`.

Reimplemented in ipc::Event, ipc::Mutex, and ipc::RdWrLock.

Here is the caller graph for this function:

6.37.4.2 virtual bool ipc::IPCObject::Init ( ) [inline], [virtual]

Sets the internal status to `true`

Returns

`true` if successful. If it is not overloaded, it always returns `true` .

Reimplemented in ipc::Event, ipc::Mutex, and ipc::RdWrLock.

Here is the caller graph for this function:

6.37.4.3 bool ipc::IPCObject::IsValid ( ) [inline]

Returns `true` if the object is valid, that is, the internal status value is `true`.

Here is the caller graph for this function:

6.37.4.4 virtual WaitResult ipc::IPCObject::Wait (int *time_out* = −1) [inline], [virtual]

Performs a wait operation with the object to get it.

Parameters

time_out	Time out (infinite by default).

Returns

`WAIT_OBJECT` if successful, `WAIT_TIMEOUT` if time out or `WAIT_ERROR` is error. If it is not overloaded, it always returns `WAIT_ERROR`.

Reimplemented in ipc::Event, ipc::Mutex, and ipc::RdWrLock.

6.37.5 Member Data Documentation

6.37.5.1 bool ipc::IPCObject::valid [private]

Internal status of the object.

As it is a private member, the derived classes must use the methods `Init` and `Dispose` to set the value of this status.

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

- ipc/ipc_object.h

6.38 data::LockedAccess Struct Reference

Struct for wrapping the basic `FILE` locked functions for reading and writing defined in `stdio.h`.

`#include <file.h>`

Collaboration diagram for data::LockedAccess:

Static Public Member Functions

- static void configure (FILE *file_ptr)
- static size_t fwrite (const void *ptr, size_t size, size_t count, FILE *file_ptr)
- static size_t fread (void *ptr, size_t size, size_t count, FILE *file_ptr)
- static int fgetc (FILE *file_ptr)
- static int fputc (int c, FILE *file_ptr)

6.38.1 Detailed Description

Struct for wrapping the basic `FILE` locked functions for reading and writing defined in `stdio.h`.

See also

File

6.38.2 Member Function Documentation

6.38.2.1 static void data::LockedAccess::configure (FILE * *file_ptr*) [inline], [static]

6.38.2.2 static int data::LockedAccess::fgetc (FILE * *file_ptr*) [inline], [static]

6.38.2.3 static int data::LockedAccess::fputc (int *c*, FILE * *file_ptr*) [inline], [static]

6.38.2.4 static size_t data::LockedAccess::fread (void * *ptr*, size_t *size*, size_t *count*, FILE * *file_ptr*) [inline], [static]

6.38.2.5 static size_t data::LockedAccess::fwrite (const void * *ptr*, size_t *size*, size_t *count*, FILE * *file_ptr*) [inline], [static]

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

- data/file.h

6.39 jpeg2000::Metadata Class Reference

Contains the indexing information associated to the meta-data of a JPEG2000 image file.

`#include <meta_data.h>`

Collaboration diagram for jpeg2000::Metadata:

Public Member Functions

- Metadata ()
  Empty constructor.
- Metadata (const Metadata &info)
  Copy constructor.
- template<typename T >
  T & SerializeWith (T &stream)
- Metadata & operator= (const Metadata &info)
  Copy assignment.
- virtual ~Metadata ()

Public Attributes

- vector< FileSegment > meta_data
  File segments of all the meta-data blocks.
- vector< PlaceHolder > place_holders
  Associated place-holders.

Friends

- ostream & operator<< (ostream &out, const Metadata &info)

6.39.1 Detailed Description

Contains the indexing information associated to the meta-data of a JPEG2000 image file.

This class can be printed and serialized.

6.39.2 Constructor & Destructor Documentation

6.39.2.1 jpeg2000::Metadata::Metadata ( ) [inline]

Empty constructor.

6.39.2.2 jpeg2000::Metadata::Metadata (const Metadata & *info*) [inline]

Copy constructor.

6.39.2.3 virtual jpeg2000::Metadata::~Metadata ( ) [inline], [virtual]

6.39.3 Member Function Documentation

6.39.3.1 Metadata& jpeg2000::Metadata::operator= (const Metadata & *info*) [inline]

Copy assignment.

Here is the call graph for this function:

6.39.3.2 template<typename T > T& jpeg2000::Metadata::SerializeWith (T & *stream*) [inline]

6.39.4 Friends And Related Function Documentation

6.39.4.1 ostream& operator<< (ostream & *out,* const Metadata & *info*) [friend]

6.39.5 Member Data Documentation

6.39.5.1 vector<FileSegment> jpeg2000::Metadata::meta_data

File segments of all the meta-data blocks.

6.39.5.2 vector<PlaceHolder> jpeg2000::Metadata::place_holders

Associated place-holders.

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

- jpeg2000/meta_data.h

6.40 ipc::Mutex Class Reference

IPC object that offers the functionality of a mutex, implemented by means of the pthread mutex API.

`#include <mutex.h>`

Inheritance diagram for ipc::Mutex:

Collaboration diagram for ipc::Mutex:

Public Types

- typedef SHARED_PTR< Mutex > Ptr

  Pointer to a Mutex object.

Public Member Functions

- virtual bool Init ()

  Initializes the object without locking the mutex.
- bool Init (bool initial_owner)

  Initializes the object.
- virtual WaitResult Wait (int time_out=-1)

  Performs a wait operation with the object to get it.
- virtual bool Dispose ()

  Release the resources associated to the IPC object and sets the internal status to `false`*.*
- bool Release ()

  Releases/unlocks the mutex.

Private Attributes

- pthread_t locker

  Id. of the thread that locks the mutex.
- pthread_mutex_t mutex

  Mutex information.

6.40.1 Detailed Description

IPC object that offers the functionality of a mutex, implemented by means of the pthread mutex API.

See also

IPCObject

6.40.2 Member Typedef Documentation

6.40.2.1 typedef SHARED_PTR<Mutex> ipc::Mutex::Ptr

Pointer to a Mutex object.

6.40.3 Member Function Documentation

6.40.3.1 bool ipc::Mutex::Dispose ( ) [virtual]

Release the resources associated to the IPC object and sets the internal status to `false`.

Returns

`true` if successful.

Reimplemented from ipc::IPCObject.

Here is the call graph for this function:

Here is the caller graph for this function:

6.40.3.2 virtual bool ipc::Mutex::Init ( ) [inline], [virtual]

Initializes the object without locking the mutex.

Returns

`true` if successful.

Reimplemented from ipc::IPCObject.

Here is the call graph for this function:

Here is the caller graph for this function:

6.40.3.3 bool ipc::Mutex::Init (bool *initial_owner*)

Initializes the object.

Parameters

initial_owner	If `true` the mutex is locked.

Returns

`true` if successful.

Here is the call graph for this function:

6.40.3.4 bool ipc::Mutex::Release ( )

Releases/unlocks the mutex.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.40.3.5 WaitResult ipc::Mutex::Wait (int *time_out* = `-1`) [virtual]

Performs a wait operation with the object to get it.

Parameters

time_out	Time out (infinite by default).

Returns

`WAIT_OBJECT` if successful, `WAIT_TIMEOUT` if time out or `WAIT_ERROR` is error.

Reimplemented from ipc::IPCObject.

Here is the call graph for this function:

Here is the caller graph for this function:

6.40.4 Member Data Documentation

6.40.4.1 pthread_t ipc::Mutex::locker [private]

Id. of the thread that locks the mutex.

6.40.4.2 pthread_mutex_t ipc::Mutex::mutex [private]

Mutex information.

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

- ipc/mutex.h
- ipc/mutex.cc

6.41 data::OutputOperator Struct Reference

This struct identifies a basic output operator to be applied to a `File` object.

`#include <serialize.h>`

Collaboration diagram for data::OutputOperator:

Static Public Member Functions

- static const char * FileAccess ()
  Returns the required file access for this operator.
- static bool SerializeBytes (File &file, void *ptr, int num_bytes)
  Performs an output (write) serialization of bytes for a file.

6.41.1 Detailed Description

This struct identifies a basic output operator to be applied to a `File` object.

See also

BaseStream
File

6.41.2 Member Function Documentation

6.41.2.1 static const char* data::OutputOperator::FileAccess ( ) [inline], [static]

Returns the required file access for this operator.

6.41.2.2 static bool data::OutputOperator::SerializeBytes (File & *file,* void * *ptr,* int *num_bytes*) [inline], [static]

Performs an output (write) serialization of bytes for a file.

Parameters

file	File to use for the operation.
ptr	Pointer to the buffer where to read the bytes.
num_bytes	Number of bytes to write to the file.

Returns

`true` if successful.

Here is the call graph for this function:

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

- data/serialize.h

6.42 data::OutputStream Class Reference

Specialization of the `BaseStream` for output serializations.

`#include <serialize.h>`

Inheritance diagram for data::OutputStream:

Collaboration diagram for data::OutputStream:

Public Member Functions

- template<typename T >
  OutputStream & Serialize (T &var)

Additional Inherited Members

6.42.1 Detailed Description

Specialization of the `BaseStream` for output serializations.

See also

BaseStream

6.42.2 Member Function Documentation

6.42.2.1 template<typename T > OutputStream& data::OutputStream::Serialize (T & *var*) [inline]

Here is the call graph for this function:

Here is the caller graph for this function:

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

- data/serialize.h

6.43 jpeg2000::Packet Class Reference

Contains the information of a packet.

`#include <packet.h>`

Collaboration diagram for jpeg2000::Packet:

Public Member Functions

- Packet ()
  Initializes the object to zero.
- Packet (int layer, int resolution, int component, Point precinct_xy)
  Initializes the object.
- Packet (const Packet &packet)
  Copy constructor.
- const Packet & operator= (const Packet &packet)
  Copy assignment.
- virtual ~Packet ()

Public Attributes

- int layer
  Quality layer.
- int component
  Component number.
- int resolution
  Resolution level.
- Point precinct_xy
  Precinct coordinate.

Friends

- ostream & operator<< (ostream &out, const Packet &packet)

6.43.1 Detailed Description

Contains the information of a packet.

This class can be printed.

6.43.2 Constructor & Destructor Documentation

6.43.2.1 jpeg2000::Packet::Packet ( ) [inline]

Initializes the object to zero.

6.43.2.2 jpeg2000::Packet::Packet (int *layer,* int *resolution,* int *component,* Point *precinct_xy*) [inline]

Initializes the object.

6.43.2.3 jpeg2000::Packet::Packet (const Packet & *packet*) [inline]

Copy constructor.

6.43.2.4 virtual jpeg2000::Packet::~Packet ( ) [inline], [virtual]

6.43.3 Member Function Documentation

6.43.3.1 const Packet& jpeg2000::Packet::operator= (const Packet & *packet*) [inline]

Copy assignment.

6.43.4 Friends And Related Function Documentation

6.43.4.1 ostream& operator<< (ostream & *out,* const Packet & *packet*) [friend]

6.43.5 Member Data Documentation

6.43.5.1 int jpeg2000::Packet::component

Component number.

6.43.5.2 int jpeg2000::Packet::layer

Quality layer.

6.43.5.3 Point jpeg2000::Packet::precinct_xy

Precinct coordinate.

6.43.5.4 int jpeg2000::Packet::resolution

Resolution level.

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

- jpeg2000/packet.h

6.44 jpeg2000::PacketIndex Class Reference

Class used for indexing the packets of a codestream image.

`#include <packet_index.h>`

Collaboration diagram for jpeg2000::PacketIndex:

Public Types

- enum { MINIMUM_OFFSET = 64 }

Public Member Functions

- PacketIndex ()
  Empty constructor.
- PacketIndex (uint64_t max_offset)
  Initializes the object.
- PacketIndex (const PacketIndex &index)
  Copy constructor.
- const PacketIndex & operator= (const PacketIndex &index)
  Copy assignment.
- PacketIndex & Add (const FileSegment &segment)
  Adds a new packet segment to the index.
- int Size () const
  Returns the number of elements of the vector.
- void Clear ()
  Clears the content.
- FileSegment operator[ ] (int i) const
  Operator used for accessing the items.
- virtual ~PacketIndex ()

Private Attributes

- vint_vector offsets
  Vector of packet offsets.
- vector< FileSegment > aux
  Vector of file segments to handle the different sets of packets that are not contiguous.

6.44.1 Detailed Description

Class used for indexing the packets of a codestream image.

The class `vint_vector` is used internally to store the offsets of the packets with the minimum required bytes.

See also

data::vint_vector

6.44.2 Member Enumeration Documentation

6.44.2.1 anonymous enum

Enumerator

MINIMUM_OFFSET All the offsets must be greater than this value.

6.44.3 Constructor & Destructor Documentation

6.44.3.1 jpeg2000::PacketIndex::PacketIndex ( ) [inline]

Empty constructor.

6.44.3.2 jpeg2000::PacketIndex::PacketIndex (uint64_t *max_offset*) [inline]

Initializes the object.

Parameters

max_offset	Maximum value for an offset.

Here is the call graph for this function:

6.44.3.3 jpeg2000::PacketIndex::PacketIndex (const **PacketIndex** & *index*) [inline]

Copy constructor.

6.44.3.4 virtual jpeg2000::PacketIndex::~PacketIndex ( ) [inline], [virtual]

6.44.4 Member Function Documentation

6.44.4.1 **PacketIndex**& jpeg2000::PacketIndex::Add (const **FileSegment** & *segment*) [inline]

Adds a new packet segment to the index.

Parameters

segment	Fiel segment associated to the packet.

Returns

The object itself.

Here is the call graph for this function:

6.44.4.2 void jpeg2000::PacketIndex::Clear ( ) [inline]

Clears the content.

Here is the call graph for this function:

6.44.4.3 const PacketIndex& jpeg2000::PacketIndex::operator= (const PacketIndex & *index*) [inline]

Copy assignment.

6.44.4.4 FileSegment jpeg2000::PacketIndex::operator[ ] (int *i*) const [inline]

Operator used for accessing the items.

Parameters

i	Item index.

Returns

File segment of the packet.

6.44.4.5 int jpeg2000::PacketIndex::Size ( ) const [inline]

Returns the number of elements of the vector.

Here is the call graph for this function:

6.44.5 Member Data Documentation

6.44.5.1 vector<FileSegment> jpeg2000::PacketIndex::aux [private]

Vector of file segments to handle the different sets of packets that are not contiguous.

6.44.5.2 vint_vector jpeg2000::PacketIndex::offsets [private]

Vector of packet offsets.

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

- jpeg2000/packet_index.h

6.45 jpip::Request::ParametersMask Union Reference

Union used to control the presence of the different JPIP parameters in a request.

`#include <request.h>`

Collaboration diagram for jpip::Request::ParametersMask:

Public Member Functions

- ParametersMask ()

  Initializes the mask to zero.
- bool HasWOI () const

  Returns `true` if the mask contains the parameters associated to the WOI (fsiz, roff and rsiz).
- void Clear ()

  Sets the mask to zero.

Public Attributes

- struct {
  int fsiz: 1
  int roff: 1
  int rsiz: 1
  int metareq: 1
  int len: 1
  int target: 1
  int cid: 1
  int cnew: 1
  int cclose: 1
  int model: 1
  int stream: 1
  int context: 1
  } items

  Parameters mask.
- int value

  Parameters mask as integer.

6.45.1 Detailed Description

Union used to control the presence of the different JPIP parameters in a request.

6.45.2 Constructor & Destructor Documentation

6.45.2.1 jpip::Request::ParametersMask::ParametersMask ( ) [inline]

Initializes the mask to zero.

6.45.3 Member Function Documentation

6.45.3.1 void jpip::Request::ParametersMask::Clear ( ) [inline]

Sets the mask to zero.

Here is the caller graph for this function:

6.45.3.2 bool jpip::Request::ParametersMask::HasWOI ( ) const [inline]

Returns `true` if the mask contains the parameters associated to the WOI (fsiz, roff and rsiz).

Here is the caller graph for this function:

6.45.4 Member Data Documentation

6.45.4.1 int jpip::Request::ParametersMask::cclose

6.45.4.2 int jpip::Request::ParametersMask::cid

6.45.4.3 int jpip::Request::ParametersMask::cnew

6.45.4.4 int jpip::Request::ParametersMask::context

6.45.4.5 int jpip::Request::ParametersMask::fsiz

6.45.4.6 struct { ... } jpip::Request::ParametersMask::items

Parameters mask.

6.45.4.7 int jpip::Request::ParametersMask::len

6.45.4.8 int jpip::Request::ParametersMask::metareq

6.45.4.9 int jpip::Request::ParametersMask::model

6.45.4.10 int jpip::Request::ParametersMask::roff

6.45.4.11 int jpip::Request::ParametersMask::rsiz

6.45.4.12 int jpip::Request::ParametersMask::stream

6.45.4.13 int jpip::Request::ParametersMask::target

6.45.4.14 int jpip::Request::ParametersMask::value

Parameters mask as integer.

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

- jpip/request.h

6.46 jpeg2000::PlaceHolder Class Reference

Contains the information of a place-holder.

`#include <place_holder.h>`

Collaboration diagram for jpeg2000::PlaceHolder:

Public Member Functions

- PlaceHolder ()
  Initializes the object.
- PlaceHolder (int id, bool is_jp2c, const FileSegment &header, uint64_t data_length)
  Initializes the object.
- PlaceHolder (const PlaceHolder &place_holder)
  Copy constructor.
- template<typename T >
  T & SerializeWith (T &stream)
- PlaceHolder & operator= (const PlaceHolder &place_holder)
  Copy assignment.
- int length () const
  Returns the length of the place-holder.
- virtual ~PlaceHolder ()

Public Attributes

- int id
  Place-holder identifier.
- bool is_jp2c
  `true` *if refers to a codestream.*
- FileSegment header
  File segment associated to the box header.
- uint64_t data_length
  Length of the place-holder data.

Friends

- ostream & operator<< (ostream &out, const PlaceHolder &place_holder)

6.46.1 Detailed Description

Contains the information of a place-holder.

This class can be printed and serialized.

6.46.2 Constructor & Destructor Documentation

6.46.2.1 jpeg2000::PlaceHolder::PlaceHolder ( ) [inline]

Initializes the object.

6.46.2.2 jpeg2000::PlaceHolder::PlaceHolder (int *id,* bool *is_jp2c,* const FileSegment & *header,* uint64_t *data_length*) [inline]

Initializes the object.

Parameters

id	Place-holder identifier.
is_jp2c	Indicates if is a codestream place-holder.
header	File segment of the associated header.
data_length	Length of the place-holder data.

6.46.2.3 jpeg2000::PlaceHolder::PlaceHolder (const PlaceHolder & *place_holder*) [inline]

Copy constructor.

6.46.2.4 virtual jpeg2000::PlaceHolder::~PlaceHolder ( ) [inline], [virtual]

6.46.3 Member Function Documentation

6.46.3.1 int jpeg2000::PlaceHolder::length ( ) const [inline]

Returns the length of the place-holder.

Here is the caller graph for this function:

6.46.3.2 PlaceHolder& jpeg2000::PlaceHolder::operator= (const PlaceHolder & *place_holder*) [inline]

Copy assignment.

6.46.3.3 template<typename T > T& jpeg2000::PlaceHolder::SerializeWith (T & *stream*) [inline]

6.46.4 Friends And Related Function Documentation

6.46.4.1 ostream& operator<< (ostream & *out,* const PlaceHolder & *place_holder*) [friend]

6.46.5 Member Data Documentation

6.46.5.1 uint64_t jpeg2000::PlaceHolder::data_length

Length of the place-holder data.

6.46.5.2 FileSegment jpeg2000::PlaceHolder::header

File segment associated to the box header.

6.46.5.3 int jpeg2000::PlaceHolder::id

Place-holder identifier.

6.46.5.4 bool jpeg2000::PlaceHolder::is_jp2c

`true` if refers to a codestream.

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

- jpeg2000/place_holder.h

6.47 jpeg2000::Point Class Reference

Represents a couple of integer values that can be used to identify a coordinate as well as a size.

`#include <point.h>`

Collaboration diagram for jpeg2000::Point:

Public Member Functions

- Point ()
  Initializes the object.
- Point (int x, int y)
  Initializes the object.
- Point (const Point &p)
  Copy constructor.
- Point & operator= (const Point &p)
  Copy assignment.
- Point & operator++ ()
  Increments by one the two values.
- Point & operator-- ()
  Decrements by one the two values.
- Point & operator+= (int val)
  Increments the two values.
- Point & operator-= (int val)
  Decrements the two values.
- Point & operator*= (int val)
  Multiplies the two values by one value.
- Point & operator/= (int val)
  Divides the two values by one value.
- template<typename T >
  T & SerializeWith (T &stream)
- virtual ~Point ()

Public Attributes

- int x

  Value X.
- int y

  Value Y.

Friends

- Point operator+ (const Point &a, int value)

  Returns the sum of a point with an integer value.
- Point operator- (const Point &a, int value)

  Returns the subtraction of a point with an integer value.
- Point operator∗ (const Point &a, int value)

  Returns the multiplication of a point with an integer value.
- Point operator/ (const Point &a, int value)

  Returns the division of a point with an integer value.
- Point operator+ (const Point &a, const Point &b)

  Returns the sum of two points.
- Point operator- (const Point &a, const Point &b)

  Returns the subtraction of two points.
- Point operator∗ (const Point &a, const Point &b)

  Returns the multiplication of two points.
- Point operator/ (const Point &a, const Point &b)

  Returns the division of two points.
- bool operator== (const Point &a, const Point &b)

  Returns `true` if the two points are equal.
- bool operator!= (const Point &a, const Point &b)

  Returns `true` if the two points are not equal.
- ostream & operator<< (ostream &out, const Point &point)

6.47.1 Detailed Description

Represents a couple of integer values that can be used to identify a coordinate as well as a size.

This class can be printed and serialized.

6.47.2 Constructor & Destructor Documentation

6.47.2.1 jpeg2000::Point::Point ( ) [inline]

Initializes the object.

6.47.2.2 jpeg2000::Point::Point (int *x*, int *y*) [inline]

Initializes the object.

Parameters

x	Value X.
y	Value Y.

6.47.2.3 jpeg2000::Point::Point (const Point & *p*) [inline]

Copy constructor.

6.47.2.4 virtual jpeg2000::Point::~Point ( ) [inline], [virtual]

6.47.3 Member Function Documentation

6.47.3.1 Point& jpeg2000::Point::operator*= (int *val*) [inline]

Multiplies the two values by one value.

Parameters

val	Value to multiply.

Returns

The object itself.

6.47.3.2 Point& jpeg2000::Point::operator++ ( ) [inline]

Increments by one the two values.

Returns

The object itself.

6.47.3.3 Point& jpeg2000::Point::operator+= (int *val*) [inline]

Increments the two values.

Parameters

val	Value to increment.

Returns

The object itself.

6.47.3.4 Point& jpeg2000::Point::operator-- ( ) [inline]

Decrements by one the two values.

Returns

The object itself.

6.47.3.5 Point& jpeg2000::Point::operator-= (int *val*) [inline]

Decrements the two values.

Parameters

val	Value to decrement.

Returns

The object itself.

6.47.3.6 Point& jpeg2000::Point::operator/= (int *val*) [inline]

Divides the two values by one value.

Parameters

val	Value to divide.

Returns

The object itself.

6.47.3.7 Point& jpeg2000::Point::operator= (const Point & *p*) [inline]

Copy assignment.

6.47.3.8 template<typename T > T& jpeg2000::Point::SerializeWith (T & *stream*) [inline]

6.47.4 Friends And Related Function Documentation

6.47.4.1 bool operator!= (const Point & *a,* const Point & *b*) [friend]

Returns `true` if the two points are not equal.

6.47.4.2 Point operator∗ (const Point & *a,* int *value*) [friend]

Returns the multiplication of a point with an integer value.

The value is multiplied to the two values of the point.

6.47.4.3 Point operator∗ (const Point & *a,* const Point & *b*) [friend]

Returns the multiplication of two points.

The operation is applied each value of each point.

6.47.4.4 Point operator+ (const Point & *a,* int *value*) [friend]

Returns the sum of a point with an integer value.

The value is added to the two values of the point.

6.47.4.5 Point operator+ (const Point & *a,* const Point & *b*) [friend]

Returns the sum of two points.

The operation is applied each value of each point.

6.47.4.6 Point operator- (const Point & *a,* int *value*) [friend]

Returns the subtraction of a point with an integer value.

The value is subtracted from the two values of the point.

6.47.4.7 Point operator- (const Point & *a,* const Point & *b*) [friend]

Returns the subtraction of two points.

The operation is applied each value of each point.

6.47.4.8 Point operator/ (const Point & *a,* int *value*) [friend]

Returns the division of a point with an integer value.

The value is divided to the two values of the point.

6.47.4.9 Point operator/ (const Point & *a,* const Point & *b*) [friend]

Returns the division of two points.

The operation is applied each value of each point.

6.47.4.10 ostream& operator<< (ostream & *out,* const Point & *point*) [friend]

6.47.4.11 bool operator== (const Point & *a,* const Point & *b*) [friend]

Returns `true` if the two points are equal.

6.47.5 Member Data Documentation

6.47.5.1 int jpeg2000::Point::x

Value X.

6.47.5.2 int jpeg2000::Point::y

Value Y.

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

- jpeg2000/point.h

6.48 net::PollFD Struct Reference

Wrapper structure for the structure `pollfd` used by the kernel `poll` functions.

`#include <poll_table.h>`

Inheritance diagram for net::PollFD:

Collaboration diagram for net::PollFD:

Public Member Functions

- PollFD (int vfd, int mask)
  Initializes the structure.
- bool operator== (int n)
  Returns `true` if the file descriptor is the same as the given value.

6.48.1 Detailed Description

Wrapper structure for the structure `pollfd` used by the kernel `poll` functions.

See also

PollTable

6.48.2 Constructor & Destructor Documentation

6.48.2.1 net::PollFD::PollFD (int *vfd,* int *mask*) [inline]

Initializes the structure.

Parameters

vfd	File descriptor.
mask	Poll mask.

6.48.3 Member Function Documentation

6.48.3.1 bool net::PollFD::operator== (int *n*) [inline]

Returns `true` if the file descriptor is the same as the given value.

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

- net/poll_table.h

6.49 net::PollTable Class Reference

This class allows to perfom polls easily over a vector of descriptors.

`#include <poll_table.h>`

Collaboration diagram for net::PollTable:

Public Member Functions

- PollTable ()
- void Add (int fd, int mask)

  Adds a new file descriptor and mask to the vector.
- int Poll (int timeout=-1)

  Peforms a poll over all the descriptors using the associated masks.
- int GetSize () const

  Returns the size of the internal vector.
- void Remove (int fd)

  Removes an item of the internal vector giving its file descriptor.
- void RemoveAt (int n)

  Remove an item of the internal vector giving its index position.
- PollFD & operator[ ] (int n)

  Indexing operator.
- virtual ~PollTable ()

Private Attributes

- vector< PollFD > fds

  Vector with the file descriptors and masks for polling.

6.49.1 Detailed Description

This class allows to perfom polls easily over a vector of descriptors.

It uses an internal STL vector of `PollFD` objects to handle dinamically the file descriptors and masks.

See also

PollFD

6.49.2 Constructor & Destructor Documentation

6.49.2.1 net::PollTable::PollTable ( ) [inline]

6.49.2.2 virtual net::PollTable::~PollTable ( ) [inline], [virtual]

6.49.3 Member Function Documentation

6.49.3.1 void net::PollTable::Add (int *fd*, int *mask*) [inline]

Adds a new file descriptor and mask to the vector.

Parameters

fd	File descriptor.
mask	Polling mask.

Here is the caller graph for this function:

6.49.3.2 int net::PollTable::GetSize ( ) const [inline]

Returns the size of the internal vector.

Here is the caller graph for this function:

6.49.3.3 PollFD& net::PollTable::operator[ ] (int *n*) [inline]

Indexing operator.

6.49.3.4 int net::PollTable::Poll (int *timeout* = -1) [inline]

Peforms a poll over all the descriptors using the associated masks.

Parameters

timeout	Time out of the poll (infinite by default).

Returns

The value given by the kernel function `poll`.

Here is the caller graph for this function:

6.49.3.5 void net::PollTable::Remove (int *fd*) [inline]

Removes an item of the internal vector giving its file descriptor.

Parameters

fd	File descriptor to remove.

Here is the caller graph for this function:

6.49.3.6 void net::PollTable::RemoveAt (int *n*) [inline]

Remove an item of the internal vector giving its index position.

Parameters

n	Position of the item to remove.

Here is the caller graph for this function:

6.49.4 Member Data Documentation

6.49.4.1 vector<PollFD> net::PollTable::fds [private]

Vector with the file descriptors and masks for polling.

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

- net/poll_table.h

6.50 http::Protocol Class Reference

Class used to identify the HTTP protocol.

`#include <protocol.h>`

Collaboration diagram for http::Protocol:

Public Member Functions

- Protocol (int mayorVersion=1, int minorVersion=1)

  Initialized the protocl with the given version.
- Protocol (const Protocol &protocol)

  Copy constructor.
- int mayorVersion () const

  Returns the mayor number of the protocol version.
- int minorVersion () const

  Returns the minor number of the protocol version.

Static Public Attributes

- static const char CRLF [ ] = "\r\n"

  String with the characters 13 (CR) and 10 (LF), the line separator used in the HTTP protocol.

Private Attributes

- int mayorVersion_

  Mayor protocol version.
- int minorVersion_

  Minor protocol version.

Friends

- ostream & operator<< (ostream &out, const Protocol &protocol)
- istream & operator>> (istream &in, Protocol &protocol)

6.50.1 Detailed Description

Class used to identify the HTTP protocol.

It is possible to use this class with standard streams.

6.50.2 Constructor & Destructor Documentation

6.50.2.1 http::Protocol::Protocol (int *mayorVersion* = `1`, int *minorVersion* = `1`) `[inline]`

Initialized the protocl with the given version.

By default the version is 1.1.

Parameters

mayorVersion	Mayor protocol version
minorVersion	Minor protocol version

6.50.2.2 http::Protocol::Protocol (const Protocol & *protocol*) [inline]

Copy constructor.

6.50.3 Member Function Documentation

6.50.3.1 int http::Protocol::mayorVersion ( ) const [inline]

Returns the mayor number of the protocol version.

6.50.3.2 int http::Protocol::minorVersion ( ) const [inline]

Returns the minor number of the protocol version.

6.50.4 Friends And Related Function Documentation

6.50.4.1 ostream& operator<< (ostream & *out,* const Protocol & *protocol*) [friend]

6.50.4.2 istream& operator>> (istream & *in,* Protocol & *protocol*) [friend]

6.50.5 Member Data Documentation

6.50.5.1 const char http::Protocol::CRLF = "\r\n" [static]

String with the characters 13 (CR) and 10 (LF), the line separator used in the HTTP protocol.

6.50.5.2 int http::Protocol::mayorVersion_ [private]

Mayor protocol version.

6.50.5.3 int http::Protocol::minorVersion_ [private]

Minor protocol version.

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

- http/protocol.h
- http/protocol.cc

6.51 jpeg2000::Range Class Reference

Represents a range of integer values, defined by two values, first and last, which are assumed to be included in the range.

`#include <range.h>`

Collaboration diagram for jpeg2000::Range:

Public Member Functions

- Range ()
  Initializes the object.
- Range (int first, int last)
  Initializes the object.
- Range (const Range &range)
  Copy constructor.
- Range & operator= (const Range &range)
  Copy assignment.
- bool IsValid () const
  Returns `true` if the first value if greater or equal to zero, and it is less or equal to the last value.
- int GetItem (int i) const
  Returns an item of the range, starting at the first value.
- int GetIndex (int item) const
  Returns the index of an item of the range.
- int Length () const
  Returns the length of the range (last - first + 1).
- virtual ~Range ()

Public Attributes

- int first
  First value of the range.
- int last
  Last value of the range.

Friends

- bool operator== (const Range &a, const Range &b)
- bool operator!= (const Range &a, const Range &b)
- ostream & operator<< (ostream &out, const Range &range)

6.51.1 Detailed Description

Represents a range of integer values, defined by two values, first and last, which are assumed to be included in the range.

Some basic operations are defined for easing the work with ranges.

6.51.2 Constructor & Destructor Documentation

6.51.2.1 jpeg2000::Range::Range ( ) [inline]

Initializes the object.

6.51.2.2 jpeg2000::Range::Range (int *first,* int *last*) [inline]

Initializes the object.

Parameters

first	First value.
last	Last value.

6.51.2.3 jpeg2000::Range::Range (const Range & *range*) [inline]

Copy constructor.

6.51.2.4 virtual jpeg2000::Range::~Range ( ) [inline], [virtual]

6.51.3 Member Function Documentation

6.51.3.1 int jpeg2000::Range::GetIndex (int *item*) const [inline]

Returns the index of an item of the range.

Parameters

item	Item of the range.

Returns

item - first.

Here is the caller graph for this function:

6.51.3.2 int jpeg2000::Range::GetItem (int *i*) const [inline]

Returns an item of the range, starting at the first value.

Parameters

i	Item index.

Returns

first + i.

Here is the caller graph for this function:

6.51.3.3 bool jpeg2000::Range::IsValid ( ) const [inline]

Returns `true` if the first value if greater or equal to zero, and it is less or equal to the last value.

6.51.3.4 int jpeg2000::Range::Length ( ) const [inline]

Returns the length of the range (last - first + 1).

Here is the caller graph for this function:

6.51.3.5 Range& jpeg2000::Range::operator= (const Range & *range*) [inline]

Copy assignment.

6.51.4 Friends And Related Function Documentation

6.51.4.1 bool operator!= (const Range & *a,* const Range & *b*) [friend]

6.51.4.2 ostream& operator<< (ostream & *out,* const Range & *range*) [friend]

6.51.4.3 bool operator== (const Range & *a,* const Range & *b*) [friend]

6.51.5 Member Data Documentation

6.51.5.1 int jpeg2000::Range::first

First value of the range.

6.51.5.2 int jpeg2000::Range::last

Last value of the range.

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

- jpeg2000/range.h

6.52 ipc::RdWrLock Class Reference

IPC object that offers the functionality of a read/write lock, implemented by means of the pthread rwlock API.

`#include <rdwr_lock.h>`

Inheritance diagram for ipc::RdWrLock:

Collaboration diagram for ipc::RdWrLock:

Public Types

- typedef SHARED_PTR< RdWrLock > Ptr
  Pointer to a RdWrLock object.

Public Member Functions

- virtual bool Init ()
  Initializes the object.
- virtual WaitResult Wait (int time_out=-1)
  Performs a wait operation with the object to get it for reading.
- WaitResult WaitForWriting (int time_out=-1)
  Performs a wait operation with the object to get it for writing.
- virtual bool Dispose ()
  Release the resources associated to the IPC object and sets the internal status to `false`.
- bool Release ()
  Releases the lock.

Private Attributes

- pthread_rwlock_t rwlock
  Read/write lock information.

6.52.1 Detailed Description

IPC object that offers the functionality of a read/write lock, implemented by means of the pthread rwlock API.

See also

IPCObject

6.52.2 Member Typedef Documentation

6.52.2.1 typedef SHARED_PTR<RdWrLock> ipc::RdWrLock::Ptr

Pointer to a RdWrLock object.

6.52.3 Member Function Documentation

6.52.3.1 bool ipc::RdWrLock::Dispose ( ) [virtual]

Release the resources associated to the IPC object and sets the internal status to `false`.

Returns

`true` if successful.

Reimplemented from ipc::IPCObject.

6.52.3.2 bool ipc::RdWrLock::Init ( ) [virtual]

Initializes the object.

Returns

`true` if successful.

Reimplemented from ipc::IPCObject.

6.52.3.3 bool ipc::RdWrLock::Release ( )

Releases the lock.

Returns

`true` if successful.

6.52.3.4 WaitResult ipc::RdWrLock::Wait (int *time_out* = `-1`) [virtual]

Performs a wait operation with the object to get it for reading.

Parameters

time_out	Time out (infinite by default).

Returns

`WAIT_OBJECT` if successful, `WAIT_TIMEOUT` if time out or `WAIT_ERROR` is error.

Reimplemented from ipc::IPCObject.

6.52.3.5 WaitResult ipc::RdWrLock::WaitForWriting (int *time_out* = `-1`)

Performs a wait operation with the object to get it for writing.

Parameters

time_out	Time out (infinite by default).

Returns

`WAIT_OBJECT` if successful, `WAIT_TIMEOUT` if time out or `WAIT_ERROR` is error.

6.52.4 Member Data Documentation

6.52.4.1 pthread_rwlock_t ipc::RdWrLock::rwlock [private]

Read/write lock information.

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

- ipc/rdwr_lock.h
- ipc/rdwr_lock.cc

6.53 jpip::Request Class Reference

Class derived from the HTTP `Request` class that contains the required code for properly analyzing a JPIP request, when this protocol is used over the HTTP.

```
#include <request.h>
```

Inheritance diagram for jpip::Request:

Collaboration diagram for jpip::Request:

Classes

- union ParametersMask

  Union used to control the presence of the different JPIP parameters in a request.

Public Types

- enum RoundDirection { ROUNDUP, ROUNDDOWN, CLOSEST }

  Enumeration of the possible round directions of a WOI for specifying the resolution levels.

Public Member Functions

- istream & ParseModel (istream &stream)

  Parses a cache model from an input stream.
- istream & GetCodedChar (istream &in, char &c)

  Gets a coded char from an input stream.
- virtual void ParseParameters (istream &stream)

  Parses the parameters of a CGI HTTP request.
- virtual void ParseParameter (istream &stream, const string &param, string &value)

  Parses one parameter of a CGI HTTP request.
- Request ()

  Empty constructor.
- void GetResolution (const CodingParameters::Ptr &coding_parameters, WOI *woi) const

  Obtains the resolution level and modifies the given WOI to adjust it according to that level.
- virtual ~Request ()

Public Attributes

- Size woi_size

  WOI size.
- Point woi_position

  WOI position.
- int min_codestream

  Minimum codestream.
- int max_codestream

  Maximum codestream.
- int length_response

  Maximum response length.
- ParametersMask mask

  Parameters mask.
- Size resolution_size

  Size of the resolution level.
- CacheModel cache_model

  Cache model.
- RoundDirection round_direction

  Round direction.

6.53.1 Detailed Description

Class derived from the HTTP `Request` class that contains the required code for properly analyzing a JPIP request, when this protocol is used over the HTTP.

See also

`http::Request`
CacheModel

6.53.2 Member Enumeration Documentation

6.53.2.1 enum jpip::Request::RoundDirection

Enumeration of the possible round directions of a WOI for specifying the resolution levels.

Enumerator

ROUNDUP Round-up.

ROUNDDOWN Round-down.

CLOSEST Closest.

6.53.3 Constructor & Destructor Documentation

6.53.3.1 jpip::Request::Request ( ) [inline]

Empty constructor.

6.53.3.2 virtual jpip::Request::~Request ( ) [inline], [virtual]

6.53.4 Member Function Documentation

6.53.4.1 istream & jpip::Request::GetCodedChar (istream & *in*, char & *c*)

Gets a coded char from an input stream.

Parameters

in	Input stream.
c	Reference to store the char.

Returns

The same input stream.

Here is the caller graph for this function:

6.53.4.2 void jpip::Request::GetResolution (const CodingParameters::Ptr & *coding_parameters*, WOI * *woi*) const [inline]

Obtains the resolution level and modifies the given WOI to adjust it according to that level.

Parameters

coding_parameters	Associated coding parameters.
woi	WOI to modify.

Here is the caller graph for this function:

6.53.4.3 istream & jpip::Request::ParseModel (istream & *stream*)

Parses a cache model from an input stream.

Parameters

stream	Input stream.

Returns

The same input stream after the parsing.

Here is the call graph for this function:

Here is the caller graph for this function:

6.53.4.4 void jpip::Request::ParseParameter (istream & *stream,* const string & *param,* string & *value*) [virtual]

Parses one parameter of a CGI HTTP request.

Parameters

stream	Input stream.
param	String to store the parameter name.
value	String to store the parameter value.

Reimplemented from http::Request.

Here is the call graph for this function:

6.53.4.5 void jpip::Request::ParseParameters (istream & *stream*) [virtual]

Parses the parameters of a CGI HTTP request.

Parameters

stream	Input stream.

Reimplemented from http::Request.

Here is the call graph for this function:

Here is the caller graph for this function:

6.53.5 Member Data Documentation

6.53.5.1 CacheModel jpip::Request::cache_model

Cache model.

6.53.5.2 int jpip::Request::length_response

Maximum response length.

6.53.5.3 ParametersMask jpip::Request::mask

Parameters mask.

6.53.5.4 int jpip::Request::max_codestream

Maximum codestream.

6.53.5.5 int jpip::Request::min_codestream

Minimum codestream.

6.53.5.6 Size jpip::Request::resolution_size

Size of the resolution level.

6.53.5.7 RoundDirection jpip::Request::round_direction

Round direction.

6.53.5.8 Point jpip::Request::woi_position

WOI position.

6.53.5.9 Size jpip::Request::woi_size

WOI size.

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

- jpip/request.h
- jpip/request.cc

6.54 http::Request Class Reference

Class used to identify a HTTP request (GET or POST).

`#include <request.h>`

Inheritance diagram for http::Request:

Collaboration diagram for http::Request:

Public Types

- enum Type { GET, POST, UNKNOWN }

  Request type enumeration.

Public Member Functions

- Request (Type type=Request::GET, const string &uri="/", const Protocol &protocol=Protocol(1, 1))

  Initializes the request.
- bool Parse (const string &line)

  Parses a request from a string.
- void ParseURI (const string &uri)

  Parses a URI from a string.
- virtual void ParseParameters (istream &stream)

  Parses the parameters from an input stream.
- virtual void ParseParameter (istream &stream, const string &param, string &value)

  Parses one parameter from an input stream.

Public Attributes

- Type type

  Request type (GET or POST)
- string object

  Object associated to the request.
- Protocol protocol

  Protocol version used.
- map< string, string > parameters

  Map with all the parameters when using the CGI form.

Friends

- istream & operator>> (istream &in, Request &request)
- ostream & operator<< (ostream &out, const Request &request)

6.54.1 Detailed Description

Class used to identify a HTTP request (GET or POST).

It is possible to use this class with standard streams.

See also

Response

6.54.2 Member Enumeration Documentation

6.54.2.1 enum http::Request::Type

Request type enumeration.

Enumerator

GET

POST

UNKNOWN

6.54.3 Constructor & Destructor Documentation

6.54.3.1 http::Request::Request (Type *type* = Request::GET, const string & *uri* = "/", const Protocol & *protocol* = Protocol(1, 1)) [inline]

Initializes the request.

Parameters

type	Request type (GET by default).
uri	Request URI ("/" by default).
protocol	Protocol version (1.1 by default).

6.54.4 Member Function Documentation

6.54.4.1 bool http::Request::Parse (const string & *line*)

Parses a request from a string.

Parameters

line	String that contains the request to parse.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.54.4.2 void http::Request::ParseParameter (istream & *stream*, const string & *param*, string & *value*) [virtual]

Parses one parameter from an input stream.

Parameters

stream	Input stream.
param	Parameter name.
value	Parameter value.

Reimplemented in jpip::Request.

Here is the caller graph for this function:

6.54.4.3 void http::Request::ParseParameters (istream & *stream*) [virtual]

Parses the parameters from an input stream.

Parameters

stream	Input stream.

Reimplemented in jpip::Request.

Here is the call graph for this function:

Here is the caller graph for this function:

6.54.4.4 void http::Request::ParseURI (const string & *uri*) [inline]

Parses a URI from a string.

Parameters

uri	String that contains the URI to parse.

Here is the call graph for this function:

Here is the caller graph for this function:

6.54.5 Friends And Related Function Documentation

6.54.5.1 ostream& operator<< (ostream & *out,* const Request & *request*) [friend]

6.54.5.2 istream& operator>> (istream & *in,* Request & *request*) [friend]

6.54.6 Member Data Documentation

6.54.6.1 string http::Request::object

Object associated to the request.

6.54.6.2 map<string, string> http::Request::parameters

Map with all the parameters when using the CGI form.

6.54.6.3 Protocol http::Request::protocol

Protocol version used.

6.54.6.4 Type http::Request::type

Request type (GET or POST)

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

- http/request.h
- http/request.cc

6.55 http::Response Class Reference

Class used to identify a HTTP response.

`#include <response.h>`

Collaboration diagram for http::Response:

Classes

- class StatusCodesInitializer

  Class used for the initializer.

Public Member Functions

- Response (int code=200, const Protocol &protocol=Protocol(1, 1))

  Initializes the response.

Public Attributes

- int code

  Status code.
- Protocol protocol

  Protocol version.

Static Public Attributes

- static map< int, string > StatusCodes

  Map with the strings associated to the most commonly used status codes.

Static Private Attributes

- static StatusCodesInitializer statusCodesInitializer

  The initializer of the `StatusCodes` member.

Friends

- ostream & operator<< (ostream &out, const Response &response)
- istream & operator>> (istream &in, Response &response)

6.55.1 Detailed Description

Class used to identify a HTTP response.

It is possible to use this class with standard streams.

See also

Request

6.55.2 Constructor & Destructor Documentation

6.55.2.1 http::Response::Response (int *code* = `200`, const Protocol & *protocol* = Protocol `(1, 1)`) `[inline]`

Initializes the response.

Parameters

code	Status code (200 by default).
protocol	Protocol version (1.1 by default).

6.55.3 Friends And Related Function Documentation

6.55.3.1 ostream& operator<< (ostream & *out,* const Response & *response*) [friend]

6.55.3.2 istream& operator>> (istream & *in,* Response & *response*) [friend]

6.55.4 Member Data Documentation

6.55.4.1 int http::Response::code

Status code.

6.55.4.2 Protocol http::Response::protocol

Protocol version.

6.55.4.3 map< int, string > http::Response::StatusCodes [static]

Map with the strings associated to the most commonly used status codes.

In order to use a new user defined status code, it is necessary to include in this map the associated string.

6.55.4.4 Response::StatusCodesInitializer http::Response::statusCodesInitializer [static], [private]

The initializer of the `StatusCodes` member.

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

- http/response.h
- http/response.cc

6.56 data::Serializer< T > Struct Template Reference

This template class allows to define a "serializer".

`#include <serialize.h>`

Collaboration diagram for data::Serializer< T >:

Static Public Member Functions

- static InputStream & Load (InputStream &stream, T &var)
- static OutputStream & Save (OutputStream &stream, T &var)

6.56.1 Detailed Description

template<typename T>
struct data::Serializer< T >

This template class allows to define a "serializer".

By default, the basic serializer calls the method `SerializeWith` of the objet to be serialized.

In order to define a serializer of any other specific type, it is required to define a specialization of this template class, and redefine the methods `Load` and `Save`.

See also

BaseStream

6.56.2 Member Function Documentation

6.56.2.1 template<typename T > static InputStream& data::Serializer< T >::Load (InputStream & *stream,* T & *var*) [inline], [static]

Here is the caller graph for this function:

6.56.2.2 template<typename T > static OutputStream& data::Serializer< T >::Save (OutputStream & *stream,* T & *var*) [inline], [static]

Here is the caller graph for this function:

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

- data/serialize.h

6.57 data::Serializer< bool > Struct Template Reference

Serializer for the `bool` type.

`#include <serialize.h>`

Collaboration diagram for data::Serializer< bool >:

Static Public Member Functions

- static InputStream & Load (InputStream &stream, bool &var)
- static OutputStream & Save (OutputStream &stream, bool &var)

6.57.1 Detailed Description

template<>
struct data::Serializer< bool >

Serializer for the `bool` type.

See also

Serializer

6.57.2 Member Function Documentation

6.57.2.1 static InputStream& data::Serializer< bool >::Load (InputStream & *stream,* bool & *var*) [inline], [static]

Here is the call graph for this function:

6.57.2.2 static OutputStream& data::Serializer< bool >::Save (OutputStream & *stream,* bool & *var*) [inline], [static]

Here is the call graph for this function:

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

- data/serialize.h

6.58 data::Serializer< int > Struct Template Reference

Serializer for the `int` type.

`#include <serialize.h>`

Collaboration diagram for data::Serializer< int >:

Static Public Member Functions

- static InputStream & Load (InputStream &stream, int &var)
- static OutputStream & Save (OutputStream &stream, int &var)

6.58.1 Detailed Description

template<>
struct data::Serializer< int >

Serializer for the `int` type.

See also

Serializer

6.58.2 Member Function Documentation

6.58.2.1 static InputStream& data::Serializer< int >::Load (InputStream & *stream,* int & *var*) [inline], [static]

Here is the call graph for this function:

6.58.2.2 static OutputStream& data::Serializer< int >::Save (OutputStream & *stream,* int & *var*) [inline], [static]

Here is the call graph for this function:

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

- data/serialize.h

6.59 data::Serializer< multimap< string, int > > Struct Template Reference

Serializer for the `multimap<string,int>` class.

`#include <serialize.h>`

Collaboration diagram for data::Serializer< multimap< string, int > >:

Static Public Member Functions

- static InputStream & Load (InputStream &stream, multimap< string, int > &var)
- static OutputStream & Save (OutputStream &stream, multimap< string, int > &var)

6.59.1 Detailed Description

template<>
struct data::Serializer< multimap< string, int > >

Serializer for the `multimap<string,int>` class.

See also

Serializer

6.59.2 Member Function Documentation

6.59.2.1 static InputStream& data::Serializer< multimap< string, int > >::Load (InputStream & *stream,* multimap< string, int > & *var*) [inline], [static]

Here is the call graph for this function:

6.59.2.2 static OutputStream& data::Serializer< multimap< string, int > >::Save (OutputStream & *stream*, multimap< string, int > & *var*) [inline], [static]

Here is the call graph for this function:

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

- data/serialize.h

6.60 data::Serializer< string > Struct Template Reference

Serializer for the `string` class.

`#include <serialize.h>`

Collaboration diagram for data::Serializer< string >:

Static Public Member Functions

- static InputStream & Load (InputStream &stream, string &var)
- static OutputStream & Save (OutputStream &stream, string &var)

6.60.1 Detailed Description

template<>
struct data::Serializer< string >

Serializer for the `string` class.

See also

Serializer

6.60.2 Member Function Documentation

6.60.2.1 static InputStream& data::Serializer< string >::Load (InputStream & *stream*, string & *var*) [inline], [static]

Here is the call graph for this function:

6.60.2.2 static OutputStream& data::Serializer< string >::Save (OutputStream & *stream*, string & *var*) [inline], [static]

Here is the call graph for this function:

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

- data/serialize.h

6.61 data::Serializer< uint64_t > Struct Template Reference

Serializer for the `uint64_t` type.

`#include <serialize.h>`

Collaboration diagram for data::Serializer< uint64_t >:

Static Public Member Functions

- static InputStream & Load (InputStream &stream, uint64_t &var)
- static OutputStream & Save (OutputStream &stream, uint64_t &var)

6.61.1 Detailed Description

template<>
struct data::Serializer< uint64_t >

Serializer for the `uint64_t` type.

See also

Serializer

6.61.2 Member Function Documentation

6.61.2.1 static InputStream& data::Serializer< uint64_t >::Load (InputStream & *stream,* uint64_t & *var*) [inline], [static]

Here is the call graph for this function:

6.61.2.2 static OutputStream& data::Serializer< uint64_t >::Save (OutputStream & *stream,* uint64_t & *var*) [inline], [static]

Here is the call graph for this function:

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

- data/serialize.h

6.62 data::Serializer< vector< T > > Struct Template Reference

Serializer for the `vector` class.

`#include <serialize.h>`

Collaboration diagram for data::Serializer< vector< T > >:

Static Public Member Functions

- static InputStream & Load (InputStream &stream, vector< T > &var)
- static OutputStream & Save (OutputStream &stream, vector< T > &var)

6.62.1 Detailed Description

template<typename T>
struct data::Serializer< vector< T > >

Serializer for the `vector` class.

See also

Serializer

6.62.2 Member Function Documentation

6.62.2.1 template<typename T > static InputStream& data::Serializer< vector< T > >::Load (InputStream & *stream,* vector< T > & *var*) [inline], [static]

Here is the call graph for this function:

6.62.2.2 template<typename T > static OutputStream& data::Serializer< vector< T > >::Save (OutputStream & *stream,* vector< T > & *var*) [inline], [static]

Here is the call graph for this function:

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

- data/serialize.h

6.63 net::Socket Class Reference

This class has been designed to work with UNIX sockets in an easy and object oriented way.

`#include <socket.h>`

Collaboration diagram for net::Socket:

Public Member Functions

- Socket ()

  Initializes the socket id with an invalid value.
- Socket (int s)

  Initializes the socket id with an integer value.
- Socket (const Socket &xs)

  Copy constructor.
- operator int () const

  This operator allows to work directly with UNIX socket API.
- bool IsValid () const
- Socket & operator= (int nsid)

  Copy asignment.
- bool OpenInet (int type=SOCK_STREAM)

  This method creates a new Internet socket, storing its identifier in the object.
- bool OpenUnix (int type=SOCK_STREAM)

  This method creates a new UNIX socket, storing its identifier in the object.
- bool ListenAt (const Address &address, int nstack=10)

  Configures the socket for listening incoming connections.
- bool ConnectTo (const Address &to_address)

  Connects the socket to a server.
- bool BindTo (const Address &address)

  Binds the socket to the specified address.
- int Accept (Address *from_address)

  If it is a server socket, it accepts a new connection.
- Socket & SetBlockingMode (bool state=true)

  Set the blocking mode of the send/receive operations.
- bool IsBlockingMode ()
- int Receive (void *buf, int len, bool prevent_block=false)

  Receives a number of bytes.
- int ReceiveFrom (Address *address, void *buf, int len, bool prevent_block=false)

  Receives a number of bytes.
- int Send (void *buf, int len, bool prevent_block=false)

  Sends a number of bytes.
- int SendTo (const Address &address, void *buf, int len, bool prevent_block=false)

  Sends a number of bytes to a specific address.
- bool SendDescriptor (const Address &address, int fd, int aux=0)

  Sends a descriptor through the socket.
- bool IsValid ()

  Returns `true` if the sockets is valid, that is, if after a polling regarding error status is not successful.
- int WaitForInput (int time_out=-1)

  Waits until input data is available (`POLLIN`).
- int WaitForOutput (int time_out=-1)

  Waits until output data can be sent (`POLLOUT`).
- bool SetNoDelay (int val=1)

  Configures the parameter `TCP_NODELAY` of the socket.
- bool ReceiveDescriptor (int *fd, int *aux=NULL)

  Receives a descriptor from a socket.
- void Close ()

  Closes the socket.
- ~Socket ()

  The destructor does not closes the socket!.

Protected Attributes

- int sid
  Socket id.

6.63.1 Detailed Description

This class has been designed to work with UNIX sockets in an easy and object oriented way.

6.63.2 Constructor & Destructor Documentation

6.63.2.1 net::Socket::Socket ( ) [inline]

Initializes the socket id with an invalid value.

6.63.2.2 net::Socket::Socket (int *s*) [inline]

Initializes the socket id with an integer value.

6.63.2.3 net::Socket::Socket (const Socket & *xs*) [inline]

Copy constructor.

6.63.2.4 net::Socket::~Socket ( ) [inline]

The destructor does not closes the socket!.

6.63.3 Member Function Documentation

6.63.3.1 int net::Socket::Accept (Address * *from_address*) [inline]

If it is a server socket, it accepts a new connection.

Parameters

from_address	Pointer to store the client address.

Returns

The integer identifier (file descriptor) of the new socket.

Here is the call graph for this function:

Here is the caller graph for this function:

6.63.3.2 bool net::Socket::BindTo (const Address & *address*) [inline]

Binds the socket to the specified address.

Parameters

address	Address to bind.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.63.3.3 void net::Socket::Close ( ) [inline]

Closes the socket.

Here is the caller graph for this function:

6.63.3.4 bool net::Socket::ConnectTo (const Address & *to_address*) [inline]

Connects the socket to a server.

Parameters

to_address	Address to connect.

Returns

`true` if successful.

Here is the call graph for this function:

6.63.3.5 bool net::Socket::IsBlockingMode ( )

Returns

The blocking mode of the send/receive operations.

6.63.3.6 bool net::Socket::IsValid ( ) const [inline]

Returns

`true` if the identifier has a valid value.

Here is the caller graph for this function:

6.63.3.7 bool net::Socket::IsValid ( )

Returns `true` if the sockets is valid, that is, if after a polling regarding error status is not successful.

6.63.3.8 bool net::Socket::ListenAt (const Address & *address*, int *nstack* = `10`) `[inline]`

Configures the socket for listening incoming connections.

Parameters

address	Address used to listen.
nstack	Maximum number of clients in listening stack.

Returns

`true` if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.63.3.9 bool net::Socket::OpenInet (int *type* = `SOCK_STREAM`) `[inline]`

This method creates a new Internet socket, storing its identifier in the object.

Parameters

type	Socket type, `SOCK_STREAM` by default.

Returns

`true` if successful.

Here is the caller graph for this function:

6.63.3.10 bool net::Socket::OpenUnix (int *type* = `SOCK_STREAM`) `[inline]`

This method creates a new UNIX socket, storing its identifier in the object.

Parameters

type	Socket type, `SOCK_STREAM` by default.

Returns

`true` if successful.

Here is the caller graph for this function:

6.63.3.11 net::Socket::operator int ( ) const [inline]

This operator allows to work directly with UNIX socket API.

6.63.3.12 Socket& net::Socket::operator= (int *nsid*) [inline]

Copy asignment.

6.63.3.13 int net::Socket::Receive (void * *buf*, int *len*, bool *prevent_block* = false)

Receives a number of bytes.

This methods allows to prevent blocking, without having into account the default blocking mode stablished.

Parameters

buf	Buffer where to store the received bytes.
len	Length of the buffer.
prevent_block	true if blocking will be prevented.

Returns

The number of received bytes.

Here is the caller graph for this function:

6.63.3.14 bool net::Socket::ReceiveDescriptor (int * *fd*, int * *aux* = NULL)

Receives a descriptor from a socket.

Parameters

fd	Variable to store the received descriptor.
aux	Auxiliary information received attached.

Returns

true if successful.

Here is the caller graph for this function:

6.63.3.15 int net::Socket::ReceiveFrom (Address * *address,* void * *buf,* int *len,* bool *prevent_block* = `false`)

Receives a number of bytes.

This methods allows to prevent blocking, without having into account the default blocking mode stablished.

Parameters

address	Pointer to store the from address.
buf	Buffer where to store the received bytes.
len	Length of the buffer.
prevent_block	true if blocking will be prevented.

Returns

The number of received bytes.

Here is the call graph for this function:

6.63.3.16 int net::Socket::Send (void * *buf*, int *len*, bool *prevent_block* = `false`)

Sends a number of bytes.

This methods allows to prevent blocking, without having into account the default blocking mode stablished.

Parameters

buf	Buffer with the bytes to sent.
len	Number of bytes to sent.
prevent_block	true if blocking will be prevented.

Returns

The number of sent bytes.

Here is the caller graph for this function:

6.63.3.17 bool net::Socket::SendDescriptor (const Address & *address*, int *fd*, int *aux* = `0`)

Sends a descriptor through the socket.

Parameters

address	Address of the socket to send the descriptor.
fd	File descriptor.
aux	Auxiliary information to send attached.

Returns

true if successful.

Here is the call graph for this function:

Here is the caller graph for this function:

6.63.3.18 int net::Socket::SendTo (const Address & *address*, void * *buf*, int *len*, bool *prevent_block* = false)

Sends a number of bytes to a specific address.

This methods allows to prevent blocking, without having into account the default blocking mode established.

Parameters

address	Address to send the bytes.
buf	Buffer with the bytes to sent.
len	Number of bytes to sent.
prevent_block	true if blocking will be prevented.

Returns

The number of sent bytes.

Here is the call graph for this function:

Here is the caller graph for this function:

6.63.3.19 Socket & net::Socket::SetBlockingMode (bool *state* = true)

Set the blocking mode of the send/receive operations.

By default, this mode is true.

Parameters

state	Blocking mode state to set.

Returns

A reference of the same object.

6.63.3.20 bool net::Socket::SetNoDelay (int *val* = 1) [inline]

Configures the parameter TCP_NODELAY of the socket.

Parameters

val	New value for the parameter (1 by default).

Returns

true if successful.

6.63.3.21 int net::Socket::WaitForInput (int *time_out* = -1)

Waits until input data is available (POLLIN).

Parameters

time_out	Time out (infinite by default).

Returns

The value returned by the poll function.

6.63.3.22 int net::Socket::WaitForOutput (int *time_out* = -1)

Waits until output data can be sent (POLLOUT).

Parameters

time_out	Time out (infinite by default).

Returns

The value returned by the poll function.

6.63.4 Member Data Documentation

6.63.4.1 int net::Socket::sid [protected]

Socket id.

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

- net/socket.h
- net/socket.cc

6.64 net::SocketBuffer Class Reference

Class derived from the STL std::streambuf to allow streaming with sockets.

```
#include <socket_stream.h>
```

Inheritance diagram for net::SocketBuffer:

Collaboration diagram for net::SocketBuffer:

Public Types

- enum { INPUT_BUFFER_LENGTH = 500, OUTPUT_BUFFER_LENGTH = 500 }

Public Member Functions

- SocketBuffer (int sid, int in_len=INPUT_BUFFER_LENGTH, int out_len=OUTPUT_BUFFER_LENGTH)
- virtual int sync ()
- virtual int_type underflow ()
- virtual int_type overflow (int_type c=EOF)
- int GetReadBytes () const
- Socket * GetSocket ()
- virtual ~SocketBuffer ()

Protected Attributes

- int sum
- int in_len
- int out_len
- char * in_buf
- char * out_buf
- Socket socket

6.64.1 Detailed Description

Class derived from the STL `std::streambuf` to allow streaming with sockets.

See the documentation related to this STL base class to understand the behaviour of the class `SocketBuffer`.

See also

std::streambuf
Socket

6.64.2 Member Enumeration Documentation

6.64.2.1 anonymous enum

Enumerator

INPUT_BUFFER_LENGTH

OUTPUT_BUFFER_LENGTH

6.64.3 Constructor & Destructor Documentation

6.64.3.1 net::SocketBuffer::SocketBuffer (int *sid*, int *in_len* = INPUT_BUFFER_LENGTH, int *out_len* = OUTPUT_BUFFER_LENGTH) [inline]

6.64.3.2 virtual net::SocketBuffer::~SocketBuffer ( ) [inline], [virtual]

6.64.4 Member Function Documentation

6.64.4.1 int net::SocketBuffer::GetReadBytes ( ) const [inline]

6.64.4.2 Socket* net::SocketBuffer::GetSocket ( ) [inline]

6.64.4.3 virtual int_type net::SocketBuffer::overflow (int_type *c* = EOF) [inline], [virtual]

Here is the call graph for this function:

6.64.4.4 virtual int net::SocketBuffer::sync ( ) [inline], [virtual]

Here is the call graph for this function:

Here is the caller graph for this function:

6.64.4.5 virtual int_type net::SocketBuffer::underflow ( ) [inline], [virtual]

Here is the call graph for this function:

6.64.5 Member Data Documentation

6.64.5.1 char* net::SocketBuffer::in_buf [protected]

6.64.5.2 int net::SocketBuffer::in_len [protected]

6.64.5.3 char* net::SocketBuffer::out_buf [protected]

6.64.5.4 int net::SocketBuffer::out_len [protected]

6.64.5.5 Socket net::SocketBuffer::socket [protected]

6.64.5.6 int net::SocketBuffer::sum [protected]

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

- net/socket_stream.h

6.65 net::SocketStream Class Reference

Class derived from `std::iostream` and `SocketBuffer` that represents a socket stream.

`#include <socket_stream.h>`

Inheritance diagram for net::SocketStream:

Collaboration diagram for net::SocketStream:

Public Member Functions

- SocketStream (int sid, int in_len=INPUT_BUFFER_LENGTH, int out_len=OUTPUT_BUFFER_LENGTH)
- Socket * operator-> ()
- virtual ~SocketStream ()

Additional Inherited Members

6.65.1 Detailed Description

Class derived from `std::iostream` and `SocketBuffer` that represents a socket stream.

See also

std::iostream
SocketBuffer

6.65.2 Constructor & Destructor Documentation

6.65.2.1 net::SocketStream::SocketStream (int *sid,* int *in_len* = INPUT_BUFFER_LENGTH, int *out_len* = OUTPUT_BUFFER_LENGTH) [inline]

6.65.2.2 virtual net::SocketStream::~SocketStream ( ) [inline], [virtual]

6.65.3 Member Function Documentation

6.65.3.1 Socket* net::SocketStream::operator-> ( ) [inline]

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

- net/socket_stream.h

6.66 http::Response::StatusCodesInitializer Class Reference

Class used for the initializer.

Collaboration diagram for http::Response::StatusCodesInitializer:

Public Member Functions

- StatusCodesInitializer ()

6.66.1 Detailed Description

Class used for the initializer.

6.66.2 Constructor & Destructor Documentation

6.66.2.1 http::Response::StatusCodesInitializer::StatusCodesInitializer ( )

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

- http/response.h
- http/response.cc

6.67 TraceSystem Class Reference

Wrapper used by the application to handle the log/trace messages by means of the log4cpp library.

`#include <trace.h>`

Collaboration diagram for TraceSystem:

Static Public Member Functions

- static bool AppendToFile (const char *name)
- static bool AppendToFile (const std::string &name)
- static log4cpp::CategoryStream logStream ()
- static log4cpp::CategoryStream errorStream ()
- static log4cpp::CategoryStream traceStream ()

Private Member Functions

- TraceSystem ()
- virtual ~TraceSystem ()
- bool AppendToFile_ (const char *name)

Private Attributes

- log4cpp::Category * category
- log4cpp::Appender * appender
- log4cpp::PatternLayout * layout
- log4cpp::Appender * file_appender
- log4cpp::PatternLayout * file_layout

Static Private Attributes

- static TraceSystem traceSystem

6.67.1 Detailed Description

Wrapper used by the application to handle the log/trace messages by means of the log4cpp library.

6.67.2 Constructor & Destructor Documentation

6.67.2.1 TraceSystem::TraceSystem ( ) [private]

6.67.2.2 TraceSystem::~TraceSystem ( ) [private], [virtual]

6.67.3 Member Function Documentation

6.67.3.1 static bool TraceSystem::AppendToFile (const char * *name*) [inline], [static]

Here is the call graph for this function:

Here is the caller graph for this function:

6.67.3.2 static bool TraceSystem::AppendToFile (const std::string & *name*) [inline], [static]

Here is the call graph for this function:

6.67.3.3 bool TraceSystem::AppendToFile_ (const char * *name*) [private]

Here is the caller graph for this function:

6.67.3.4 static log4cpp::CategoryStream TraceSystem::errorStream ( ) [inline], [static]

6.67.3.5 static log4cpp::CategoryStream TraceSystem::logStream ( ) [inline], [static]

6.67.3.6 static log4cpp::CategoryStream TraceSystem::traceStream ( ) [inline], [static]

6.67.4 Member Data Documentation

6.67.4.1 log4cpp::Appender* TraceSystem::appender [private]

6.67.4.2 log4cpp::Category* TraceSystem::category [private]

6.67.4.3 log4cpp::Appender* TraceSystem::file_appender [private]

6.67.4.4 log4cpp::PatternLayout* TraceSystem::file_layout [private]

6.67.4.5 log4cpp::PatternLayout* TraceSystem::layout [private]

6.67.4.6 TraceSystem TraceSystem::traceSystem [static], [private]

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

- trace.h
- trace.cc

6.68 ui Struct Reference

Collaboration diagram for ui:

Static Public Member Functions

- template<typename T >
  static void read (T &v, T mn, T mx)

6.68.1 Member Function Documentation

6.68.1.1 template<typename T > static void ui::read (T & *v*, T *mn*, T *mx*) [inline], [static]

Here is the caller graph for this function:

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

- packet_information.cc

6.69 net::UnixAddress Class Reference

Class to identify and handle an UNIX address.

`#include <address.h>`

Inheritance diagram for net::UnixAddress:

Collaboration diagram for net::UnixAddress:

Public Member Functions

- UnixAddress ()
  Initializes the address to zero.
- UnixAddress (const UnixAddress &address)
  Copy constructor.
- UnixAddress (const char *path)
  Initializes the address with given path.
- UnixAddress & operator= (const UnixAddress &address)
  Copy assignment.
- UnixAddress & Reset ()
  Removes the file associated to the UNIX address.
- virtual sockaddr * GetSockAddr () const
  Overloaded from the base class to use the internal address structure.
- virtual int GetSize () const
  Overloaded from the base class to use the internal address structure.
- string GetPath () const
  Returns the address path.

Private Attributes

- sockaddr_un sock_addr

  Internal address structure.

6.69.1 Detailed Description

Class to identify and handle an UNIX address.

The used internal address structure is `sockaddr_un`.

See also

Address

6.69.2 Constructor & Destructor Documentation

6.69.2.1 net::UnixAddress::UnixAddress ( ) [inline]

Initializes the address to zero.

6.69.2.2 net::UnixAddress::UnixAddress (const UnixAddress & *address*) [inline]

Copy constructor.

6.69.2.3 net::UnixAddress::UnixAddress (const char * *path*) [inline]

Initializes the address with given path.

Parameters

path	Address path.

6.69.3 Member Function Documentation

6.69.3.1 string net::UnixAddress::GetPath ( ) const [inline]

Returns the address path.

6.69.3.2 virtual int net::UnixAddress::GetSize ( ) const [inline], [virtual]

Overloaded from the base class to use the internal address structure.

Implements net::Address.

6.69.3.3 virtual sockaddr* net::UnixAddress::GetSockAddr ( ) const [inline], [virtual]

Overloaded from the base class to use the internal address structure.

Implements net::Address.

6.69.3.4 UnixAddress& net::UnixAddress::operator= (const UnixAddress & *address*) [inline]

Copy assignment.

6.69.3.5 UnixAddress& net::UnixAddress::Reset ( ) [inline]

Removes the file associated to the UNIX address.

Here is the caller graph for this function:

6.69.4 Member Data Documentation

6.69.4.1 sockaddr_un net::UnixAddress::sock_addr [private]

Internal address structure.

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

- net/address.h

6.70 data::UnlockedAccess Struct Reference

Struct for wrapping the basic `FILE` unlocked functions for reading and writing defined in `stdio_exts.h`.

`#include <file.h>`

Collaboration diagram for data::UnlockedAccess:

Static Public Member Functions

- static void configure (FILE *file_ptr)
- static size_t fwrite (const void *ptr, size_t size, size_t count, FILE *file_ptr)
- static size_t fread (void *ptr, size_t size, size_t count, FILE *file_ptr)
- static int fgetc (FILE *file_ptr)
- static int fputc (int c, FILE *file_ptr)

6.70.1 Detailed Description

Struct for wrapping the basic `FILE` unlocked functions for reading and writing defined in `stdio_exts.h`.

See also

File

6.70.2 Member Function Documentation

6.70.2.1 static void data::UnlockedAccess::configure (FILE * *file_ptr*) [inline], [static]

6.70.2.2 static int data::UnlockedAccess::fgetc (FILE * *file_ptr*) [inline], [static]

6.70.2.3 static int data::UnlockedAccess::fputc (int *c,* FILE * *file_ptr*) [inline], [static]

6.70.2.4 static size_t data::UnlockedAccess::fread (void * *ptr,* size_t *size,* size_t *count,* FILE * *file_ptr*) [inline], [static]

6.70.2.5 static size_t data::UnlockedAccess::fwrite (const void * *ptr,* size_t *size,* size_t *count,* FILE * *file_ptr*) [inline], [static]

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

- data/file.h

6.71 data::vint_vector Class Reference

This class has been implemented with the same philosophy that the class STL vector, but specifically designed to store integers with a length in bytes that can be not multiple from 2 (e.g.

```
#include <vint_vector.h>
```

Collaboration diagram for data::vint_vector:

Public Member Functions

- vint_vector ()

  Initializes the vector to store 64-bit integers.
- vint_vector (int num_bytes)

  Initializes the vector to store integers with the number of bytes given as parameter.
- vint_vector (const vint_vector &v)

  Copy constructor.
- const vint_vector & operator= (const vint_vector &v)

  Copy assignment.
- void set_num_bytes (int num_bytes)

  Changes the number of bytes of the integer values.
- int num_bytes () const

  Returns the number of bytes used.
- int data_bytes () const

  Returns the current number of bytes stored.
- uint64_t operator[ ] (int index) const

  Operator overloading for indexing the integer values.
- void push_back (uint64_t value)

  Adds a new item to the end of the vector.
- void clear ()

  Clears the content.
- int size () const

  Returns the size of the vector, in number of items.
- uint64_t & back ()

  Return the reference of the last item of the vector.
- virtual ~vint_vector ()

Private Attributes

- uint64_t mask

  Mask used for accessing the data.
- int8_t num_bytes_

  Number of bytes used for the integers.
- vector< uint8_t > data

6.71.1 Detailed Description

This class has been implemented with the same philosophy that the class STL vector, but specifically designed to store integers with a length in bytes that can be not multiple from 2 (e.g.

integers of 3 bytes). This class internally handles a vector of 1-byte integers.

See also

vector

6.71.2 Constructor & Destructor Documentation

6.71.2.1 data::vint_vector::vint_vector ( ) [inline]

Initializes the vector to store 64-bit integers.

6.71.2.2 data::vint_vector::vint_vector (int *num_bytes*) [inline]

Initializes the vector to store integers with the number of bytes given as parameter.

Parameters

num_bytes	Number of bytes of each integer.

6.71.2.3 data::vint_vector::vint_vector (const vint_vector & *v*) [inline]

Copy constructor.

6.71.2.4 virtual data::vint_vector::~vint_vector ( ) [inline], [virtual]

6.71.3 Member Function Documentation

6.71.3.1 uint64_t& data::vint_vector::back ( ) [inline]

Return the reference of the last item of the vector.

Here is the caller graph for this function:

6.71.3.2 void data::vint_vector::clear ( ) [inline]

Clears the content.

Here is the caller graph for this function:

6.71.3.3 int data::vint_vector::data_bytes ( ) const [inline]

Returns the current number of bytes stored.

6.71.3.4 int data::vint_vector::num_bytes ( ) const [inline]

Returns the number of bytes used.

6.71.3.5 const vint_vector& data::vint_vector::operator= (const vint_vector & *v*) [inline]

Copy assignment.

6.71.3.6 uint64_t data::vint_vector::operator[ ] (int *index*) const [inline]

Operator overloading for indexing the integer values.

Parameters

index	Index of the item to return.

Returns

Value of the item, always as a `uint64_t`.

6.71.3.7 void data::vint_vector::push_back (uint64_t *value*) [inline]

Adds a new item to the end of the vector.

Parameters

value	Value to add to the vector.

Here is the caller graph for this function:

6.71.3.8 void data::vint_vector::set_num_bytes (int *num_bytes*) [inline]

Changes the number of bytes of the integer values.

All the current content is removed.

Parameters

num_bytes	New number of bytes to use.

Here is the caller graph for this function:

6.71.3.9 int data::vint_vector::size ( ) const [inline]

Returns the size of the vector, in number of items.

Here is the caller graph for this function:

6.71.4 Member Data Documentation

6.71.4.1 vector<uint8_t> data::vint_vector::data [private]

6.71.4.2 uint64_t data::vint_vector::mask [private]

Mask used for accessing the data.

6.71.4.3 int8_t data::vint_vector::num_bytes_ [private]

Number of bytes used for the integers.

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

- data/vint_vector.h

6.72 jpip::WOI Class Reference

Class that identifies a WOI (Window Of Interest).

`#include <woi.h>`

Collaboration diagram for jpip::WOI:

Public Member Functions

- WOI ()
  Initializes the resolution level to zero.
- WOI (const Point &position, const Size &size, int resolution)
  Initializes the object.
- WOI (const WOI &woi)
  Copy constructor.
- WOI & operator= (const WOI &woi)
  Copy assignment.
- virtual ~WOI ()

Public Attributes

- Size size
  Size of the WOI (width and height)
- Point position
  Position of the upper-left corner of the WOI.
- int resolution
  Resolution level where the WOI is located (0 == the highest)

Friends

- bool operator== (const WOI &a, const WOI &b)
  Returns `true` if the two given WOIs are equal.
- bool operator!= (const WOI &a, const WOI &b)
  Returns `true` if the two given WOIs are not equal.
- ostream & operator<< (ostream &out, const WOI &woi)

6.72.1 Detailed Description

Class that identifies a WOI (Window Of Interest).

This term refers, from the point of view of the JPIP protocol, to a rectangular region of an image, for a resolution level. This class can be printed.

See also

Point

6.72.2 Constructor & Destructor Documentation

6.72.2.1 jpip::WOI::WOI ( ) [inline]

Initializes the resolution level to zero.

6.72.2.2 jpip::WOI::WOI (const Point & *position,* const Size & *size,* int *resolution*) [inline]

Initializes the object.

Parameters

position	Position of the WOI.
size	Size of the WOI.
resolution	Resolution level of the WOI.

6.72.2.3 jpip::WOI::WOI (const WOI & *woi*) [inline]

Copy constructor.

6.72.2.4 virtual jpip::WOI::~WOI ( ) [inline], [virtual]

6.72.3 Member Function Documentation

6.72.3.1 WOI& jpip::WOI::operator= (const WOI & *woi*) [inline]

Copy assignment.

6.72.4 Friends And Related Function Documentation

6.72.4.1 bool operator!= (const WOI & *a,* const WOI & *b*) [friend]

Returns `true` if the two given WOIs are not equal.

6.72.4.2 ostream& operator<< (ostream & *out,* const WOI & *woi*) [friend]

6.72.4.3 bool operator== (const WOI & *a,* const WOI & *b*) [friend]

Returns `true` if the two given WOIs are equal.

6.72.5 Member Data Documentation

6.72.5.1 Point jpip::WOI::position

Position of the upper-left corner of the WOI.

6.72.5.2 int jpip::WOI::resolution

Resolution level where the WOI is located (0 == the highest)

6.72.5.3 Size jpip::WOI::size

Size of the WOI (width and height)

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

- jpip/woi.h

6.73 jpip::WOIComposer Class Reference

By means of this class it is possible to find out the which packets of an image are associated to a WOI.

`#include <woi_composer.h>`

Collaboration diagram for jpip::WOIComposer:

Public Member Functions

- WOIComposer ()
  Initializes the object.
- WOIComposer (const WOIComposer &composer)
  Copy constructor.
- WOIComposer (CodingParameters::Ptr coding_parameters)
  Initializes the object.
- void Reset (const WOI &woi, CodingParameters::Ptr coding_parameters)
  Resets the packets navigation and starts a new one.
- WOIComposer & operator= (const WOIComposer &composer)
  Copy assignment.
- Packet GetCurrentPacket () const
  Returns the current packet.
- bool GetNextPacket (Packet *packet=NULL)
  Moves to the next packet of the WOI.
- virtual ~WOIComposer ()

Private Attributes

- Point pxy1
  Upper-left corner of the WOI.
- Point pxy2
  Bottom-right corner of the WOI.
- bool more_packets
  Flag to control the last packet.
- int max_resolution
  Maximum resolution.
- Size min_precinct_xy
  Minimum precinct.
- Size max_precinct_xy
  Maximum precinct.
- Packet current_packet
  Current packet.
- CodingParameters::Ptr coding_parameters
  Pointer to the associated coding parameters.

6.73.1 Detailed Description

By means of this class it is possible to find out the which packets of an image are associated to a WOI.

Given a WOI and the coding parameters of an image, the code of this class allows to navigate, following the LRCP order, through all the associated packets.

See also

WOI
CodingParameters

6.73.2 Constructor & Destructor Documentation

6.73.2.1 jpip::WOIComposer::WOIComposer ( ) [inline]

Initializes the object.

No packets are available.

6.73.2.2 jpip::WOIComposer::WOIComposer (const WOIComposer & *composer*) [inline]

Copy constructor.

6.73.2.3 jpip::WOIComposer::WOIComposer (CodingParameters::Ptr *coding_parameters*) [inline]

Initializes the object.

No packets are available.

Parameters

coding_parameters	Pointer to the coding parameters.

6.73.2.4 virtual jpip::WOIComposer::~WOIComposer ( ) [inline], [virtual]

6.73.3 Member Function Documentation

6.73.3.1 Packet jpip::WOIComposer::GetCurrentPacket ( ) const [inline]

Returns the current packet.

Here is the caller graph for this function:

6.73.3.2 bool jpip::WOIComposer::GetNextPacket (Packet * *packet* = NULL) [inline]

Moves to the next packet of the WOI.

Parameters

packet	Pointer to store the current packet (not the next one).

Returns

`true` if successful.

Here is the caller graph for this function:

6.73.3.3 WOIComposer& jpip::WOIComposer::operator= (const WOIComposer & *composer*) [inline]

Copy assignment.

6.73.3.4 void jpip::WOIComposer::Reset (const WOI & *woi*, CodingParameters::Ptr *coding_parameters*) [inline]

Resets the packets navigation and starts a new one.

Sets the current packet to the first packet of the WOI, assuming a LRCP order.

Parameters

woi	New WOI to use.
coding_parameters	Coding parameters to use.

Here is the caller graph for this function:

6.73.4 Member Data Documentation

6.73.4.1 CodingParameters::Ptr jpip::WOIComposer::coding_parameters [private]

Pointer to the associated coding parameters.

6.73.4.2 Packet jpip::WOIComposer::current_packet [private]

Current packet.

6.73.4.3 Size jpip::WOIComposer::max_precinct_xy [private]

Maximum precinct.

6.73.4.4 int jpip::WOIComposer::max_resolution [private]

Maximum resolution.

6.73.4.5 Size jpip::WOIComposer::min_precinct_xy [private]

Minimum precinct.

6.73.4.6 bool jpip::WOIComposer::more_packets [private]

Flag to control the last packet.

6.73.4.7 Point jpip::WOIComposer::pxy1 [private]

Upper-left corner of the WOI.

6.73.4.8 Point jpip::WOIComposer::pxy2 [private]

Bottom-right corner of the WOI.

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

- jpip/woi_composer.h

Chapter 7

File Documentation

7.1 app_config.cc File Reference

```
#include <libconfig.h++>
#include "app_config.h"
```
Include dependency graph for app_config.cc:

7.2 app_config.h File Reference

```
#include <string>
#include <iostream>
```
Include dependency graph for app_config.h: This graph shows which files directly or indirectly include this file:

Classes

- class AppConfig

  Contains the configuration parameters of the application.

7.3 app_info.cc File Reference

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <sys/file.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <sys/sysctl.h>
#include "app_info.h"
```
Include dependency graph for app_info.cc:

Macros

- #define LOCK_FILE "/tmp/esa_jpip_server.lock"

7.3.1 Macro Definition Documentation

7.3.1.1 #define LOCK_FILE "/tmp/esa_jpip_server.lock"

7.4 app_info.h File Reference

```
#include <assert.h>
#include <iostream>
#include <iomanip>
#include <proc/readproc.h>
```

Include dependency graph for app_info.h: This graph shows which files directly or indirectly include this file:

Classes

- class AppInfo

  Contains the run-time information of the application.
- struct AppInfo::Data

  Contains the data block that is maintained in shared memory.

7.5 args_parser.cc File Reference

```
#include <string>
#include <iomanip>
#include <iostream>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <dirent.h>
#include "trace.h"
#include "args_parser.h"
```

Include dependency graph for args_parser.cc:

7.6 args_parser.h File Reference

```
#include "app_info.h"
#include "app_config.h"
```

Include dependency graph for args_parser.h: This graph shows which files directly or indirectly include this file:

Classes

- class ArgsParser

  Class that allows to parse and handle the application command line parameters.

7.7 base.cc File Reference

```
#include "base.h"
```
Include dependency graph for base.cc:

7.8 base.h File Reference

```
#include <map>
#include <vector>
#include <sstream>
```
Include dependency graph for base.h: This graph shows which files directly or indirectly include this file:

Classes

- struct base

  Contains a set of useful static methods used by the application.

7.9 client_info.cc File Reference

```
#include "client_info.h"
```
Include dependency graph for client_info.cc:

7.10 client_info.h File Reference

```
#include <time.h>
```
Include dependency graph for client_info.h: This graph shows which files directly or indirectly include this file:

Classes

- class ClientInfo

  Contains information of a connected client.

7.11 client_manager.cc File Reference

```
#include "trace.h"
#include "client_manager.h"
#include "jpip/jpip.h"
#include "jpip/request.h"
#include "jpip/databin_server.h"
#include "http/header.h"
#include "http/response.h"
#include "net/socket_stream.h"
#include "jpeg2000/index_manager.h"
```

Include dependency graph for client_manager.cc:

7.12 client_manager.h File Reference

```
#include "app_info.h"
#include "app_config.h"
#include "client_info.h"
#include "jpeg2000/index_manager.h"
```

Include dependency graph for client_manager.h: This graph shows which files directly or indirectly include this file:

Classes

- class ClientManager

  Handles a client connection with a dedicated thread.

7.13 data/data.h File Reference

Namespaces

- data

  Contains a set of classes to easy the handling of data and files, as well as the serialization.

7.14 data/file.cc File Reference

```
#include "file.h"
```

Include dependency graph for file.cc:

7.15 data/file.h File Reference

```
#include <stdio.h>
#include <assert.h>
#include <stdint.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string>
#include <stdio_ext.h>
#include "tr1_compat.h"
```

Include dependency graph for file.h: This graph shows which files directly or indirectly include this file:

Classes

- struct data::LockedAccess

  Struct for wrapping the basic `FILE` locked functions for reading and writing defined in `stdio.h`.
- struct data::UnlockedAccess

  Struct for wrapping the basic `FILE` unlocked functions for reading and writing defined in `stdio_exts.h`.
- class data::BaseFile< IO >

  This is a wrapper class for the `FILE` functions that provides all the functionality to handle files safely.

Namespaces

- data

  Contains a set of classes to easy the handling of data and files, as well as the serialization.

Typedefs

- typedef BaseFile< LockedAccess > data::File

  Specialization of the class `BaseFile` with locked access.
- typedef BaseFile< UnlockedAccess > data::FastFile

  Specialization of the class `BaseFile` with unlocked access.

7.16 data/file_segment.cc File Reference

```
#include "file_segment.h"
```
Include dependency graph for file_segment.cc:

Namespaces

- data

  Contains a set of classes to easy the handling of data and files, as well as the serialization.

7.17 data/file_segment.h File Reference

```
#include <iostream>
#include <stdint.h>
#include <assert.h>
```
Include dependency graph for file_segment.h: This graph shows which files directly or indirectly include this file:

Classes

- class data::FileSegment

  Identifies a data segment of a file.

Namespaces

- data

  Contains a set of classes to easy the handling of data and files, as well as the serialization.

7.18 data/serialize.cc File Reference

```
#include "serialize.h"
```
Include dependency graph for serialize.cc:

7.19 data/serialize.h File Reference

```
#include <stdint.h>
#include <string>
#include <vector>
#include <map>
#include "file.h"
```
Include dependency graph for serialize.h: This graph shows which files directly or indirectly include this file:

Classes

- struct data::InputOperator

  This struct identifies a basic input operator to be applied to a `File` object.
- struct data::OutputOperator

  This struct identifies a basic output operator to be applied to a `File` object.
- class data::BaseStream< StreamClass, StreamOperator >

  This template is used as the base for the input/output stream classes.
- struct data::Serializer< T >

  This template class allows to define a "serializer".
- class data::InputStream

  Specialization of the `BaseStream` for input serializations.
- class data::OutputStream

  Specialization of the `BaseStream` for output serializations.
- struct data::Serializer< bool >

  Serializer for the `bool` type.
- struct data::Serializer< int >

  Serializer for the `int` type.
- struct data::Serializer< uint64_t >

  Serializer for the `uint64_t` type.
- struct data::Serializer< string >

  Serializer for the `string` class.
- struct data::Serializer< vector< T > >

  Serializer for the `vector` class.
- struct data::Serializer< multimap< string, int > >

  Serializer for the `multimap<string,int>` class.

Namespaces

- data

  Contains a set of classes to easy the handling of data and files, as well as the serialization.

7.20 data/vint_vector.cc File Reference

```
#include "vint_vector.h"
```
Include dependency graph for vint_vector.cc:

7.21 data/vint_vector.h File Reference

```
#include <vector>
#include <stdint.h>
#include <assert.h>
#include <algorithm>
```
Include dependency graph for vint_vector.h: This graph shows which files directly or indirectly include this file:

Classes

- class data::vint_vector

  This class has been implemented with the same philosophy that the class STL vector, but specifically designed to store integers with a length in bytes that can be not multiple from 2 (e.g.

Namespaces

- data

  Contains a set of classes to easy the handling of data and files, as well as the serialization.

7.22 esa_jpip_server.cc File Reference

```
#include <sys/wait.h>
#include <signal.h>
#include "trace.h"
#include "version.h"
#include "app_info.h"
#include "app_config.h"
#include "args_parser.h"
#include "client_info.h"
#include "client_manager.h"
#include "net/poll_table.h"
#include "net/socket_stream.h"
#include "jpeg2000/file_manager.h"
#include "jpeg2000/index_manager.h"
```
Include dependency graph for esa_jpip_server.cc:

Macros

- #define SERVER_NAME "ESA JPIP Server"
- #define SERVER_APP_NAME "esa_jpip_server"
- #define CONFIG_FILE "server.cfg"
- #define POLLRDHUP (0)

Functions

- int ChildProcess ()
- void * ClientThread (void *arg)
- bool ParseArguments (int argc, char **argv)
- void SIGCHLD_handler (int signal)
- int main (int argc, char **argv)

Variables

- AppConfig cfg
- int base_id = 0
- AppInfo app_info
- Socket child_socket
- PollTable poll_table
- bool child_lost = false
- IndexManager index_manager
- UnixAddress child_address ("/tmp/child_unix_address")
- UnixAddress father_address ("/tmp/father_unix_address")

7.22.1 Macro Definition Documentation

7.22.1.1 #define CONFIG_FILE "server.cfg"

7.22.1.2 #define POLLRDHUP (0)

7.22.1.3 #define SERVER_APP_NAME "esa_jpip_server"

7.22.1.4 #define SERVER_NAME "ESA JPIP Server"

7.22.2 Function Documentation

7.22.2.1 int ChildProcess ( )

Here is the call graph for this function:

Here is the caller graph for this function:

7.22.2.2 void * ClientThread (void * *arg*)

Here is the call graph for this function:

Here is the caller graph for this function:

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

Here is the call graph for this function:

7.22.2.4 bool ParseArguments (int *argc,* char ** *argv*)

7.22.2.5 void SIGCHLD_handler (int *signal*)

Here is the caller graph for this function:

7.22.3 Variable Documentation

7.22.3.1 AppInfo app_info

7.22.3.2 int base_id = 0

7.22.3.3 AppConfig cfg

7.22.3.4 UnixAddress child_address("/tmp/child_unix_address")

7.22.3.5 bool child_lost = false

7.22.3.6 Socket child_socket

7.22.3.7 UnixAddress father_address("/tmp/father_unix_address")

7.22.3.8 IndexManager index_manager

7.22.3.9 PollTable poll_table

7.23 http/header.cc File Reference

```
#include "header.h"
```

Include dependency graph for header.cc:

Namespaces

- http

  Contains the definition of a set of classes for working easily with the protocol HTTP.

7.24 http/header.h File Reference

```
#include <string.h>
#include <iostream>
#include <assert.h>
#include "protocol.h"
```

Include dependency graph for header.h: This graph shows which files directly or indirectly include this file:

Classes

- class http::HeaderName

  Container for the strings associated to the most common HTTP headers, used for the specialization of the class `HeaderBase`.
- class http::HeaderBase< NAME >

  Template class used to identify a HTTP header.
- class http::HeaderBase< HeaderName::UNDEFINED >

  Specialization of the `HeaderBase` template class with the `HeaderName::UNDEFINED` value.
- class http::Header

  Class used to handle a HTTP header.

Namespaces

- http

  Contains the definition of a set of classes for working easily with the protocol HTTP.

7.25 http/http.h File Reference

Namespaces

- http

  Contains the definition of a set of classes for working easily with the protocol HTTP.

7.26 http/protocol.cc File Reference

```
#include "protocol.h"
```

Include dependency graph for protocol.cc:

Namespaces

- http

  Contains the definition of a set of classes for working easily with the protocol HTTP.

7.27 http/protocol.h File Reference

```
#include <string>
#include <iostream>
#include <assert.h>
```

Include dependency graph for protocol.h: This graph shows which files directly or indirectly include this file:

Classes

- class http::Protocol

  Class used to identify the HTTP protocol.

Namespaces

- http

  Contains the definition of a set of classes for working easily with the protocol HTTP.

7.28 http/request.cc File Reference

```
#include "trace.h"
#include "request.h"
```

Include dependency graph for request.cc:

Namespaces

- http

  Contains the definition of a set of classes for working easily with the protocol HTTP.

Functions

- istream & http::operator>> (istream &in, Request &request)
- ostream & http::operator<< (ostream &out, const Request &request)

7.29 jpip/request.cc File Reference

```
#include "trace.h"
#include "request.h"
```

Include dependency graph for request.cc:

Namespaces

- jpip

  Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.30 http/request.h File Reference

```
#include <map>
#include <string>
#include <vector>
#include <sstream>
#include <iostream>
#include <stdlib.h>
#include "header.h"
#include "protocol.h"
```

Include dependency graph for request.h: This graph shows which files directly or indirectly include this file:

Classes

- class http::Request

  Class used to identify a HTTP request (GET or POST).

Namespaces

- http

  Contains the definition of a set of classes for working easily with the protocol HTTP.

7.31 jpip/request.h File Reference

```
#include <vector>
#include <string>
#include <iostream>
#include "jpip/woi.h"
#include "jpip/cache_model.h"
#include "http/request.h"
#include "jpeg2000/point.h"
#include "jpeg2000/coding_parameters.h"
#include <string.h>
```

Include dependency graph for request.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpip::Request

  Class derived from the HTTP `Request` class that contains the required code for properly analyzing a JPIP request, when this protocol is used over the HTTP.
- union jpip::Request::ParametersMask

  Union used to control the presence of the different JPIP parameters in a request.

Namespaces

- jpip

  Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.32 http/response.cc File Reference

```
#include "response.h"
```
Include dependency graph for response.cc:

Namespaces

- http

  Contains the definition of a set of classes for working easily with the protocol HTTP.

7.33 http/response.h File Reference

```
#include <map>
#include <string>
#include <sstream>
#include <iostream>
#include "protocol.h"
```
Include dependency graph for response.h: This graph shows which files directly or indirectly include this file:

Classes

- class http::Response

  Class used to identify a HTTP response.
- class http::Response::StatusCodesInitializer

  Class used for the initializer.

Namespaces

- http

  Contains the definition of a set of classes for working easily with the protocol HTTP.

7.34 ipc/event.cc File Reference

```
#include <errno.h>
#include <assert.h>
#include <sys/time.h>
#include "event.h"
```
Include dependency graph for event.cc:

Namespaces

- ipc

  Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.35 ipc/event.h File Reference

```
#include <pthread.h>
#include "ipc_object.h"
```
Include dependency graph for event.h: This graph shows which files directly or indirectly include this file:

Classes

- class ipc::Event

  IPC object that offers the functionality of an event (Windows IPC object), implemented by means of a combination of the pthread mutex and conditional variables API.

Namespaces

- ipc

  Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.36 ipc/ipc.h File Reference

Namespaces

- ipc

  Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.37 ipc/ipc_object.cc File Reference

```
#include "ipc_object.h"
```
Include dependency graph for ipc_object.cc:

7.38 ipc/ipc_object.h File Reference

```
#include "tr1_compat.h"
```
Include dependency graph for ipc_object.h: This graph shows which files directly or indirectly include this file:

Classes

- class ipc::IPCObject

  Class base of all the IPC classes that has the basic operations (`Init`, `Wait` and `Dispose`) to be overloaded.

Namespaces

- ipc

  Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

Enumerations

- enum ipc::WaitResult { ipc::WAIT_OBJECT = 0, ipc::WAIT_TIMEOUT, ipc::WAIT_ERROR }

  Enumeration of the possible values returned when a wait operation is performed for an IPC object.

7.39 ipc/mutex.cc File Reference

```
#include <assert.h>
#include <errno.h>
#include <sched.h>
#include "mutex.h"
```

Include dependency graph for mutex.cc:

Namespaces

- ipc

  Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.40 ipc/mutex.h File Reference

```
#include <pthread.h>
#include "ipc_object.h"
```

Include dependency graph for mutex.h: This graph shows which files directly or indirectly include this file:

Classes

- class ipc::Mutex

  IPC object that offers the functionality of a mutex, implemented by means of the pthread mutex API.

Namespaces

- ipc

  Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.41 ipc/rdwr_lock.cc File Reference

```
#include "rdwr_lock.h"
#include <assert.h>
#include <errno.h>
#include <sched.h>
#include <iostream>
```

Include dependency graph for rdwr_lock.cc:

Namespaces

- ipc

  Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.42 ipc/rdwr_lock.h File Reference

```
#include <pthread.h>
#include "ipc_object.h"
```

Include dependency graph for rdwr_lock.h: This graph shows which files directly or indirectly include this file:

Classes

- class ipc::RdWrLock

  IPC object that offers the functionality of a read/write lock, implemented by means of the pthread rwlock API.

Namespaces

- ipc

  Contains classes for working with the IPC mechanisms available in Linux using the `pthread` library.

7.43 jpeg2000/codestream_index.cc File Reference

```
#include "codestream_index.h"
```

Include dependency graph for codestream_index.cc:

7.44 jpeg2000/codestream_index.h File Reference

```
#include <vector>
#include "base.h"
#include "data/file_segment.h"
```

Include dependency graph for codestream_index.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::CodestreamIndex

  Class used for indexing the information of a JPEG2000 codestream.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.45 jpeg2000/coding_parameters.cc File Reference

```
#include "coding_parameters.h"
```
Include dependency graph for coding_parameters.cc:

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.46 jpeg2000/coding_parameters.h File Reference

```
#include <vector>
#include <math.h>
#include "tr1_compat.h"
#include "base.h"
#include "point.h"
#include "trace.h"
#include "packet.h"
```
Include dependency graph for coding_parameters.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::CodingParameters

  Contains the coding parameters of a JPEG2000 image codestream.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.47 jpeg2000/file_manager.cc File Reference

```
#include "file_manager.h"
#include "trace.h"
```
Include dependency graph for file_manager.cc:

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

Macros

- #define EOC_MARKER 0xFFD9
- #define SOC_MARKER 0xFF4F
- #define SIZ_MARKER 0xFF51
- #define COD_MARKER 0xFF52
- #define SOT_MARKER 0xFF90
- #define PLT_MARKER 0xFF58
- #define SOD_MARKER 0xFF93
- #define JP2C_BOX_ID 0x6A703263
- #define XML__BOX_ID 0x786D6C20
- #define ASOC_BOX_ID 0x61736F63
- #define NLST_BOX_ID 0x6E6C7374
- #define JPCH_BOX_ID 0x6A706368
- #define FTBL_BOX_ID 0x6674626C
- #define DBTL_BOX_ID 0x6474626C
- #define URL__BOX_ID 0x75726C20
- #define FLST_BOX_ID 0x666C7374

7.47.1 Macro Definition Documentation

7.47.1.1 #define ASOC_BOX_ID 0x61736F63

7.47.1.2 #define COD_MARKER 0xFF52

7.47.1.3 #define DBTL_BOX_ID 0x6474626C

7.47.1.4 #define EOC_MARKER 0xFFD9

7.47.1.5 #define FLST_BOX_ID 0x666C7374

7.47.1.6 #define FTBL_BOX_ID 0x6674626C

7.47.1.7 #define JP2C_BOX_ID 0x6A703263

7.47.1.8 #define JPCH_BOX_ID 0x6A706368

7.47.1.9 #define NLST_BOX_ID 0x6E6C7374

7.47.1.10 #define PLT_MARKER 0xFF58

7.47.1.11 #define SIZ_MARKER 0xFF51

7.47.1.12 #define SOC_MARKER 0xFF4F

7.47.1.13 #define SOD_MARKER 0xFF93

7.47.1.14 #define SOT_MARKER 0xFF90

7.47.1.15 #define URL__BOX_ID 0x75726C20

7.47.1.16 #define XML__BOX_ID 0x786D6C20

7.48 jpeg2000/file_manager.h File Reference

```
#include <sys/stat.h>
#include "data/serialize.h"
#include "image_info.h"
```

Include dependency graph for file_manager.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::FileManager

  Manages the image files of a repository, allowing read their indexing information, with a caching mechanism for efficiency.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.49 jpeg2000/image_index.cc File Reference

```
#include "trace.h"
#include "image_index.h"
```

Include dependency graph for image_index.cc:

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.50 jpeg2000/image_index.h File Reference

```
#include "trace.h"
#include <list>
#include <vector>
#include "base.h"
#include "range.h"
#include "image_info.h"
#include "packet_index.h"
#include "ipc/rdwr_lock.h"
```

Include dependency graph for image_index.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::ImageIndex

  Contains the indexing information of a JPEG2000 image file that is managed by the index manager.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.51 jpeg2000/image_info.cc File Reference

```
#include "image_info.h"
```

Include dependency graph for image_info.cc:

7.52 jpeg2000/image_info.h File Reference

```
#include <map>
#include "base.h"
#include "data/file.h"
#include "codestream_index.h"
#include "coding_parameters.h"
#include "meta_data.h"
```

Include dependency graph for image_info.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::ImageInfo

  Contains the indexing information of a JPEG2000 image.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.53 jpeg2000/index_manager.cc File Reference

```
#include <assert.h>
#include "trace.h"
#include "index_manager.h"
```

Include dependency graph for index_manager.cc:

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.54 jpeg2000/index_manager.h File Reference

```
#include <list>
#include "ipc/mutex.h"
#include "image_index.h"
#include "file_manager.h"
```

Include dependency graph for index_manager.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::IndexManager

  Manages the indexing information of a repository fo images.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.55 jpeg2000/jpeg2000.h File Reference

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.56 jpeg2000/meta_data.cc File Reference

```
#include "meta_data.h"
```

Include dependency graph for meta_data.cc:

7.57 jpeg2000/meta_data.h File Reference

```
#include <vector>
#include "base.h"
#include "jpeg2000/place_holder.h"
```

Include dependency graph for meta_data.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::Metadata

  Contains the indexing information associated to the meta-data of a JPEG2000 image file.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.58 jpeg2000/packet.cc File Reference

```
#include "packet.h"
```
Include dependency graph for packet.cc:

7.59 jpeg2000/packet.h File Reference

```
#include "point.h"
```
Include dependency graph for packet.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::Packet

  Contains the information of a packet.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.60 jpeg2000/packet_index.cc File Reference

```
#include "packet_index.h"
```
Include dependency graph for packet_index.cc:

7.61 jpeg2000/packet_index.h File Reference

```
#include "data/vint_vector.h"
#include "data/file_segment.h"
```
Include dependency graph for packet_index.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::PacketIndex

  Class used for indexing the packets of a codestream image.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.62 jpeg2000/place_holder.cc File Reference

`#include "place_holder.h"`
Include dependency graph for place_holder.cc:

7.63 jpeg2000/place_holder.h File Reference

`#include "data/file_segment.h"`
Include dependency graph for place_holder.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::PlaceHolder

  Contains the information of a place-holder.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.64 jpeg2000/point.cc File Reference

`#include "point.h"`
Include dependency graph for point.cc:

7.65 jpeg2000/point.h File Reference

`#include <iostream>`
Include dependency graph for point.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::Point

  Represents a couple of integer values that can be used to identify a coordinate as well as a size.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

Typedefs

- typedef Point jpeg2000::Size

  It is a synonymous of the class `Point`.

7.66 jpeg2000/range.cc File Reference

```
#include "range.h"
```

Include dependency graph for range.cc:

7.67 jpeg2000/range.h File Reference

```
#include <iostream>
#include <assert.h>
```

Include dependency graph for range.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpeg2000::Range

  Represents a range of integer values, defined by two values, first and last, which are assumed to be included in the range.

Namespaces

- jpeg2000

  Set of classes for handling (reading and indexing) image files with the format defined in the Part 1 and 2 of the JPEG2000 standard.

7.68 jpip/cache_model.cc File Reference

```
#include "cache_model.h"
```

Include dependency graph for cache_model.cc:

7.69 jpip/cache_model.h File Reference

```
#include <vector>
#include <iostream>
#include <limits.h>
#include "base.h"
#include "jpip/jpip.h"
#include "data/serialize.h"
```

Include dependency graph for cache_model.h: This graph shows which files directly or indirectly include this file:

Classes

- struct jpip::DataBinSelector< BIN_CLASS >

  Template class that is specialized for allowing basic operations (add and get) with cache models depending on the data-bin classes.
- class jpip::CacheModel

  The cache model of a JPIP client is handled using this class.
- class jpip::CacheModel::Codestream

  Sub-class of the cache model class used to identify a codestream.
- struct jpip::DataBinSelector< DataBinClass::META_DATA >
- struct jpip::DataBinSelector< DataBinClass::MAIN_HEADER >
- struct jpip::DataBinSelector< DataBinClass::TILE_HEADER >
- struct jpip::DataBinSelector< DataBinClass::PRECINCT >

Namespaces

- jpip

  Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.70 jpip/databin_server.cc File Reference

```
#include "databin_server.h"
#include "data/file_segment.h"
```

Include dependency graph for databin_server.cc:

Namespaces

- jpip

  Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.71 jpip/databin_server.h File Reference

```
#include <utility>
#include "trace.h"
#include "data/file.h"
#include "jpip/woi.h"
#include "jpip/request.h"
#include "jpip/cache_model.h"
#include "jpip/woi_composer.h"
#include "jpip/databin_writer.h"
#include "jpeg2000/range.h"
#include "jpeg2000/image_index.h"
```

Include dependency graph for databin_server.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpip::DataBinServer

  Contains the core functionality of a (JPIP) data-bin server, which maintains a cache model and is capable of generating data chunks of variable length;.

Namespaces

- jpip

  Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.72 jpip/databin_writer.cc File Reference

```
#include "databin_writer.h"
```
Include dependency graph for databin_writer.cc:

Namespaces

- jpip

  Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.73 jpip/databin_writer.h File Reference

```
#include <stdint.h>
#include "jpip.h"
#include "data/file.h"
#include "data/file_segment.h"
#include "jpeg2000/place_holder.h"
```
Include dependency graph for databin_writer.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpip::DataBinWriter

  Class used to generate data-bin segments and write them into a memory buffer.

Namespaces

- jpip

  Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.74 jpip/jpip.cc File Reference

```
#include "jpip.h"
```
Include dependency graph for jpip.cc:

Namespaces

- jpip

  Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.75 jpip/jpip.h File Reference

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

Classes

- class jpip::DataBinClass

  Class that contains the definitions of all the data-bin classes defined for the JPIP protocol.
- class jpip::EOR

  Class that contains all the definitions of the EOF messages defined for the JPIP protocol.

Namespaces

- jpip

  Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.76 jpip/woi.cc File Reference

```
#include "woi.h"
```

Include dependency graph for woi.cc:

7.77 jpip/woi.h File Reference

```
#include <iostream>
#include "jpeg2000/point.h"
```

Include dependency graph for woi.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpip::WOI

  Class that identifies a WOI (Window Of Interest).

Namespaces

- jpip

  Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.78 jpip/woi_composer.cc File Reference

7.79 jpip/woi_composer.h File Reference

```
#include "woi.h"
#include "jpeg2000/packet.h"
#include "jpeg2000/coding_parameters.h"
```

Include dependency graph for woi_composer.h: This graph shows which files directly or indirectly include this file:

Classes

- class jpip::WOIComposer

  By means of this class it is possible to find out the which packets of an image are associated to a WOI.

Namespaces

- jpip

  Set of classes related to the JPIP protocol, defined in the Part 9 of the JPEG2000 standard.

7.80 net/address.cc File Reference

```
#include "address.h"
```

Include dependency graph for address.cc:

7.81 net/address.h File Reference

```
#include <string>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdlib.h>
#include <string.h>
#include <sys/un.h>
#include <unistd.h>
```

Include dependency graph for address.h: This graph shows which files directly or indirectly include this file:

Classes

- class net::Address

  Abstract base class to wrap the `sockaddr` derived structures.
- class net::InetAddress

  Class to identify and handle an Internet address.
- class net::UnixAddress

  Class to identify and handle an UNIX address.

Namespaces

- net

  Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

7.82 net/net.h File Reference

Namespaces

- net

  Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

7.83 net/poll_table.cc File Reference

```
#include "poll_table.h"
```

Include dependency graph for poll_table.cc:

7.84 net/poll_table.h File Reference

```
#include <vector>
#include <poll.h>
#include <algorithm>
```

Include dependency graph for poll_table.h: This graph shows which files directly or indirectly include this file:

Classes

- struct net::PollFD

  Wrapper structure for the structure `pollfd` used by the kernel `poll` functions.
- class net::PollTable

  This class allows to perfom polls easily over a vector of descriptors.

Namespaces

- net

  Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

7.85 net/socket.cc File Reference

```
#include <fcntl.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <assert.h>
#include "socket.h"
#include "poll_table.h"
```

Include dependency graph for socket.cc:

Namespaces

- net

  Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

Macros

- #define POLLRDHUP (0)

7.85.1 Macro Definition Documentation

7.85.1.1 #define POLLRDHUP (0)

7.86 net/socket.h File Reference

```
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/times.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string>
#include "address.h"
```

Include dependency graph for socket.h: This graph shows which files directly or indirectly include this file:

Classes

- class net::Socket

  This class has been designed to work with UNIX sockets in an easy and object oriented way.

Namespaces

- net

  Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

7.87 net/socket_stream.cc File Reference

```
#include "socket_stream.h"
```

Include dependency graph for socket_stream.cc:

7.88 net/socket_stream.h File Reference

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include "socket.h"
```

Include dependency graph for socket_stream.h: This graph shows which files directly or indirectly include this file:

Classes

- class net::SocketBuffer

  Class derived from the STL `std::streambuf` to allow streaming with sockets.
- class net::SocketStream

  Class derived from `std::iostream` and `SocketBuffer` that represents a socket stream.

Namespaces

- net

  Contains classes to easy the utilization of sockets, specially implemented for UNIX systems.

7.89 packet_information.cc File Reference

```
#include "trace.h"
#include "app_config.h"
#include "jpip/woi_composer.h"
#include "jpeg2000/index_manager.h"
```

Include dependency graph for packet_information.cc:

Classes

- struct ui

Functions

- int main (void)

7.89.1 Function Documentation

7.89.1.1 int main (void)

Here is the call graph for this function:

7.90 tr1_compat.h File Reference

```
#include <tr1/memory>
```
Include dependency graph for tr1_compat.h: This graph shows which files directly or indirectly include this file:

Macros

- #define SHARED_PTR std::tr1::shared_ptr

7.90.1 Macro Definition Documentation

7.90.1.1 #define SHARED_PTR std::tr1::shared_ptr

7.91 trace.cc File Reference

```
#include "trace.h"
```
Include dependency graph for trace.cc:

7.92 trace.h File Reference

```
#include <string>
#include <iostream>
#include <log4cpp/Category.hh>
#include <log4cpp/FileAppender.hh>
#include <log4cpp/PatternLayout.hh>
#include <log4cpp/OstreamAppender.hh>
```
Include dependency graph for trace.h: This graph shows which files directly or indirectly include this file:

Classes

- class TraceSystem

  Wrapper used by the application to handle the log/trace messages by means of the log4cpp library.

Macros

- #define LOG4CPP_FIX_ERROR_COLLISION 1
- #define _RED "31m"
- #define _GREEN "32m"
- #define _YELLOW "33m"
- #define _BLUE "34m"
- #define _SET_COLOR(a) "\033[" a
- #define _RESET_COLOR() "\033[0m"
- #define LOG(a) (TraceSystem::logStream() << a << log4cpp::eol)
- #define LOGC(c, a) (TraceSystem::logStream() << _SET_COLOR(c) << a << _RESET_COLOR() << log4cpp::eol)
- #define ERROR(a) (TraceSystem::errorStream() << _SET_COLOR(_RED) << __FILE__ << ":" << __↩ LINE__ << ": ERROR: " << a << _RESET_COLOR() << log4cpp::eol)
- #define TRACE(a) {}
- #define CERR(a) (cerr << _SET_COLOR(_RED) << a << "!" << _RESET_COLOR() << endl, -1)

7.92.1 Macro Definition Documentation

7.92.1.1 #define _BLUE "34m"

7.92.1.2 #define _GREEN "32m"

7.92.1.3 #define _RED "31m"

7.92.1.4 #define _RESET_COLOR( ) "\033[0m"

7.92.1.5 #define _SET_COLOR(*a*) "\033[" a

7.92.1.6 #define _YELLOW "33m"

7.92.1.7 #define CERR(*a*) (cerr << _SET_COLOR(_RED) << a << "!" << _RESET_COLOR() << endl, -1)

7.92.1.8 #define ERROR(*a*) (TraceSystem::errorStream() << _SET_COLOR(_RED) << __FILE__ << ":" << __LINE__ << ": ERROR: " << a << _RESET_COLOR() << log4cpp::eol)

7.92.1.9 #define LOG(*a*) (TraceSystem::logStream() << a << log4cpp::eol)

7.92.1.10 #define LOG4CPP_FIX_ERROR_COLLISION 1

7.92.1.11 #define LOGC(*c, a*) (TraceSystem::logStream() << _SET_COLOR(c) << a << _RESET_COLOR() << log4cpp::eol)

7.92.1.12 #define TRACE(*a*) {}

7.93 version.h File Reference

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

Macros

- #define VERSION "0.2"

7.93.1 Macro Definition Documentation

7.93.1.1 #define VERSION "0.2"

Index