Dharma Software Development Kit

API Reference

Interactive Objects, Inc.

November 2001

Author: John Locke

Table of Contents

Dharma SDK Module Index

Dharma SDK Modules

Here is a list of all modules:

Dharma SDK Hierarchical Index

Dharma SDK Class Hierarchy

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

Dharma SDK Compound Index

Dharma SDK Compound List

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

Dharma SDK Module Documentation

Codec Manager

The codec manager is responsible for locating and allocating codecs in the system. It does this by querying the registry for available codecs, then generating a list of supported file types at startup. This list is then queried whenever a load by type is requested.

Compounds

- class **CCodecManager**
  The CCodecManager singleton.

Detailed Description

The codec manager is responsible for locating and allocating codecs in the system. It does this by querying the registry for available codecs, then generating a list of supported file types at startup. This list is then queried whenever a load by type is requested.

Codec Interface

Dadio(tm) relies on an abstract interface for dealing with arbitrary codecs. This interface is the **ICodec** (*p.112*) interface, as defined in ./player/codec/common/include/Codec.h. The purpose of this document is to define the **ICodec** (*p.112*) interface and each of its member functions, and to describe their expected behavior in the Dadio(tm) OS.

Compounds

- struct **CodecFunctionMap_s**
  Short table of static functions that a codec must expose. This is auto-generated by the registration interface.

- class **ICodec**
  Dadio(tm) relies on an abstract interface for dealing with arbitrary codecs. This interface is the ICodec interface, as defined in ./player/codec/common/include/Codec.h. The purpose of this document is to define the ICodec interface and each of its member functions, and to describe their expected behavior in the Dadio(tm) OS.

- struct **set_stream_event_data_s**
  Structure created when a song is successfully set in the media player. This is passed along through the EVENT_STREAM_SET event. The structure and the metadata should be deleted after the event is received (which is the default action if the event is passed to the play manager).

- struct **stream_info_s**
  Used to propagate information about the bitstream out of the codec to the system.

Defines

- #define **CODEC_TYPE_ID** 0x0a

Variables

- const ERESULT **CODEC_NO_ERROR**
- const ERESULT **CODEC_DRM_SUCCESS**
- const ERESULT **CODEC_NO_WORK**
- const ERESULT **CODEC_BAD_FORMAT**
- const ERESULT **CODEC_DECODE_ERROR**
- const ERESULT **CODEC_END_OF_FILE**
- const ERESULT **CODEC_FAIL**
- const ERESULT **CODEC_DRM_FAIL**
- **stream_info_s stream_info_t**
- **set_stream_event_data_s set_stream_event_data_t**
- **CodecFunctionMap_s CodecFunctionMap**

Detailed Description

Dadio(tm) relies on an abstract interface for dealing with arbitrary codecs. This interface is the **ICodec** (*p.112*) interface, as defined in ./player/codec/common/include/Codec.h. The purpose of this document is to define the **ICodec** (*p.112*) interface and each of its member functions, and to describe their expected behavior in the Dadio(tm) OS.

Define Documentation

#define CODEC_TYPE_ID 0x0a

Define the codec error zone for eresult error codes.

Variable Documentation

const ERESULT CODEC_BAD_FORMAT

The input stream is not in the expected format.

const ERESULT CODEC_DECODE_ERROR

The decoder had an error on the bitstream.

const ERESULT CODEC_DRM_FAIL

DRM Authentication failure.

const ERESULT CODEC_DRM_SUCCESS

DRM Authentication succeeded for the input stream.

const ERESULT CODEC_END_OF_FILE

An EOF was received from the input stream.

const ERESULT CODEC_FAIL

General purpose failure.

const ERESULT CODEC_NO_ERROR

No error was encountered during the operation.

const ERESULT CODEC_NO_WORK

The codec was asked to decode a frame, but either no input data is available, or no output space is available.

struct CodecFunctionMap_s CodecFunctionMap

Short table of static functions that a codec must expose. This is auto-generated by the registration interface.

struct set_stream_event_data_s set_stream_event_data_t

Structure created when a song is successfully set in the media player. This is passed along through the EVENT_STREAM_SET event. The structure and the metadata should be deleted after the event is received (which is the default action if the event is passed to the play manager).

struct stream_info_s stream_info_t

Used to propagate information about the bitstream out of the codec to the system.

Codec Registration

Registration interface.

Defines

- #define **DEFINE_CODEC**(codecname, codecID, can_probe, extensions...)
- #define **REGISTER_CODEC**(classname, codecID)

Detailed Description

Registration interface.

Define Documentation

#define DEFINE_CODEC(codecname, codecID, can_probe, extensions...)

Fill out the codec class definition with additional members necessary for codecs to work in Dadio(tm). Additionally, this satisfies the underlying requirements of the ident (**IIdentifiableObject** (*p.126*)) baseclass. This macro should be used you in your class declaration (header). The usage would be like:

```
DEFINE CODEC( "iObjects MP3 Codec", 783, true, "mp3" );
DEFINE_CODEC( "iObjects PCM Codec", 741, false, "raw", "pcm" );
```

□ codecname A string name for the codec

□ codecID A unique identifier for the codec

□ can_probe Either true or false indicating whether the system can probe bitstreams against this codec. In the case of PCM and similar pass through codecs, this should be false

□ extensions... A variable length list of strings that indicate which extensions are supported by this codec. This is case insensitive.

#define REGISTER_CODEC(classname, codecID)

Create a static initializer to register your codec with the system registry, and provide some initialized data structures to support generic instantiation. This macro should be used in your implementation file (source).

□ classname The name of your class (i.e. CMP3Codec)

□ codecID The unique identifier specified in DEFINE_CODEC.

Content Manager

Both media files and playlist files are tracked by a content manager in the Dadio(tm) system. The **IContentManager** (*p.114*) interface describes functions for simple addition and retrieval of records. The **IQueryableContentManager** (*p.140*) interface adds functions for creating lists of media records based on artist, album, and genre metadata.

Compounds

- struct **cm_key_value_record_t**
  Contains a key/value pair from the content manager.

- struct **content_record_update_s**
  Used for passing information about the media and playlist records available from a data source to the content manager.

- class **IContentManager**
  The IContentManager provides record tracking for media content accross several data sources.

- class **IContentRecord**

The base class for both media content records and playlist content records. The content record has an ID unique across the content manager, a URL unique across the system, and the ID of the data source it's stored on. Content records are also considered verified if they've been found to reside on a data source (as opposed to solely existing in the database from a previous run). They also have a status which specifies their ability to be run through a codec for playback.

- class **IMediaContentRecord**
  *A subclass of both **IContentRecord** (p.118) and **IMetadata** (p.130), used for representing tracks that can be decoded by the media player. In addition to the functionality of the **IContentRecord** (p.118) and **IMetadata** (p.130) classes, the media content record also retains a codec ID.*

- class **IPlaylistContentRecord**
  *A subclass of **IContentRecord** (p.118), used for representing playlist files that can can be loaded by the playlist format manager. In addition to the functionality of the **IContentRecord** (p.118) class, the playlist content record also retains a playlist format ID.*

- class **IQueryableContentManager**
  *This class extends the **IContentManager** (p.114) by providing functions for querying content based on artist, album, and genre keys.*

- struct **media_record_info_s**
  Used for passing information about a track available from the data source to the content manager (and back to the data source manager for metadata retrieval in a two-pass update).

- struct **playlist_record_s**
  Used for passing information about a playlist file available from the data source to the content manager.

Defines

- #define **CMK_ALL**

Variables

- **media_record_info_s media_record_info_t**
- **playlist_record_s playlist_record_t**
- **content_record_update_s content_record_update_t**

Detailed Description

Both media files and playlist files are tracked by a content manager in the Dadio(tm) system. The **IContentManager** (*p.114*) interface describes functions for simple addition and retrieval of records. The **IQueryableContentManager** (*p.140*) interface adds functions for creating lists of media records based on artist, album, and genre metadata.

Define Documentation

#define CMK_ALL

Requests that all values for the given key parameter be returned.

Variable Documentation

struct content_record_update_s content_record_update_t

Used for passing information about the media and playlist records available from a data source to the content manager.

struct media_record_info_s media_record_info_t

Used for passing information about a track available from the data source to the content manager (and back to the data source manager for metadata retrieval in a two-pass update).

struct playlist_record_s playlist_record_t

Used for passing information about a playlist file available from the data source to the content manager.

Metadata

The **IMetadata** (*p.130*) class is used by the Dadio(tm) system to associate a set of key/value pairs with a media item. **IMetadata** (*p.130*) records are created mainly by two sources: the data source during content update, and the media player when a track is set. The content manager provides its own **IMetadata** (*p.130*) subclass, and the media player can be told what type of **IMetadata** (*p.130*) subclass to produce.

Compounds

- class **CMetadataTable**
  The CMetadataTable class matches metadata attribute IDs to types. It's a singleton class that when created adds all the standard metadata types to its table. If client code wants to add a new metadata attribute, then it must pick an attribute ID not currently in use and add it to this table along with a type for the metadata.

- class **IMetadata**
  The metadata class is an abstract representation of a collection of key/value pairs for a track. Functions are provided for setting, getting, and clearing attributes.

Defines

- #define **METADATA_TYPE_ID** 0x0e
- #define **MDA_INVALID_ID** 0
- #define **MDA_TITLE**
- #define **MDA_ALBUM**
- #define **MDA_GENRE**
- #define **MDA_FILE_NAME**
- #define **MDA_FILE_SIZE**
- #define **MDA_ALBUM_TRACK_NUMBER**
- #define **MDA_COMMENT**
- #define **MDA_YEAR**
- #define **MDA_HAS_DRM**
- #define **MDA_DURATION**
- #define **MDA_SAMPLING_FREQUENCY**
- #define **MDA_CHANNELS**
- #define **MDA_BITRATE**
- #define **MDT_INVALID_TYPE** 0
- #define **MDT_INT**
- #define **MDT_TCHAR**

Variables

- const ERESULT **METADATA_NO_ERROR**
- const ERESULT **METADATA_ERROR**
- const ERESULT **METADATA_NOT_USED**
- const ERESULT **METADATA_NO_VALUE_SET**
- const ERESULT **METADATA_TABLE_INVALID_TYPE**
- const ERESULT **METADATA_TABLE_ID_IN_USE**

Detailed Description

The **IMetadata** (*p.130*) class is used by the Dadio(tm) system to associate a set of key/value pairs with a media item. **IMetadata** (*p.130*) records are created mainly by two sources: the data source during content update, and the media player when a track is set. The content manager provides its own **IMetadata** (*p.130*) subclass, and the media player can be told what type of **IMetadata** (*p.130*) subclass to produce.

Define Documentation

#define MDA_ALBUM

TCHAR. The album the track belongs to.

#define MDA_ALBUM_TRACK_NUMBER

int. The 1-based index of the track on the album.

#define MDA_BITRATE

int. Bitrate of the track, in bits per second.

#define MDA_CHANNELS

int. Number of channels.

#define MDA_COMMENT

TCHAR. Comments on the track stored in the codec.

#define MDA_DURATION

int. Duration of the track in seconds.

#define MDA_FILE_NAME

TCHAR. The track's file name in the data source.

#define MDA_FILE_SIZE

int. The size of the track in bytes.

#define MDA_GENRE

TCHAR. The genre of the track.

#define MDA_HAS_DRM

int. 1 = the track has drm, 0 = no drm.

#define MDA_INVALID_ID 0

0 is reserved for indicating an invalid metadata ID.

#define MDA_SAMPLING_FREQUENCY

int. Sampling frequency of the track in hz.

#define MDA_TITLE

TCHAR. The title of the track.

#define MDA_YEAR

int. Year of the track.

#define MDT_INT

Integer type

#define MDT_INVALID_TYPE 0

0 is reserved for indicating an invalid metadata type

#define MDT_TCHAR

TCHAR type

#define METADATA_TYPE_ID 0x0e

The metadata error zone for eresult error codes.

Variable Documentation

const ERESULT METADATA_ERROR

An unspecified error occurred during the operation.

const ERESULT METADATA_NOT_USED

The attribute is not used.

const ERESULT METADATA_NO_ERROR

No error was encountered during the operation.

const ERESULT METADATA_NO_VALUE_SET

The attribute is used but no value has been set.

const ERESULT METADATA_TABLE_ID_IN_USE

The ID for the metadata attribute is already in use.

const ERESULT METADATA_TABLE_INVALID_TYPE

The ID for the given attribute type is not valid.

Events

The following is a list of system events used by the SDK. Descriptions of the data parameter filled in by the GetEvent call and the behavior of the default event handler are included.

Defines

- #define **EVENT_KEY_PRESS**
- #define **EVENT_KEY_HOLD**
- #define **EVENT_KEY_RELEASE**
- #define **EVENT_STREAM_SET**
- #define **EVENT_STREAM_END**
- #define **EVENT_STREAM_FAIL**
- #define **EVENT_STREAM_PROGRESS**
- #define **EVENT_STREAM_PLAYING**
- #define **EVENT_STREAM_PAUSED**
- #define **EVENT_STREAM_STOPPED**
- #define **EVENT_CONTENT_UPDATE_BEGIN**
- #define **EVENT_CONTENT_UPDATE**
- #define **EVENT_CONTENT_UPDATE_END**
- #define **EVENT_CONTENT_UPDATE_ERROR**
- #define **EVENT_CONTENT_METADATA_UPDATE_BEGIN**
- #define **EVENT_CONTENT_METADATA_UPDATE**
- #define **EVENT_CONTENT_METADATA_UPDATE_END**
- #define **EVENT_MEDIA_REMOVED**
- #define **EVENT_MEDIA_INSERTED**

Detailed Description

The following is a list of system events used by the SDK. Descriptions of the data parameter filled in by the GetEvent call and the behavior of the default event handler are included.

Define Documentation

#define EVENT_CONTENT_METADATA_UPDATE

Sent from the data source during the second pass of a content update. The data parameter is a pointer to a content_record_update_t struct populated with content. The default event handler calls NotifyContentMetadataUpdate, which calls AddContentRecords in the content manager to add the new records. When finished it deletes the content_record_update_t struct.

#define EVENT_CONTENT_METADATA_UPDATE_BEGIN

Sent from the data source after the first pass of a content update (only if there's going to be a second pass). The data parameter is an integer that specifies the data source's instance ID. The default event handler tells the data source manager to send the EVENT_CONTENT_METADATA_UPDATE_END message when the second pass is completed.

#define EVENT_CONTENT_METADATA_UPDATE_END

Sent from the data source manager after the second pass of a content update is finished. The data parameter is an integer that specifies the data source's instance ID. The default event handler ignores this event.

#define EVENT_CONTENT_UPDATE

Sent from the data source during a content update when the number of records in the chunk size has been reached. The data parameter is a pointer to a content_record_update_t struct populated with content. The default event handler calls NotifyContentUpdate, which calls AddContentRecords in the content manager to add the new records, then either deletes the content_record_update_t struct or passes it back to the data source manager through the GetContentMetadata in a two-pass system.

#define EVENT_CONTENT_UPDATE_BEGIN

Sent from the data source when starting a content update. The data parameter is an integer that specifies the data source's instance ID. The default event handler calls MarkRecordsFromDataSourceUnverified in the content manager.

#define EVENT_CONTENT_UPDATE_END

Sent from the data source when a content update is completed. The data parameter is an integer that specifies the data source's instance ID. The default event handler calls DeleteUnverifiedRecordsFromDataSource in the content manager.

#define EVENT_CONTENT_UPDATE_ERROR

Sent from the data source when an error occurs during a content update. The data parameter is an integer that specifies the data source's instance ID. The default event handler calls DeleteUnverifiedRecordsFromDataSource in the content manager.

#define EVENT_KEY_HOLD

Sent from the keyboard driver when a key that's in the hold map is held down. The data parameter is the value of the key in the hold map. The default event handler ignores this event.

#define EVENT_KEY_PRESS

Sent from the keyboard driver when a key that's in the press map is pressed. The data parameter is the value of the key in the press map. The default event handler ignores this event.

#define EVENT_KEY_RELEASE

Sent from the keyboard driver when a key that's in the release map is released. The data parameter is the value of the key in the release map. The default event handler ignores this event.

#define EVENT_MEDIA_INSERTED

Sent from the data source when it detects its media (e.g., CD-ROM, CF card) has been inserted. The data parameter is an integer that specifies the data source's instance ID. The default event handler calls NotifyMediaInserted in the content manager, which tells the data source manager to begin a content update using the data source's default parameters.

#define EVENT_MEDIA_REMOVED

Sent from the data source when it detects its media (e.g., CD-ROM, CF card) has been removed. The data parameter is an integer that specifies the data source's instance ID. The default event handler calls NotifyContentMetadataUpdate in the content manager, which stops the track playing on the media player if its data source is the removed media, and then removes all of the content records for that data source from the play list.

#define EVENT_STREAM_END

Sent from the media player when the end of a song is reached during playback. The default event handler sets the next track and continues playback.

#define EVENT_STREAM_FAIL

Sent from the media player if playback is interrupted because of an error. The default event handler sets the next track and continues playback.

#define EVENT_STREAM_PAUSED

Sent from the media player when playback is paused. The default event handler ignores this event.

#define EVENT_STREAM_PLAYING

Sent from the media player when playback begins. The default event handler ignores this event.

#define EVENT_STREAM_PROGRESS

Sent from the media player during playback to notify the interface of the current track time. The data parameter is an integer that specifies the track time in seconds. The default event handler ignores this event.

#define EVENT_STREAM_SET

Sent from the media player when an entry is successfully set. The data parameter is a pointer to a set_stream_event_data_t struct. The default event handler simply deletes the pMediaPlayerMetadata pointer, then deletes the set_stream_event_data_t itself. Thus, it's easiest to copy whatever metadata information needed for display before passing the event to the default handler for cleanup.

#define EVENT_STREAM_STOPPED

Sent from the media player when playback is stopped. The default event handler ignores this event.

Media Player

Dadio(tm) uses the Media Player as a central control for audio playback and output. The Media Player is responsible for opening new tracks, decoding the bitstream, and providing track progress. Events are generated from the media player for state changes (such as stream end) and track progress. Much of the functionality provided by the media player is wrapped by the play manager, allowing the play manager to coordinate operations that require multiple modules.

Compounds

- class **CMediaPlayer**
  *CMediaPlayer is the basic interface for the media player. This is a singleton class that can be referenced through the **GetInstance**() (p.67) routine.*

- struct **playstream_settings_s**
  The sequence of codecs->filters->outputstream is considered to be the playstream. Currently, the playstream_settings_t structure allows the user to specify a list of filters to put in the playstream and an outputstream to use for playback.

Defines

- #define **MEDIAPLAYER_ERROR_ZONE** 0x0c

Typedefs

- typedef **IMetadata * FNCreateMetadata** ()

Variables

- const ERESULT **MP_NO_ERROR**
- const ERESULT **MP_ERROR**
- const ERESULT **MP_NOT_CONFIGURED**
- **playstream_settings_s playstream_settings_t**

Detailed Description

Dadio(tm) uses the Media Player as a central control for audio playback and output. The Media Player is responsible for opening new tracks, decoding the bitstream, and providing track progress. Events are generated from the media player for state changes (such as stream end) and track progress. Much of the functionality provided by the media player is wrapped by the play manager, allowing the play manager to coordinate operations that require multiple modules.

Define Documentation

#define MEDIAPLAYER_ERROR_ZONE 0x0c

Define the error zone for media player eresult codes.

Typedef Documentation

typedef IMetadata FNCreateMetadata()*

Prototype for a function the mediaplayer can call to create an appropriate **IMetadata** (*p.130*) object.

Variable Documentation

const ERESULT MP_ERROR

A generic failure occurred.

const ERESULT MP_NOT_CONFIGURED

A play control was issued on the media player prior to configuration.

const ERESULT MP_NO_ERROR

No error was encountered during the operation.

struct playstream_settings_s playstream_settings_t

The sequence of codecs->filters->outputstream is considered to be the playstream. Currently, the playstream_settings_t structure allows the user to specify a list of filters to put in the playstream and an outputstream to use for playback.

Play Manager

The play manager singleton is the nerve center of the Dadio(tm) system and is the usual point of contact between the system and application-specific client code.

Compounds

- class **CPlayManager**
  The play manager is the nerve center of the system. It maintains pointers to the content manager and current playlist. It has a default event handler. It exposes functions for play control. It's the object that kicks off content updates.

Defines

- #define **PLAYMANAGER_ERROR_ZONE** 0xac

Variables

- const ERESULT **PM_NO_ERROR**
- const ERESULT **PM_ERROR**
- const ERESULT **PM_NO_GOOD_TRACKS**
- const ERESULT **PM_PLAYLIST_END**
- const ERESULT **PM_PLAYING**

Detailed Description

The play manager singleton is the nerve center of the Dadio(tm) system and is the usual point of contact between the system and application-specific client code.

Define Documentation

#define PLAYMANAGER_ERROR_ZONE 0xac

Define the play manager error zone for ERESULT error codes.

Variable Documentation

const ERESULT PM_ERROR

An unspecified error occured.

const ERESULT PM_NO_ERROR

No error was encountered during the operation.

const ERESULT PM_NO_GOOD_TRACKS

All tracks in the playlist are bad. Returned from a call to NextTrack, PreviousTrack, or a stream end event.

const ERESULT PM_PLAYING

The next/previous track was set and is playing. Returned from a call to NextTrack, PreviousTrack, or a stream end event. If the functions return PM_NO_ERROR, then the next/previous track was set successfully but is not playing back.

const ERESULT PM_PLAYLIST_END

No next/previous track. Returned from a call to NextTrack, PreviousTrack, or a stream end event.

Filter Interface

Dadio(tm) uses an asbtract interface for filters, which allow components to examine and modify the current PCM audio stream. This is the **IFilter** (*p.124*) interface, as defined in ./player/datastream/filter/include/Filter.h. The purpose of this document is to define the **IFilter** (*p.124*) interface and each of its member functions, and to describe their expected behavior in the Dadio(tm) OS.

Compounds

- struct **IFilter**
  Dadio(tm) uses an asbtract interface for filters, which allow components to examine and modify the current PCM audio stream. This is the IFilter interface, as defined in ./player/datastream/filter/include/Filter.h. The purpose of this document is to define the IFilter

interface and each of its member functions, and to describe their expected behavior in the Dadio(tm) OS.

Defines

- #define **FILTER_TYPE_ID** 0x0b

Variables

- const ERESULT **FILTER_NO_ERROR**
- const ERESULT **FILTER_NO_WORK**
- const ERESULT **FILTER_EOF**
- const ERESULT **FILTER_FAIL**

Detailed Description

Dadio(tm) uses an asbtract interface for filters, which allow components to examine and modify the current PCM audio stream. This is the **IFilter** (*p.124*) interface, as defined in ./player/datastream/filter/include/Filter.h. The purpose of this document is to define the **IFilter** (*p.124*) interface and each of its member functions, and to describe their expected behavior in the Dadio(tm) OS.

Define Documentation

#define FILTER_TYPE_ID 0x0b

Define the filter error zone for eresult error codes.

Variable Documentation

const ERESULT FILTER_EOF

An EOF was received from the input buffer.

const ERESULT FILTER_FAIL

A general error was encountered.

const ERESULT FILTER_NO_ERROR

No error was encountered during the operation.

const ERESULT FILTER_NO_WORK

No work can be processed; that is, either the output buffer is full, or the input buffer is empty.

Filter Registration

Registration interface.

Defines

- #define **DEFINE_FILTER**(filtername, filterID)
- #define **REGISTER_FILTER**(classname, filterID)

Detailed Description

Registration interface.

Define Documentation

#define DEFINE_FILTER(filtername, filterID)

Fill out the filter class definition with additional members necessary for filters to work in Dadio(tm). Additionally, this satisfies the underlying requirements of the **IIdentifiableObject** (*p.126*) baseclass. This macro should be used in your definition (header). The usage would be like:

```
DEFINE_FILTER( "iObjects sample rate converter", 685 );
DEFINE_FILTER( "iObjects disk writer", 691 );
```

- filtername A string name for the filter
- filterID A unique identifier for the filter.

#define REGISTER_FILTER(classname, filterID)

Create a static initializer to register your filter with the system registry. This macro should be used in your implementation (source).

- classname The name of your class (i.e. CSRCFilter)
- filterID The unique identifier specified in DEFINE_FILTER.

Input Streams

InputStreams provide an abstract way for Dadio(tm) to read in data. They are used in various parts of the system, and are required for decoding. Several precompiled input streams are provided for use. Input streams provide an optional registration interface also.

Compounds

- struct **IInputStream**
  The IInputStream interface defines the basic interface for read only stream access. Additionally, support is provided for seekable streams.

Defines

- #define **INPUT_TYPE_ID** 0x0c
- #define **DEFINE_INPUTSTREAM**(isname, isID)
- #define **REGISTER_INPUTSTREAM**(classname, isID)

Variables

- const ERESULT **INPUTSTREAM_NO_ERROR**
- const ERESULT **INPUTSTREAM_ERROR**

Detailed Description

InputStreams provide an abstract way for Dadio(tm) to read in data. They are used in various parts of the system, and are required for decoding. Several precompiled input streams are provided for use. Input streams provide an optional registration interface also.

Define Documentation

#define DEFINE_INPUTSTREAM(isname, isID)

Registration for input streams is optional. If you elect not to register your input streams, you still must use the **IDENTIFY_OBJECT**() (*p.36*) macro in your class definition (header) in order to satisfy the requirements of the **IIdentifiableObject** (*p.126*) base class. Otherwise, this macro is used in your definition to complete your input stream definition.

- □ isname String name for your input stream.
- □ isID Unique identifier for this input stream.

#define INPUT_TYPE_ID 0x0c

The error zone for input stream error codes.

#define REGISTER_INPUTSTREAM(classname, isID)

Create a static initializer to register your input stream with the system registry. This macro should be used in your implementation (source) file.

- □ classname The name of your input stream class (i.e. CMemInputStream)
- □ isID The unique identifier used in DEFINE_INPUTSTREAM.

Variable Documentation

const ERESULT INPUTSTREAM_ERROR

A generic error was encountered during the operation.

const ERESULT INPUTSTREAM_NO_ERROR

No error was encountered during the operation.

Output Streams

OutputStreams provide an abstract way for Dadio(tm) components to output data. The most basic example of an output stream is WaveOut, which writes pcm data to the audio driver. A handful of precompiled output streams are provided with Dadio(tm).

Compounds

- struct **IOutputStream**
  Interface for output streams. If you elect to not use the registration interface for output streams, you still must use IDENTIFY_OBJECT in your class definition.

Defines

- #define **OUTPUT_TYPE_ID** 0x0d
- #define **DEFINE_OUTPUTSTREAM**(outname, outID)
- #define **REGISTER_OUTPUTSTREAM**(classname, outID)

Variables

- const ERESULT **OUTPUTSTREAM_NO_ERROR**
- const ERESULT **OUTPUTSTREAM_ERROR**

Detailed Description

OutputStreams provide an abstract way for Dadio(tm) components to output data. The most basic example of an output stream is WaveOut, which writes pcm data to the audio driver. A handful of precompiled output streams are provided with Dadio(tm).

Define Documentation

#define DEFINE_OUTPUTSTREAM(outname, outID)

Fill out the output stream class definition with additional members required for output stream registration and object identification. This should be used in your definition (header). The usage would be like:

```
DEFINE_OUTPUTSTREAM( "iObjects wave output", 937 )
```

- outname A string name for your output stream
- outID A unique identifier for your output stream.

#define OUTPUT_TYPE_ID 0x0d

The error zone for output stream error codes.

#define REGISTER_OUTPUTSTREAM(classname, outID)

Create a static initializer to register your output stream with the system registry. This should be used in your implementation

- classname The name of your class (i.e. CWaveOutput)
- outID The unique identifer used in your DEFINE_OUTPUTSTREAM call.

Variable Documentation

const ERESULT OUTPUTSTREAM_ERROR

A generic error was encountered during the operation.

const ERESULT OUTPUTSTREAM_NO_ERROR

No error was encountered during the operation.

Wave Output

Compounds

- class **CWaveOutStream**
  PCM is written to the audio driver through the CWaveOutStream class. This class does not perform any operations on the audio data, and assumes that any necessary sample rate conversion has already been applied.

Defines

- #define **WAVEOUT_KEY**
- #define **KEY_WAVEOUT_SET_SAMPLERATE**

Define Documentation

#define KEY_WAVEOUT_SET_SAMPLERATE

The ioctl key to configure hardware sample rate conversion, if available.

Parameters:

Value Should be a pointer to an unsigned int with the sample frequency of the input data.

#define WAVEOUT_KEY

The key used to uniquely identify the wave output stream.

Commercial IR

Compounds

- class **CIR**
  CIR is the singleton IR driver object.

- struct **ir_map_s**
  The ir_map_t structure specifies what events the IR driver should generate for various button presses it receives. This allows an application to mask changes in the physical button layout of a remote by simply shuffling the order of the button mapping. Additionally it provides configuration for repeat events. The IR driver does not support release events.

Defines

- #define **IR_REPEAT_ENABLE** 0x01

Variables

- **ir_map_s ir_map_t**

Define Documentation

#define IR_REPEAT_ENABLE 0x01

This flag enables button repeat events from the ir driver.

Variable Documentation

struct ir_map_s ir_map_t

The ir_map_t structure specifies what events the IR driver should generate for various button presses it receives. This allows an application to mask changes in the physical button layout of a remote by simply shuffling the order of the button mapping. Additionally it provides configuration for repeat events. The IR driver does not support release events.

Keyboard driver

The keyboard driver provides generic key support for the Cirrus logic 7312 processor. Alternate hardware button matrices can be supported by adjusting the configuration in the key_map_t structure, avoiding any need to rework code for physical layout changes.

Compounds

- class **CKeyboard**
  The keyboard driver processes keyboard interrupts, scans the keyboard matrix, and generates events based on the current keymap. It allows the user application to mask and unmask event generation for software key locks.

- struct **key_map_s**
  The key_map_t structure specifies what events the keyboard driver should trigger based on the key input. It also allows the application to specify the keyboard layout and customize keyboard driver details such as repeat delay and rate. press_map, hold_map, and release_map should point to event arrays of size num_buttons.

Defines

- #define **KEY_REPEAT_ENABLE** 0x01

Variables

- **key_map_s key_map_t**

Detailed Description

The keyboard driver provides generic key support for the Cirrus logic 7312 processor. Alternate hardware button matrices can be supported by adjusting the configuration in the key_map_t structure, avoiding any need to rework code for physical layout changes.

Define Documentation

#define KEY_REPEAT_ENABLE 0x01

This flag enables key repeat events from the keyboard driver.

Variable Documentation

struct key_map_s key_map_t

The key_map_t structure specifies what events the keyboard driver should trigger based on the key input. It also allows the application to specify the keyboard layout and customize keyboard driver details such as repeat delay and rate. press_map, hold_map, and release_map should point to event arrays of size num_buttons.

Playlist Format

Dadio(tm) uses a registration interface to maintain a list of playlist formats it can read and write. New playlist formats can be added by using the macro REGISTER_PLAYLIST_FORMAT and providing functions for loading and saving playlist to and from streams.

Compounds

- class **CPlaylistFormatManager**
  The playlist format manager keeps track of the playlist file formats available in the system and the functions for loading and saving them.

Defines

- #define **PLAYLIST_FORMAT_TYPE_ID** 0x06
- #define **REGISTER_PLAYLIST_FORMAT**(formatnamestring, playlistformatID, loadplaylist, saveplaylist, extensions...)

Typedefs

- typedef ERESULT **FNLoadPlaylist** (const char *szURL, **IPlaylist** *pPlaylist, bool bVerifyContent)
- typedef ERESULT **FNSavePlaylist** (const char *szURL, **IPlaylist** *pPlaylist)

Variables

- const ERESULT **PLAYLIST_FORMAT_NO_ERROR**
- const ERESULT **PLAYLIST_FORMAT_FILE_OPEN_ERROR**
- const ERESULT **PLAYLIST_FORMAT_UNKNOWN_FORMAT**
- const ERESULT **PLAYLIST_FORMAT_BAD_FORMAT**
- const ERESULT **PLAYLIST_FORMAT_READ_ERROR**
- const ERESULT **PLAYLIST_FORMAT_WRITE_ERROR**
- const ERESULT **PLAYLIST_FORMAT_BAD_URL**

Detailed Description

Dadio(tm) uses a registration interface to maintain a list of playlist formats it can read and write. New playlist formats can be added by using the macro REGISTER_PLAYLIST_FORMAT and providing functions for loading and saving playlist to and from streams.

Define Documentation

#define PLAYLIST_FORMAT_TYPE_ID 0x06

Define the playlist format error zone for ERESULT error codes.

#define REGISTER_PLAYLIST_FORMAT(formatnamestring, playlistformatID, loadplaylist, saveplaylist, extensions...)

Fill out the playlist format class definition with additional members necessary for playlist formats to work in Dadio(tm). Additionally, this satisfies the underlying requirements of the ident (**IIdentifiableObject** (*p.126*)) baseclass. The usage would be like:

```
REGISTER_PLAYLIST_FORMAT( "Winamp M3U playlist format", 97, LoadM3UPlaylist,
SaveM3UPlaylist, "m3u" );
REGISTER_PLAYLIST_FORMAT( "Dadio playlist format", 98, LoadDPLPlaylist,
SaveDPLPlaylist, "dpl" );
```

- formatnamestring A string name for the playlist format
- playlistformatID A unique non-zero identifier for the playlist format
- loadplaylist The function used to load this playlist format
- saveplaylist The function used to save this playlist format
- extensions... A variable length list of strings that indicate which extensions are supported by this playlist format. This is case insensitive.

Typedef Documentation

typedef ERESULT FNLoadPlaylist(const char szURL, IPlaylist* pPlaylist, bool bVerifyContent)*

The function prototype for loading playlists.

Parameters:

szURL The URL of the file to open for reading.
pPlaylist A playlist object to add entries to.
bVerifyContent If true, then entries found in the playlist file should be checked to see if they exist in the content manager. If a playlist entry has no matching record in the content manager, then it should not be added to the playlist. If *bVerifyContent* is false, then all entries should be added to the content manager as well as the playlist.

typedef ERESULT FNSavePlaylist(const char szURL, IPlaylist* pPlaylist)*

The function prototype for saving playlists.

Parameters:

szURL The URL of the file to open for writing.

pPlaylist The playlist whose entries are to be written to file.

Variable Documentation

const ERESULT PLAYLIST_FORMAT_BAD_FORMAT

An error occurred when loading the playlist file.

const ERESULT PLAYLIST_FORMAT_BAD_URL

No corresponding data source was found for the URL.

const ERESULT PLAYLIST_FORMAT_FILE_OPEN_ERROR

An error occurred opening the file.

const ERESULT PLAYLIST_FORMAT_NO_ERROR

No error was encountered during the operation.

const ERESULT PLAYLIST_FORMAT_READ_ERROR

An error occurred reading the file.

const ERESULT PLAYLIST_FORMAT_UNKNOWN_FORMAT

No matching playlist format was found.

const ERESULT PLAYLIST_FORMAT_WRITE_ERROR

An error occurred writing the file.

Object Identification

Dadio(tm) uses a system wide registry to allow objects and data to be found and allocated by an ID instead of by key. Classes that need to be accessible through this registry have a common base, in that they are all uniquely identifiable. All Dadio(tm) classes that are accessible through the registry derive from IdentifiableObject.

Compounds

- struct **IIdentifiableObject**
  The IIdentifiableObject interface. The methods defined by this interface do not need to be implemented directly ever.

Defines

- #define **ASSEMBLE_ID**(typeID, classID)
- #define **IDENTIFY_OBJECT**(typeID, classID)

Detailed Description

Dadio(tm) uses a system wide registry to allow objects and data to be found and allocated by an ID instead of by key. Classes that need to be accessible through this registry have a common base, in that they are all uniquely identifiable. All Dadio(tm) classes that are accessible through the registry derive from IdentifiableObject.

Define Documentation

#define ASSEMBLE_ID(typeID, classID)

Macro that correctly assembles an object ID given a type id and a class id

- typeID The type ID
- classID The class ID.

#define IDENTIFY_OBJECT(typeID, classID)

Macro that expands to correct implementations of the **IIdentifiableObject** (*p.126*) routines. This macro is to be used in your definition in preference to directly implementing the routines.\arg typeID The type ID

- classID The class ID.

tchar

The TCHAR type is a double-byte character suitable for Unicode strings. This module contains functions for manipulating TCHAR strings.

Functions

- TCHAR * **tstrcat** (TCHAR *strDestination, const TCHAR *strSource)
- TCHAR * **tstrncat** (TCHAR *strDestination, const TCHAR *strSource, int count)
- int **tstrcmp** (const TCHAR *string1, const TCHAR *string2)
- int **tstrncmp** (const TCHAR *string1, const TCHAR *string2, int count)
- TCHAR * **tstrcpy** (TCHAR *strDestination, const TCHAR *strSource)
- TCHAR * **tstrncpy** (TCHAR *strDestination, const TCHAR *strSource, int count)
- int **tstrlen** (const TCHAR *string)
- TCHAR * **CharToTchar** (TCHAR *strDestination, const char *strSource)
- TCHAR * **CharToTcharN** (TCHAR *strDestination, const char *strSource, int count)
- char * **TcharToChar** (char *strDestination, const TCHAR *strSource)
- char * **TcharToCharN** (char *strDestination, const TCHAR *strSource, int count)
- TCHAR * **tstrdup** (const TCHAR *string)

Detailed Description

The TCHAR type is a double-byte character suitable for Unicode strings. This module contains functions for manipulating TCHAR strings.

Function Documentation

TCHAR CharToTchar (TCHAR ** strDestination*, const char ** strSource*)*

Converts the source single-byte string into a double-byte string. Returns a pointer to *strDestination*, the converted string.

TCHAR CharToTcharN (TCHAR ** strDestination*, const char ** strSource*, int* count*)*

Converts the specified number of characters from the source single-byte string into a double-byte string. Returns a pointer to *strDestination*, the converted string.

char TcharToChar (char ** strDestination*, const TCHAR ** strSource*)*

Converts the source double-byte string into a single-byte string. Returns a pointer to *strDestination*, the converted string.

char TcharToCharN (char ** strDestination*, const TCHAR ** strSource*, int* count*)*

Converts the specified number of characters from the source double-byte string into a single-byte string. Returns a pointer to *strDestination*, the converted string.

TCHAR tstrcat (TCHAR ** strDestination*, const TCHAR ** strSource*)*

Appends the source string to the destination string. Both strings must be null-terminated. Returns a pointer to *strDestination*, the concatenated string.

*int tstrcmp (const TCHAR ** string1*, const TCHAR ** string2*)*

Compares the two strings.

Return values:

-1 If *string1 < string2*
0 If *string1 == string2*
1 if *string1 > string2*.

TCHAR tstrcpy (TCHAR ** strDestination*, const TCHAR ** strSource*)*

Copies the source string to the destination string. Returns a pointer to *strDestination*, the copied string.

TCHAR tstrdup (const TCHAR ** string*)*

Duplicates a string. Returns a pointer to the newly-allocated copy of the string.

*int tstrlen (const TCHAR ** string*)*

Returns the length of the null-terminated string passed in.

TCHAR tstrncat (TCHAR ** strDestination*, const TCHAR ** strSource*, int* count*)*

Appends the specified number of characters from the source string to the destination string. The destination string must be null-terminated. After concatenation the destination string will be null-terminated, no matter if the source string is shorter or longer than the number of characters appended.

*int tstrncmp (const TCHAR ** string1*, const TCHAR ** string2*, int* count*)*

Compares the first *count* characters of the two strings.

Return values:

-1 If *string1 < string2*
0 If *string1 == string2*
1 if *string1 > string2*.

TCHAR tstrncpy (TCHAR ** strDestination*, const TCHAR ** strSource*, int* count*)*

Copies the specified number of characters from the source string to the destination string. Returns a pointer to *strDestination*, the copied string.

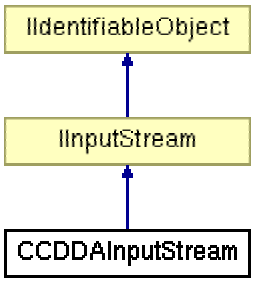
Dharma SDK Class Documentation

CCDDAInputStream Class Reference

Dadio(tm) provides a simple way to access CD audio tracks through the CCDDAInputStream. The location of tracks is determined by the data read back from the CD table of contents (TOC), and this input stream simply performs raw audio reads from the drive.

`#include <CDDAInputStream.h>`

Inheritance diagram for CCDDAInputStream:



Public Methods

- ERESULT **Open** (const char *Source)
- ERESULT **Open** (**CCDDataSource** *pDataSource, int iLBAStart, int iLBALength)

Detailed Description

Dadio(tm) provides a simple way to access CD audio tracks through the CCDDAInputStream. The location of tracks is determined by the data read back from the CD table of contents (TOC), and this input stream simply performs raw audio reads from the drive.

Member Function Documentation

ERESULT CCDDAInputStream::Open (CCDDataSource * **pDataSource**, *int* **iLBAStart**, *int* **iLBALength**)

Open a stream using the specified LBA range on the given CD data source. This replaces the default **IInputStream::Open** (*p.128*) call.

ERESULT CCDDAInputStream::Open (const char * **Source**) `[virtual]`

The default open routine - this returns an error, since more information must be provided about the item being opened.

Reimplemented from **IInputStream** (*p.128*).

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

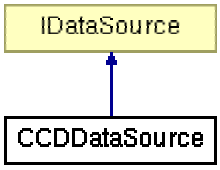
- **CDDAInputStream.h**

CCDDataSource Class Reference

Implements the **IDataSource** (*p.120*) class for CD drives.

`#include <CDDataSource.h>`

Inheritance diagram for CCDDataSource:



Public Methods

- Cyg_ErrNo **GetMediaStatus** (bool bSendEvent)
- void **SetDefaultRefreshMode** (**RefreshMode** mode)
- **RefreshMode GetDefaultRefreshMode** () const
- void **SetDefaultUpdateChunkSize** (int iUpdateChunkSize)
- int **GetDefaultUpdateChunkSize** () const
- bool **GetRootURLPrefix** (char *szRootURLPrefix, int iMaxLength) const
- ERESULT **ListAllEntries** (**RefreshMode** mode, int iUpdateChunkSize)
- void **GetContentMetadata** (**content_record_update_t** *pContentUpdate)
- **IInputStream** * **OpenInputStream** (const char *szURL)
- **IOutputStream** * **OpenOutputStream** (const char *szURL)
- long **Read** (long lba, long sectors, void *buffer)

Static Public Methods

- CCDDataSource * **Open** (const char *szDeviceName, const char *szMountDirectory)

Detailed Description

Implements the **IDataSource** (*p.120*) class for CD drives.

Member Function Documentation

*void CCDDataSource::GetContentMetadata (content_record_update_t * pContentUpdate) [virtual]*

Retrieves metadata for each media content record in the passed-in list.

Reimplemented from **IDataSource** (*p.121*).

RefreshMode CCDDataSource::GetDefaultRefreshMode () const `[virtual]`

Returns the current default refresh mode of the data source.

Reimplemented from **IDataSource** (*p.121*).

int CCDDataSource::GetDefaultUpdateChunkSize () const `[virtual]`

Returns the default update chunk size of the data source This value is the maximum number of records that will be sent for each EVENT_CONTENT_UPDATE message. 0 indicates that all records will be sent in one single message.

Reimplemented from **IDataSource** (*p.121*).

Cyg_ErrNo CCDDataSource::GetMediaStatus (bool **bSendEvent***)*

Queries the media status to see if a disc has been removed or inserted.

*bool CCDDataSource::GetRootURLPrefix (char ** **szRootURLPrefix***, int* **iMaxLength***) const* `[virtual]`

Copies the string the data source uses to prefix its URLs into the given string provided.

Reimplemented from **IDataSource** (*p.122*).

ERESULT CCDDataSource::ListAllEntries (RefreshMode **mode***, int* **iUpdateChunkSize***)* `[virtual]`

Asks the source to pass content updates lists through the event system that contain all of the content it can access. If bGetMetadata is true, then each entry is opened and passed to the matching codec for metadata retrieval. iUpdateChunkSize specifies how many media records to send back at a time. If iUpdateChunkSize is zero, then all records are sent back at once.

Reimplemented from **IDataSource** (*p.122*).

CCDDataSource CCDDataSource::Open (const char ** **szDeviceName***, const char ** **szMountDirectory***)* `[static]`

Attempts to open the drive with the given device name. Returns a pointer to a new CCDataSource object if successful, 0 otherwise.

IInputStream CCDDataSource::OpenInputStream (const char ** **szURL***)* `[virtual]`

Asks the source to open this URL for reading. Returns 0 if the URL was unable to be opened, otherwise it returns the proper subclass of **IInputStream** (*p.127*) for this file type.

Reimplemented from **IDataSource** (*p.122*).

IOutputStream CCDDataSource::OpenOutputStream (const char ** **szURL***)* `[inline, virtual]`

Asks the source to open this URL for writing. Returns 0 if the URL was unable to be opened, otherwise it returns the proper subclass of **IOutputStream** (*p.132*) for this file type.

Reimplemented from **IDataSource** (*p.122*).

long CCDDataSource::Read (long **lba***, long* **sectors***, void ** **buffer***)*

Reads data from the CD.

Parameters:

lba The logical block address to start reading from.
sectors The number of sectors to read.
buffer A pointer to a pre-allocated buffer to receive the data.

Return values:

>=0 The number of sectors read.
<0 An error code.

void CCDDataSource::SetDefaultRefreshMode (RefreshMode mode*) [virtual]*

Sets the default refresh mode of the data source. If DSR_DEFAULT is passed in, then DSR_ONE_PASS_WITH_METADATA is used.

Reimplemented from **IDataSource** (*p.122*).

void CCDDataSource::SetDefaultUpdateChunkSize (int iUpdateChunkSize*) [virtual]*

Sets the default update chunk size of the data source This value is the maximum number of records that should be sent for each EVENT_CONTENT_UPDATE message. 0 indicates that all records should be sent in one single message. If DS_DEFAULT_CHUNK_SIZE is passed in, then 0 is used.

Reimplemented from **IDataSource** (*p.122*).

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

- **CDDataSource.h**

CCodecManager Class Reference

The CCodecManager singleton.

`#include <CodecManager.h>`

Public Methods

- **ICodec * FindCodec** (unsigned int iCodecID)
- **ICodec * FindCodec** (const char *szExtension)
- RegKey **FindCodecID** (const char *szExtension)
- **ICodec * TryCodec** (int iCodecIndex)

Static Public Methods

- CCodecManager * **GetInstance** ()

Detailed Description

The CCodecManager singleton.

Member Function Documentation

ICodec CCodecManager::FindCodec (const char * szExtension)*

Given a file extension, find a suitable codec, allocate it, and return a pointer to the object, or NULL if no suitable codec was found.

ICodec CCodecManager::FindCodec (unsigned int iCodecID)*

Given a specific CodecID, find the codec, allocate it, and return a pointer to the newly allocated object.

*RegKey CCodecManager::FindCodecID (const char * szExtension)*

Rather than allocate the codec, simply see if one exists that claims to support this file type. This allows the codec ID to be associated with a content record, so when a track is actually loaded the codec manager doesn't have to perform a search for the appropriate codec.x.

CCodecManager CCodecManager::GetInstance () `[static]`*

Returns a pointer to the singleton codec manager.

ICodec CCodecManager::TryCodec (int iCodecIndex)*

Probe through available codecs looking for one that will support the given input stream.

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

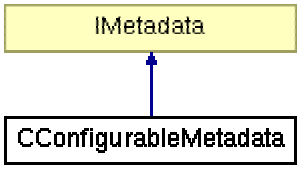
- **CodecManager.h**

CConfigurableMetadata Class Reference

The configurable metadata class allows clients to specify which attributes should be stored by this class. Attributes added to the class are added to all instances of the CConfigurableMetadata.

`#include <ConfigurableMetadata.h>`

Inheritance diagram for CConfigurableMetadata:



Static Public Methods

- void **AddAttribute** (int iAttributeID)
- void **RemoveAllAttributes** ()

Detailed Description

The configurable metadata class allows clients to specify which attributes should be stored by this class. Attributes added to the class are added to all instances of the CConfigurableMetadata.

Member Function Documentation

void CConfigurableMetadata::AddAttribute (int iAttributeID*)* `[static]`

Add a metadata type to be used by all CConfigurableMetadata objects.

void CConfigurableMetadata::RemoveAllAttributes () `[static]`

Removes all metadata types from the list.

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

- **ConfigurableMetadata.h**

CDataSourceManager Class Reference

The data source manager keeps a list of all data sources added to the system. It has a worker thread responsible for invoking content and metadata updates on the data sources.

`#include <DataSourceManager.h>`

Public Methods

- void **AddDataSource** (**IDataSource** *pDataSource)
- void **RemoveDataSource** (**IDataSource** *pDataSource)
- int **GetDataSourceCount** ()
- **IDataSource** * **GetDataSourceByID** (int iDataSourceID)
- **IDataSource** * **GetDataSourceByIndex** (int index)
- **IDataSource** * **GetDataSourceByClassID** (int iDataSourceClassID, int index)
- **IDataSource** * **GetDataSourceByURL** (const char *szURL)
- **IInputStream** * **OpenInputStream** (**IContentRecord** *pRecord)
- **IInputStream** * **OpenInputStream** (const char *szURL)
- **IOutputStream** * **OpenOutputStream** (**IContentRecord** *pRecord)
- **IOutputStream** * **OpenOutputStream** (const char *szURL)
- void **RefreshContent** (int iDataSourceID, **IDataSource::RefreshMode** mode, int iUpdateChunkSize)
- void **GetContentMetadata** (**content_record_update_t** *pContentUpdate)
- void **NotifyContentMetadataUpdateEnd** (int iDataSourceID)

Static Public Methods

- CDataSourceManager * **GetInstance** ()
- void **Destroy** ()

Detailed Description

The data source manager keeps a list of all data sources added to the system. It has a worker thread responsible for invoking content and metadata updates on the data sources.

Member Function Documentation

*void CDataSourceManager::AddDataSource (IDataSource * pDataSource)*

Adds a data source to the list.

void CDataSourceManager::Destroy () [static]

Destroy the singleton global data source manager.

*void CDataSourceManager::GetContentMetadata (content_record_update_t * pContentUpdate)*

Given a list of content records, this passes them to the data source for metadata retrieval.

IDataSource CDataSourceManager::GetDataSourceByClassID (int iDataSourceClassID, int index)*

Returns a pointer to the data source of the given type using a zero-based index. Returns 0 if no matching data source was found.

IDataSource CDataSourceManager::GetDataSourceByID (int iDataSourceID)*

Returns a pointer to a data source with the given instance ID. Returns 0 if no matching data source was found.

IDataSource CDataSourceManager::GetDataSourceByIndex (int index)*

Returns a pointer to a data source at the given zero-based index. Returns 0 if the index is out-of-bounds.

IDataSource CDataSourceManager::GetDataSourceByURL (const char * szURL)*

Compares each data source's URL prefix to the given URL. Returns a pointer to the data source that matches the URL prefix. Returns 0 if no matching data source was found.

int CDataSourceManager::GetDataSourceCount ()

Returns the number of data sources in the system.

CDataSourceManager CDataSourceManager::GetInstance () [static]*

Returns a pointer to the global data source manager.

void CDataSourceManager::NotifyContentMetadataUpdateEnd (int iDataSourceID)

Used to send an EVENT_CONTENT_METADATA_UPDATE_END message.

IInputStream CDataSourceManager::OpenInputStream (const char * szURL)*

Tells the data source manager to find the data source associated with this URL and asks the source to open this record for reading. Returns 0 if the URL was unable to be opened, otherwise it returns the proper subclass of **IInputStream** (*p.127*) for this file type.

IInputStream CDataSourceManager::OpenInputStream (IContentRecord * pRecord)*

Tells the data source manager to find the data source associated with this content record and asks the source to open this record for reading. Returns 0 if the record was unable to be opened, otherwise it returns the proper subclass of **IInputStream** (*p.127*) for this file type.

IOutputStream CDataSourceManager::OpenOutputStream (const char * szURL)*

Tells the data source manager to find the data source associated with this content record and asks the source to open this record for writing. Returns 0 if the record was unable to be opened, otherwise it returns the proper subclass of **IOutputStream** (*p.132*) for this file type.

IOutputStream CDataSourceManager::OpenOutputStream (IContentRecord * pRecord)*

Tells the data source manager to find the data source associated with this content record and asks the source to open this record for writing. Returns 0 if the record was unable to be opened, otherwise it returns the proper subclass of **IOutputStream** (*p.132*) for this file type.

void CDataSourceManager::RefreshContent (int iDataSourceID*, IDataSource::RefreshMode* mode*, int* iUpdateChunkSize*)*

Starts a content refresh on the specified data source.

*void CDataSourceManager::RemoveDataSource (IDataSource **pDataSource*)*

Removes a data source from the list.

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

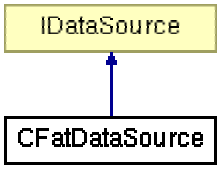
- **DataSourceManager.h**

CFatDataSource Class Reference

Implements the **IDataSource** (*p.120*) class for storage devices that use the FAT file system.

`#include <FatDataSource.h>`

Inheritance diagram for CFatDataSource:



Public Methods

- void **SetDefaultRefreshMode** (**RefreshMode** mode)
- **RefreshMode GetDefaultRefreshMode** () const
- void **SetDefaultUpdateChunkSize** (int iUpdateChunkSize)
- int **GetDefaultUpdateChunkSize** () const
- bool **GetRootURLPrefix** (char *szRootURLPrefix, int iMaxLength) const
- ERESULT **ListAllEntries** (**RefreshMode** mode, int iUpdateChunkSize)
- void **GetContentMetadata** (**content_record_update_t** *pContentUpdate)
- **IInputStream** * **OpenInputStream** (const char *szURL)
- **IOutputStream** * **OpenOutputStream** (const char *szURL)

Static Public Methods

- CFatDataSource * **Open** (int iDriveNumber)

Detailed Description

Implements the **IDataSource** (*p.120*) class for storage devices that use the FAT file system.

Member Function Documentation

*void CFatDataSource::GetContentMetadata (content_record_update_t * pContentUpdate) [virtual]*

Retrieves metadata for each media content record in the passed-in list.

Reimplemented from **IDataSource** (*p.121*).

RefreshMode CFatDataSource::GetDefaultRefreshMode () const `[virtual]`

Returns the current default refresh mode of the data source.

Reimplemented from **IDataSource** (*p.121*).

int CFatDataSource::GetDefaultUpdateChunkSize () const `[virtual]`

Returns the default update chunk size of the data source This value is the maximum number of records that will be sent for each EVENT_CONTENT_UPDATE message. 0 indicates that all records will be sent in one single message.

Reimplemented from **IDataSource** (*p.121*).

bool CFatDataSource::GetRootURLPrefix (char * **szRootURLPrefix**, *int* **iMaxLength**) *const* `[virtual]`

Copies the string the data source uses to prefix its URLs into the given string provided.

Reimplemented from **IDataSource** (*p.122*).

ERESULT CFatDataSource::ListAllEntries (RefreshMode **mode**, *int* **iUpdateChunkSize**) `[virtual]`

Asks the source to pass content updates lists through the event system that contain all of the content it can access. If bGetMetadata is true, then each entry is opened and passed to the matching codec for metadata retrieval. iUpdateChunkSize specifies how many media records to send back at a time. If iUpdateChunkSize is zero, then all records are sent back at once.

Reimplemented from **IDataSource** (*p.122*).

CFatDataSource CFatDataSource::Open (int* **iDriveNumber**) `[static]`

Attempts to open the drive with the given drive number. Returns a pointer to a new CFatDataSource object if successful, 0 otherwise.

IInputStream CFatDataSource::OpenInputStream (const char* * **szURL**) `[virtual]`

Asks the source to open this URL for reading. Returns 0 if the URL was unable to be opened, otherwise it returns the proper subclass of **IInputStream** (*p.127*) for this file type.

Reimplemented from **IDataSource** (*p.122*).

IOutputStream CFatDataSource::OpenOutputStream (const char* * **szURL**) `[virtual]`

Asks the source to open this URL for writing. Returns 0 if the URL was unable to be opened, otherwise it returns the proper subclass of **IOutputStream** (*p.132*) for this file type.

Reimplemented from **IDataSource** (*p.122*).

void CFatDataSource::SetDefaultRefreshMode (RefreshMode **mode**) `[virtual]`

Sets the default refresh mode of the data source. If DSR_DEFAULT is passed in, then DSR_ONE_PASS_WITH_METADATA is used.

Reimplemented from **IDataSource** (*p.122*).

void CFatDataSource::SetDefaultUpdateChunkSize (int iUpdateChunkSize*)* `[virtual]`

Sets the default update chunk size of the data source This value is the maximum number of records that should be sent for each EVENT_CONTENT_UPDATE message. 0 indicates that all records should be sent in one single message. If DS_DEFAULT_CHUNK_SIZE is passed in, then 0 is used.

Reimplemented from **IDataSource** (*p.122*).

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

- **FatDataSource.h**

CFatFile Class Reference

Since FAT filesystems have both read and write ability, we provide a common class to implement FAT file i/o operations, and wrap this in other useful classes (such as **CFatFileInputStream** (*p.54*) and **CFatFileOutputStream** (*p.55*)).

```
#include <FatFile.h>
```

Public Types

- enum **FileSeekPos** { **SeekStart** = 0, **SeekCurrent**, **SeekEnd** }
- enum **FileMode** { **ReadOnly**, **WriteOnly**, **ReadWrite**, **WriteCreate**, **WriteAppend** }

Public Methods

- bool **Open** (const char *Filename, **FileMode** Mode=ReadOnly)
- bool **Close** ()
- int **Read** (void *Buffer, int Count)
- int **Write** (const void *Buffer, int Count)
- bool **Flush** ()
- int **Ioctl** (int Key, void *Value)
- int **Seek** (**FileSeekPos** Origin, int Offset)
- int **GetOffset** () const
- int **Length** () const

Static Public Methods

- bool **Delete** (const char *Filename)

Detailed Description

Since FAT filesystems have both read and write ability, we provide a common class to implement FAT file i/o operations, and wrap this in other useful classes (such as **CFatFileInputStream** (*p.54*) and **CFatFileOutputStream** (*p.55*)).

Member Enumeration Documentation

enum CFatFile::FileMode

An enumeration for determining file open modes.

enum CFatFile::FileSeekPos

An enumeration for controlling seek origins.

Member Function Documentation

bool CFatFile::Close ()

Close the open file.

bool CFatFile::Delete (const char * **Filename***)* `[static]`

Static routine to delete a file from the filesystem.

bool CFatFile::Flush ()

Flush any buffered data to the disk.

int CFatFile::GetOffset () const

Returns the current offset (position) within the file.

int CFatFile::Ioctl (int **Key***, void* * **Value***)*

A generic interface for disk commands, not currently supported.

int CFatFile::Length () const

Return the length of the current file.

bool CFatFile::Open (const char * **Filename***, FileMode* **Mode** **=** ***ReadOnly)***

Attempt to open the given file in the specified file mode.

int CFatFile::Read (void * **Buffer***, int* **Count***)*

Read the specified number of bytes from the file to the buffer.

int CFatFile::Seek (FileSeekPos **Origin***, int* **Offset***)*

Seek from the specified origin to the offset.

int CFatFile::Write (const void * **Buffer***, int* **Count***)*

Write the specified number of bytes from the buffer to the file.

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

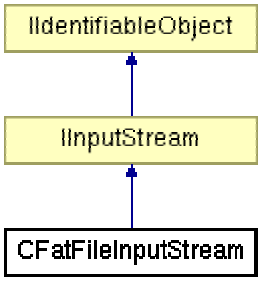
- **FatFile.h**

CFatFileInputStream Class Reference

The CFatFileInputStream provides an abstract input stream wrapper for the underlying **CFatFile** (*p.52*) object.

`#include <FileInputStream.h>`

Inheritance diagram for CFatFileInputStream:



Detailed Description

The CFatFileInputStream provides an abstract input stream wrapper for the underlying **CFatFile** (*p.52*) object.

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

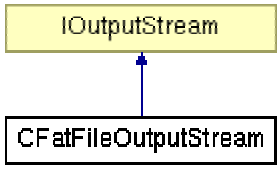
- **FileInputStream.h**

CFatFileOutputStream Class Reference

The CFatFileOutputStream provides an abstract output stream wrapper for the underlying **CFatFile** (*p.52*) object.

`#include <FileOutputStream.h>`

Inheritance diagram for CFatFileOutputStream:



Detailed Description

The CFatFileOutputStream provides an abstract output stream wrapper for the underlying **CFatFile** (*p.52*) object.

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

- **FileOutputStream.h**

CFilterManager Class Reference

The filter manager provides a simple way for the media player to locate and load available filters in the system. The filters can then be incorporated into the playstream, and the media player will take care of releasing them when it is done with them. This is a singleton class.

`#include <FilterManager.h>`

Public Methods

- **IFilter** * **LoadFilter** (unsigned int FilterID)

Static Public Methods

- CFilterManager * **GetInstance** ()

Detailed Description

The filter manager provides a simple way for the media player to locate and load available filters in the system. The filters can then be incorporated into the playstream, and the media player will take care of releasing them when it is done with them. This is a singleton class.

Member Function Documentation

CFilterManager CFilterManager::GetInstance () `[static]`*

Get a pointer to the single filter manager.

IFilter CFilterManager::LoadFilter (unsigned int* FilterID*)*

Load a filter by key, returning a pointer to the newly created instance. If the filter is not found, return NULL.

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

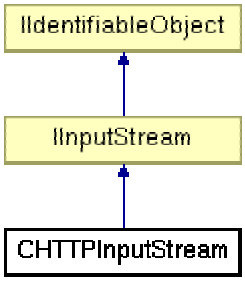
- **FilterManager.h**

CHTTPInputStream Class Reference

Dadio(tm) provides a method for shoutcast streams and http based streams to be played via the CHTTPInputStream. This class uses a **CNetStream** (*p.80*) object to do basic network i/o, and internally implements a mini http client to perform get requests and parse results.

`#include <HTTPInputStream.h>`

Inheritance diagram for CHTTPInputStream:



Detailed Description

Dadio(tm) provides a method for shoutcast streams and http based streams to be played via the CHTTPInputStream. This class uses a **CNetStream** (*p.80*) object to do basic network i/o, and internally implements a mini http client to perform get requests and parse results.

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

- **HTTPInputStream.h**

CIR Class Reference

CIR is the singleton IR driver object.

`#include <IR.h>`

Public Methods

- void **SetIRMap** (const ir_map_t *pIRmap)
- void **LockIR** ()
- void **UnlockIR** ()

Static Public Methods

- CIR * **GetInstance** ()

Detailed Description

CIR is the singleton IR driver object.

Member Function Documentation

CIR CIR::GetInstance () [static]*

Get a pointer to the driver instance.

void CIR::LockIR ()

Prevent the IR driver from generating events.

*void CIR::SetIRMap (const ir_map_t * pIRmap)*

Set an IR map for the driver. This can only be called once currently.

void CIR::UnlockIR ()

If the IR driver has been locked, unlock it.

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

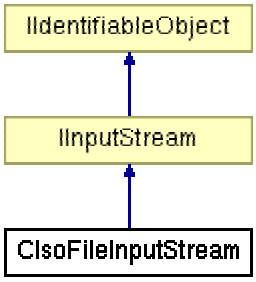
- **IR.h**

CIsoFileInputStream Class Reference

Dadio(tm) provides a precompiled class to access files found on ISO9660 (data CD) filesystems.

`#include <IsoFileInputStream.h>`

Inheritance diagram for CIsoFileInputStream:



Detailed Description

Dadio(tm) provides a precompiled class to access files found on ISO9660 (data CD) filesystems.

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

- **IsoFileInputStream.h**

CKeyboard Class Reference

The keyboard driver processes keyboard interrupts, scans the keyboard matrix, and generates events based on the current keymap. It allows the user application to mask and unmask event generation for software key locks.

`#include <Keyboard.h>`

Public Methods

- void **SetKeymap** (const **key_map_t** *pKeymap)
- void **LockKeyboard** ()
- void **UnlockKeyboard** ()

Static Public Methods

- CKeyboard * **GetInstance** ()

Detailed Description

The keyboard driver processes keyboard interrupts, scans the keyboard matrix, and generates events based on the current keymap. It allows the user application to mask and unmask event generation for software key locks.

Member Function Documentation

CKeyboard CKeyboard::GetInstance () `[static]`*

Get a pointer to the one global instance of the keyboard driver.

void CKeyboard::LockKeyboard ()

Prevent the keyboard driver from generating events.

*void CKeyboard::SetKeymap (const key_map_t * pKeymap)*

Set a keymap for this keyboard. This can only be called once currently

Parameters:

pKeymap A key_map_t defining the keyboard layout and events to generate.

void CKeyboard::UnlockKeyboard ()

If the keyboard driver has been locked, unlock it.

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

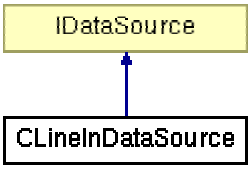
- **Keyboard.h**

CLineInDataSource Class Reference

The line in data source allows an application to connect to the line in device (the ADC on the board).

`#include <LineInDataSource.h>`

Inheritance diagram for CLineInDataSource:



Public Methods

- void **SetDefaultRefreshMode** (**RefreshMode** mode)
- **RefreshMode GetDefaultRefreshMode** () const
- void **SetDefaultUpdateChunkSize** (int iUpdateChunkSize)
- int **GetDefaultUpdateChunkSize** () const
- bool **GetRootURLPrefix** (char *szRootURLPrefix, int iMaxLength) const
- ERESULT **ListAllEntries** (**RefreshMode** mode, int iUpdateChunkSize)
- void **GetContentMetadata** (**content_record_update_t** *pContentUpdate)
- **IInputStream** * **OpenInputStream** (const char *szURL)
- **IOutputStream** * **OpenOutputStream** (const char *szURL)
- **IMediaContentRecord** * **GenerateEntry** (**IContentManager** *pContentManager, const char *pURL)

Detailed Description

The line in data source allows an application to connect to the line in device (the ADC on the board).

Member Function Documentation

IMediaContentRecord CLineInDataSource::GenerateEntry (IContentManager* pContentManager, const char * pURL)*

Add an entry to the content manager by parsing the given URL.

*void CLineInDataSource::GetContentMetadata (content_record_update_t * pContentUpdate) [inline, virtual]*

Not implemented. Entries are added manually by calling GenerateEntry.

Reimplemented from **IDataSource** (*p.121*).

RefreshMode CLineInDataSource::GetDefaultRefreshMode () const `[inline, virtual]`

Not implemented. Entries are added manually by calling GenerateEntry.

Reimplemented from **IDataSource** (*p.121*).

int CLineInDataSource::GetDefaultUpdateChunkSize () const `[inline, virtual]`

Not implemented. Entries are added manually by calling GenerateEntry.

Reimplemented from **IDataSource** (*p.121*).

bool CLineInDataSource::GetRootURLPrefix (char **szRootURLPrefix**, *int* **iMaxLength**) *const* `[virtual]`

Copies the string the data source uses to prefix its URLs into the given string provided.

Reimplemented from **IDataSource** (*p.122*).

ERESULT CLineInDataSource::ListAllEntries (RefreshMode **mode**, *int* **iUpdateChunkSize**) `[inline, virtual]`

Not implemented. Entries are added manually by calling GenerateEntry.

Reimplemented from **IDataSource** (*p.122*).

IInputStream CLineInDataSource::OpenInputStream (const char* * **szURL**) `[virtual]`

Asks the source to open this URL for reading. Returns 0 if the URL was unable to be opened, otherwise it returns the proper subclass of **IInputStream** (*p.127*) for this file type.

Reimplemented from **IDataSource** (*p.122*).

IOutputStream CLineInDataSource::OpenOutputStream (const char* * **szURL**) `[inline, virtual]`

Asks the source to open this URL for writing. Returns 0 if the URL was unable to be opened, otherwise it returns the proper subclass of **IOutputStream** (*p.132*) for this file type.

Reimplemented from **IDataSource** (*p.122*).

void CLineInDataSource::SetDefaultRefreshMode (RefreshMode **mode**) `[inline, virtual]`

Not implemented. Entries are added manually by calling GenerateEntry.

Reimplemented from **IDataSource** (*p.122*).

void CLineInDataSource::SetDefaultUpdateChunkSize (int **iUpdateChunkSize**) `[inline, virtual]`

Not implemented. Entries are added manually by calling GenerateEntry.

Reimplemented from **IDataSource** (*p.122*).

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

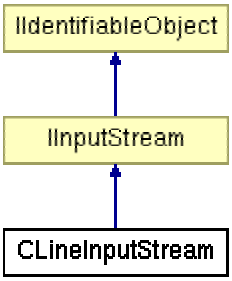
- **LineInDataSource.h**

CLineInputStream Class Reference

The line input stream provides an inputstream wrapper to the ADC audio input driver. Unlike other input streams, line input must be read in real-time, otherwise input frames will be dropped.

`#include <LineInputStream.h>`

Inheritance diagram for CLineInputStream:



Detailed Description

The line input stream provides an inputstream wrapper to the ADC audio input driver. Unlike other input streams, line input must be read in real-time, otherwise input frames will be dropped.

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

- **LineInputStream.h**

cm_key_value_record_t Struct Reference

Contains a key/value pair from the content manager.

`#include <QueryableContentManager.h>`

Detailed Description

Contains a key/value pair from the content manager.

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

- **QueryableContentManager.h**

CMediaPlayer Class Reference

CMediaPlayer is the basic interface for the media player. This is a singleton class that can be referenced through the **GetInstance**() (*p.67*) routine.

```
#include <MediaPlayer.h>
```

Public Types

- enum **PlayState** { **PLAYING** = 0, **PAUSED**, **STOPPED**, **NOT_CONFIGURED** }

Public Methods

- ERESULT **SetPlaystream** (const **playstream_settings_t** *pSettings)
- void **SetCreateMetadataFunction** (**FNCreateMetadata** *pfnCreateMetadata)
- ERESULT **SetSong** (**IPlaylistEntry** *pNewSong, bool bBufferedOpen=false)
- void **Deconfigure** ()
- StreamNode_t * **GetRoot** () const
- StreamNode_t * **AddNode** (StreamNode_t *pRoot, unsigned int ChildID)
- ERESULT **RemoveNode** (unsigned int NodeID)
- ERESULT **RemoveNode** (StreamNode_t *pNode)
- ERESULT **Play** ()
- ERESULT **Pause** ()
- ERESULT **Stop** ()
- ERESULT **Seek** (unsigned long secSeek)
- **PlayState GetPlayState** () const
- unsigned long **GetTrackTime** () const
- unsigned long **GetTrackLength** () const

Static Public Methods

- CMediaPlayer * **GetInstance** ()

Detailed Description

CMediaPlayer is the basic interface for the media player. This is a singleton class that can be referenced through the **GetInstance**() (*p.67*) routine.

Member Enumeration Documentation

enum CMediaPlayer::PlayState

Enumeration to indicate the current playstate.

Member Function Documentation

StreamNode_t CMediaPlayer::AddNode (StreamNode_t* **pRoot**, unsigned int **ChildID**)*

Not currently implemented.

void CMediaPlayer::Deconfigure ()

Deconfigure will unload any objects that are currently loaded, and return the mediaplayer to the NOT_CONFIGURED state.

CMediaPlayer CMediaPlayer::GetInstance () `[static]`*

Get a pointer to the singleton media player.

PlayState CMediaPlayer::GetPlayState () const

Query for current play state.

StreamNode_t CMediaPlayer::GetRoot () const*

Not currently implemented.

unsigned long CMediaPlayer::GetTrackLength () const

Returns the total playing time of the track.

unsigned long CMediaPlayer::GetTrackTime () const

Returns the current playing time of the track.

ERESULT CMediaPlayer::Pause ()

Pause playback.

ERESULT CMediaPlayer::Play ()

Start playback, or resume if paused.

ERESULT CMediaPlayer::RemoveNode (StreamNode_t **pNode**)*

Not currently implemented.

*ERESULT CMediaPlayer::RemoveNode (unsigned int **NodeID**)*

Not currently implemented.

*ERESULT CMediaPlayer::Seek (unsigned long **secSeek**)*

Seek to a new offset in the song.

void CMediaPlayer::SetCreateMetadataFunction (FNCreateMetadata **pfnCreateMetadata**)*

Sets the function that will be called to generate a new **IMetadata** (*p.130*) record to be populated during SetSong. If pfnCreateMetadata is 0, then no metadata record will be generated during SetSong (and no metadata will be retrieved).

ERESULT CMediaPlayer::SetPlaystream (const playstream_settings_t **pSettings**)*

Sets the current playstream. This will take effect when the next call to SetSong occurs.

ERESULT CMediaPlayer::SetSong (IPlaylistEntry **pNewSong**, bool **bBufferedOpen** = false)*

SetSong will use the playstream settings to generate a Playstream. if there are no settings it will fail.

ERESULT CMediaPlayer::Stop ()

Stop playback.

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

- **MediaPlayer.h**

CMetadataTable Class Reference

The CMetadataTable class matches metadata attribute IDs to types. It's a singleton class that when created adds all the standard metadata types to its table. If client code wants to add a new metadata attribute, then it must pick an attribute ID not currently in use and add it to this table along with a type for the metadata.

`#include <MetadataTable.h>`

Public Methods

- int **FindMetadataTypeByID** (int iAttributeID) const
- ERESULT **AddMetadataAttribute** (int iAttributeID, int iAttributeType)

Static Public Methods

- CMetadataTable * **GetInstance** ()
- void **Destroy** ()

Detailed Description

The CMetadataTable class matches metadata attribute IDs to types. It's a singleton class that when created adds all the standard metadata types to its table. If client code wants to add a new metadata attribute, then it must pick an attribute ID not currently in use and add it to this table along with a type for the metadata.

Member Function Documentation

ERESULT CMetadataTable::AddMetadataAttribute (int iAttributeID*, int* iAttributeType*)*

Adds a metadata attribute with the specified ID and type to the table.

Return values:

METADATA_NO_ERROR The attribute was added successfully.
METADATA_TABLE_ID_IN_USE There already exists and attribute with the same ID.
METADATA_TABLE_INVALID_TYPE The type passed in is invalid.

void CMetadataTable::Destroy () `[static]`

Destroy the singleton global metadata table.

int CMetadataTable::FindMetadataTypeByID (int iAttributeID*) const*

Given an attribute ID this function returns the type for that attribute or MDA_INVALID_ID if no matching metadata ID was found.

CMetadataTable CMetadataTable::GetInstance () `[static]`*

Returns a pointer to the global metadata table.

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

- **MetadataTable.h**

CMetakitContentManager Class Reference

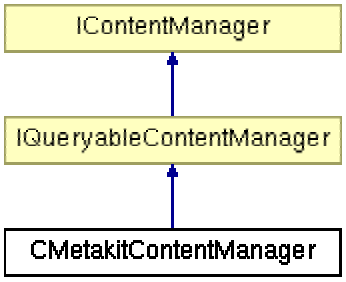
The metakit content manager is a concrete implementation of the **IContentManager** (*p.114*) interface. Content records are stored in the metakit database and can be queried by artist, album, and genre. Functions for loading and saving the content manager's state are also provided. This class has its own metadata type that stores title, artist, album, and genre. More metadata attributes can be added through the AddStoredMetadata function. Attributes added this way will persist when the database is saved.

Warning:

If used in persistent mode, then it must be created AFTER fat system initialization.

```
#include <MetakitContentManager.h>
```

Inheritance diagram for CMetakitContentManager:



Public Methods

- ERESULT **AddStoredMetadata** (int iAttributeID, const void *pUnsetValue)
- bool **LoadStateFromStream** (**IInputStream** *pInputStream)
- bool **SaveStateToStream** (**IOutputStream** *pOutputStream)
- void **Clear** ()
- void **Commit** ()
- void **AddContentRecords** (**content_record_update_t** *pContentUpdate)
- int **GetMediaRecordCount** ()
- **IMediaContentRecord** * **AddMediaRecord** (**media_record_info_t** &mediaContent, bool *pbAlreadyExists=0)
- void **DeleteMediaRecord** (int iContentRecordID)
- **IMediaContentRecord** * **GetMediaRecord** (int iContentRecordID)
- **IMediaContentRecord** * **GetMediaRecord** (const char *szURL)
- int **GetPlaylistRecordCount** ()
- **IPlaylistContentRecord** * **AddPlaylistRecord** (**playlist_record_t** &playlistRecord)
- void **DeletePlaylistRecord** (int iContentRecordID)
- void **DeleteRecordsFromDataSource** (int iDataSourceID)
- void **MarkRecordsFromDataSourceUnverified** (int iDataSourceID)
- void **DeleteUnverifiedRecordsFromDataSource** (int iDataSourceID)

- const TCHAR * **GetArtistByKey** (int iArtistKey) const
- const TCHAR * **GetAlbumByKey** (int iAlbumKey) const
- const TCHAR * **GetGenreByKey** (int iGenreKey) const
- int **GetArtistKey** (const TCHAR *szArtist) const
- int **GetAlbumKey** (const TCHAR *szAlbum) const
- int **GetGenreKey** (const TCHAR *szGenre) const
- void **GetAllMediaRecords** (MediaRecordList &records) const
- void **GetMediaRecordsByDataSourceID** (MediaRecordList &records, int iDataSourceID) const
- void **GetMediaRecords** (MediaRecordList &records, int iArtistKey=CMK_ALL, int iAlbumKey=CMK_ALL, int iGenreKey=CMK_ALL, int iDataSourceID=CMK_ALL) const
- void **GetArtists** (ContentKeyValueVector &keyValues, int iAlbumKey=CMK_ALL, int iGenreKey=CMK_ALL, int iDataSourceID=CMK_ALL) const
- void **GetAlbums** (ContentKeyValueVector &keyValues, int iArtistKey=CMK_ALL, int iGenreKey=CMK_ALL, int iDataSourceID=CMK_ALL) const
- void **GetGenres** (ContentKeyValueVector &keyValues, int iArtistKey=CMK_ALL, int iAlbumKey=CMK_ALL, int iDataSourceID=CMK_ALL) const
- void **GetAllPlaylistRecords** (PlaylistRecordList &records) const
- void **GetPlaylistRecordsByDataSourceID** (PlaylistRecordList &records, int iDataSourceID) const

---

Detailed Description

The metakit content manager is a concrete implementation of the **IContentManager** (*p.114*) interface. Content records are stored in the metakit database and can be queried by artist, album, and genre. Functions for loading and saving the content manager's state are also provided. This class has its own metadata type that stores title, artist, album, and genre. More metadata attributes can be added through the AddStoredMetadata function. Attributes added this way will persist when the database is saved.

Warning:

If used in persistent mode, then it must be created AFTER fat system initialization.

Member Function Documentation

*void CMetakitContentManager::AddContentRecords (content_record_update_t * pContentUpdate) [virtual]*

Adds a list of media and playlist records to the content manager.

Reimplemented from **IContentManager** (*p.115*).

IMediaContentRecord CMetakitContentManager::AddMediaRecord (media_record_info_t & mediaContent, bool * pbAlreadyExists = 0) [virtual]*

Adds an entry to the content manager. Returns a pointer to the content record. If pbAlreadyExists is not null, then its value is set to true if the content record with the same URL already exists in the content manager, false otherwise.

Reimplemented from **IContentManager** (*p.115*).

IPlaylistContentRecord CMetakitContentManager::AddPlaylistRecord (playlist_record_t &* playlistRecord*)* `[virtual]`

Adds a playlist record to the content manager. Returns a pointer to the record.

Reimplemented from **IContentManager** (*p.115*).

ERESULT CMetakitContentManager::AddStoredMetadata (int iAttributeID*, const void ** pUnsetValue*)*

Tells the content manager to use the given metadata attribute.

Parameters:

iAttributeID The ID of the attribute to add. If this attribute isn't a default attribute that came with the SDK, then it must have been added to the **CMetadataTable** (*p.69*).
pUnsetValue Value to be used inside the content manager to represent that the value of this attribute has not been set. (That is, when GetAttribute() is called, if the value of the attribute is equal to pUnsetValue, then METADATA_NO_VALUE_SET is returned.)

Return values:

MCM_UNKNOWN_METADATA_ID The attribute ID isn't in the **CMetadataTable** (*p.69*).
MCM_UNKNOWN_METADATA_TYPE The attribute has a bad type in the **CMetadataTable** (*p.69*).
MCM_NO_ERROR The attribute was added successfully.

void CMetakitContentManager::Clear () `[virtual]`

Clears the content manager and deletes its content records.

Reimplemented from **IContentManager** (*p.115*).

void CMetakitContentManager::Commit ()

Saves the database.

void CMetakitContentManager::DeleteMediaRecord (int iContentRecordID*)* `[virtual]`

Removes a record from the content manager.

Reimplemented from **IContentManager** (*p.115*).

void CMetakitContentManager::DeletePlaylistRecord (int iContentRecordID*)* `[virtual]`

Removes a playlist record from the content manager.

Reimplemented from **IContentManager** (*p.115*).

void CMetakitContentManager::DeleteRecordsFromDataSource (int iDataSourceID*)* `[virtual]`

Removes all records from the given data source from the content manager.

Reimplemented from **IContentManager** (*p.116*).

void CMetakitContentManager::DeleteUnverifiedRecordsFromDataSource (int iDataSourceID*)* `[virtual]`

Removes all records from the given data source that are marked as unverified.

Reimplemented from **IContentManager** (*p.116*).

const TCHAR CMetakitContentManager::GetAlbumByKey (int* iAlbumKey*) const* `[virtual]`

Returns the string of the album with the given key. Returns 0 if no album with a matching key was found.

Reimplemented from **IQueryableContentManager** (*p.141*).

*int CMetakitContentManager::GetAlbumKey (const TCHAR ** szAlbum*) const* `[virtual]`

Returns the album key that matches the album string passed in, or 0 if no matching album was found.

Reimplemented from **IQueryableContentManager** (*p.141*).

void CMetakitContentManager::GetAlbums (ContentKeyValueVector & keyValues*, int* iArtistKey = *CMK_ALL, int* iGenreKey = *CMK_ALL, int* iDataSourceID = *CMK_ALL) const* `[virtual]`

Searches the content manager for albums that have tracks with the specified artist, genre, and data source keys.

Parameters:

keyValues List used to append matching album key/value pairs.

Reimplemented from **IQueryableContentManager** (*p.141*).

void CMetakitContentManager::GetAllMediaRecords (MediaRecordList & records*) const* `[virtual]`

Appends all media content records in the manager to the record list passed in.

Reimplemented from **IContentManager** (*p.116*).

void CMetakitContentManager::GetAllPlaylistRecords (PlaylistRecordList & records*) const* `[virtual]`

Appends all playlist content records in the manager to the record list passed in.

Reimplemented from **IContentManager** (*p.116*).

const TCHAR CMetakitContentManager::GetArtistByKey (int* iArtistKey*) const* `[virtual]`

Returns the string of the artist with the given key. Returns 0 if no artist with a matching key was found.

Reimplemented from **IQueryableContentManager** (*p.141*).

*int CMetakitContentManager::GetArtistKey (const TCHAR ** szArtist*) const* `[virtual]`

Returns the artist key that matches the artist string passed in, or 0 if no matching artist was found.

Reimplemented from **IQueryableContentManager** (*p.141*).

void CMetakitContentManager::GetArtists (ContentKeyValueVector & keyValues*, int* iAlbumKey = *CMK_ALL, int* iGenreKey = *CMK_ALL, int* iDataSourceID = *CMK_ALL) const* `[virtual]`

Searches the content manager for artists that have tracks with the specified album, genre, and data source keys.

Parameters:

keyValues List used to append matching artist key/value pairs.

Reimplemented from **IQueryableContentManager** (*p.141*).

const TCHAR CMetakitContentManager::GetGenreByKey (int* iGenreKey*) const* `[virtual]`

Returns the string of the genre with the given key. Returns 0 if no genre with a matching key was found.

Reimplemented from **IQueryableContentManager** (*p.142*).

*int CMetakitContentManager::GetGenreKey (const TCHAR ** szGenre*) const* `[virtual]`

Returns the genre key that matches the genre string passed in, or 0 if no matching genre was found.

Reimplemented from **IQueryableContentManager** (*p.142*).

void CMetakitContentManager::GetGenres (ContentKeyValueVector & keyValues*, int* iArtistKey = *CMK_ALL, int* iAlbumKey = *CMK_ALL, int* iDataSourceID = *CMK_ALL) const* `[virtual]`

Searches the content manager for genre that have tracks with the specified artist, album, and data source keys.

Parameters:

keyValues List used to append matching genre key/value pairs.

Reimplemented from **IQueryableContentManager** (*p.142*).

IMediaContentRecord CMetakitContentManager::GetMediaRecord (const char ** szURL*)* `[virtual]`

Returns a pointer to the media content record with the specified URL. Returns 0 if no record with a matching URL was found.

Reimplemented from **IContentManager** (*p.116*).

IMediaContentRecord CMetakitContentManager::GetMediaRecord (int* iContentRecordID*)* `[virtual]`

Returns a pointer to the media content record with the specified record ID. Returns 0 if no record with a matching ID was found.

Reimplemented from **IContentManager** (*p.116*).

int CMetakitContentManager::GetMediaRecordCount () `[virtual]`

Returns the number of media records in the content manager.

Reimplemented from **IContentManager** (*p.116*).

void CMetakitContentManager::GetMediaRecords (MediaRecordList & records*, int* iArtistKey = *CMK_ALL, int* iAlbumKey = *CMK_ALL, int* iGenreKey = *CMK_ALL, int* iDataSourceID = *CMK_ALL) const* `[virtual]`

Searches the content manager for media records that match the artist, album, genre, and data source keys.

Parameters:

records List used to append matching records.

Reimplemented from **IQueryableContentManager** (*p.142*).

void CMetakitContentManager::GetMediaRecordsByDataSourceID (MediaRecordList & records*, int* iDataSourceID*) const* `[virtual]`

Appends all media content records from the given data source in the manager to the record list passed in.

Reimplemented from **IContentManager** (*p.116*).

int CMetakitContentManager::GetPlaylistRecordCount () `[virtual]`

Returns the number of playlist records in the content manager.

Reimplemented from **IContentManager** (*p.116*).

void CMetakitContentManager::GetPlaylistRecordsByDataSourceID (PlaylistRecordList & records*, int* iDataSourceID*) const* `[virtual]`

Appends all playlist content records from the given data source in the manager to the record list passed in.

Reimplemented from **IContentManager** (*p.117*).

*bool CMetakitContentManager::LoadStateFromStream (IInputStream ** pInputStream*)*

Loads content records from a stream

Parameters:

pInputStream An open input stream to read the state from.

Return values:

true if the state was loaded,
false if an error occured loading state.

void CMetakitContentManager::MarkRecordsFromDataSourceUnverified (int iDataSourceID*)* `[virtual]`

Marks all records from the given data source as unverified.

Reimplemented from **IContentManager** (*p.117*).

*bool CMetakitContentManager::SaveStateToStream (IOutputStream ** pOutputStream*)*

Saves content records to a stream.

Parameters:

pOutputStream An open output stream to write the state to.

Return values:

true if the state was saved,
false if an error occured saving state.

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

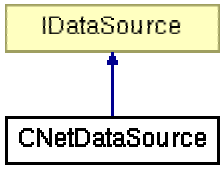
- **MetakitContentManager.h**

CNetDataSource Class Reference

A network based data source that lets clients specify URLs to add to the content manager.

`#include <NetDataSource.h>`

Inheritance diagram for CNetDataSource:



Public Methods

- void **SetDefaultRefreshMode** (**RefreshMode** mode)
- **RefreshMode GetDefaultRefreshMode** () const
- void **SetDefaultUpdateChunkSize** (int iUpdateChunkSize)
- int **GetDefaultUpdateChunkSize** () const
- **IInputStream** * **OpenInputStream** (const char *szURL)
- **IOutputStream** * **OpenOutputStream** (const char *szURL)
- **IMediaContentRecord** * **GenerateEntry** (**IContentManager** *pContentManager, const char *pURL)
- bool **IsInitialized** () const

Detailed Description

A network based data source that lets clients specify URLs to add to the content manager.

Member Function Documentation

IMediaContentRecord CNetDataSource::GenerateEntry (IContentManager * pContentManager, const char * pURL)*

Add an entry to the content manager by parsing the given URL.

RefreshMode CNetDataSource::GetDefaultRefreshMode () const [inline, virtual]

Not implemented. Entries are added manually by calling GenerateEntry.

Reimplemented from **IDataSource** (*p.121*).

int CNetDataSource::GetDefaultUpdateChunkSize () const [inline, virtual]

Not implemented. Entries are added manually by calling GenerateEntry.

Reimplemented from **IDataSource** (*p.121*).

bool CNetDataSource::IsInitialized () const

Returns true if the network is initialized, false otherwise.

IInputStream CNetDataSource::OpenInputStream (const char * szURL) [virtual]*

Asks the source to open this URL for reading. Returns 0 if the URL was unable to be opened, otherwise it returns the proper subclass of **IInputStream** (*p.127*) for this file type.

Reimplemented from **IDataSource** (*p.122*).

IOutputStream CNetDataSource::OpenOutputStream (const char * szURL) [inline, virtual]*

Asks the source to open this URL for writing. Returns 0 if the URL was unable to be opened, otherwise it returns the proper subclass of **IOutputStream** (*p.132*) for this file type.

Reimplemented from **IDataSource** (*p.122*).

void CNetDataSource::SetDefaultRefreshMode (RefreshMode mode) [inline, virtual]

Not implemented. Entries are added manually by calling GenerateEntry.

Reimplemented from **IDataSource** (*p.122*).

void CNetDataSource::SetDefaultUpdateChunkSize (int iUpdateChunkSize) [inline, virtual]

Not implemented. Entries are added manually by calling GenerateEntry.

Reimplemented from **IDataSource** (*p.122*).

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

- **NetDataSource.h**

CNetStream Class Reference

Dadio(tm) provides a simple class to read and write to network streams. This allows input streams and output streams to be derived from a common base, in addition to allowing applications to abstractly interact with network services.

`#include <NetStream.h>`

Public Methods

- bool **Open** (unsigned int address, unsigned int port)
- bool **Open** (const char *address, unsigned int port)
- bool **Close** ()
- int **Read** (void *Buffer, int Count)
- int **Write** (const void *Buffer, int Count)
- bool **Flush** ()
- int **Ioctl** (int Key, void *Value)
- int **Position** () const

Detailed Description

Dadio(tm) provides a simple class to read and write to network streams. This allows input streams and output streams to be derived from a common base, in addition to allowing applications to abstractly interact with network services.

Member Function Documentation

bool CNetStream::Close ()

Close the current connection.

bool CNetStream::Flush ()

Flush the current stream. This is currently a no-op.

int CNetStream::Ioctl (int Key*, void ** Value*)*

Issue a generic ioctl to the stream, not currently supported.

*bool CNetStream::Open (const char ** address*, unsigned int* port*)*

Open a connection to the given address/port pair. address can be a pointer to a string representing a dotted notation IP, or can be a pointer to a string with a valid hostname. In the case of the latter, a DNS lookup will be performed to attempt to locate the host.

bool CNetStream::Open (unsigned int address*, unsigned int* port*)*

Open a connection to the given address/port pair

Parameters:

address The network byte order address
port The host byte order port.

int CNetStream::Position () const

Return the current offset (position) within the stream.

int CNetStream::Read (void * **Buffer***, int* **Count***)*

Read from the network stream.

int CNetStream::Write (const void * **Buffer***, int* **Count***)*

Write to the network stream.

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

- **NetStream.h**

CodecFunctionMap_s Struct Reference

Short table of static functions that a codec must expose. This is auto-generated by the registration interface.

`#include <Codec.h>`

Detailed Description

Short table of static functions that a codec must expose. This is auto-generated by the registration interface.

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

- **Codec.h**

content_record_update_s Struct Reference

Used for passing information about the media and playlist records available from a data source to the content manager.

`#include <ContentManager.h>`

Public Attributes

- int **iDataSourceID**
- bool **bTwoPass**
- IMediaRecordInfoVector **media**
- IPlaylistRecordInfoVector **playlists**

Detailed Description

Used for passing information about the media and playlist records available from a data source to the content manager.

Member Data Documentation

bool content_record_update_s::bTwoPass

Pass-through option specified by the type of search. If true, then when AddContentRecords is called the content manager should prune the media record info vector to contain only records whose metadata needs retrieving.

int content_record_update_s::iDataSourceID

Data source that supplied the content.

IMediaRecordInfoVector content_record_update_s::media

List of media content records to add/update.

IPlaylistRecordInfoVector content_record_update_s::playlists

List of playlist content records to add/update.

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

- **ContentManager.h**

COutputManager Class Reference

The output manager provides a simple way for the media player to locate and load available output streams in the system. The output streams can then be incorporated into the playstream, and the media player will take care of releasing them when it is done with them. This is a singleton class.

`#include <OutputManager.h>`

Public Methods

- **IOutputStream * LoadOutputStream** (unsigned int OutputID)

Static Public Methods

- COutputManager *** GetInstance** ()

Detailed Description

The output manager provides a simple way for the media player to locate and load available output streams in the system. The output streams can then be incorporated into the playstream, and the media player will take care of releasing them when it is done with them. This is a singleton class.

Member Function Documentation

COutputManager COutputManager::GetInstance () `[static]`*

Get a pointer to the single instance of the output manager.

IOutputStream COutputManager::LoadOutputStream (unsigned int* OutputID*)*

Load an output stream based on output stream ID, return a pointer to the newly allocated object. If the requested output stream is not found, return NULL.

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

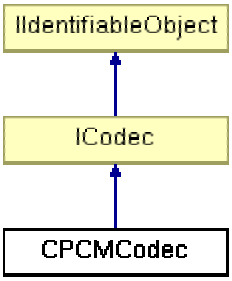
- **OutputManager.h**

CPCMCodec Class Reference

Dadio(tm) has a precompiled PCM codec capable of playing back 44.1kHz stereo 16bit raw PCM streams such as those found on audio CDs. This codec does not look for any header on incoming bitstreams and assumes data being given to it is in this format.

`#include <PCMCodec.h>`

Inheritance diagram for CPCMCodec:



Public Methods

- ERESULT **SetSong** (**IInputStream** *pInputStream, **stream_info_t** &streamInfo, **IMetadata** *pMetadata=0)

Detailed Description

Dadio(tm) has a precompiled PCM codec capable of playing back 44.1kHz stereo 16bit raw PCM streams such as those found on audio CDs. This codec does not look for any header on incoming bitstreams and assumes data being given to it is in this format.

Member Function Documentation

ERESULT CPCMCodec::SetSong (IInputStream * pInputStream, *stream_info_t &* streamInfo, *IMetadata* * pMetadata = *0)* `[virtual]`

SetSong on the PCM codec is slightly different from other codecs, in that it always succeeds. As a result of this, this codec cannot be probed to determine whether or not it supports a given format.

Reimplemented from **ICodec** (*p.113*).

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

- **PCMCodec.h**

CPlaylistFormatManager Class Reference

The playlist format manager keeps track of the playlist file formats available in the system and the functions for loading and saving them.

`#include <PlaylistFormatManager.h>`

Public Methods

- unsigned int **FindPlaylistFormat** (const char *szExtension)
- ERESULT **LoadPlaylist** (int iPlaylistFormatID, const char *szURL, **IPlaylist** *pPlaylist, bool bVerifyContent=false)
- ERESULT **SavePlaylist** (int iPlaylistFormatID, const char *szURL, **IPlaylist** *pPlaylist)

Static Public Methods

- CPlaylistFormatManager * **GetInstance** ()

Detailed Description

The playlist format manager keeps track of the playlist file formats available in the system and the functions for loading and saving them.

Member Function Documentation

*unsigned int CPlaylistFormatManager::FindPlaylistFormat (const char * szExtension)*

Given a file extension, this function searches the list of registered playlist formats for a matching extension. If found, then the ID of the playlist format is returned. If no matching playlist format is found, then 0 is returned.

CPlaylistFormatManager CPlaylistFormatManager::GetInstance () [static]*

Get a pointer to the single instance of the playlist format manager.

*ERESULT CPlaylistFormatManager::LoadPlaylist (int iPlaylistFormatID, const char * szURL, IPlaylist * pPlaylist, bool bVerifyContent = false)*

Load the playlist at the given URL into the playlist provided.

Parameters:

iPlaylistFormatID The ID in the registry that matches the playlist format. Usually attained by calling FindPlaylistFormat.
szURL The URL of the playlist file to open.
pPlaylist Pointer to a playlist to append entries to.
bVerifyContent If bVerifyContent is true, then entries will be checked against the content manager before being added, and files not found there will not be added. If bVerifyContent is

false, then entries not found in the content manager will be added to it. HTTP streams are not verified; they are always added.

ERESULT CPlaylistFormatManager::SavePlaylist (int iPlaylistFormatID*, const char ** szURL*, IPlaylist ** pPlaylist*)*

Save the given playlist to a stream to be opened by the given URL.

Parameters:

iPlaylistFormatID The ID in the registry that matches the playlist format. Usually attained by calling FindPlaylistFormat.
szURL The URL of the playlist file to open.
pPlaylist Pointer to a playlist whose entries are to be save out.

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

- **PlaylistFormatManager.h**

CPlayManager Class Reference

The play manager is the nerve center of the system. It maintains pointers to the content manager and current playlist. It has a default event handler. It exposes functions for play control. It's the object that kicks off content updates.

`#include <PlayManager.h>`

Public Methods

- void **AddDataSource** (**IDataSource** *pDS)
- void **RefreshAllContent** (**IDataSource::RefreshMode** mode=IDataSource::DSR_DEFAULT, int iUpdateChunkSize=DS_DEFAULT_CHUNK_SIZE)
- void **RefreshContent** (int iDataSourceID, **IDataSource::RefreshMode** mode=IDataSource::DSR_DEFAULT, int iUpdateChunkSize=DS_DEFAULT_CHUNK_SIZE)
- void **NotifyContentUpdate** (**content_record_update_t** *pContentUpdate)
- void **NotifyContentMetadataUpdate** (**content_record_update_t** *pContentUpdate)
- void **SetContentManager** (**IContentManager** *pCM)
- **IContentManager** * **GetContentManager** () const
- void **SetPlaylist** (**IPlaylist** *pPL)
- **IPlaylist** * **GetPlaylist** () const
- **IPlaylist::PlaylistMode GetPlaylistMode** () const
- void **SetPlaylistMode** (**IPlaylist::PlaylistMode** mode)
- ERESULT **Play** ()
- ERESULT **Pause** ()
- ERESULT **Stop** ()
- ERESULT **Seek** (unsigned long secSeek)
- **CMediaPlayer::PlayState GetPlayState** () const
- ERESULT **NextTrack** ()
- ERESULT **PreviousTrack** ()
- ERESULT **HandleEvent** (int key, void *data)
- void **NotifyMediaRemoved** (int iDataSourceID)
- void **NotifyMediaInserted** (int iDataSourceID)

Static Public Methods

- CPlayManager * **GetInstance** ()

Detailed Description

The play manager is the nerve center of the system. It maintains pointers to the content manager and current playlist. It has a default event handler. It exposes functions for play control. It's the object that kicks off content updates.

Member Function Documentation

*void CPlayManager::AddDataSource (IDataSource * pDS)*

Given an arbitrary (opened) data source, add it to the system list of available data sources.

IContentManager CPlayManager::GetContentManager () const*

Returns a pointer to the content manager.

CPlayManager CPlayManager::GetInstance () [static]*

Get a pointer to the one global instance of the play manager.

CMediaPlayer::PlayState CPlayManager::GetPlayState () const

Query for current play state.

IPlaylist CPlayManager::GetPlaylist () const*

Returns a pointer to the current playlist.

IPlaylist::PlaylistMode CPlayManager::GetPlaylistMode () const

Returns the current mode for traversing the playlist.

*ERESULT CPlayManager::HandleEvent (int key, void * data)*

The system event handler. This should be called from the client event loop to process system-specific events and to handle event cleanup.

ERESULT CPlayManager::NextTrack ()

Go to the next track in the playlist. If an entry is unable to be set in the media player, then the next track will be tried. If all tracks following the current one are bad, then the current track is re-set. If the current track can't be set, then the play manager attempts to set a track previous to the current one.

Return values:

PM_PLAYING The next track was set successfully and is currently playing.
PM_NO_ERROR The next track was set successfully.
PM_NO_GOOD_TRACKS All tracks in the playlist are invalid.
PM_PLAYLIST_END The next track couldn't be set because the end of the playlist was reached.

*void CPlayManager::NotifyContentMetadataUpdate (content_record_update_t * pContentUpdate)*

Updates existing content records with metadata retrieved from a data source and a codec. Called by the default handler. Only call this function if you are bypassing the default event handler during content update.

*void CPlayManager::NotifyContentUpdate (content_record_update_t * pContentUpdate)*

Updates the content manager with content received from a source. Called by the default handler. Only call this function if you are bypassing the default event handler during content update.

void CPlayManager::NotifyMediaInserted (int iDataSourceID)

Informs the play manager that a data source's media has been inserted. Called by the default handler. Only call this function if you are bypassing the default event handler during content update.

void CPlayManager::NotifyMediaRemoved (int iDataSourceID)

Informs the play manager that a data source's media has been removed. Called by the default handler. Only call this function if you are bypassing the default event handler during content update.

ERESULT CPlayManager::Pause ()

Pause playback of the current entry in the media player.

ERESULT CPlayManager::Play ()

Begin playback of the current entry in the media player. If the media player hasn't been configured, then the current playlist entry is set as the current entry.

ERESULT CPlayManager::PreviousTrack ()

Go to the previous track in the playlist. If an entry is unable to be set in the media player, then the previous track will be tried. If all tracks prior the current one are bad, then the current track is re-set. If the current track can't be set, then the play manager attempts to set a track following the current one.

Return values:

PM_PLAYING The previous track was set successfully and is currently playing.
PM_NO_ERROR The previous track was set successfully.
PM_NO_GOOD_TRACKS All tracks in the playlist are invalid.
PM_PLAYLIST_END The previous track couldn't be set because the end of the playlist was reached.

void CPlayManager::RefreshAllContent (IDataSource::RefreshMode mode = *IDataSource::DSR_DEFAULT, int* iUpdateChunkSize = *DS_DEFAULT_CHUNK_SIZE)*

Starts a content refresh from the available data sources

Parameters:

mode Which refresh should be used by each data source.
iUpdateChunkSize How many records should be accumulated before sending off an update event. If 0, then only one event will be sent per data source.

void CPlayManager::RefreshContent (int iDataSourceID*, IDataSource::RefreshMode* mode = *IDataSource::DSR_DEFAULT, int* iUpdateChunkSize = *DS_DEFAULT_CHUNK_SIZE)*

Starts a content refresh on the specified data source

Parameters:

mode Which refresh should be used by the data source.
iUpdateChunkSize How many records should be accumulated before sending off an update event. If 0, then the data source will populate all content before sending an event.

ERESULT CPlayManager::Seek (unsigned long secSeek*)*

Seek to the specified time offset in the file.

*void CPlayManager::SetContentManager (IContentManager ** pCM*)*

Associates the play manager with a content manager. A content manager must be set before content refreshes can begin.

*void CPlayManager::SetPlaylist (IPlaylist ** pPL*)*

Associates the play manager with a playlist.

void CPlayManager::SetPlaylistMode (IPlaylist::PlaylistMode mode*)*

Sets the play mode. If the mode is RANDOM or REPEAT_RANDOM, then the playlist is reshuffled with the current entry set as the first entry in the random list.

ERESULT CPlayManager::Stop ()

Stop playback of the current entry in the media player.

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

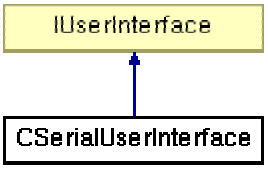
- **PlayManager.h**

CSerialUserInterface Class Reference

The CSerialUserInterface class implents the **IUserInterface** (*p.145*) class to provide text-based updates of the player state over the serial line for the demo players. It is not part of the SDK proper, but is provided to support the demo applications.

`#include <SerialUserInterface.h>`

Inheritance diagram for CSerialUserInterface:



Detailed Description

The CSerialUserInterface class implents the **IUserInterface** (*p.145*) class to provide text-based updates of the player state over the serial line for the demo players. It is not part of the SDK proper, but is provided to support the demo applications.

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

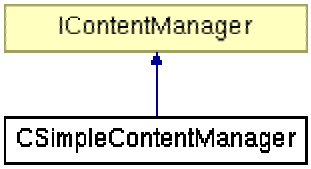
- **SerialUserInterface.h**

CSimpleContentManager Class Reference

The CSimpleContentManager implements the **IContentManager** (*p.114*) interface in a simple, non-optimized manner.

`#include <SimpleContentManager.h>`

Inheritance diagram for CSimpleContentManager:



Public Methods

- **CSimpleContentManager** (**FNCreateMetadata** *pfnCreateMetadata)
- **IMetadata * CreateMetadataRecord** () const
- void **Clear** ()
- void **AddContentRecords** (**content_record_update_t** *pContentUpdate)
- int **GetMediaRecordCount** ()
- **IMediaContentRecord * AddMediaRecord** (**media_record_info_t** &mediaContent, bool *pbAlreadyExists=0)
- void **DeleteMediaRecord** (int iContentRecordID)
- **IMediaContentRecord * GetMediaRecord** (int iContentRecordID)
- **IMediaContentRecord * GetMediaRecord** (const char *szURL)
- int **GetPlaylistRecordCount** ()
- **IPlaylistContentRecord * AddPlaylistRecord** (**playlist_record_t** &playlistRecord)
- void **DeletePlaylistRecord** (int iContentRecordID)
- void **DeleteRecordsFromDataSource** (int iDataSourceID)
- void **MarkRecordsFromDataSourceUnverified** (int iDataSourceID)
- void **DeleteUnverifiedRecordsFromDataSource** (int iDataSourceID)

Detailed Description

The CSimpleContentManager implements the **IContentManager** (*p.114*) interface in a simple, non-optimized manner.

Constructor & Destructor Documentation

*CSimpleContentManager::CSimpleContentManager (FNCreateMetadata * pfnCreateMetadata)*

pfnCreateMetadata is a pointer to the function that will be called to generate a new **IMetadata** (*p.130*) record to be populated during content refreshes. If pfnCreateMetadata is 0, then no metadata record will be generated (and no metadata will be retrieved).

Member Function Documentation

*void CSimpleContentManager::AddContentRecords (content_record_update_t * pContentUpdate) [virtual]*

Called from the event handler when a data source returns a list of records.

Reimplemented from **IContentManager** (*p.115*).

IMediaContentRecord CSimpleContentManager::AddMediaRecord (media_record_info_t & mediaContent, bool * pbAlreadyExists = 0) [virtual]*

Adds an entry to the content manager. Returns a pointer to the content record. If pbAlreadyExists is not null, then its value is set to true if the content record with the same URL already exists in the content manager, false otherwise.

Reimplemented from **IContentManager** (*p.115*).

IPlaylistContentRecord CSimpleContentManager::AddPlaylistRecord (playlist_record_t & playlistRecord) [virtual]*

Adds a playlist record to the content manager. Returns a pointer to the record.

Reimplemented from **IContentManager** (*p.115*).

void CSimpleContentManager::Clear () [virtual]

Clears the content manager and deletes its content records.

Reimplemented from **IContentManager** (*p.115*).

IMetadata CSimpleContentManager::CreateMetadataRecord () const [virtual]*

Creates an empty metadata record.

Reimplemented from **IContentManager** (*p.115*).

void CSimpleContentManager::DeleteMediaRecord (int iContentRecordID) [virtual]

Removes a record from the content manager.

Reimplemented from **IContentManager** (*p.115*).

void CSimpleContentManager::DeletePlaylistRecord (int iContentRecordID) [virtual]

Removes a playlist record from the content manager.

Reimplemented from **IContentManager** (*p.115*).

void CSimpleContentManager::DeleteRecordsFromDataSource (int iDataSourceID*)* `[virtual]`

Removes all records from the given data source from the content manager.

Reimplemented from **IContentManager** (*p.116*).

void CSimpleContentManager::DeleteUnverifiedRecordsFromDataSource (int iDataSourceID*)* `[virtual]`

Removes all records from the given data source that are marked as unverified.

Reimplemented from **IContentManager** (*p.116*).

IMediaContentRecord CSimpleContentManager::GetMediaRecord (const char ** szURL*)* `[virtual]`

Returns a pointer to the media content record with the specified URL. Returns 0 if no record with a matching URL was found.

Reimplemented from **IContentManager** (*p.116*).

IMediaContentRecord CSimpleContentManager::GetMediaRecord (int* iContentRecordID*)* `[virtual]`

Returns a pointer to the media content record with the specified record ID. Returns 0 if no record with a matching ID was found.

Reimplemented from **IContentManager** (*p.116*).

int CSimpleContentManager::GetMediaRecordCount () `[virtual]`

Returns the number of media records in the content manager.

Reimplemented from **IContentManager** (*p.116*).

int CSimpleContentManager::GetPlaylistRecordCount () `[virtual]`

Returns the number of playlist records in the content manager.

Reimplemented from **IContentManager** (*p.116*).

void CSimpleContentManager::MarkRecordsFromDataSourceUnverified (int iDataSourceID*)* `[virtual]`

Marks all records from the given data source as unverified.

Reimplemented from **IContentManager** (*p.117*).

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

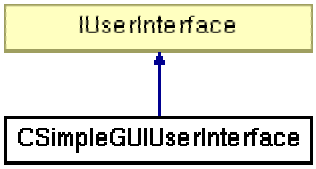
- **SimpleContentManager.h**

CSimpleGUIUserInterface Class Reference

The CSimpleGUIUserInterface class implents the **IUserInterface** (*p.145*) class to provide a simple graphical user interface for the demo players. It is not part of the SDK proper, but is provided to support the demo applications.

`#include <SimpleGUIUserInterface.h>`

Inheritance diagram for CSimpleGUIUserInterface:



Detailed Description

The CSimpleGUIUserInterface class implents the **IUserInterface** (*p.145*) class to provide a simple graphical user interface for the demo players. It is not part of the SDK proper, but is provided to support the demo applications.

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

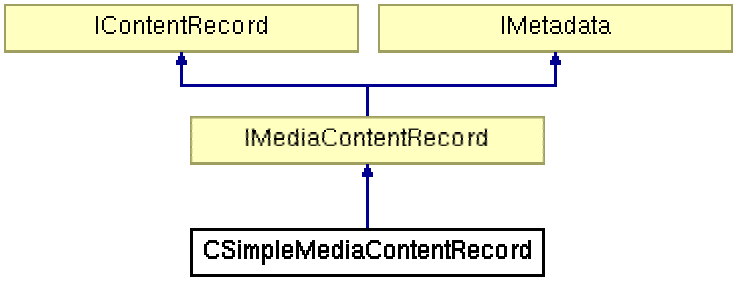
- **SimpleGUIUserInterface.h**

CSimpleMediaContentRecord Class Reference

A demo derivation of the **IMediaContentRecord** (*p.129*) interface. This object is not part of the SDK proper.

`#include <SimpleContentManager.h>`

Inheritance diagram for CSimpleMediaContentRecord:



Detailed Description

A demo derivation of the **IMediaContentRecord** (*p.129*) interface. This object is not part of the SDK proper.

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

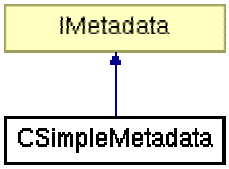
- **SimpleContentManager.h**

CSimpleMetadata Class Reference

This class is a simple implementation of the **IMetadata** (*p.130*) class, providing support for the MDA_TITLE, MDA_ARTIST, MDA_ALBUM, and MDA_GENRE fields.

`#include <SimpleMetadata.h>`

Inheritance diagram for CSimpleMetadata:



Public Methods

- const TCHAR * **GetTitle** () const
- const TCHAR * **GetArtist** () const
- const TCHAR * **GetAlbum** () const
- const TCHAR * **GetGenre** () const

Detailed Description

This class is a simple implementation of the **IMetadata** (*p.130*) class, providing support for the MDA_TITLE, MDA_ARTIST, MDA_ALBUM, and MDA_GENRE fields.

Member Function Documentation

const TCHAR CSimpleMetadata::GetAlbum () const* `[inline]`

Convenience function for getting the album.

Return values:

0 No album set.

const TCHAR CSimpleMetadata::GetArtist () const* `[inline]`

Convenience function for getting the artist.

Return values:

0 No artist set.

const TCHAR CSimpleMetadata::GetGenre () const* `[inline]`

Convenience function for getting the genre.

Return values:

0 No genre set.

const TCHAR CSimpleMetadata::GetTitle () const* `[inline]`

Convenience function for getting the title.

Return values:

0 No title set.

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

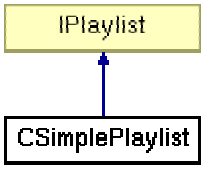
- **SimpleMetadata.h**

CSimplePlaylist Class Reference

Dadio(tm) is packaged with an example simple playlist. For most basic players this should suffice. This code is not part of the SDK proper, but is instead an example of an implementation of the SDK **IPlaylist** (*p.134*) API.

`#include <SimplePlaylist.h>`

Inheritance diagram for CSimplePlaylist:



Public Methods

- const char * **GetName** () const
- void **Clear** ()
- int **GetSize** ()
- bool **IsEmpty** () const
- **IPlaylistEntry** * **AddEntry** (**IMediaContentRecord** *pContentRecord)
- **IPlaylistEntry** * **InsertEntry** (**IMediaContentRecord** *pContentRecord, int index)
- void **AddEntries** (MediaRecordList &records)
- void **DeleteEntry** (**IPlaylistEntry** *pEntry)
- void **DeleteEntriesFromDataSource** (int iDataSourceID)
- void **ReshuffleRandomEntries** ()
- **IPlaylistEntry** * **GetEntry** (int index, **PlaylistMode** mode=NORMAL)
- int **GetEntryIndex** (const **IPlaylistEntry** *pEntry)
- **IPlaylistEntry** * **GetCurrentEntry** ()
- **IPlaylistEntry** * **SetCurrentEntry** (**IPlaylistEntry** *pEntry)
- **IPlaylistEntry** * **GetNextEntry** (**IPlaylistEntry** *pEntry, **PlaylistMode** mode)
- **IPlaylistEntry** * **SetNextEntry** (**PlaylistMode** mode)
- **IPlaylistEntry** * **GetPreviousEntry** (**IPlaylistEntry** *pEntry, **PlaylistMode** mode)
- **IPlaylistEntry** * **SetPreviousEntry** (**PlaylistMode** mode)

Detailed Description

Dadio(tm) is packaged with an example simple playlist. For most basic players this should suffice. This code is not part of the SDK proper, but is instead an example of an implementation of the SDK **IPlaylist** (*p.134*) API.

Member Function Documentation

void CSimplePlaylist::AddEntries (MediaRecordList & records*)* `[virtual]`

Adds all entries from the content record list to the end of the playlist.

Reimplemented from **IPlaylist** (*p.135*).

IPlaylistEntry CSimplePlaylist::AddEntry (IMediaContentRecord ** pContentRecord*)* `[virtual]`

Adds an entry to the end of the playlist. Returns a pointer to the new playlist entry.

Reimplemented from **IPlaylist** (*p.135*).

void CSimplePlaylist::Clear () `[virtual]`

Clears the playlist and frees up its memory.

Reimplemented from **IPlaylist** (*p.135*).

void CSimplePlaylist::DeleteEntriesFromDataSource (int iDataSourceID*)* `[virtual]`

Removes all entries from the given data source from the playlist.

Reimplemented from **IPlaylist** (*p.135*).

*void CSimplePlaylist::DeleteEntry (IPlaylistEntry ** pEntry*)* `[virtual]`

Removes an entry from the playlist.

Reimplemented from **IPlaylist** (*p.135*).

IPlaylistEntry CSimplePlaylist::GetCurrentEntry ()* `[virtual]`

Returns a pointer to the current entry.

Reimplemented from **IPlaylist** (*p.136*).

IPlaylistEntry CSimplePlaylist::GetEntry (int* index*, PlaylistMode* mode *= NORMAL)* `[virtual]`

Returns a pointer to the entry at the specified zero-based index using the given playlist mode. If the index is out-of-range, then 0 is returned.

Reimplemented from **IPlaylist** (*p.136*).

*int CSimplePlaylist::GetEntryIndex (const IPlaylistEntry ** pEntry*)* `[virtual]`

Returns the zero-based index of the given playlist entry. Returns -1 if the entry is not in the playlist.

Reimplemented from **IPlaylist** (*p.136*).

const char CSimplePlaylist::GetName () const* `[virtual]`

Returns the playlist's name.

Reimplemented from **IPlaylist** (*p.136*).

IPlaylistEntry CSimplePlaylist::GetNextEntry (IPlaylistEntry ** pEntry*, PlaylistMode* mode*)* `[virtual]`

Returns a pointer to the next entry in the playlist as determined by the mode. Returns 0 if the end of the list was reached and the play mode isn't repeating.

Reimplemented from **IPlaylist** (*p.136*).

IPlaylistEntry CSimplePlaylist::GetPreviousEntry (IPlaylistEntry ** pEntry*, PlaylistMode* mode*)* `[virtual]`

Returns a pointer to the previous entry in the playlist as determined by the mode. Returns 0 if the end of the list was reached and the play mode isn't repeating.

Reimplemented from **IPlaylist** (*p.136*).

int CSimplePlaylist::GetSize () `[virtual]`

Returns the number of entries in the playlist.

Reimplemented from **IPlaylist** (*p.136*).

IPlaylistEntry CSimplePlaylist::InsertEntry (IMediaContentRecord ** pContentRecord*, int* index*)* `[virtual]`

Inserts an entry to the playlist at the specified zero-based index. Returns a pointer to the new playlist entry.

Reimplemented from **IPlaylist** (*p.136*).

bool CSimplePlaylist::IsEmpty () const `[virtual]`

Returns true if the playlist is empty, false otherwise.

Reimplemented from **IPlaylist** (*p.137*).

void CSimplePlaylist::ReshuffleRandomEntries () `[virtual]`

Reshuffles the random links in the playlist. The current entry will be set as the first random entry.

Reimplemented from **IPlaylist** (*p.137*).

IPlaylistEntry CSimplePlaylist::SetCurrentEntry (IPlaylistEntry ** pEntry*)* `[virtual]`

Sets the current entry pointer to the given record. If the entry isn't in the list, then the current entry pointer is set to 0.

Reimplemented from **IPlaylist** (*p.137*).

IPlaylistEntry CSimplePlaylist::SetNextEntry (PlaylistMode* mode*)* `[virtual]`

Sets the current entry to the next entry in the playlist as determined by the mode. Returns a pointer to the new current entry, or 0 if the end of the list was reached (and the play mode isn't repeating).

Reimplemented from **IPlaylist** (*p.137*).

IPlaylistEntry CSimplePlaylist::SetPreviousEntry (PlaylistMode* mode*)* `[virtual]`

Sets the current entry to the previous entry in the playlist as determined by the mode. Returns a pointer to the new current entry, or 0 if the end of the list was reached (and the play mode isn't repeating).

Reimplemented from **IPlaylist** (*p.137*).

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

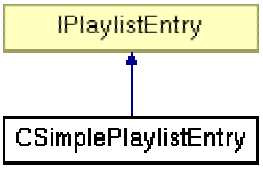
- **SimplePlaylist.h**

CSimplePlaylistEntry Class Reference

CSimplePlaylist (*p.100*) consists of CSimplePlaylistEntry objects, which define the necessary information for a playlist entry in this derived playlist. This is not part of the SDK proper, and is provided as a demonstration of an implementation of **IPlaylistEntry** (*p.139*).

`#include <SimplePlaylist.h>`

Inheritance diagram for CSimplePlaylistEntry:



Detailed Description

CSimplePlaylist (*p.100*) consists of CSimplePlaylistEntry objects, which define the necessary information for a playlist entry in this derived playlist. This is not part of the SDK proper, and is provided as a demonstration of an implementation of **IPlaylistEntry** (*p.139*).

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

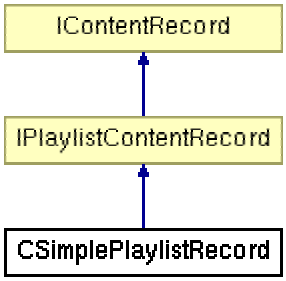
- **SimplePlaylist.h**

CSimplePlaylistRecord Class Reference

A demo derivation of the **IPlaylistContentRecord** (*p.138*) interface. This class is not part of the sdk proper.

`#include <SimpleContentManager.h>`

Inheritance diagram for CSimplePlaylistRecord:



Detailed Description

A demo derivation of the **IPlaylistContentRecord** (*p.138*) interface. This class is not part of the sdk proper.

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

- **SimpleContentManager.h**

CVolumeControl Class Reference

The volume control implementation is a non-thread safe singleton.

`#include <VolumeControl.h>`

Public Methods

- void **SetVolumeRange** (int iRange)
- void **SetVolumeRange** (int iValueCount, int *iValues)
- int **GetVolumeRange** () const
- int **VolumeUp** ()
- int **VolumeDown** ()
- int **GetVolume** () const
- int **SetVolume** (int level)
- void **SetTrebleRange** (int iRange)
- void **SetTrebleRange** (int iValueCount, int *iValues)
- int **GetTrebleRange** () const
- int **TrebleUp** ()
- int **TrebleDown** ()
- int **GetTreble** () const
- int **SetTreble** (int level)
- void **SetBassRange** (int iRange)
- void **SetBassRange** (int iValueCount, int *iValues)
- int **GetBassRange** () const
- int **BassUp** ()
- int **BassDown** ()
- int **GetBass** () const
- int **SetBass** (int level)

Static Public Methods

- CVolumeControl * **GetInstance** ()
- void **Destroy** ()

Detailed Description

The volume control implementation is a non-thread safe singleton.

Member Function Documentation

int CVolumeControl::BassDown ()

Lower the bass.

int CVolumeControl::BassUp ()

Raise the bass.

void CVolumeControl::Destroy () `[static]`

Destroy the singleton.

int CVolumeControl::GetBass () const

Get the current bass.

int CVolumeControl::GetBassRange () const

Get the current bass range.

CVolumeControl CVolumeControl::GetInstance ()* `[static]`

Get a pointer to the single instance of the volume control.

int CVolumeControl::GetTreble () const

Get the current treble.

int CVolumeControl::GetTrebleRange () const

Get the current treble range.

int CVolumeControl::GetVolume () const

Get the current volume.

int CVolumeControl::GetVolumeRange () const

Get the current volume range.

int CVolumeControl::SetBass (int **level***)*

Set the current bass.

void CVolumeControl::SetBassRange (int **iValueCount***, int ** **iValues***)*

Set the bass map and range.

void CVolumeControl::SetBassRange (int **iRange***)*

Set the bass range and let the control generate a map.

int CVolumeControl::SetTreble (int **level***)*

Set the current treble.

void CVolumeControl::SetTrebleRange (int **iValueCount***, int ** **iValues***)*

Set the treble map and range.

void CVolumeControl::SetTrebleRange (int **iRange***)*

Set the treble range and let the control generate a map.

int CVolumeControl::SetVolume (int **level***)*

Set the current volume.

void CVolumeControl::SetVolumeRange (int **iValueCount***, int ** **iValues***)*

Specify a volume map and range.

void CVolumeControl::SetVolumeRange (int **iRange***)*

Set a volume range and let the control generate a volume map.

int CVolumeControl::TrebleDown ()

Lower the treble.

int CVolumeControl::TrebleUp ()

Raise the treble.

int CVolumeControl::VolumeDown ()

Lower the volume.

int CVolumeControl::VolumeUp ()

Raise the volume.

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

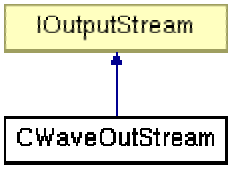
- **VolumeControl.h**

CWaveOutStream Class Reference

PCM is written to the audio driver through the CWaveOutStream class. This class does not perform any operations on the audio data, and assumes that any necessary sample rate conversion has already been applied.

`#include <WaveOut.h>`

Inheritance diagram for CWaveOutStream:



Public Methods

- int **Ioctl** (int Key, void *Value)

Detailed Description

PCM is written to the audio driver through the CWaveOutStream class. This class does not perform any operations on the audio data, and assumes that any necessary sample rate conversion has already been applied.

Member Function Documentation

int CWaveOutStream::Ioctl (int Key*, void** * Value*)* `[virtual]`

CWaveOutStream supports configuration of hardware sample rate conversion through an ioctl.

Reimplemented from **IOutputStream** (*p.133*).

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

- **WaveOut.h**

drive_geometry_s Struct Reference

This structure is used in our block device interface to specify information about a physical disk. In situations where the physical layout of a device does not match the C/H/S style hard drive layout, it is acceptable to specify the device as having 1 cylinder, 1 head, and a number of sectors equal to the total blocks.

`#include <blk_dev.h>`

Public Attributes

- cyg_uint16 **cyl**
- cyg_uint16 **hd**
- cyg_uint16 **sec**
- cyg_uint16 **bytes_p_sec**
- cyg_uint32 **num_blks**
- unsigned char **serial_num** [40+1]
- unsigned char **model_num** [40+1]

Detailed Description

This structure is used in our block device interface to specify information about a physical disk. In situations where the physical layout of a device does not match the C/H/S style hard drive layout, it is acceptable to specify the device as having 1 cylinder, 1 head, and a number of sectors equal to the total blocks.

Member Data Documentation

cyg_uint16 drive_geometry_s::bytes_p_sec

Bytes per sector.

cyg_uint16 drive_geometry_s::cyl

Number of cylinders.

cyg_uint16 drive_geometry_s::hd

Number of heads.

unsigned char drive_geometry_s::model_num[40 + 1]

Model number for the drive.

cyg_uint32 drive_geometry_s::num_blks

Total blocks (C*H*S).

cyg_uint16 drive_geometry_s::sec

Number of sectors.

unsigned char drive_geometry_s::serial_num[40 + 1]

Serial number for the drive.

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

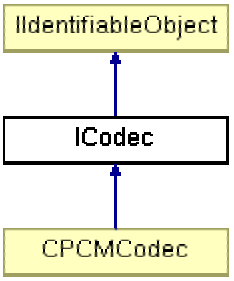
- **blk_dev.h**

ICodec Class Reference

Dadio(tm) relies on an abstract interface for dealing with arbitrary codecs. This interface is the ICodec interface, as defined in ./player/codec/common/include/Codec.h. The purpose of this document is to define the ICodec interface and each of its member functions, and to describe their expected behavior in the Dadio(tm) OS.

`#include <Codec.h>`

Inheritance diagram for ICodec:



Public Methods

- virtual ERESULT **DecodeFrame** (unsigned long &TimePos)=0
- virtual ERESULT **SetSong** (**IInputStream** *pInputStream, **stream_info_t** &streamInfo, **IMetadata** *pMetadata=0)=0
- virtual ERESULT **Seek** (unsigned long &secSeek)=0
- virtual ERESULT **GetMetadata** (**IMetadata** *pMetadata, **IInputStream** *pInputStream)=0
- virtual rbuf_writer_t * **GetWriteBuf** () const=0
- virtual void **SetWriteBuf** (rbuf_writer_t *pW)=0
- virtual int **GetOutputUnitSize** () const=0
- virtual **IInputStream** * **GetInputStream** () const=0
- virtual void **Stats** ()=0

Detailed Description

Dadio(tm) relies on an abstract interface for dealing with arbitrary codecs. This interface is the ICodec interface, as defined in ./player/codec/common/include/Codec.h. The purpose of this document is to define the ICodec interface and each of its member functions, and to describe their expected behavior in the Dadio(tm) OS.

Member Function Documentation

virtual ERESULT ICodec::DecodeFrame (unsigned long & TimePos*)* `[pure virtual]`

Decode a frame of audio data. Calculate the current time offset within the stream (in seconds) and store this value in TimePos.

virtual IInputStream ICodec::GetInputStream () const* `[pure virtual]`

Return a pointer to the current input stream.

*virtual ERESULT ICodec::GetMetadata (IMetadata ** pMetadata*, IInputStream ** pInputStream*)* `[pure virtual]`

Attempt to fetch metadata for a given track. If pInputStream is specified, it will attempt to open pInputStream and find metadata in that stream. Otherwise, it will provide metadata found for the currently set track.

virtual int ICodec::GetOutputUnitSize () const `[pure virtual]`

Indicate how much data will be produced by the codec per **DecodeFrame**() (*p.113*) call. This is used to appropriately shape buffers after the codec in the data stream.

virtual rbuf_writer_t ICodec::GetWriteBuf () const* `[pure virtual]`

Return a handle to the write buffer for this codec.

virtual ERESULT ICodec::Seek (unsigned long & secSeek*)* `[pure virtual]`

Attempt to seek to the specified time offset within the file (in seconds). If the input stream does not support seeking, it is appropriate to return CODEC_FAIL here.

*virtual ERESULT ICodec::SetSong (IInputStream ** pInputStream*, stream_info_t &* streamInfo*, IMetadata ** pMetadata *= 0)* `[pure virtual]`

Initialize the codec to use the given input stream. If the codec can correctly interpret the stream format, it must populate the streamInfo structure with the appropriate information regarding the stream. If metadata is available and the pMetadata argument is non-null, then pMetadata should be populated with the available metadata for the stream.

Reimplemented in **CPCMCodec** (*p.85*).

*virtual void ICodec::SetWriteBuf (rbuf_writer_t ** pW*)* `[pure virtual]`

Set the write buffer for this codec. The write buffer is used to store decoded data. After **SetSong**() (*p.113*) has successfully completed, the codec will be associated with a write buffer for it to store decoded data in.

virtual void ICodec::Stats () `[pure virtual]`

Routine that can optionally be used to print diagnostic information about the codec after a stream has completed.

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

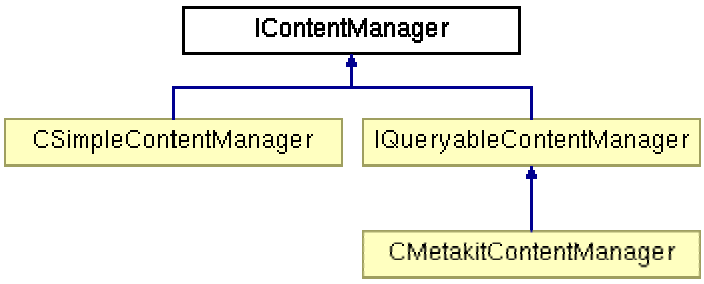
- **Codec.h**

IContentManager Class Reference

The IContentManager provides record tracking for media content accross several data sources.

`#include <ContentManager.h>`

Inheritance diagram for IContentManager:



Public Methods

- virtual **IMetadata** * **CreateMetadataRecord** () const=0
- virtual void **Clear** ()=0
- virtual void **AddContentRecords** (**content_record_update_t** *pContentUpdate)=0
- virtual int **GetMediaRecordCount** ()=0
- virtual **IMediaContentRecord** * **AddMediaRecord** (**media_record_info_t** &mediaContent, bool *pbAlreadyExists=0)=0
- virtual void **DeleteMediaRecord** (int iContentRecordID)=0
- virtual **IMediaContentRecord** * **GetMediaRecord** (int iContentRecordID)=0
- virtual **IMediaContentRecord** * **GetMediaRecord** (const char *szURL)=0
- virtual int **GetPlaylistRecordCount** ()=0
- virtual **IPlaylistContentRecord** * **AddPlaylistRecord** (**playlist_record_t** &playlistRecord)=0
- virtual void **DeletePlaylistRecord** (int iContentRecordID)=0
- virtual void **DeleteRecordsFromDataSource** (int iDataSourceID)=0
- virtual void **MarkRecordsFromDataSourceUnverified** (int iDataSourceID)=0
- virtual void **DeleteUnverifiedRecordsFromDataSource** (int iDataSourceID)=0
- virtual void **GetAllMediaRecords** (MediaRecordList &records) const=0
- virtual void **GetMediaRecordsByDataSourceID** (MediaRecordList &records, int iDataSourceID) const=0
- virtual void **GetAllPlaylistRecords** (PlaylistRecordList &records) const=0
- virtual void **GetPlaylistRecordsByDataSourceID** (PlaylistRecordList &records, int iDataSourceID) const=0

Detailed Description

The IContentManager provides record tracking for media content accross several data sources.

Member Function Documentation

*virtual void IContentManager::AddContentRecords (content_record_update_t * pContentUpdate) [pure virtual]*

Adds a list of media and playlist records to the content manager.

Reimplemented in **CMetakitContentManager** (*p.72*), and **CSimpleContentManager** (*p.94*).

virtual IMediaContentRecord IContentManager::AddMediaRecord (media_record_info_t & mediaContent, bool * pbAlreadyExists = 0) [pure virtual]*

Adds a media entry to the content manager. Returns a pointer to the content record. If pbAlreadyExists is not null, then its value is set to true if the content record with the same URL already exists in the content manager, false otherwise.

Reimplemented in **CMetakitContentManager** (*p.73*), and **CSimpleContentManager** (*p.94*).

virtual IPlaylistContentRecord IContentManager::AddPlaylistRecord (playlist_record_t & playlistRecord) [pure virtual]*

Adds a playlist record to the content manager. Returns a pointer to the record.

Reimplemented in **CMetakitContentManager** (*p.73*), and **CSimpleContentManager** (*p.94*).

virtual void IContentManager::Clear () [pure virtual]

Clears the content manager and deletes its content records.

Reimplemented in **CMetakitContentManager** (*p.73*), and **CSimpleContentManager** (*p.94*).

virtual IMetadata IContentManager::CreateMetadataRecord () const [pure virtual]*

Creates an empty metadata record.

Reimplemented in **CSimpleContentManager** (*p.94*).

virtual void IContentManager::DeleteMediaRecord (int iContentRecordID) [pure virtual]

Removes a media record from the content manager.

Reimplemented in **CMetakitContentManager** (*p.73*), and **CSimpleContentManager** (*p.94*).

virtual void IContentManager::DeletePlaylistRecord (int iContentRecordID) [pure virtual]

Removes a playlist record from the content manager.

Reimplemented in **CMetakitContentManager** (*p.73*), and **CSimpleContentManager** (*p.94*).

virtual void IContentManager::DeleteRecordsFromDataSource (int iDataSourceID*)* `[pure virtual]`

Removes all records from the given data source from the content manager.

Reimplemented in **CMetakitContentManager** (*p.73*), and **CSimpleContentManager** (*p.95*).

virtual void IContentManager::DeleteUnverifiedRecordsFromDataSource (int iDataSourceID*)* `[pure virtual]`

Removes all records from the given data source that are marked as unverified.

Reimplemented in **CMetakitContentManager** (*p.74*), and **CSimpleContentManager** (*p.95*).

virtual void IContentManager::GetAllMediaRecords (MediaRecordList & records*) const* `[pure virtual]`

Appends all media content records in the manager to the record list passed in.

Reimplemented in **CMetakitContentManager** (*p.74*).

virtual void IContentManager::GetAllPlaylistRecords (PlaylistRecordList & records*) const* `[pure virtual]`

Appends all playlist content records in the manager to the record list passed in.

Reimplemented in **CMetakitContentManager** (*p.74*).

virtual IMediaContentRecord IContentManager::GetMediaRecord (const char ** szURL*)* `[pure virtual]`

Returns a pointer to the media content record with the specified URL. Returns 0 if no record with a matching URL was found.

Reimplemented in **CMetakitContentManager** (*p.75*), and **CSimpleContentManager** (*p.95*).

virtual IMediaContentRecord IContentManager::GetMediaRecord (int* iContentRecordID*)* `[pure virtual]`

Returns a pointer to the media content record with the specified record ID. Returns 0 if no record with a matching ID was found.

Reimplemented in **CMetakitContentManager** (*p.75*), and **CSimpleContentManager** (*p.95*).

virtual int IContentManager::GetMediaRecordCount () `[pure virtual]`

Returns the number of media records in the content manager.

Reimplemented in **CMetakitContentManager** (*p.75*), and **CSimpleContentManager** (*p.95*).

virtual void IContentManager::GetMediaRecordsByDataSourceID (MediaRecordList & records*, int* iDataSourceID*) const* `[pure virtual]`

Appends all media content records from the given data source in the manager to the record list passed in.

Reimplemented in **CMetakitContentManager** (*p.76*).

virtual int IContentManager::GetPlaylistRecordCount () `[pure virtual]`

Returns the number of playlist records in the content manager.

Reimplemented in **CMetakitContentManager** (*p.76*), and **CSimpleContentManager** (*p.95*).

virtual void IContentManager::GetPlaylistRecordsByDataSourceID (PlaylistRecordList & records*, int* iDataSourceID*) const* `[pure virtual]`

Appends all playlist content records from the given data source in the manager to the record list passed in.

Reimplemented in **CMetakitContentManager** (*p.76*).

virtual void IContentManager::MarkRecordsFromDataSourceUnverified (int iDataSourceID*)* `[pure virtual]`

Marks all records from the given data source as unverified.

Reimplemented in **CMetakitContentManager** (*p.76*), and **CSimpleContentManager** (*p.95*).

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

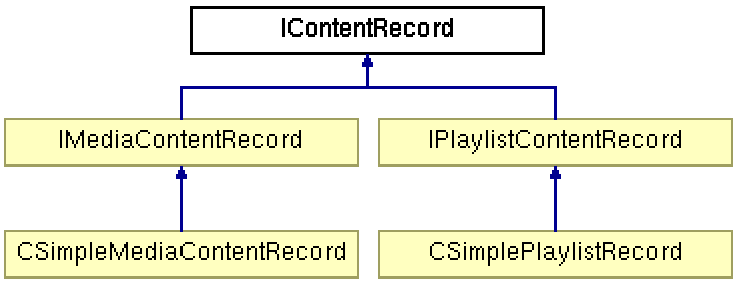
- **ContentManager.h**

IContentRecord Class Reference

The base class for both media content records and playlist content records. The content record has an ID unique across the content manager, a URL unique across the system, and the ID of the data source it's stored on. Content records are also considered verified if they've been found to reside on a data source (as opposed to solely existing in the database from a previous run). They also have a status which specifies their ability to be run through a codec for playback.

`#include <ContentManager.h>`

Inheritance diagram for IContentRecord:



Public Types

- enum **ContentRecordStatus** { **CR_OKAY** = 0, **CR_DRM_FAILURE**, **CR_DECODE_ERROR**, **CR_NO_CODEC** }

Public Methods

- virtual int **GetID** () const=0
- virtual const char * **GetURL** () const=0
- virtual int **GetDataSourceID** () const=0
- virtual bool **IsVerified** () const=0
- virtual void **SetVerified** (bool bVerified)=0
- virtual **ContentRecordStatus GetStatus** () const=0
- virtual void **SetStatus** (**ContentRecordStatus** status)=0

Detailed Description

The base class for both media content records and playlist content records. The content record has an ID unique across the content manager, a URL unique across the system, and the ID of the data source it's stored on. Content records are also considered verified if they've been found to reside on a data source (as opposed to solely existing in the database from a previous run). They also have a status which specifies their ability to be run through a codec for playback.

Member Enumeration Documentation

enum IContentRecord::ContentRecordStatus

Describes the playability status of this content record.

Enumeration values:

CR_OKAY The content record is either good or hasn't been tested.
CR_DRM_FAILURE The content record can't be played because of licensing issues.
CR_DECODE_ERROR Errors in decoding prevent this track from being played.
CR_NO_CODEC No matching codec for this content record was found.

Member Function Documentation

virtual int IContentRecord::GetDataSourceID () const `[pure virtual]`

Returns the ID of the data source this content record comes from.

virtual int IContentRecord::GetID () const `[pure virtual]`

Returns the ID of the content record, which must be unique across records in the manager.

virtual ContentRecordStatus IContentRecord::GetStatus () const `[pure virtual]`

Gets the status of the content record.

virtual const char IContentRecord::GetURL () const* `[pure virtual]`

Returns the unique URL of the content record.

virtual bool IContentRecord::IsVerified () const `[pure virtual]`

Returns true if the file has been verified as existing on its data source.

virtual void IContentRecord::SetStatus (ContentRecordStatus status*)* `[pure virtual]`

Sets the status of the content record. Called primarily from the media player during SetSong. This is separate from verified/unverified so that bad content can be left in the content manager after refreshes (instead of flushing bad content and adding it again on the refresh).

virtual void IContentRecord::SetVerified (bool bVerified*)* `[pure virtual]`

Set this content records as verified (i.e., existing on a data source) or unverified.

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

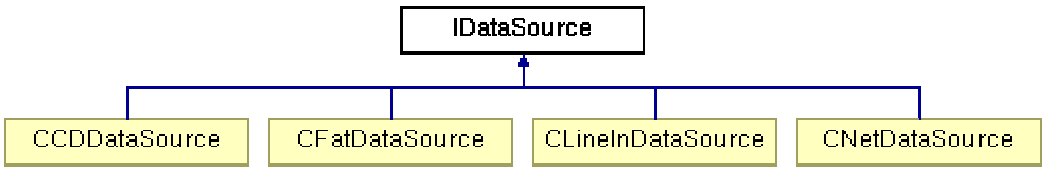
- **ContentManager.h**

IDataSource Class Reference

The data source is an abstract representation of a provider of content. It has methods for obtaining a list of content on the data source, fetching metadata for that content, and creating input streams for a given piece of content.

`#include <DataSource.h>`

Inheritance diagram for IDataSource:



Public Types

- enum **RefreshMode** { **DSR_DEFAULT** = 0, **DSR_ONE_PASS**, **DSR_ONE_PASS_WITH_METADATA**, **DSR_TWO_PASS** }

Public Methods

- virtual int **GetClassID** () const=0
- virtual int **GetInstanceID** () const=0
- virtual bool **GetRootURLPrefix** (char *szRootURLPrefix, int iMaxLength) const=0
- virtual void **SetDefaultRefreshMode** (**RefreshMode** mode)=0
- virtual **RefreshMode GetDefaultRefreshMode** () const=0
- virtual void **SetDefaultUpdateChunkSize** (int iUpdateChunkSize)=0
- virtual int **GetDefaultUpdateChunkSize** () const=0
- virtual ERESULT **ListAllEntries** (**RefreshMode** mode, int iUpdateChunkSize)=0
- virtual void **GetContentMetadata** (**content_record_update_t** *pContentUpdate)=0
- virtual **IInputStream * OpenInputStream** (const char *szURL)=0
- virtual **IOutputStream * OpenOutputStream** (const char *szURL)=0

Protected Methods

- virtual void **SetInstanceID** (int iDataSourceID)=0

Detailed Description

The data source is an abstract representation of a provider of content. It has methods for obtaining a list of content on the data source, fetching metadata for that content, and creating input streams for a given piece of content.

Member Enumeration Documentation

enum IDataSource::RefreshMode

The refresh mode specifies how a list of content is to be obtained from the data source.

Enumeration values:

DSR_DEFAULT The default method for the data source, chosen from one of the settings below.
DSR_ONE_PASS List all available entries in one pass without fetching metadata.
DSR_ONE_PASS_WITH_METADATA List all available entries in one pass and try to fetch metadata for each entry.
DSR_TWO_PASS List all available entries in one pass without fetching metadata. The play manager will then call GetMetadata for each new entry in the content manager.

Member Function Documentation

virtual int IDataSource::GetClassID () const `[pure virtual]`

Returns an ID unique to the type of data source instantiated by the object.

*virtual void IDataSource::GetContentMetadata (content_record_update_t * pContentUpdate)* `[pure virtual]`

Retrieves metadata for each media content record in the passed-in list.

Reimplemented in **CCDDataSource** (*p.41*), **CFatDataSource** (*p.49*), and **CLineInDataSource** (*p.61*).

virtual RefreshMode IDataSource::GetDefaultRefreshMode () const `[pure virtual]`

Returns the current default refresh mode of the data source.

Reimplemented in **CCDDataSource** (*p.42*), **CFatDataSource** (*p.50*), **CLineInDataSource** (*p.62*), and **CNetDataSource** (*p.78*).

virtual int IDataSource::GetDefaultUpdateChunkSize () const `[pure virtual]`

Returns the default update chunk size of the data source This value is the maximum number of records that will be sent for each EVENT_CONTENT_UPDATE message. 0 indicates that all records will be sent in one single message.

Reimplemented in **CCDDataSource** (*p.42*), **CFatDataSource** (*p.50*), **CLineInDataSource** (*p.62*), and **CNetDataSource** (*p.79*).

virtual int IDataSource::GetInstanceID () const `[pure virtual]`

Returns an ID unique to the instantiation of the object.

*virtual bool IDataSource::GetRootURLPrefix (char * szRootURLPrefix, int iMaxLength) const [pure virtual]*

Copies the string the data source uses to prefix its URLs into the given string provided.

Reimplemented in **CCDDataSource** (*p.42*), **CFatDataSource** (*p.50*), and **CLineInDataSource** (*p.62*).

virtual ERESULT IDataSource::ListAllEntries (RefreshMode mode, int iUpdateChunkSize) [pure virtual]

Asks the source to pass content updates lists through the event system that contain all of the content it can access.

Parameters:

mode Specifies if metadata should be retrieved in one pass, two passes, or not at all.
iUpdateChunkSize Specifies how many media records to send back at a time. If iUpdateChunkSize is zero, then all records are sent back at once.

Reimplemented in **CCDDataSource** (*p.42*), **CFatDataSource** (*p.50*), and **CLineInDataSource** (*p.62*).

virtual IInputStream IDataSource::OpenInputStream (const char * szURL) [pure virtual]*

Asks the source to open this URL for reading. Returns 0 if the URL was unable to be opened, otherwise it returns the proper subclass of **IInputStream** (*p.127*) for this file type.

Reimplemented in **CCDDataSource** (*p.42*), **CFatDataSource** (*p.50*), **CLineInDataSource** (*p.62*), and **CNetDataSource** (*p.79*).

virtual IOutputStream IDataSource::OpenOutputStream (const char * szURL) [pure virtual]*

Asks the source to open this URL for writing. Returns 0 if the URL was unable to be opened, otherwise it returns the proper subclass of **IOutputStream** (*p.132*) for this file type.

Reimplemented in **CCDDataSource** (*p.42*), **CFatDataSource** (*p.50*), **CLineInDataSource** (*p.62*), and **CNetDataSource** (*p.79*).

virtual void IDataSource::SetDefaultRefreshMode (RefreshMode mode) [pure virtual]

Sets the default refresh mode of the data source. If DSR_DEFAULT is passed in, then the original default refresh setting for the data source should be used.

Reimplemented in **CCDDataSource** (*p.43*), **CFatDataSource** (*p.50*), **CLineInDataSource** (*p.62*), and **CNetDataSource** (*p.79*).

virtual void IDataSource::SetDefaultUpdateChunkSize (int iUpdateChunkSize) [pure virtual]

Sets the default update chunk size of the data source This value is the maximum number of records that should be sent for each EVENT_CONTENT_UPDATE message. 0 indicates that all records should be sent in one single message. DS_DEFAULT_CHUNK_SIZE tells the data source to use its original default chunk size.

Reimplemented in **CCDDataSource** (*p.43*), **CFatDataSource** (*p.51*), **CLineInDataSource** (*p.62*), and **CNetDataSource** (*p.79*).

virtual void IDataSource::SetInstanceID (int iDataSourceID*)* `[protected, pure virtual]`

Called by the data source manager to assign an ID to this data source.

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

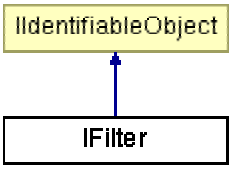
- **DataSource.h**

IFilter Struct Reference

Dadio(tm) uses an asbtract interface for filters, which allow components to examine and modify the current PCM audio stream. This is the IFilter interface, as defined in ./player/datastream/filter/include/Filter.h. The purpose of this document is to define the IFilter interface and each of its member functions, and to describe their expected behavior in the Dadio(tm) OS.

`#include <Filter.h>`

Inheritance diagram for IFilter:



Public Methods

- virtual ERESULT **DoWork** ()=0
- virtual ERESULT **Ioctl** (int Key, void *Value)=0
- virtual int **GetInputUnitSize** () const=0
- virtual int **GetOutputUnitSize** () const=0
- virtual int **SetWriteBuf** (rbuf_writer_t *WriteBuf)=0
- virtual int **SetReadBuf** (rbuf_reader_t *ReadBuf)=0
- virtual rbuf_writer_t * **GetWriteBuf** () const=0
- virtual rbuf_reader_t * **GetReadBuf** () const=0

Detailed Description

Dadio(tm) uses an asbtract interface for filters, which allow components to examine and modify the current PCM audio stream. This is the IFilter interface, as defined in ./player/datastream/filter/include/Filter.h. The purpose of this document is to define the IFilter interface and each of its member functions, and to describe their expected behavior in the Dadio(tm) OS.

Member Function Documentation

virtual ERESULT IFilter::DoWork () `[pure virtual]`

Process a chunk of audio data. The size of a chunk depends on the input space, output space, and the amount of data the filter can process. This routine is called until the filter indicates that no work is available.

virtual int IFilter::GetInputUnitSize () const `[pure virtual]`

Give the media player an idea of the minimum input unit for this filter. For example, a filter that requires 128 samples per channel of 16 bit stereo PCM to operate would return (16bits / 8) * 128 * 2 = 512.

virtual int IFilter::GetOutputUnitSize () const `[pure virtual]`

Give the media player an idea of the minimum output unit for this filter. For the most part, this will either be equal to the input unit size, unless the filter changes the amount of data being pushed through.

virtual rbuf_reader_t IFilter::GetReadBuf () const* `[pure virtual]`

Called by the media player to find out what read handle we are using. This is typically called at the end of a song, when the media player wants to clean up the playstream.

virtual rbuf_writer_t IFilter::GetWriteBuf () const* `[pure virtual]`

Called by the media player to find out what write handle we are using. This is typically called at the end of a song, when the media player wants to clean up the playstream.

virtual ERESULT IFilter::Ioctl (int Key*, void** Value*)* `[pure virtual]`

A generic Ioctl interface to allow custom configuration.

*virtual int IFilter::SetReadBuf (rbuf_reader_t** ReadBuf*)* `[pure virtual]`

Called by the media player to assign a read handle for this filter. When the **DoWork**() (*p.124*) call is issued, the filter will be expected to read data it processes from this handle.

*virtual int IFilter::SetWriteBuf (rbuf_writer_t** WriteBuf*)* `[pure virtual]`

Called by the media player to assign a write handle for this filter. When the **DoWork**() (*p.124*) call is issued, the filter will be expected to write data it processes to this handle.

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

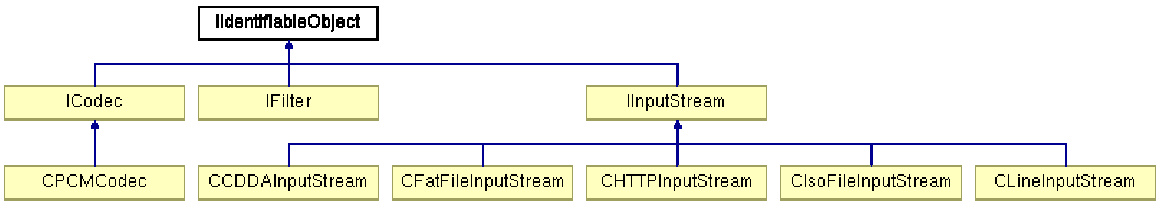
- **Filter.h**

IIdentifiableObject Struct Reference

The IIdentifiableObject interface. The methods defined by this interface do not need to be implemented directly ever.

`#include <IdentifiableObject.h>`

Inheritance diagram for IIdentifiableObject:



Detailed Description

The IIdentifiableObject interface. The methods defined by this interface do not need to be implemented directly ever.

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

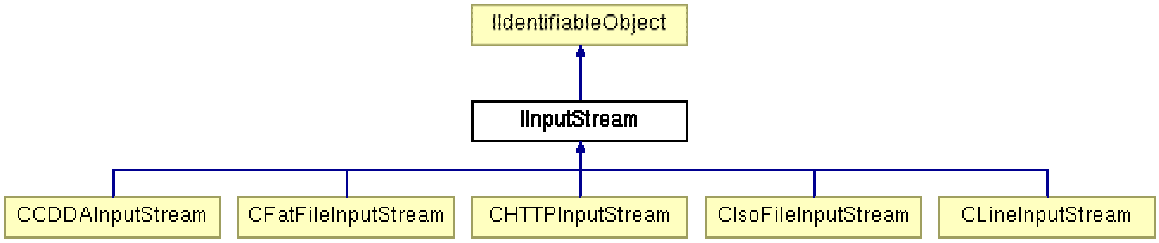
- **IdentifiableObject.h**

IInputStream Struct Reference

The IInputStream interface defines the basic interface for read only stream access. Additionally, support is provided for seekable streams.

`#include <InputStream.h>`

Inheritance diagram for IInputStream:



Public Types

- enum **InputSeekPos** { **SeekStart**, **SeekCurrent**, **SeekEnd** }

Public Methods

- virtual ERESULT **Open** (const char *Source)=0
- virtual ERESULT **Close** ()=0
- virtual int **Read** (void *Buffer, int Count)=0
- virtual int **Ioctl** (int Key, void *Value)=0
- virtual int **GetInputUnitSize** ()=0
- virtual bool **CanSeek** () const=0
- virtual int **Seek** (**InputSeekPos** SeekOrigin, int Offset)=0
- virtual int **Position** () const=0
- virtual int **Length** () const=0

Detailed Description

The IInputStream interface defines the basic interface for read only stream access. Additionally, support is provided for seekable streams.

Member Enumeration Documentation

enum IInputStream::InputSeekPos

Enumeration for seek origin.

Member Function Documentation

virtual bool IInputStream::CanSeek () const `[pure virtual]`

Allows the input stream to indicate whether or not seek is supported.

virtual ERESULT IInputStream::Close () `[pure virtual]`

Close the input stream if currently open.

virtual int IInputStream::GetInputUnitSize () `[pure virtual]`

Called by codecs to determine the minimum unit to read in from this input stream. Not all codecs have been updated to use this routine.

virtual int IInputStream::Ioctl (int Key*, void ** Value*)* `[pure virtual]`

A generic interface for controlling the input stream.

virtual int IInputStream::Length () const `[pure virtual]`

If available, return the length of the stream.

*virtual ERESULT IInputStream::Open (const char ** Source*)* `[pure virtual]`

Open the input stream to the given source.

Reimplemented in **CCDDAInputStream** (*p.40*).

virtual int IInputStream::Position () const `[pure virtual]`

Give back our current position within the stream.

*virtual int IInputStream::Read (void ** Buffer*, int* Count*)* `[pure virtual]`

Read the specified number of bytes into Buffer.

virtual int IInputStream::Seek (InputSeekPos SeekOrigin*, int* Offset*)* `[pure virtual]`

Attempt to seek on the input stream.

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

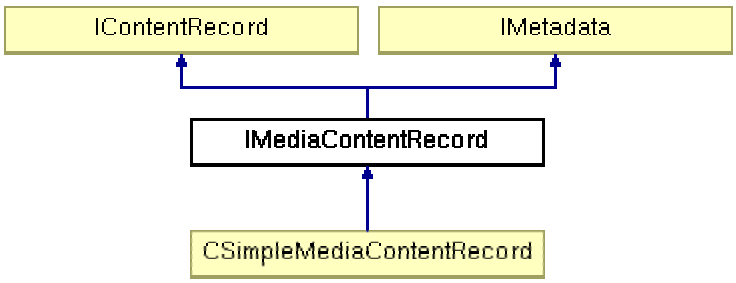
- **InputStream.h**

IMediaContentRecord Class Reference

A subclass of both **IContentRecord** (*p.118*) and **IMetadata** (*p.130*), used for representing tracks that can be decoded by the media player. In addition to the functionality of the **IContentRecord** (*p.118*) and **IMetadata** (*p.130*) classes, the media content record also retains a codec ID.

`#include <ContentManager.h>`

Inheritance diagram for IMediaContentRecord:



Public Methods

- virtual int **GetCodecID** () const=0

Detailed Description

A subclass of both **IContentRecord** (*p.118*) and **IMetadata** (*p.130*), used for representing tracks that can be decoded by the media player. In addition to the functionality of the **IContentRecord** (*p.118*) and **IMetadata** (*p.130*) classes, the media content record also retains a codec ID.

Member Function Documentation

virtual int IMediaContentRecord::GetCodecID () const `[pure virtual]`

Returns the ID of the codec used to decode this track.

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

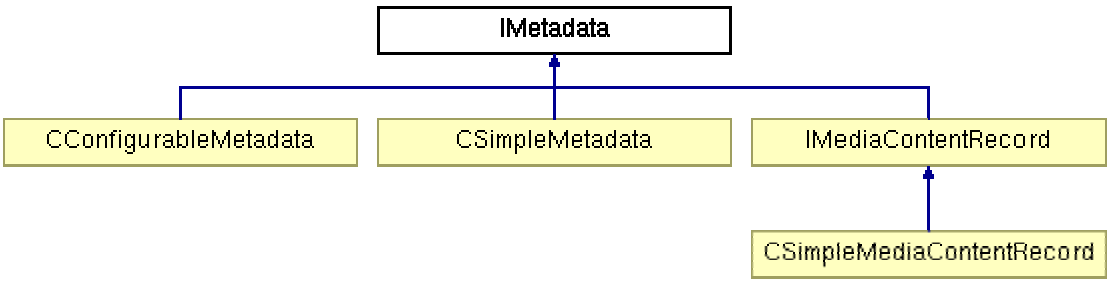
- **ContentManager.h**

IMetadata Class Reference

The metadata class is an abstract representation of a collection of key/value pairs for a track. Functions are provided for setting, getting, and clearing attributes.

`#include <Metadata.h>`

Inheritance diagram for IMetadata:



Public Methods

- virtual IMetadata * **Copy** () const=0
- virtual bool **UsesAttribute** (int iAttributeID) const=0
- virtual ERESULT **SetAttribute** (int iAttributeID, void *pAttributeValue)=0
- virtual ERESULT **UnsetAttribute** (int iAttributeID)=0
- virtual ERESULT **GetAttribute** (int iAttributeID, void **ppAttributeValue) const=0
- virtual void **ClearAttributes** ()=0
- virtual void **MergeAttributes** (const IMetadata *pMetadata, bool bOverwrite=false)=0

Detailed Description

The metadata class is an abstract representation of a collection of key/value pairs for a track. Functions are provided for setting, getting, and clearing attributes.

Member Function Documentation

virtual void IMetadata::ClearAttributes () [pure virtual]

Clears the value of all attributes.

virtual IMetadata IMetadata::Copy () const [pure virtual]*

Makes a copy of the metadata object.

virtual ERESULT IMetadata::GetAttribute (int iAttributeID*, void ** ppAttributeValue*) const [pure virtual]*

Gets the value of an attribute.

Return values:

METADATA_NO_ERROR The value was retrieved successfully.
METADATA_NOT_USED The attribute isn't used by this metadata object.
METADATA_NO_VALUE_SET The attribute is used by this metadata object but no value has been set.

virtual void IMetadata::MergeAttributes (const IMetadata * **pMetadata**, *bool* **bOverwrite** = *false*) *[pure virtual]*

Merges the attributes used by this metadata object with the values for those attributes from the metadata object passed in. Attributes used in the passed-in metadata that are not used by this object are not copied.

Parameters:

bOverwrite If false, only attributes in this object that have no value set are merged.
bOverwrite If true, all attributes in this object are merged, even if a previous value has been set.

virtual ERESULT IMetadata::SetAttribute (int **iAttributeID**, *void* * **pAttributeValue**) *[pure virtual]*

Sets a value of an attribute.

Return values:

METADATA_NO_ERROR The value was set successfully.
METADATA_NOT_USED The attribute isn't used by this metadata object.

virtual ERESULT IMetadata::UnsetAttribute (int **iAttributeID**) *[pure virtual]*

Unsets a value of an attribute and frees any memory associated with it.

Return values:

METADATA_NO_ERROR The value was unset successfully.
METADATA_NOT_USED The attribute isn't used by this metadata object.

virtual bool IMetadata::UsesAttribute (int **iAttributeID**) *const [pure virtual]*

Returns true if the attribute is used by this metadata object, false otherwise.

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

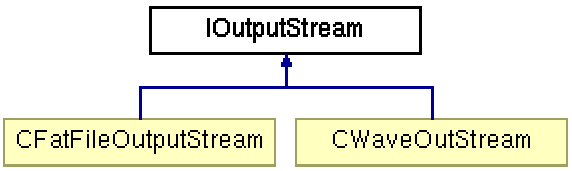
- **Metadata.h**

IOutputStream Struct Reference

Interface for output streams. If you elect to not use the registration interface for output streams, you still must use IDENTIFY_OBJECT in your class definition.

`#include <OutputStream.h>`

Inheritance diagram for IOutputStream:



Public Types

- enum **OutputSeekPos** { **SeekStart**, **SeekCurrent**, **SeekEnd** }

Public Methods

- virtual ERESULT **Open** (const char *Source)=0
- virtual ERESULT **Close** ()=0
- virtual int **Write** (const void *Buffer, int Count)=0
- virtual int **Ioctl** (int Key, void *Value)=0
- virtual bool **Flush** ()=0
- virtual int **GetOutputUnitSize** ()=0
- virtual bool **CanSeek** ()=0
- virtual int **Seek** (**OutputSeekPos** SeekOrigin, int Offset)=0

Detailed Description

Interface for output streams. If you elect to not use the registration interface for output streams, you still must use IDENTIFY_OBJECT in your class definition.

Member Enumeration Documentation

enum IOutputStream::OutputSeekPos

Enumeration for seek origin.

Member Function Documentation

virtual bool IOutputStream::CanSeek () `[pure virtual]`

Indicate whether or not this output stream supports seeking.

virtual ERESULT IOutputStream::Close () `[pure virtual]`

Close the output stream if currently open.

virtual bool IOutputStream::Flush () `[pure virtual]`

If possible, flush the output stream, forcing all buffered data to be written.

virtual int IOutputStream::GetOutputUnitSize () `[pure virtual]`

Give an indication as to how large of a unit we write out.

virtual int IOutputStream::Ioctl (int Key*, void ** Value*)* `[pure virtual]`

A generic interface to issue proprietary commands to the output stream.

Reimplemented in **CWaveOutStream** (*p.109*).

*virtual ERESULT IOutputStream::Open (const char ** Source*)* `[pure virtual]`

Attempt to open the given location.

virtual int IOutputStream::Seek (OutputSeekPos SeekOrigin*, int* Offset*)* `[pure virtual]`

Attempt to seek the output stream.

*virtual int IOutputStream::Write (const void ** Buffer*, int* Count*)* `[pure virtual]`

Write the given number of bytes from the buffer to the stream.

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

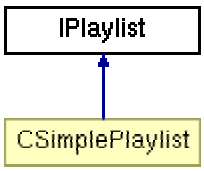
- **OutputStream.h**

IPlaylist Class Reference

The playlist class is an abstract base class that describes functions to access a sequential list of content records.

`#include <Playlist.h>`

Inheritance diagram for IPlaylist:



Public Types

- enum **PlaylistMode** { **NORMAL** = 0, **RANDOM**, **REPEAT_ALL**, **REPEAT_RANDOM**, **REPEAT_TRACK** }

Public Methods

- virtual const char * **GetName** () const=0
- virtual void **Clear** ()=0
- virtual int **GetSize** ()=0
- virtual bool **IsEmpty** () const=0
- virtual **IPlaylistEntry** * **AddEntry** (**IMediaContentRecord** *pContentRecord)=0
- virtual **IPlaylistEntry** * **InsertEntry** (**IMediaContentRecord** *pContentRecord, int index)=0
- virtual void **AddEntries** (MediaRecordList &records)=0
- virtual void **DeleteEntry** (**IPlaylistEntry** *pEntry)=0
- virtual void **DeleteEntriesFromDataSource** (int iDataSourceID)=0
- virtual void **ReshuffleRandomEntries** ()=0
- virtual **IPlaylistEntry** * **GetEntry** (int index, **PlaylistMode** mode=NORMAL)=0
- virtual int **GetEntryIndex** (const **IPlaylistEntry** *pEntry)=0
- virtual **IPlaylistEntry** * **GetCurrentEntry** ()=0
- virtual **IPlaylistEntry** * **SetCurrentEntry** (**IPlaylistEntry** *pEntry)=0
- virtual **IPlaylistEntry** * **GetNextEntry** (**IPlaylistEntry** *pEntry, **PlaylistMode** mode)=0
- virtual **IPlaylistEntry** * **SetNextEntry** (**PlaylistMode** mode)=0
- virtual **IPlaylistEntry** * **GetPreviousEntry** (**IPlaylistEntry** *pEntry, **PlaylistMode** mode)=0
- virtual **IPlaylistEntry** * **SetPreviousEntry** (**PlaylistMode** mode)=0

Detailed Description

The playlist class is an abstract base class that describes functions to access a sequential list of content records.

Member Enumeration Documentation

enum IPlaylist::PlaylistMode

Used to determine how to traverse the list of entries.

Enumeration values:

NORMAL Use the normal sequence when determining the next track.
RANDOM Use the random sequence when determining the next track.
REPEAT_ALL Use the normal sequence when determining the next track. When the current track is the last playlist entry, the next track becomes the first playlist entry.
REPEAT_RANDOM Use the random sequence when determining the next track. If the current track is the last entry in the random sequence, use the first entry in the random sequence as the next track.
REPEAT_TRACK Use the current track as the next track.

Member Function Documentation

virtual void IPlaylist::AddEntries (MediaRecordList & records*)* `[pure virtual]`

Adds all entries from the content record list to the end of the playlist.

Reimplemented in **CSimplePlaylist** (*p.101*).

virtual IPlaylistEntry IPlaylist::AddEntry (IMediaContentRecord ** pContentRecord*)* `[pure virtual]`

Adds an entry to the end of the playlist. Returns a pointer to the new playlist entry.

Reimplemented in **CSimplePlaylist** (*p.101*).

virtual void IPlaylist::Clear () `[pure virtual]`

Clears the playlist and frees up its memory.

Reimplemented in **CSimplePlaylist** (*p.101*).

virtual void IPlaylist::DeleteEntriesFromDataSource (int iDataSourceID*)* `[pure virtual]`

Removes all entries from the given data source from the playlist.

Reimplemented in **CSimplePlaylist** (*p.101*).

*virtual void IPlaylist::DeleteEntry (IPlaylistEntry ** pEntry*)* `[pure virtual]`

Removes an entry from the playlist.

Reimplemented in **CSimplePlaylist** (*p.101*).

virtual IPlaylistEntry IPlaylist::GetCurrentEntry ()* `[pure virtual]`

Returns a pointer to the current entry.

Reimplemented in **CSimplePlaylist** (*p.101*).

virtual IPlaylistEntry IPlaylist::GetEntry (int* index*, PlaylistMode* mode = *NORMAL)* `[pure virtual]`

Returns a pointer to the entry at the specified zero-based index using the given playlist mode. If the index is out-of-range, then 0 is returned.

Reimplemented in **CSimplePlaylist** (*p.101*).

virtual int IPlaylist::GetEntryIndex (const IPlaylistEntry * pEntry*)* `[pure virtual]`

Returns the zero-based index of the given playlist entry. Returns -1 if the entry is not in the playlist.

Reimplemented in **CSimplePlaylist** (*p.101*).

virtual const char IPlaylist::GetName () const* `[pure virtual]`

Returns the playlist's name.

Reimplemented in **CSimplePlaylist** (*p.101*).

virtual IPlaylistEntry IPlaylist::GetNextEntry (IPlaylistEntry* * pEntry*, PlaylistMode* mode*)* `[pure virtual]`

Returns a pointer to the next entry in the playlist as determined by the mode. Returns 0 if the end of the list was reached and the play mode isn't repeating.

Reimplemented in **CSimplePlaylist** (*p.102*).

virtual IPlaylistEntry IPlaylist::GetPreviousEntry (IPlaylistEntry* * pEntry*, PlaylistMode* mode*)* `[pure virtual]`

Returns a pointer to the previous entry in the playlist as determined by the mode. Returns 0 if the end of the list was reached and the play mode isn't repeating.

Reimplemented in **CSimplePlaylist** (*p.102*).

virtual int IPlaylist::GetSize () `[pure virtual]`

Returns the number of entries in the playlist.

Reimplemented in **CSimplePlaylist** (*p.102*).

virtual IPlaylistEntry IPlaylist::InsertEntry (IMediaContentRecord** pContentRecord*, int* index*)* `[pure virtual]`

Inserts an entry to the playlist at the specified zero-based index. Returns a pointer to the new playlist entry.

Reimplemented in **CSimplePlaylist** (*p.102*).

virtual bool IPlaylist::IsEmpty () const `[pure virtual]`

Returns true if the playlist is empty, false otherwise.

Reimplemented in **CSimplePlaylist** (*p.102*).

virtual void IPlaylist::ReshuffleRandomEntries () [pure virtual]

Reshuffles the random links in the playlist. The current entry will be set as the first random entry.

Reimplemented in **CSimplePlaylist** (*p.102*).

virtual IPlaylistEntry IPlaylist::SetCurrentEntry (IPlaylistEntry* **pEntry***) [pure virtual]*

Sets the current entry pointer to the given record. If the entry isn't in the list, then the current entry pointer is set to 0.

Reimplemented in **CSimplePlaylist** (*p.102*).

virtual IPlaylistEntry IPlaylist::SetNextEntry (PlaylistMode* **mode***) [pure virtual]*

Sets the current entry to the next entry in the playlist as determined by the mode. Returns a pointer to the new current entry, or 0 if the end of the list was reached (and the play mode isn't repeating).

Reimplemented in **CSimplePlaylist** (*p.102*).

virtual IPlaylistEntry IPlaylist::SetPreviousEntry (PlaylistMode* **mode***) [pure virtual]*

Sets the current entry to the previous entry in the playlist as determined by the mode. Returns a pointer to the new current entry, or 0 if the end of the list was reached (and the play mode isn't repeating).

Reimplemented in **CSimplePlaylist** (*p.103*).

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

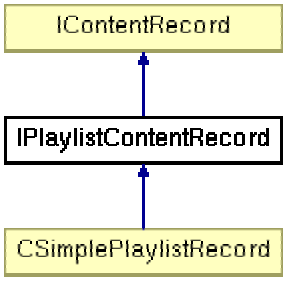
- **Playlist.h**

IPlaylistContentRecord Class Reference

A subclass of **IContentRecord** (*p.118*), used for representing playlist files that can can be loaded by the playlist format manager. In addition to the functionality of the **IContentRecord** (*p.118*) class, the playlist content record also retains a playlist format ID.

`#include <ContentManager.h>`

Inheritance diagram for IPlaylistContentRecord:



Public Methods

- virtual int **GetPlaylistFormatID** () const=0

Detailed Description

A subclass of **IContentRecord** (*p.118*), used for representing playlist files that can can be loaded by the playlist format manager. In addition to the functionality of the **IContentRecord** (*p.118*) class, the playlist content record also retains a playlist format ID.

Member Function Documentation

virtual int IPlaylistContentRecord::GetPlaylistFormatID () const `[pure virtual]`

Returns the ID of the playlist format used to load/save this playlist.

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

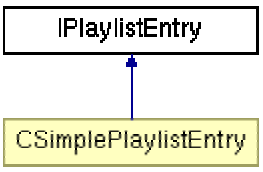
- **ContentManager.h**

IPlaylistEntry Class Reference

The playlist entry class is used to access the individual content records that comprise the playlist.

`#include <Playlist.h>`

Inheritance diagram for IPlaylistEntry:



Public Methods

- virtual int **GetIndex** () const=0

Detailed Description

The playlist entry class is used to access the individual content records that comprise the playlist.

Member Function Documentation

virtual int IPlaylistEntry::GetIndex () const `[pure virtual]`

Returns the index of this entry in the playlist.

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

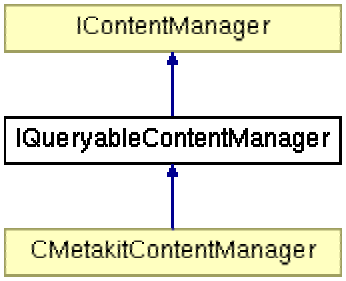
- **Playlist.h**

IQueryableContentManager Class Reference

This class extends the **IContentManager** (*p.114*) by providing functions for querying content based on artist, album, and genre keys.

`#include <QueryableContentManager.h>`

Inheritance diagram for IQueryableContentManager:



Public Methods

- virtual int **GetArtistKey** (const TCHAR *szArtist) const=0
- virtual int **GetAlbumKey** (const TCHAR *szAlbum) const=0
- virtual int **GetGenreKey** (const TCHAR *szGenre) const=0
- virtual const TCHAR * **GetArtistByKey** (int iArtistKey) const=0
- virtual const TCHAR * **GetAlbumByKey** (int iAlbumKey) const=0
- virtual const TCHAR * **GetGenreByKey** (int iGenreKey) const=0
- virtual void **GetMediaRecords** (MediaRecordList &records, int iArtistKey=CMK_ALL, int iAlbumKey=CMK_ALL, int iGenreKey=CMK_ALL, int iDataSourceID=CMK_ALL) const=0
- virtual void **GetArtists** (ContentKeyValueVector &keyValues, int iAlbumKey=CMK_ALL, int iGenreKey=CMK_ALL, int iDataSourceID=CMK_ALL) const=0
- virtual void **GetAlbums** (ContentKeyValueVector &keyValues, int iArtistKey=CMK_ALL, int iGenreKey=CMK_ALL, int iDataSourceID=CMK_ALL) const=0
- virtual void **GetGenres** (ContentKeyValueVector &keyValues, int iArtistKey=CMK_ALL, int iAlbumKey=CMK_ALL, int iDataSourceID=CMK_ALL) const=0

Detailed Description

This class extends the **IContentManager** (*p.114*) by providing functions for querying content based on artist, album, and genre keys.

Member Function Documentation

virtual const TCHAR IQueryableContentManager::GetAlbumByKey (int* iAlbumKey*) const* `[pure virtual]`

Returns the string of the album with the given key. Returns 0 if no album with a matching key was found.

Reimplemented in **CMetakitContentManager** (*p. 74*).

*virtual int IQueryableContentManager::GetAlbumKey (const TCHAR ** szAlbum*) const* `[pure virtual]`

Returns the album key that matches the album string passed in, or 0 if no matching album was found.

Reimplemented in **CMetakitContentManager** (*p. 74*).

virtual void IQueryableContentManager::GetAlbums (ContentKeyValueVector & keyValues*, int* iArtistKey = *CMK_ALL, int* iGenreKey = *CMK_ALL, int* iDataSourceID = *CMK_ALL) const* `[pure virtual]`

Searches the content manager for albums that have tracks with the specified artist, genre, and data source keys.

Parameters:

keyValues List used to append matching album key/value pairs.

Reimplemented in **CMetakitContentManager** (*p. 74*).

virtual const TCHAR IQueryableContentManager::GetArtistByKey (int* iArtistKey*) const* `[pure virtual]`

Returns the string of the artist with the given key. Returns 0 if no artist with a matching key was found.

Reimplemented in **CMetakitContentManager** (*p. 74*).

*virtual int IQueryableContentManager::GetArtistKey (const TCHAR ** szArtist*) const* `[pure virtual]`

Returns the artist key that matches the artist string passed in, or 0 if no matching artist was found.

Reimplemented in **CMetakitContentManager** (*p. 74*).

virtual void IQueryableContentManager::GetArtists (ContentKeyValueVector & keyValues*, int* iAlbumKey = *CMK_ALL, int* iGenreKey = *CMK_ALL, int* iDataSourceID = *CMK_ALL) const* `[pure virtual]`

Searches the content manager for artists that have tracks with the specified album, genre, and data source keys.

Parameters:

keyValues List used to append matching artist key/value pairs.

Reimplemented in **CMetakitContentManager** (*p. 75*).

virtual const TCHAR IQueryableContentManager::GetGenreByKey (int* iGenreKey*) const* `[pure virtual]`

Returns the string of the genre with the given key. Returns 0 if no genre with a matching key was found.

Reimplemented in **CMetakitContentManager** (*p.75*).

*virtual int IQueryableContentManager::GetGenreKey (const TCHAR * szGenre) const [pure virtual]*

Returns the genre key that matches the genre string passed in, or 0 if no matching genre was found.

Reimplemented in **CMetakitContentManager** (*p.75*).

virtual void IQueryableContentManager::GetGenres (ContentKeyValueVector & keyValues, int iArtistKey = CMK_ALL, int iAlbumKey = CMK_ALL, int iDataSourceID = CMK_ALL) const [pure virtual]

Searches the content manager for genre that have tracks with the specified artist, album, and data source keys.

Parameters:

keyValues List used to append matching genre key/value pairs.

Reimplemented in **CMetakitContentManager** (*p.75*).

virtual void IQueryableContentManager::GetMediaRecords (MediaRecordList & records, int iArtistKey = CMK_ALL, int iAlbumKey = CMK_ALL, int iGenreKey = CMK_ALL, int iDataSourceID = CMK_ALL) const [pure virtual]

Searches the content manager for media records that match the artist, album, genre, and data source keys.

Parameters:

records List used to append matching records.

Reimplemented in **CMetakitContentManager** (*p.76*).

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

- **QueryableContentManager.h**

ir_map_s Struct Reference

The ir_map_t structure specifies what events the IR driver should generate for various button presses it receives. This allows an application to mask changes in the physical button layout of a remote by simply shuffling the order of the button mapping. Additionally it provides configuration for repeat events. The IR driver does not support release events.

`#include <IR.h>`

Public Attributes

- int **num_buttons**
- int **repeat_flags**
- int **filter_start**
- int **filter_rate**
- const unsigned int *const **press_map**
- const unsigned int *const **hold_map**

Detailed Description

The ir_map_t structure specifies what events the IR driver should generate for various button presses it receives. This allows an application to mask changes in the physical button layout of a remote by simply shuffling the order of the button mapping. Additionally it provides configuration for repeat events. The IR driver does not support release events.

Member Data Documentation

int ir_map_s::filter_rate

Rate at which to filter repeat events. If the remote triggers repeat events too fast, raising this number will cause the driver to ignore more hardware events.

int ir_map_s::filter_start

Number of repeat events to drop initially before actually generating the event.

const unsigned int const ir_map_s::hold_map*

Pointer to an array of events that are fired on button hold.

int ir_map_s::num_buttons

Number of buttons on the remote.

const unsigned int const ir_map_s::press_map*

Pointer to an array of events that are fired on button press.

int ir_map_s::repeat_flags

Flags regarding button repeat behavior.

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

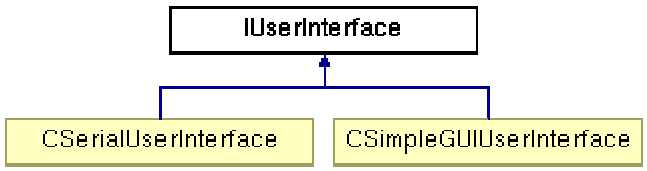
- **IR.h**

IUserInterface Class Reference

This is an abstract base class used for simple demo UI purposes. Functions are provided for notifying the user of player events. It's not part of the SDK proper, but exists to give an example of what a basic user interface might look like.

`#include <UserInterface.h>`

Inheritance diagram for IUserInterface:



Detailed Description

This is an abstract base class used for simple demo UI purposes. Functions are provided for notifying the user of player events. It's not part of the SDK proper, but exists to give an example of what a basic user interface might look like.

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

- **UserInterface.h**

key_map_s Struct Reference

The key_map_t structure specifies what events the keyboard driver should trigger based on the key input. It also allows the application to specify the keyboard layout and customize keyboard driver details such as repeat delay and rate. press_map, hold_map, and release_map should point to event arrays of size num_buttons.

```
#include <Keyboard.h>
```

Public Attributes

- int **num_columns**
- int **num_rows**
- int **num_buttons**
- int **repeat_flags**
- int **tick_length**
- int **repeat_rate**
- int **initial_delay**
- const unsigned int *const **press_map**
- const unsigned int *const **hold_map**
- const unsigned int *const **release_map**

Detailed Description

The key_map_t structure specifies what events the keyboard driver should trigger based on the key input. It also allows the application to specify the keyboard layout and customize keyboard driver details such as repeat delay and rate. press_map, hold_map, and release_map should point to event arrays of size num_buttons.

Member Data Documentation

const unsigned int const key_map_s::hold_map*

Pointer to an array of events to be fired if the button is being held down.

int key_map_s::initial_delay

Number of ticks before the first event is fired.

int key_map_s::num_buttons

Number of actual buttons.

int key_map_s::num_columns

Number of columns on the keyboard.

int key_map_s::num_rows

Number of rows on the keyboard.

const unsigned int const key_map_s::press_map*

Pointer to an array of events that are fired on button press.

const unsigned int const key_map_s::release_map*

Pointer to an array of events that are fired on button release.

int key_map_s::repeat_flags

Flags regarding key repeat behavior.

int key_map_s::repeat_rate

Number of ticks between events.

int key_map_s::tick_length

Length of a keyboard tick in 10ms units.

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

- **Keyboard.h**

media_record_info_s Struct Reference

Used for passing information about a track available from the data source to the content manager (and back to the data source manager for metadata retrieval in a two-pass update).

`#include <ContentManager.h>`

Public Attributes

- char * **szURL**
- bool **bVerified**
- int **iDataSourceID**
- int **iCodecID**
- **IMetadata * pMetadata**

Detailed Description

Used for passing information about a track available from the data source to the content manager (and back to the data source manager for metadata retrieval in a two-pass update).

Member Data Documentation

bool media_record_info_s::bVerified

True if this record has been verified to exist on the data source.

int media_record_info_s::iCodecID

ID of the matching codec of this content.

int media_record_info_s::iDataSourceID

ID of the data source that hosts the content.

IMetadata media_record_info_s::pMetadata*

Associated metadata; can be 0.

char media_record_info_s::szURL*

The unique URL of this media record.

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

- **ContentManager.h**

playlist_record_s Struct Reference

Used for passing information about a playlist file available from the data source to the content manager.

`#include <ContentManager.h>`

Public Attributes

- char * **szURL**
- bool **bVerified**
- int **iDataSourceID**
- int **iPlaylistFormatID**

Detailed Description

Used for passing information about a playlist file available from the data source to the content manager.

Member Data Documentation

bool playlist_record_s::bVerified

True if this record has been verified to exist on the data source.

int playlist_record_s::iDataSourceID

ID of the data source that hosts the playlist.

int playlist_record_s::iPlaylistFormatID

ID of the matching playlist format.

char playlist_record_s::szURL*

The unique URL of this playlist record.

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

- **ContentManager.h**

playstream_settings_s Struct Reference

The sequence of codecs->filters->outputstream is considered to be the playstream. Currently, the playstream_settings_t structure allows the user to specify a list of filters to put in the playstream and an outputstream to use for playback.

`#include <MediaPlayer.h>`

Detailed Description

The sequence of codecs->filters->outputstream is considered to be the playstream. Currently, the playstream_settings_t structure allows the user to specify a list of filters to put in the playstream and an outputstream to use for playback.

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

- **MediaPlayer.h**

set_stream_event_data_s Struct Reference

Structure created when a song is successfully set in the media player. This is passed along through the EVENT_STREAM_SET event. The structure and the metadata should be deleted after the event is received (which is the default action if the event is passed to the play manager).

`#include <Codec.h>`

Public Attributes

- **IPlaylistEntry * pPlaylistEntry**
- **IMetadata * pMediaPlayerMetadata**
- **stream_info_t streamInfo**

Detailed Description

Structure created when a song is successfully set in the media player. This is passed along through the EVENT_STREAM_SET event. The structure and the metadata should be deleted after the event is received (which is the default action if the event is passed to the play manager).

Member Data Documentation

IMetadata set_stream_event_data_s::pMediaPlayerMetadata*

Metadata retrieved from the codec during SetSong.

IPlaylistEntry set_stream_event_data_s::pPlaylistEntry*

Pointer to the playlist entry that was set.

stream_info_t set_stream_event_data_s::streamInfo

Playback parameters of the stream.

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

- **Codec.h**

SimpleList Class Template Reference

The SimpleList is a template class for generically storing a doubly-linked list of items. Its content can be traversed by the use of iterators.

`#include <SimpleList.h>`

Public Methods

- SimpleList & **operator=** (const SimpleList &sl)
- SimpleList & **Append** (const SimpleList &sl)
- void **PushFront** (DataType data)
- void **PushBack** (DataType data)
- void **Insert** (DataType data, int index)
- void **Clear** ()
- int **Size** () const
- bool **IsEmpty** () const
- DataType **PopFront** ()
- DataType **PopBack** ()
- DataType **Remove** (**SimpleListIterator**< DataType > it)
- **SimpleListIterator**< DataType > **GetHead** () const
- **SimpleListIterator**< DataType > **GetTail** () const
- **SimpleListIterator**< DataType > **GetEnd** () const

Detailed Description

template<class DataType> class SimpleList< DataType >

The SimpleList is a template class for generically storing a doubly-linked list of items. Its content can be traversed by the use of iterators.

Member Function Documentation

template<class DataType> SimpleList& SimpleList< DataType >::Append (const SimpleList< DataType > & sl) `[inline]`

Append the contents of the given list to the end of this list.

template<class DataType> void SimpleList< DataType >::Clear () `[inline]`

Erase the nodes but don't delete the data.

template<class DataType> SimpleListIterator<DataType> SimpleList< DataType >::GetEnd () const `[inline]`

Return an iterator that points to an empty element. Use this to test if an iterator has passed the end of the list.

template<class DataType> SimpleListIterator<DataType> SimpleList< DataType >::GetHead () const `[inline]`

Return an iterator that points to the first element in the list.

template<class DataType> SimpleListIterator<DataType> SimpleList< DataType >::GetTail () const `[inline]`

Return an iterator that points to the last element in the list.

template<class DataType> void SimpleList< DataType >::Insert (DataType data*, int* index*)* `[inline]`

Inserts the data type into the list at the zero-based index. The results are undefined if the index is out-of-range.

template<class DataType> bool SimpleList< DataType >::IsEmpty () const `[inline]`

Returns true if there are no items in the list, false otherwise.

template<class DataType> DataType SimpleList< DataType >::PopBack () `[inline]`

Pops the last item off the list and returns it. The results are undefined if the list is empty.

template<class DataType> DataType SimpleList< DataType >::PopFront () `[inline]`

Pops the first item off the list and returns it. The results are undefined if the list is empty.

template<class DataType> void SimpleList< DataType >::PushBack (DataType data*)* `[inline]`

Add an element to the end of the list.

template<class DataType> void SimpleList< DataType >::PushFront (DataType data*)* `[inline]`

Add an element to the beginning of the list.

template<class DataType> DataType SimpleList< DataType >::Remove (SimpleListIterator< DataType > it*)* `[inline]`

Removes a node from the list. The iterator is invalid after this operation.

template<class DataType> int SimpleList< DataType >::Size () const `[inline]`

Returns the number of nodes in the list.

template<class DataType> SimpleList& SimpleList< DataType >::operator= (const SimpleList< DataType > & sl*)* `[inline]`

Copy an existing list.

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

- **SimpleList.h**

SimpleListIterator Class Template Reference

The SimpleListIterator is used to traverse a **SimpleList** (*p.152*).

`#include <SimpleList.h>`

Public Methods

- DataType & **operator *** ()
- bool **operator==** (const SimpleListIterator< DataType > &rhs)
- bool **operator!=** (const SimpleListIterator< DataType > &rhs)
- SimpleListIterator< DataType > & **operator+=** (int rhs)
- SimpleListIterator< DataType > & **operator++** ()
- SimpleListIterator< DataType > & **operator-=** (int rhs)
- SimpleListIterator< DataType > & **operator--** ()

Friends

- SimpleListIterator< DataType > **operator+** (const SimpleListIterator< DataType > &lhs, int rhs)
- SimpleListIterator< DataType > **operator-** (const SimpleListIterator< DataType > &lhs, int rhs)

Detailed Description

template<class DataType> class SimpleListIterator< DataType >

The SimpleListIterator is used to traverse a **SimpleList** (*p.152*).

Member Function Documentation

*template<class DataType> DataType& SimpleListIterator< DataType >::operator * () [inline]*

Get access to the data stored at this position in the list.

template<class DataType> bool SimpleListIterator< DataType >::operator!= (const SimpleListIterator< DataType > & rhs) [inline]

Test for inequality.

template<class DataType> SimpleListIterator<DataType>& SimpleListIterator< DataType >::operator++ () [inline]

Go to the next element in the list.

template<class DataType> SimpleListIterator<DataType>& SimpleListIterator< DataType >::operator+= (int rhs) [inline]

Advance the iterator by the given number of indices. The results are undefined if the end of the list is passed.

template<class DataType> SimpleListIterator<DataType>& SimpleListIterator< DataType >::operator-- () `[inline]`

Go to the previous element in the list.

template<class DataType> SimpleListIterator<DataType>& SimpleListIterator< DataType >::operator-= (int rhs*)* `[inline]`

Rewind the iterator by the given number of indices. The results are undefined if the end of the list is passed.

template<class DataType> bool SimpleListIterator< DataType >::operator== (const SimpleListIterator< DataType > & rhs*)* `[inline]`

Test for equality.

Friends And Related Function Documentation

template<class DataType> SimpleListIterator<DataType> operator+ (const SimpleListIterator< DataType > & lhs*, int* rhs*)* `[friend]`

Copy the given iterator and advance it by the given number of indices. The results are undefined if the end of the list is passed.

template<class DataType> SimpleListIterator<DataType> operator- (const SimpleListIterator< DataType > & lhs*, int* rhs*)* `[friend]`

Copy the given iterator and rewind it by the given number of indices. The results are undefined if the end of the list is passed.

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

- **SimpleList.h**

SimpleMap Class Template Reference

The SimpleMap is a template class for generically storing elements indexed by keys. The IndexType must have comparison operators < and >.

`#include <SimpleMap.h>`

Public Methods

- bool **GetEntry** (int index, IndexType *pKey, DataType *pValue)
- void **AddEntry** (IndexType key, DataType value)
- bool **FindEntry** (const IndexType key, DataType *pValue)
- bool **FindEntry** (const IndexType key, DataType *pValue) const
- void **RemoveEntryAtIndex** (int index, IndexType *pKey, DataType *pValue)
- bool **RemoveEntryByKey** (const IndexType key, DataType *pValue)
- void **Clear** ()

Detailed Description

template<class IndexType, class DataType> class SimpleMap< IndexType, DataType >

The SimpleMap is a template class for generically storing elements indexed by keys. The IndexType must have comparison operators < and >.

Member Function Documentation

template<class IndexType, class DataType> void SimpleMap< IndexType, DataType >::AddEntry (IndexType **key**, *DataType* **value**) `[inline]`*

Adds an entry to the dictionary. If a key with the same name already exists, then its value is overwritten.

template<class IndexType, class DataType> void SimpleMap< IndexType, DataType >::Clear () `[inline]`

Clears the map of all entries.

template<class IndexType, class DataType> bool SimpleMap< IndexType, DataType >::FindEntry (const IndexType **key**, *DataType* * **pValue**) const `[inline]`*

Gets a value given a key. If the key isn't in the dictionary then false is returned.

template<class IndexType, class DataType> bool SimpleMap< IndexType, DataType >::FindEntry (const IndexType **key**, *DataType* * **pValue**) `[inline]`*

Gets a value given a key. If the key isn't in the map then false is returned.

template<class IndexType, class DataType> bool SimpleMap< IndexType, DataType >::GetEntry (int index*, IndexType* * pKey*, DataType* * pValue*)* *[inline]*

Gets a key and value given an index. If the index is out-of-range then false is returned.

template<class IndexType, class DataType> void SimpleMap< IndexType, DataType >::RemoveEntryAtIndex (int index*, IndexType* * pKey*, DataType* * pValue*)* *[inline]*

Removes the entry at the given index from the map. This is an inefficient operation and should be used sparingly.

template<class IndexType, class DataType> bool SimpleMap< IndexType, DataType >::RemoveEntryByKey (const IndexType key*, DataType* * pValue*)* *[inline]*

Removes the entry with the given key from the map. This is an inefficient operation and should be used sparingly. If the key isn't in the map then false is returned.

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

- **SimpleMap.h**

SimpleVector Class Template Reference

The SimpleVector is a template class for generically storing an array of items. Its content can be randomly accessed by index. The array starts empty, but grows as items are added to it.

`#include <SimpleVector.h>`

Public Methods

- **SimpleVector** (int blockSize=5)
- **SimpleVector** (const SimpleVector &sv)
- SimpleVector & **operator=** (const SimpleVector &sv)
- void **PushBack** (DataType pData)
- DataType **PopBack** ()
- DataType **Remove** (unsigned int index)
- void **Clear** ()
- int **Size** () const
- bool **IsEmpty** () const
- const DataType & **operator[]** (int pos) const
- DataType & **operator[]** (int pos)
- typedef bool **CompareDataFunction** (const DataType &a, const DataType &b)
- void **Sort** (CompareDataFunction &compareDataFunction)
- int **Find** (const DataType &data)
- bool **Insert** (DataType pData, unsigned int index)
- bool **Insert** (DataType pData, CompareDataFunction &compareDataFunction)
- void **Move** (int indexA, int indexB)

Detailed Description

template<class DataType> class SimpleVector< DataType >

The SimpleVector is a template class for generically storing an array of items. Its content can be randomly accessed by index. The array starts empty, but grows as items are added to it.

Constructor & Destructor Documentation

template<class DataType> SimpleVector< DataType >::SimpleVector (int blockSize = *5)* `[inline]`

Construct a new, empty vector.

Parameters:

blockSize The number of items to grow the array by at a time.

template<class DataType> SimpleVector< DataType >::SimpleVector (const SimpleVector< DataType > & sv*)* `[inline]`

Copy an existing vector.

Member Function Documentation

template<class DataType> void SimpleVector< DataType >::Clear () `[inline]`

Clear the vector but don't delete the data.

template<class DataType> typedef bool SimpleVector< DataType >::CompareDataFunction (const DataType & a*, const DataType &* b*)*

A CompareDataFunction should return true if a <= b, false otherwise.

template<class DataType> int SimpleVector< DataType >::Find (const DataType & data*)* `[inline]`

Searches for a matching value in the vector. Returns the index of the item if found, -1 otherwise.

template<class DataType> bool SimpleVector< DataType >::Insert (DataType pData*, CompareDataFunction &* compareDataFunction*)* `[inline]`

Insert a record in the sorted array in its proper position, based on the comparison function.

template<class DataType> bool SimpleVector< DataType >::Insert (DataType pData*, unsigned int* index*)* `[inline]`

Insert a record at the specified index in the vector. If the index is out-of-range, then false is returned.

template<class DataType> bool SimpleVector< DataType >::IsEmpty () const `[inline]`

Returns true if there are no items in the array, false otherwise.

template<class DataType> void SimpleVector< DataType >::Move (int indexA*, int* indexB*)* `[inline]`

Moves a record in the array to the new index, and pushes the records in between the two indices up or down accordingly. No range-checking is done, so make sure the indices are in bounds.

template<class DataType> DataType SimpleVector< DataType >::PopBack () `[inline]`

Remove a record from the end of the vector Popping from an empty vector has undefined results.

template<class DataType> void SimpleVector< DataType >::PushBack (DataType pData*)* `[inline]`

Add a record to the end of the vector. If the vector is full, then grow it by the block size.

template<class DataType> DataType SimpleVector< DataType >::Remove (unsigned int index*)* `[inline]`

Remove a record from the specified position in the vector. This is an inefficient operation and should be used sparingly.

template<class DataType> int SimpleVector< DataType >::Size () const `[inline]`

Return the number of filled blocks in the list.

template<class DataType> void SimpleVector< DataType >::Sort (CompareDataFunction & compareDataFunction*)* `[inline]`

Sort the vector in ascending order, based on the comparison function.

template<class DataType> SimpleVector& SimpleVector< DataType >::operator= (const SimpleVector< DataType > & sv*)* `[inline]`

Copy an existing vector.

template<class DataType> DataType& SimpleVector< DataType >::operator[] (int pos*)* `[inline]`

Returns a reference to the item at the given index.

template<class DataType> const DataType& SimpleVector< DataType >::operator[] (int pos*) const* `[inline]`

Returns a const reference to the item at the given index.

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

- **SimpleVector.h**

stream_info_s Struct Reference

Used to propagate information about the bitstream out of the codec to the system.

`#include <Codec.h>`

Public Attributes

- unsigned long **Duration**
- unsigned long **SamplingFrequency**
- unsigned long **Channels**
- unsigned long **OutputChannels**
- unsigned long **Bitrate**

Detailed Description

Used to propagate information about the bitstream out of the codec to the system.

Member Data Documentation

unsigned long stream_info_s::Bitrate

The bitrate of the track (in bits/second).

unsigned long stream_info_s::Channels

The number of channels of audio (currently only 1 or 2 are supported).

unsigned long stream_info_s::Duration

The duration of the track (in seconds).

unsigned long stream_info_s::OutputChannels

The number of channels actually output (may differ from Channels if channel duplication is supported).

unsigned long stream_info_s::SamplingFrequency

The sampling frequency of the track (in Hz).

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

- **Codec.h**

Index