# CloudTV Nano SDK (C++ Southbound API)

4.4

Generated by Doxygen 1.8.5

Mon Nov 6 2017 10:23:13

# Contents

# Chapter 1

# Introduction


Figure 1.1: width=150px

## 1.1 Summary

The CloudTV™ Nano SDK can set up and control CloudTV sessions, providing the same capabilities across many different device types. The CloudTV Nano SDK supports cable set-top boxes as well as IP set-top boxes.

The CloudTV™ Nano SDK can be provided as a compiled software library, offering client APIs that a controlling Application (Guide/Middleware) can use to set up a CloudTV session. A device abstraction layer implements all device specific functionality. Porting to a specific device is done by implementing the device abstraction or Device Porting Layer for that device.

The Nano Client family of SDK's support the BCP and RFB-TV protocols. BCP is intended for cable systems with limited return paths (ALOHA). The RFB-TV protocol is intended for faster return paths. There are more features in the RFB--TV protocol, so the APIs are different between the two protocols. Therefore the APIs are split into two separate SDK versions. This SDK — the CloudTV Nano SDK — is written in C++ and only supports the RFB-TV protocol.

## 1.2 Copyright

## 1.3 Other Restricted Rights



U.S. Patents listed at http://www.activevideo.com/patents

## 1.4 Contact

### 1.4.1 ActiveVideo — Main Headquarters

333 W. San Carlos St.

Suite 400

San Jose, CA 95110

United States

Toll-free: 1-800-926-8398

Main: 1-408-931-9200

Fax: 1-408-931-9100

e-mail: info@activevideo.com

### 1.4.2 ActiveVideo — European Headquarters

Joop van den Endeplein 1

Mediacentrum 3745

1217 WJ Hilversum

The Netherlands

Main: +31 (0)35 6774131

# Chapter 2

# Content of the package

## 2.1 Introduction

The CloudTV Nano SDK porting layer, so-called Southbound API, is an abstraction layer that allows Nano library to be platform independent and portable.

The code is divided in two main groups:

- **OS specific code**: this layer abstracts the Operating System of the platform. ActiveVideo provides with two reference implementations, one based in POSIX compliant Operating Systems and the other based in Windows. These reference implementations should help the integrator to CloudTV Nano SDK to create a new, if necessary.
- **Generic code**: this layer has to be customized for each integration because it platform dependent.

## 2.2 Directory structure

The porting_layer directory has the following structure

```
porting_layer
    |-> Makefile        Makefile to compile libporting_layer.a
    |-> ...             Header files of the porting layer
    |-> src/            Directory where the implementation resides
        |-> generic/    Generic implementation that is OS independent
        |-> posix/      Reference implementation for POSIX compliant OS
        |-> windows/    Reference implementation for Window
```

## 2.3 Where to start

If the integrator requires to create a new porting layer, he should initially look at the available source code.

If he needs to customize the generic layer to provide with a different abstraction layer to the Middleware or the platform, the starting point should be *porting_layer/src/generic* where most of the code should be reusable.

On the other hand, if need is driven by an unsupported Operating System, then a new directory should be created in *porting_layer/src/<new OS>*. For example, if the OS where CloudTV Nano SDK has to execute is VxWorks, the new source code should be placed in: *porting_layer/src/vxworks*

The Makefile in *porting_layer* can be reused to build a new porting layer. The relevant variables to be modified are *SRC_DIR* and *SOURCE_FILES*.

# Chapter 3

# Hierarchical Index

## 3.1 Class Hierarchy

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

# Chapter 4

# Data Structure Index

## 4.1 Data Structures

Here are the data structures with brief descriptions:

# Chapter 5

# Data Structure Documentation

## 5.1 Atomic< T > Struct Template Reference

Generic interface for 'atomic' variables.

### Public Member Functions

- Atomic (const T &value)

  *Construct an atomic value.*
- T operator= (const T &value)

  *Assign new value atomically.*
- operator T () const

  *Get the value atomically.*
- T operator++ ()

  *Increment the value atomically.*

### 5.1.1 Detailed Description

**template<typename T>struct ctvc::Atomic< T >**

This template class helps in creating variables that need to be thread safe.

**Note**

Only operators that are actually used in the SDK are implemented.

## 5.2 ClientContext Class Reference

ClientContext stores all client-specific context information such as device manufacturer or device model.

### Public Member Functions

- void set_manufacturer (const char *manufacturer)

*Set the device manufacturer, e.g. "Arris".*

- const char * get_manufacturer () const

  *Returns the device manufacturer that was previously set.*

- void set_device_type (const char *devicetype)

  *Set the device type/model, e.g. "VIP1113".*

- const char * get_device_type () const

  *Returns the device type/model that was previously set.*

- void set_unique_id (const char *unique_id)

  *Set the unique device identifier, e.g. the MAC hardware address or serial number.*

- const char * get_unique_id () const

  *Returns the unique device identifier that was previously set.*

- void set_ca_path (const char *path)

  *Set the path to the CA certificate file, in PEM format.*

- const char * get_ca_path () const

  *Returns the path to the CA certificate file that was previously set.*

- void set_ca_client_path (const char *path)

  *Set the path to the TLS certificate file, in PEM format.*

- const char * get_ca_client_path () const

  *Returns the path to the TLS certificate file.*

- void set_private_key_path (const char *path)

  *Set the path to the TLS private key file, in PEM format.*

- const char * get_private_key_path () const

  *Returns the path to the TLS private key file.*

- void register_log_output (ILogOutput &log_output)

  *Registers a private logging output with the porting layer.*

- void unregister_log_output (ILogOutput &log_output)

  *Unregisters a private logging output with the porting layer.*

- void set_log_format (const char *log_format)

  *Sets the log formatting string. All successive log messages will be printed using this format.*

- void log_message (LogMessageType message_type, const char *file, int line, const char *function, const char *message) const

  *Forwards a log message to all registered ILogOutput interfaces.*

- void log_message (LogMessageType message_type, const char *message) const

  *Forwards a log message to all registered ILogOutput interfaces.*

- void set_base_store_path (const char *path)

  *Set base store path for get/set/delete_secure_data and cookie files.*

- DataStore & get_data_store ()

  *Get access to the DataStore object of this client.*

- X11KeyMap & get_keymap ()

  *Get key map for translating native keys to X11 key codes.*

## Static Public Member Functions

- static ClientContext & instance ()

  *Get the one-and-only instance as per the singleton pattern.*

### 5.2.1 Detailed Description

This class follows the singleton pattern, thus it is not necessary to be explicitly passed to the Session object. However, its values shall be filled in before setting up a new session with the CloudTV platform.

### 5.2.2 Member Function Documentation

#### 5.2.2.1 const char* get_ca_client_path ( ) const

**Returns**

String with the path to the TLS certificate file. This is an empty string if it was not previously set.

#### 5.2.2.2 const char* get_ca_path ( ) const

**Returns**

String indicating the path to the CA certificate file. This is an emtpy string if it was not previously set.

#### 5.2.2.3 DataStore& get_data_store ( ) [inline]

**Returns**

Reference to the DataStore object that ClientContext holds.

#### 5.2.2.4 const char* get_device_type ( ) const

**Returns**

String with device type/model. This is an empty string if it was not previously set.

#### 5.2.2.5 X11KeyMap& get_keymap ( )

**Returns**

The key translation map.

#### 5.2.2.6 const char* get_manufacturer ( ) const

**Returns**

String with the device manufacturer. This is an empty string if it was not previously set.

#### 5.2.2.7 const char* get_private_key_path ( ) const

**Returns**

String with the path to the TLS private key file. This is an empty string if it was not previously set.

#### 5.2.2.8 const char* get_unique_id ( ) const

**Returns**

String with the unique device identifier. This is an empty string if it was not previously set.

#### 5.2.2.9 static ClientContext& instance ( ) [static]

**Returns**

Reference to the ClientContext singleton.

#### 5.2.2.10 void log_message ( LogMessageType *message_type*, const char * *file*, int *line*, const char * *function*, const char * *message* ) const

**Parameters**

| | | |
|---|---|---|
| in | *message_type* | Log level |

**See Also**

Log.h

**Parameters**

| | | |
|---|---|---|
| in | *file* | Name of the source file issuing the log. |
| in | *line* | Line number of the source code issuing the log. |
| in | *function* | Function that issues the log. |
| in | *message* | String containing the log message If no logging output is set, the porting layer will output all logging to stderr. |

#### 5.2.2.11 void log_message ( LogMessageType *message_type*, const char * *message* ) const [inline]

**Parameters**

| | | |
|---|---|---|
| in | *message_type* | Log level |

**See Also**

Log.h

**Parameters**

| | | |
|---|---|---|
| in | *message* | String containing the log message If no logging output is set, the porting layer will output all logging to stderr. |

#### 5.2.2.12 void register_log_output ( ILogOutput & *log_output* )

**Parameters**

| | | |
|---|---|---|
| in | *log_output* | Reference to an ILogOutput interface Re-registering an object that was already registered has no effect. |

#### 5.2.2.13 void set_base_store_path ( const char * *path* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *path* | The path |

#### 5.2.2.14 void set_ca_client_path ( const char * *path* )

**Parameters**

| | | |
|---|---|---|
| in | *path* | Name of the file containing the certificate |

#### 5.2.2.15 void set_ca_path ( const char * *path* )

**Parameters**

| | | |
|---|---|---|
| in | *path* | Name of the file containing the certificate |

#### 5.2.2.16 void set_device_type ( const char * *devicetype* )

**Note**

This is a mandatory parameter that must be set by the client.

**Parameters**

| | | |
|---|---|---|
| in | *devicetype* | String indicating the device type/model. |

#### 5.2.2.17 void set_log_format ( const char * *log_format* )

**Parameters**

| | | |
|---|---|---|
| in | *log_format* | The format string to specify. If 0, the default format is selected. The format string uses a simplified printf-style format: All characters will be copied as-is, except when prepended with a ” character or when explicitly excluded.. The ” character escape sequences are the following: "%%": Print a single ” character. "%T": : Print the time of the log message (in hh:mm:ss.ms format). "%t": Print the type (debug, info, warning, error) of the log message. "%F": Print the function name from which the log was called. "%f": Print the file name from which the log was called. "%l": Print the line number from which the log was called. "%n": Print the name of the thread from which the log was called. "%m": Print the log message contents. "%[": Text to include if a log message is output (and excluded if an empty log message is output). "%]": End of text section to include if a log message is output. |

**Note**

The default log format is "<%T> Type:<%t> at %f:%l, %F%[, Message:<%m>%]\r\n"

#### 5.2.2.18 void set_manufacturer ( const char * *manufacturer* )

**Note**

This is a mandatory parameter that must be set by the client.

**Parameters**

| | | |
|---|---|---|
| in | *manufacturer* | String indicating the device manufacturer. |

#### 5.2.2.19 void set_private_key_path ( const char * *path* )

**Parameters**

| | | |
|---|---|---|
| in | *path* | Name of the file containing the key |

#### 5.2.2.20 void set_unique_id ( const char * *unique_id* )

**Note**

This is a mandatory parameter that must be set by the client.

**Parameters**

| | | |
|---|---|---|
| in | *unique_id* | String indicating the unique device identifier. |

#### 5.2.2.21 void unregister_log_output ( ILogOutput & *log_output* )

**Parameters**

| | | |
|---|---|---|
| in | *log_output* | Reference to an ILogOutput interface Re-unregistering an object that was already unregistered has no effect. |

## 5.3 Condition Class Reference

Generic condition variable interface.

### Data Structures

- struct ICondition

  *Interface for the implementation of Condition.*

## Public Member Functions

- void lock ()

  *Lock operation to protect a critical region.*
- void unlock ()

  *Unlock operation to indicate the end of a critical region.*
- bool trylock ()

  *Try to lock the mutex.*
- void notify ()

  *Notify a thread that is waiting (to be notified).*
- void wait_without_lock ()

  *Wait until notified.*
- bool wait_without_lock (uint32_t timeout_in_ms)

  *Wait until notified while using a timeout.*

### 5.3.1 Member Function Documentation

#### 5.3.1.1 void lock ( ) [inline], [virtual]

This method will wait until the mutex can be acquired. The mutex is recursive, thus if the same thread locks twice the same mutex, it won't block the execution.

Implements IMutex.

#### 5.3.1.2 void notify ( ) [inline]

**See Also**

wait()

#### 5.3.1.3 bool trylock ( ) [inline], [virtual]

This method never blocks. If the mutex cannot be acquired, it will return false.

**Return values**

| | |
|---|---|
| *true* | If the mutex has been acquired. |
| *false* | If the mutex could not be acquired. |

Implements IMutex.

#### 5.3.1.4 void wait_without_lock ( ) [inline]

The corresponding mutex will be atomically released when the wait starts. As soon as the calling thread gets notified, the mutex ownership will be gained (again atomically).

**See Also**

notify()

#### 5.3.1.5 bool wait_without_lock ( uint32_t *timeout_in_ms* ) [inline]

If locked, the corresponding mutex will be atomically released when the wait starts. As soon as the calling thread gets notified, the mutex ownership will be gained (again atomically). If not notified, the method will wait for at most *timeout_in_ms* milliseconds for *notify()* to be called by another thread. If *notify()* is called before the timeout expires, *wait_without_lock()* returns true. If *notify()* was not called within *timeout_in_ms* milliseconds, *wait_without_lock()* returns false. In both cases the Mutex will be locked again, before returning from this call unless it was not locked upon enty.

**Parameters**

| | | |
|---|---|---|
| in | *timeout_in_ms* | The maximum time to wait for another thread to call *notify()* before returning false. |

**Return values**

| | |
|---|---|
| *true* | if successful. |
| *false* | if a timeout occurred. |

**See Also**

notify()

## 5.4 DataStore Class Reference

Class for (secure) storage of the data.

### Public Member Functions

- void set_base_store_path (const char *path)

  *Set base store path for get/set/delete_data.*
- ResultCode set_data (const char *id, const uint8_t *data, uint32_t length)

  *This is called to save persistent data.*
- ResultCode set_data (const char *id, const std::vector< uint8_t > &data)

  *This is called to save persistent data from an std::vector.*
- ResultCode set_data (const char *id, const std::string &data)

  *This is called to save persistent data from an std::string.*
- ResultCode get_data (const char *id, uint8_t *data, uint32_t size, uint32_t *length)

  *This is called to get stored persistent data.*
- ResultCode get_data (const char *id, std::vector< uint8_t > &data)

  *This is called to get stored persistent data into an std::vector.*
- ResultCode get_data (const char *id, std::string &data)

  *This is called to get stored persistent data into an std::string.*
- ResultCode delete_data (const char *id)

  *This is called to delete persistent data.*

### Static Public Attributes

- static const ResultCode INVALID_PARAMETER

  *The parameter is not valid.*
- static const ResultCode COULD_NOT_OPEN_ITEM

*The requested item cannot be opened.*

- static const ResultCode READ_ERROR

  *Error during reading of the item.*

- static const ResultCode WRITE_ERROR

  *Error during writing of the item.*

- static const ResultCode COULD_NOT_REMOVE_ITEM

  *The item could not be deleted.*

### 5.4.1 Detailed Description

Nano SDK uses this class to store data in the platform.

### 5.4.2 Member Function Documentation

#### 5.4.2.1 ResultCode delete_data ( const char * *id* )

**Parameters**

| | | |
|---|---|---|
| in | *id* | The id of the data. |

**Returns**

ResultCode

#### 5.4.2.2 ResultCode get_data ( const char * *id*, uint8_t * *data*, uint32_t *size*, uint32_t * *length* )

**Parameters**

| | | |
|---|---|---|
| in | *id* | The id of the data. |
| out | *data* | Buffer for the data to be retrieved; if null, only the length will be returned. |
| in | *size* | The size of data buffer (if data != null). |
| out | *length* | The length of data that is or will be retrieved, may return 0 if the data is empty. |

**Returns**

ResultCode

#### 5.4.2.3 ResultCode get_data ( const char * *id*, std::vector< uint8_t > & *data* )

**Parameters**

| | | |
|---|---|---|
| in | *id* | The id of the data. |
| out | *data* | The data that is retrieved, may be empty. |

**Returns**

ResultCode

#### 5.4.2.4 ResultCode get_data ( const char * *id*, std::string & *data* )

**Parameters**

| | | |
|---|---|---|
| in | *id* | The id of the data. |
| out | *data* | The data that is retrieved, may be empty. |

**Returns**

ResultCode

#### 5.4.2.5 void set_base_store_path ( const char * *path* )

**Parameters**

| | | |
|---|---|---|
| in | *path* | The path. |

#### 5.4.2.6 ResultCode set_data ( const char * *id*, const uint8_t * *data*, uint32_t *length* )

**Parameters**

| | | |
|---|---|---|
| in | *id* | The id of the data. |
| in | *data* | The data to be saved. |
| in | *length* | The length of the data, may be 0. |

**Returns**

ResultCode

**Note**

The DataStore may implement mechanisms to prevent writing the same data multiple times.

#### 5.4.2.7 ResultCode set_data ( const char * *id*, const std::vector< uint8_t > & *data* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *id* | The id of the data. |
| in | *data* | The data to be saved, may be empty. |

**Returns**

ResultCode

**Note**

The DataStore may implement mechanisms to prevent writing the same data multiple times.

#### 5.4.2.8 ResultCode set_data ( const char * *id*, const std::string & *data* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *id* | The id of the data. |
| in | *data* | The data to be saved, may be an empty string. |

**Returns**

ResultCode

**Note**

The DataStore may implement mechanisms to prevent writing the same data multiple times.

### 5.4.3 Field Documentation

#### 5.4.3.1 const ResultCode COULD_NOT_OPEN_ITEM [static]

**See Also**

ResultCode

#### 5.4.3.2 const ResultCode COULD_NOT_REMOVE_ITEM [static]

**See Also**

ResultCode

#### 5.4.3.3 const ResultCode INVALID_PARAMETER [static]

**See Also**

ResultCode

#### 5.4.3.4 const ResultCode READ_ERROR [static]

**See Also**

ResultCode

#### 5.4.3.5 const ResultCode WRITE_ERROR [static]

**See Also**

ResultCode

## 5.5 Condition::ICondition Struct Reference

Interface for the implementation of Condition.

### Public Member Functions

- virtual void **notify** ()=0
- virtual void **wait_without_lock** ()=0
- virtual bool **wait_without_lock** (uint32_t)=0

### 5.5.1 Detailed Description

Condition uses an object that implements this interface to expose its functionality. There is a one-to-one mapping between ICondition methods and Condition's, i.e. they have the same syntax and semantics

**See Also**

Condition

## 5.6 ILogOutput Struct Reference

Log output forwarding interface.

### Public Member Functions

- virtual void log_message (LogMessageType message_type, const char *message)=0

  *Receives a formatted log message string with a severance as indicated by message_type.*

### 5.6.1 Member Function Documentation

#### 5.6.1.1 virtual void log_message ( LogMessageType *message_type*, const char * *message* ) [pure virtual]

**Parameters**

| | | |
|---|---|---|
| in | *message_type* | Log level |

**See Also**

Log.h

**Parameters**

| | | |
|---|---|---|
| in | *message* | String containing the log message |

## 5.7 IMutex Struct Reference

Abstract interface for the implementation of Mutex.

### Public Member Functions

- virtual void **lock** ()=0
- virtual void **unlock** ()=0
- virtual bool **trylock** ()=0

### 5.7.1 Detailed Description

**See Also**

Mutex and Condition

## 5.8 Thread::IRunnable Struct Reference

Interface of the object that Thread will execute in parallel.

### Public Member Functions

- virtual bool run ()=0

  *This function will run in its own thread.*

### 5.8.1 Member Function Documentation

#### 5.8.1.1 virtual bool run ( ) [pure virtual]

**Return values**

| | |
|---|---|
| *false* | It will be run again (looped) |
| *true* | It will stop and the thread will exit |

## 5.9 Semaphore::ISemaphore Struct Reference

Interface for the implementation of Semaphore.

### Public Member Functions

- virtual void **post** ()=0
- virtual void **wait** ()=0
- virtual bool **wait** (uint32_t timeout_in_ms)=0
- virtual bool **trywait** ()=0

### 5.9.1 Detailed Description

Semaphore uses an object that implements this interface to expose its functionality. There is a one-to-one mapping between ISemaphore methods and Semaphore's, i.e. they have the same syntax and semantics

**See Also**

Semaphore

## 5.10 Socket::ISocket Struct Reference

Interface for the implementation of socket functionality.

### Public Member Functions

- virtual void **open** ()=0
- virtual void **close** ()=0
- virtual ResultCode **connect** (const char ∗host, int port)=0
- virtual ResultCode **bind** (const char ∗host, int port)=0
- virtual ResultCode **send** (const uint8_t ∗data, uint32_t length)=0
- virtual ResultCode **receive** (uint8_t ∗data, uint32_t size, uint32_t &length)=0
- virtual ResultCode **set_receive_buffer_size** (uint32_t size)=0
- virtual ResultCode **set_reuse_address** (bool on)=0
- virtual ResultCode **set_non_blocking** (bool on)=0

#### 5.10.1 Detailed Description

Socket uses an object that implements this interface to expose socket functionality. There is a one-to-one mapping between ISocket methods and Socket's, i.e. they have the same syntax and semantics

**See Also**

Socket

## 5.11 Thread::IThread Struct Reference

Interface for the implementation of Thread.

### Public Member Functions

- virtual ResultCode **start** (IRunnable &runnable, Priority priority)=0
- virtual void **stop** ()=0
- virtual ResultCode **wait_until_stopped** ()=0
- virtual bool **is_running** ()=0
- virtual bool **must_stop** ()=0
- virtual ResultCode **stop_and_wait_until_stopped** ()=0
- virtual const std::string & **get_name** () const =0

#### 5.11.1 Detailed Description

Thread uses an object that implements this interface to expose the threading functions. There is a one-to-one mapping between IThread methods and Thread's, i.e. they have the same syntax and semantics

**See Also**

Thread N.B. the only exception is the static method Tread::self() because by definition static methods cannot be part of an interface. Furthermore, the implementation is hidden by default.

## 5.12 Keyboard Class Reference

Generic platform-independent keyboard interface Mainly meant for use with example applications, not intended for use with real clients.

### Static Public Member Functions

- static int get_key ()

  *Get a key from the console keyboard.*

### Static Public Attributes

- static const int TIMEOUT_IN_MS = 10

  *Time to block when no key is pressed.*
- static const int ESC_SEQ = 0x1000

  *Special key codes Common ASCII key codes like 'A' are not defined here.*
- static const int BACKSPACE_KEY = '\b'

  *Backspace key, same as '\b'.*
- static const int ENTER_KEY = '\r'

  *Enter key, same as '\r'.*
- static const int ESC_KEY = 0x1B

  *ESC-key (if not followed by a '[' character).*
- static const int DEL_KEY = 0x7F

  *DEL key.*
- static const int UP_KEY = 0x100

  *Cursor up key.*
- static const int DOWN_KEY = 0x101

  *Cursor down key.*
- static const int LEFT_KEY = 0x102

  *Cursor left key.*
- static const int RIGHT_KEY = 0x103

  *Cursor right key.*

## 5.12.1 Member Function Documentation

#### 5.12.1.1 static int get_key ( ) [static]

**Returns**

Key value (typically ASCII), EOF if end-of-file is reached or 0 if no key was pressed.

**Note**

*get_key()* typically blocks for at most *TIMEOUT_IN_MS* milliseconds waiting for a key to be pressed.
Escape sequences are returned as specific values. An ESC-'['-<key> sequence is returned as value ESC_SEQ + x, where x is the ASCII key value after the escape sequence. For example, the 'UP' arrow key typically encodes as ESC-'['-'A', which will then be returned as the value ESC_SEQ + 'A'. Other escape sequences are not translated and typically ignore the first ESC key. And ESC-ESC sequence returns as a single ESC_KEY character, 0x1B.

## 5.12.2 Field Documentation

#### 5.12.2.1 const int ESC_SEQ = 0x1000 [static]

ESC-key followed by a '[' character. The key value returned will be this value plus an escape-specific key value. For instance, the escape sequence ESC-'['-'E' will return the key value 0x1045, which equals ESC_SEQ + 'E'. (This holds for unrecognized escape sequences; Recognized sequences like ESC-'['-'A' will be returned as UP_KEY, for instance.)

## 5.13 X11KeyMap::KeyMap Struct Reference

A single key map entry for use in add_mapping()

### Data Fields

- int from_key

  *Platform key code to be mapped from.*
- X11KeyCode to_key

  *X11 key code to be generated.*

## 5.14 Mutex Class Reference

Generic mutex interface.

### Public Member Functions

- void lock ()

  *Lock operation to protect a critical region.*
- void unlock ()

  *Unlock operation to indicate the end of a critical region.*
- bool trylock ()

  *Try to lock the mutex.*

### 5.14.1 Detailed Description

Mutex uses an object that implements this interface to expose its functionality. There is a one-to-one mapping between IMutex methods and Mutex's, i.e. they have the same syntax and semantics

**See Also**

Mutex

### 5.14.2 Member Function Documentation

#### 5.14.2.1 void lock ( ) [inline], [virtual]

This method will wait until the mutex can be acquired. The mutex is recursive, thus if the same thread locks twice the same mutex, it won't block the execution.

Implements IMutex.

#### 5.14.2.2 bool trylock ( ) [inline], [virtual]

This method never blocks. If the mutex cannot be acquired, it will return false.

**Return values**

| | |
|---|---|
| *true* | If the mutex has been acquired. |
| *false* | If the mutex could not be acquired. |

Implements IMutex.

## 5.15 ResultCode Class Reference

Generic class to return the result of the methods.

### Public Member Functions

- ResultCode ()

  *Default constructor for non-initialized code.*
- int get_code () const

  *Return the unique code number of the result.*
- const char * get_description () const

  *Return a textual description of the result.*
- bool operator== (const ResultCode &rhs) const

  *Comparison operator.*
- bool operator!= (const ResultCode &rhs) const

  *Comparison operator.*
- ResultCode & operator= (const ResultCode &rhs)

  *Assignment operator.*
- bool is_ok () const

  *Return true if the object is ResultCode::SUCCESS.*
- bool is_error () const

  *Return true if the object is not ResultCode::SUCCESS.*
- ResultCode & operator|= (const ResultCode &rhs)

  *Combine this result code with the right hand side code. If this code is_ok(), the right hand side is taken. If this code is_error(), the code is not changed.*

### Static Public Attributes

- static const ResultCode SUCCESS

  *Operation succeeded.*

### 5.15.1 Detailed Description

The design of this class has the following features:

- Minimum overhead when the objects are copied (e.g. when returning an object)
- The object can be compared
- Type check during compiling time
- Access to a textual description of the result (

**See Also**

get_description())

- Each defined ResultCode has an unique code number (

**See Also**

get_code())

### 5.15.2 Member Function Documentation

#### 5.15.2.1 int get_code ( ) const [inline]

**Returns**

Unique code number of the result

#### 5.15.2.2 const char* get_description ( ) const

**Returns**

Pointer to the array that contains the textual description of the result

#### 5.15.2.3 bool is_error ( ) const [inline]

**Returns**

True is the object is not ResultCode::SUCCESS. False otherwise.

#### 5.15.2.4 bool is_ok ( ) const [inline]

**Returns**

True is the object is ResultCode::SUCCESS. False otherwise.

#### 5.15.2.5 bool operator!= ( const ResultCode & *rhs* ) const [inline]

**Parameters**

| | | |
|---|---|---|
| in | *rhs* | Operand on the right hand side of "!=" sign |

**Returns**

True, if both ResultCode objects are different. False otherwise.

#### 5.15.2.6 ResultCode& operator= ( const ResultCode & *rhs* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *rhs* | Operand on the right hand side of "=" sign |

**Returns**

Reference to this

#### 5.15.2.7 bool operator== ( const ResultCode & *rhs* ) const [inline]

**Parameters**

| | | |
|---|---|---|
| in | *rhs* | Operand on the right hand side of "==" sign |

**Returns**

True, if both ResultCode objects are the same. False otherwise.

#### 5.15.2.8 ResultCode& operator|= ( const ResultCode & *rhs* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *rhs* | Operand on the right hand side of "\|=" sign |

**Returns**

Reference to this

## 5.16 Semaphore Class Reference

Generic platform-independent semaphore interface.

### Data Structures

- struct ISemaphore

  *Interface for the implementation of Semaphore.*

### Public Member Functions

- Semaphore ()

  *Semaphores are counting semaphore objects used for synchronization between threads. They are constructed with a count of 0.*
- void post ()

  *Post to the semaphore; the semaphore's count is incremented by 1. If any thread is blocked on wait(), one of them is released now.*
- void wait ()

  *Wait for the semaphore to have a count greater than 0. If so, the semaphore's count is decremented and the call returns. If not, the method will wait indefinitely until some other thread calls post().*

- bool wait (uint32_t timeout_in_ms)

  *Wait for the semaphore to have a count greater than 0 using a timeout. If the count is greater than 0 when the call is made, the semaphore's count is decremented and the call returns immediately with the return value true. If not, the method will wait for at most timeout_in_ms milliseconds for post() to be called by another thread. If post() is called before the timeout expires, wait() returns true. If post() was not called within timeout_in_ms milliseconds, wait() returns false.*
- bool trywait ()

  *Check whether the semaphore has a count greater than 0. If so, the semaphore's count is decremented and the call returns immediately with the return value true. If not, the method returns immediately with the return value false.*

### 5.16.1 Member Function Documentation

#### 5.16.1.1 bool trywait ( ) [inline]

**Return values**

| | |
|---|---|
| *true* | if successful; the semaphore count will be decremented by one. |
| *false* | if unsuccessful; the semaphore count will not have changed and still be 0. |

#### 5.16.1.2 bool wait ( uint32_t *timeout_in_ms* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *timeout_in_ms* | The maximum time to wait for another thread to call *post()* before returning false. |

**Return values**

| | |
|---|---|
| *true* | if successful; the semaphore count will be decremented by one. |
| *false* | if a timeout occurred; the semaphore count will not have changed and still be 0. |

## 5.17 Socket Class Reference

Generic socket interface.

### Data Structures

- struct ISocket

  *Interface for the implementation of socket functionality.*

### Public Member Functions

- Socket (ISocket &impl)

  *Constructor of Socket.*
- void open ()

  *Open the socket.*
- void close ()

  *Close the socket.*
- ResultCode connect (const char *host, int port)

*Establish a connection with a host in the specified port.*

- ResultCode bind (const char *host, int port)

  *Bind a port to a local address and in a specific port.*
- ResultCode send (const uint8_t *data, uint32_t length)

  *Send data to the host.*
- ResultCode receive (uint8_t *data, uint32_t size, uint32_t &length)

  *Receive data from the socket.*
- ResultCode set_receive_buffer_size (uint32_t size)

  *Set the size of the socket buffer of the platform.*
- ResultCode set_reuse_address (bool on)

  *The local address can be re-used for binding operations.*

- ISocket & **get_impl** ()

### Static Public Member Functions

- static ResultCode get_local_address (std::string &local_address)

  *Get the address that is bound to the network adapter (usually DHCP assigned).*

### Static Public Attributes

- static const ResultCode SOCKET_NOT_OPEN

  *The socket is not open.*
- static const ResultCode READ_ERROR

  *Error reading from the socket.*
- static const ResultCode WRITE_ERROR

  *Error writing to the socket.*
- static const ResultCode BIND_ERROR

  *Error when binding the socket to a port.*
- static const ResultCode HOST_NOT_FOUND

  *Host not found error.*
- static const ResultCode CONNECTION_REFUSED

  *The connection could not be established because it was refused by the server.*
- static const ResultCode CONNECT_FAILED

  *The connection could not be established.*
- static const ResultCode CONNECT_TIMEOUT

  *Timeout while trying to establish a connection.*
- static const ResultCode LISTEN_FAILED

  *The port could not be listened.*
- static const ResultCode SOCKET_OPTION_ACCESS_FAILED

  *Error in the given options.*
- static const ResultCode THREAD_SHUTDOWN

  *A blocking call was interrupted because the calling thread is shut down.*

### Protected Attributes

- ISocket & **m_impl**

### 5.17.1 Detailed Description

The implementation is done as part of the porting layer. All implementation is forwarded to the private Impl class. Coding this inline should speed-up code and reduce code size.

This class should not be used directly, but one of the derived classes instead

**See Also**

UdpSocket, TcpSocket, SslSocket

### 5.17.2 Constructor & Destructor Documentation

#### 5.17.2.1 Socket ( ISocket & *impl* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *impl* | Object that implements ISocket interface to implement the required functionality |

### 5.17.3 Member Function Documentation

#### 5.17.3.1 ResultCode bind ( const char * *host*, int *port* ) [inline]

After successfully binding the port, data can be received using the method receive()

**Parameters**

| | | |
|---|---|---|
| in | *host* | Local address in either dotted notation or FQDN |
| in | *port* | Local port |

**Return values**

| | |
|---|---|
| *ResultCode::SUCCESS* | If socket has been successfully bound. |
| *SOCKET_NOT_OPEN* | When the socket has not been previously opened |
| *HOST_NOT_FOUND* | When the hostname cannot be resolved or be reached |
| *BIND_ERROR* | When the given host/port could not be bound |

#### 5.17.3.2 ResultCode connect ( const char * *host*, int *port* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *host* | Destination address in either dotted notation or FQDN |
| in | *port* | Destination port |

**Return values**

| | |
|---|---|
| *ResultCode::SUCCESS* | When the connection has been established |
| *SOCKET_NOT_OPEN* | When the socket has not been previously opened |
| *HOST_NOT_FOUND* | When the hostname cannot be resolved or be reached |

| | |
|---|---|
| *CONNECTION_REFUSED* | When the server has refused the connection request |

#### 5.17.3.3 static ResultCode get_local_address ( std::string & *local_address* ) [static]

**Parameters**

| | | |
|---|---|---|
| out | *local_address* | The local address in dotted IP notatation (e.g. 172.178.16.128). |

**Return values**

| | |
|---|---|
| *ResultCode::SUCCESS* | If the operation succeeded. |
| *SOCKET_NOT_OPEN* | When the temporary socket failed to open. |
| *SOCKET_OPTION_ACCES-S_FAILED* | If the operation failed. |

#### 5.17.3.4 ResultCode receive ( uint8_t * *data*, uint32_t *size*, uint32_t & *length* ) [inline]

This method only can be used after a successful binding.

**Parameters**

| | | |
|---|---|---|
| in | *data* | Buffer where the received data will be stored |
| in | *size* | Size of the buffer "data" |
| out | *length* | Number of bytes received |

**Return values**

| | |
|---|---|
| *ResultCode::SUCCESS* | If the operation succeeded. This does not imply that actual data was received. If length == 0, this indicates that the connection was closed by the peer. |
| *SOCKET_NOT_OPEN* | When the socket has not been previously opened. |
| *READ_ERROR* | When there was an error with the reading, e.g. the socket was not opened. |
| *THREAD_SHUTDOWN* | When the call was interrupted because the calling thread is shut down. |

#### 5.17.3.5 ResultCode send ( const uint8_t * *data*, uint32_t *length* ) [inline]

This method only can be used after a successful connection has been established

**See Also**

connect().

**Parameters**

| | | |
|---|---|---|
| in | *data* | Array of bytes to be sent |
| in | *length* | Number of bytes to be sent |

**Return values**

| | |
|---|---|
| *ResultCode::SUCCESS* | If the entire buffer has been successfully sent |
| *SOCKET_NOT_OPEN* | When the socket has not been previously opened |
| *WRITE_ERROR* | When there was an error with the writing, e.g. the socket was not opened or the connection is closed. |

#### 5.17.3.6 ResultCode set_receive_buffer_size ( uint32_t *size* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *size* | Size of the socket buffer |

**Returns**

ResultCode::SUCCESS If the new size has been successfully set.

**Return values**

| | |
|---|---|
| *SOCKET_NOT_OPEN* | When the socket has not been previously opened. |
| *SOCKET_OPTION_ACCES-S_FAILED* | If the operation failed. |

#### 5.17.3.7 ResultCode set_reuse_address ( bool *on* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *on* | Whether the local address can be reused. |

**Return values**

| | |
|---|---|
| *ResultCode::SUCCESS* | If the operation succeeded. |
| *SOCKET_NOT_OPEN* | When the socket has not been previously opened. |
| *SOCKET_OPTION_ACCES-S_FAILED* | If the operation failed. |

## 5.18 SslSocket Class Reference

SSL socket interface.

### Additional Inherited Members

## 5.19 TcpSocket Class Reference

TCP socket interface.

### Public Member Functions

- virtual ResultCode listen (uint32_t backlog)

*Listen into an specific port for incoming connection request.*

- virtual TcpSocket * accept ()

  *Accept a new connection.*

- virtual ResultCode set_no_delay (bool on)

  *If set, the data will be sent as soon as possible.*

## Protected Member Functions

- **TcpSocket** (ISocket &)

## Additional Inherited Members

### 5.19.1 Member Function Documentation

#### 5.19.1.1 virtual TcpSocket* accept ( ) [virtual]

This method can only be used after listen() succeeded.

**Returns**

A pointer to an TcpSocket object to send or receive data or 0 if accept() failed or was interrupted because the calling thread is shut down.

#### 5.19.1.2 virtual ResultCode listen ( uint32_t *backlog* ) [virtual]

**Parameters**

| | | |
|---|---|---|
| in | *backlog* | Maximum queue length. When the backlog is exhausted, new connection will be rejected. |

**Return values**

| | |
|---|---|
| *ResultCode::SUCCESS* | If the operation succeeded. |
| *SOCKET_NOT_OPEN* | When the socket has not been previously opened. |
| *LISTEN_FAILED* | If the operation failed. |

#### 5.19.1.3 virtual ResultCode set_no_delay ( bool *on* ) [virtual]

When the amount of data is small, the platform will try to wait for more data before sending it. If this flag is set, the data is sent as soon as possible.

**Parameters**

| | | |
|---|---|---|
| in | *on* | True to send the data as soon as possible. |

**Return values**

| | |
|---|---|
| *ResultCode::SUCCESS* | If the operation succeeded. |
| *SOCKET_NOT_OPEN* | When the socket has not been previously opened. |
| *SOCKET_OPTION_ACCES-S_FAILED* | If the operation failed. |

## 5.20 Thread Class Reference

Generic thread interface.

### Data Structures

- struct IRunnable

  *Interface of the object that Thread will execute in parallel.*
- struct IThread

  *Interface for the implementation of Thread.*

### Public Types

- enum Priority {
  PRIO_LOW, PRIO_NORMAL,
  PRIO_HIGH, PRIO_HIGHEST }

  *Priority levels at which a thread can run.*

### Public Member Functions

- Thread (const std::string &thread_name)

  *Constructor.*
- ResultCode start (IRunnable &runnable, Priority priority)

  *Execute the function in a separate thread.*
- void stop ()

  *Stop the current running thread (if any)*
- ResultCode wait_until_stopped ()

  *Wait for the thread until it stops.*
- bool is_running ()

  *Check if Thread is currently executing an IRunnable object.*
- bool must_stop ()

  *Check if Thread must stop.*
- ResultCode stop_and_wait_until_stopped ()

  *Stop the thread and wait until it finishes.*
- const std::string & get_name () const

  *Get a reference to the thread name.*

### Static Public Member Functions

- static void sleep (uint32_t time_in_milliseconds)

  *Sleep for the given number of milliseconds.*

- static Thread * self ()

  *Get a pointer to the current thread.*

### Static Public Attributes

- static const ResultCode THREAD_ALREADY_STARTED

  *The thread was already started.*

- static const ResultCode CANNOT_CREATE_THREAD

  *The thread could not be created.*

- static const ResultCode CANNOT_SET_THREAD_PRIORITY

  *The thread priority could not be set.*

- static const ResultCode FAILED_WAITING_FOR_THREAD_TO_FINISH

  *The thread could not be joined.*

## 5.20.1 Detailed Description

The implementation is done as part of the porting layer. All implementation is forwarded to the private Impl class. Coding this inline should speed-up code and reduce code size.

## 5.20.2 Member Enumeration Documentation

### 5.20.2.1 enum Priority

**Enumerator**

***PRIO_LOW*** Priority below normal. Such a thread should only run if no other threads can run at that moment.

***PRIO_NORMAL*** Normal priority, at which threads run that would otherwise not have their priority set. Typically the same prority as that of the main() thread.

***PRIO_HIGH*** Priority above normal. Such a thread should run if possible (i.e. if not blocked), unless a higher priority thread can run.

***PRIO_HIGHEST*** Highest priority. Such a thread should run at all times if it can (i.e. if not blocked).

## 5.20.3 Constructor & Destructor Documentation

### 5.20.3.1 Thread ( const std::string & *thread_name* )

**Parameters**

| | | |
|---|---|---|
| in | *thread_name* | Name of the current thread. |

### 5.20.4 Member Function Documentation

#### 5.20.4.1 const std::string& get_name ( ) const [inline]

**Returns**

Reference to the thread name.

#### 5.20.4.2 bool is_running ( ) [inline]

**Return values**

| | |
|---|---|
| *True* | if the thread is running. |
| *False* | if the thread is not running. |

#### 5.20.4.3 bool must_stop ( ) [inline]

**Return values**

| | |
|---|---|
| *True* | if the thread is running and has been signaled to stop. |
| *False* | if the thread is not running or not signaled to stop. |

#### 5.20.4.4 static Thread* self ( ) [static]

**Returns**

Pointer to current thread or NULL if if not on an explicit thread, e.g. the main thread.

#### 5.20.4.5 static void sleep ( uint32_t *time_in_milliseconds* ) [static]

**Parameters**

| | | |
|---|---|---|
| in | *time_in_milliseconds* | Number of milliseconds to sleep. |

#### 5.20.4.6 ResultCode start ( IRunnable & *runnable*, Priority *priority* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *runnable* | Object with run() function to be executed in a separated thread. |
| in | *priority* | Priority at which the thread is expected to run. |

**Return values**

| | |
|---|---|
| *ResultCode::SUCCESS* | When the thread has started successfully. |

| | |
|---|---|
| *CANNOT_CREATE_THREAD* | If the thread couldn't be started, e.g. lack of resources. |
| *THREAD_ALREADY_STARTED* | If there is already an IRunnable being executed. |
| *CANNOT_SET_THREAD_PRIORITY* | If the requested thread priority cannot be set. |

#### 5.20.4.7 ResultCode stop_and_wait_until_stopped ( ) [inline]

**See Also**

stop() and wait_until_stopped()

**Returns**

**See Also**

wait_until_stopped()

#### 5.20.4.8 ResultCode wait_until_stopped ( ) [inline]

stop() shall be called previously

**Return values**

| | |
|---|---|
| *ResultCode::SUCCESS* | If the thread has successfully finished |
| *FAILED_WAITING_FOR_THREAD_TO_FINISH* | When an error condition ocurred, e.g. no thread was started. |

## 5.21 TimeStamp Class Reference

Generic time and time stamp interface.

### Public Member Functions

- TimeStamp ()

  *Construct a new TimeStamp object.*
- TimeStamp (const TimeStamp &rhs)

  *Construct a new TimeStamp object with initial value.*
- TimeStamp & operator= (const TimeStamp &rhs)

  *Assigns a new value to the TimeStamp, replacing its current contents.*
- bool is_valid () const

  *Check if the stored time value is valid.*
- bool is_absolute () const

  *Check if the stored time value is absolute.*

- bool is_relative () const

  *Check if the stored time value is relative.*
- bool is_comparable (const TimeStamp &rhs) const

  *Check if the stored time value can be compared to the other.*
- void invalidate ()

  *Make the time stamp invalid.*
- int64_t get_as_microseconds () const

  *Get the time value in microseconds.*
- int64_t get_as_milliseconds () const

  *Get the time value in milliseconds.*
- int64_t get_as_seconds () const

  *Get the time value in seconds.*
- TimeStamp & add_microseconds (int64_t delta_in_us)

  *Add microseconds to a TimeStamp.*
- TimeStamp & add_milliseconds (int64_t delta_in_ms)

  *Add milliseconds to a TimeStamp.*
- TimeStamp & add_seconds (int32_t delta_in_s)

  *Add seconds to a TimeStamp.*

- bool operator== (const TimeStamp &rhs) const

  *Comparison operators. Assumes is_comparable() to be true.*
- bool **operator>=** (const TimeStamp &rhs) const
- bool **operator<=** (const TimeStamp &rhs) const
- bool **operator!=** (const TimeStamp &rhs) const
- bool **operator<** (const TimeStamp &rhs) const
- bool **operator>** (const TimeStamp &rhs) const

- TimeStamp & operator+= (const TimeStamp &rhs)

  *Arithmetic operators.*
- TimeStamp & **operator-=** (const TimeStamp &rhs)
- TimeStamp **operator+** (const TimeStamp &rhs) const
- TimeStamp **operator-** (const TimeStamp &rhs) const

### Static Public Member Functions

- static TimeStamp now ()

  *Sample the current time.*
- static TimeStamp zero ()

  *Return a relative time of 0.*

### 5.21.1 Detailed Description

The implementation is done as part of the porting layer. The inline coding is done to reduce code size, increase speed and reduce memory usage.

### 5.21.2 Constructor & Destructor Documentation

#### 5.21.2.1 TimeStamp ( const TimeStamp & *rhs* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *rhs* | Initial value. |

### 5.21.3 Member Function Documentation

#### 5.21.3.1 TimeStamp& add_microseconds ( int64_t *delta_in_us* ) [inline]

**Note**

Assumes the time stamp is valid

**Parameters**

| | | |
|---|---|---|
| in | *delta_in_us* | Time in microseconds. |

**Returns**

TimeStamp reference to this.

**Note**

This alters the current object value.

#### 5.21.3.2 TimeStamp& add_milliseconds ( int64_t *delta_in_ms* ) [inline]

**Note**

Assumes the time stamp is valid

**Parameters**

| | | |
|---|---|---|
| in | *delta_in_ms* | Time in milliseconds. |

**Returns**

TimeStamp reference to this.

**Note**

This alters the current object value.

#### 5.21.3.3 TimeStamp& add_seconds ( int32_t *delta_in_s* ) [inline]

**Note**

Assumes the time stamp is valid.

**Parameters**

| | | |
|---|---|---|
| in | *delta_in_s* | Time in seconds. |

**Returns**

TimeStamp reference to this.

**Note**

This alters the current object value.

#### 5.21.3.4 int64_t get_as_microseconds ( ) const [inline]

**Returns**

Time stamp value in microseconds. May return absolute or relative time, depending on is_absolute()

**Note**

Assumes the time stamp is valid

#### 5.21.3.5 int64_t get_as_milliseconds ( ) const [inline]

**Returns**

Time stamp value in milliseconds. May return absolute or relative time, depending on is_absolute()

**Note**

Assumes the time stamp is valid

#### 5.21.3.6 int64_t get_as_seconds ( ) const [inline]

**Returns**

Time stamp value in seconds. May return absolute or relative time, depending on is_absolute()

**Note**

Assumes the time stamp is valid

#### 5.21.3.7 bool is_absolute ( ) const [inline]

**Return values**

| | |
|---|---|
| *true* | if absolute, false otherwise. |

#### 5.21.3.8 bool is_comparable ( const TimeStamp & *rhs* ) const [inline]

**Parameters**

| | | |
|---|---|---|
| in | *rhs* | Reference to object being compared. |

**Return values**

| | |
|---|---|
| *true* | if so, false otherwise. If true, both time stamps are valid and of the same type. |

#### 5.21.3.9 bool is_relative ( ) const [inline]

**Return values**

| | |
|---|---|
| *true* | if relative, false otherwise. |

#### 5.21.3.10 bool is_valid ( ) const [inline]

**Return values**

| | |
|---|---|
| *true* | if valid, false otherwise. |

#### 5.21.3.11 TimeStamp& operator= ( const TimeStamp & *rhs* ) [inline]

**Parameters**

| | | |
|---|---|---|
| in | *rhs* | New value. |

**Returns**

Reference to this object.

## 5.22 UdpSocket Class Reference

UDP socket interface.

### Additional Inherited Members

## 5.23 X11KeyMap Class Reference

Class for the key mapping between the platform keycodes and X11 keycodes.

## Data Structures

- struct KeyMap

  *A single key map entry for use in add_mapping()*

## Public Member Functions

- void add_mapping (int from_key, X11KeyCode to_key)

  *Add a new mapping of a native key code to an X11 key code.*
- void add_mapping (const KeyMap map[], unsigned int n_entries)

  *Add a new set of translations from native key code to X11 key code.*
- X11KeyCode translate (int native_key) const

  *Translate a native key code to an X11 key code.*

### 5.23.1 Member Function Documentation

#### 5.23.1.1 void add_mapping ( int *from_key*, X11KeyCode *to_key* )

**Parameters**

| | | |
|---|---|---|
| in | *from_key* | The native (remote control) key code. |
| in | *to_key* | The X11KeyCode it maps to. |

#### 5.23.1.2 void add_mapping ( const KeyMap *map[]*, unsigned int *n_entries* )

**Parameters**

| | | |
|---|---|---|
| in | *map* | The map structure array pointer. |
| in | *n_entries* | The number of entries in the array. |

#### 5.23.1.3 X11KeyCode translate ( int *native_key* ) const

**Parameters**

| | | |
|---|---|---|
| in | *native_key* | The native (remote control) key code. |

**Returns**

X11 translated code for *native_key* or the native key if no mapping exists at all, or X11_INVALID if a keymap was set, but no mapping exists for *native_key*.

# Index