

libcot
0:0:0

Generated by Doxygen 1.5.6

Thu Aug 13 14:09:24 2009

Contents

1	oBIX Client Library Documentation	1
1.1	Compilation	2
2	Data Structure Index	2
2.1	Data Structures	2
3	File Index	2
3.1	File List	2
4	Data Structure Documentation	3
4.1	_oBIX_BatchResult Struct Reference	3
4.1.1	Detailed Description	3
4.1.2	Field Documentation	3
5	File Documentation	4
5.1	bool.h File Reference	4
5.1.1	Detailed Description	4
5.1.2	Define Documentation	4
5.1.3	Typedef Documentation	5
5.2	example_timer.c File Reference	5
5.2.1	Detailed Description	6
5.2.2	Define Documentation	6
5.2.3	Function Documentation	6
5.2.4	Variable Documentation	8
5.3	ixml_ext.h File Reference	9
5.3.1	Detailed Description	10
5.3.2	Function Documentation	10
5.4	log_utils.h File Reference	14
5.4.1	Detailed Description	16
5.4.2	Define Documentation	16
5.4.3	Typedef Documentation	17
5.4.4	Enumeration Type Documentation	17
5.4.5	Function Documentation	18
5.4.6	Variable Documentation	18
5.5	mainpage.h File Reference	18
5.5.1	Detailed Description	19
5.6	obix_client.h File Reference	19

5.6.1	Detailed Description	21
5.6.2	Usage	21
5.6.3	Typedef Documentation	22
5.6.4	Enumeration Type Documentation	23
5.6.5	Function Documentation	24
5.7	obix_utils.h File Reference	33
5.7.1	Detailed Description	36
5.7.2	Function Documentation	36
5.7.3	Variable Documentation	37
5.8	ptask.h File Reference	39
5.8.1	Detailed Description	40
5.8.2	Usage	40
5.8.3	Typedef Documentation	41
5.8.4	Function Documentation	41
5.9	xml_config.h File Reference	44
5.9.1	Detailed Description	46
5.9.2	Function Documentation	46
5.9.3	Variable Documentation	49

1 oBIX Client Library Documentation

oBIX Client Library (libcot) is a small library written in C, which allows creating lightweight oBIX client applications such as device adapters.

Library API consists of [obix_client.h](#) which declares oBIX Client API and several utility header files which provides helper tools commonly needed by oBIX client applications:

- [ptask.h](#) - Allows scheduling tasks to be executed in a separate thread;
- [obix_utils.h](#) - Contains names of most essential oBIX object, attributes, etc.
- [xml_config.h](#) - Helps to load application settings from XML file;
- [log_utils.h](#) - Provides interface to the logging utility which is used by the whole library.
- [ixml_ext.h](#) - Contains utilities for manipulating XML DOM structures.

All parts of the library produce log messages which give better understanding of what happens inside and investigate occurred problems. By default the logging system writes everything (including all debug messages) to *stdout* which can be inconvenient. Log settings can be changed either manually (using functions defined at [log_utils.h](#)) or during oBIX Client API initialization (using [obix_loadConfigFile\(\)](#)).

The example usage of the library can be found at [example_timer.c](#)

1.1 Compilation

The following string will compile application which uses oBIX Client Library:

```
gcc -I<cot_headers> -L<cot_lib> -lcot-client <source> -o <output_name>
```

where

- *<cot_headers>* - Path to header files of *libcot* (usually it is *<installation_prefix>/include/cot/*).
- *<cot_lib>* - Path to library binaries of *libcot* (usually it is *<installation_prefix>/lib*).
- *<sources>* - Your source files to be compiled.
- *<output_name>* - Name of the output binary.

Author:

Andrey Litvinov

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

[_oBIX_BatchResult](#) (Contains outputs of the command, which was executed in a [Batch](#)) 3

3 File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

bool.h (Defines boolean data type)	4
example_timer.c (Simple oBIX Timer device implementation, which demonstrates usage of oBIX client library)	5
example_timer_config.xml	??
ixml_ext.h (Defines utility methods for work with XML DOM structure)	9
log_utils.h (Definitions of logging tools)	14
mainpage.h (Main page of the documentation generated by Doxygen)	18
obix_client.h (OBIX Client API)	19
obix_utils.h (Contains oBIX keywords (object names, contracts, facets, etc) and some utility functions)	33
ptask.h (Periodic Task - tool for asynchronous task execution)	39

[xml_config.h](#) (Declares configuration API)

44

4 Data Structure Documentation

4.1 `_oBIX_BatchResult` Struct Reference

Contains outputs of the command, which was executed in a [Batch](#).

```
#include <obix_client.h>
```

Data Fields

- `IXML_Element * obj`
XML object returned by the function, if available (e.g.
- `int status`
Return value of the executed command.
- `char * value`
String value returned by the function, if available (e.g.

4.1.1 Detailed Description

Contains outputs of the command, which was executed in a [Batch](#).

Definition at line 467 of file `obix_client.h`.

4.1.2 Field Documentation

4.1.2.1 `IXML_Element* _oBIX_BatchResult::obj`

XML object returned by the function, if available (e.g.
for [obix_batch_read\(\)](#)).

Definition at line 483 of file `obix_client.h`.

4.1.2.2 `int _oBIX_BatchResult::status`

Return value of the executed command.

It is identical to the return value of the corresponding command executed without Batch.

Definition at line 473 of file `obix_client.h`.

4.1.2.3 `char* _oBIX_BatchResult::value`

String value returned by the function, if available (e.g.
for [obix_batch_readValue\(\)](#)).

Definition at line 478 of file `obix_client.h`.

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

- [obix_client.h](#)

5 File Documentation

5.1 bool.h File Reference

Defines boolean data type.

Defines

- #define [FALSE](#) 0
This is false.
- #define [TRUE](#) 1
That's true.

Typedefs

- typedef int [BOOL](#)
Boolean data type which is so natural for all programmers.

5.1.1 Detailed Description

Defines boolean data type.

Just in case if nobody defined it before.

Author:

Andrey Litvinov

Version:

1.0

Definition in file [bool.h](#).

5.1.2 Define Documentation

5.1.2.1 #define FALSE 0

This is *false*.

Definition at line 43 of file bool.h.

5.1.2.2 #define TRUE 1

That's *true*.

Definition at line 38 of file bool.h.

Referenced by main().

5.1.3 Typedef Documentation

5.1.3.1 typedef int BOOL

Boolean data type which is so natural for all programmers.

Definition at line 34 of file bool.h.

5.2 example_timer.c File Reference

Simple oBIX Timer device implementation, which demonstrates usage of oBIX client library.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <obix_utils.h>
#include <obix_client.h>
#include <ptask.h>
```

Defines

- #define [CONNECTION_ID](#) 0
ID of the connection which is described in configuration file.

Functions

- char * [getDeviceData](#) (char *deviceUri)
Generates device data in oBIX format.
- int [main](#) (int argc, char **argv)
Entry point of the Timer application.
- int [resetListener](#) (int connectionId, int deviceId, int listenerId, const char *newValue)
Handles changes of "reset" value.
- void [timerTask](#) (void *arg)
Updates timer value and writes it to oBIX server.

Variables

- `Task_Thread * _taskThread`
Separate thread for timer value updating.
- `long _time`
Elapsed time is stored here.
- `pthread_mutex_t _time_mutex = PTHREAD_MUTEX_INITIALIZER`
Need mutex for synchronization, because _time variable is accessed from two threads.
- `int _timerTaskId`
ID of the task which increases time every second.
- `const char * DEVICE_DATA`
Data which is posted to oBIX server.

5.2.1 Detailed Description

Simple oBIX Timer device implementation, which demonstrates usage of oBIX client library.

It shows the time elapsed after timer was started or reset by user. Device registers itself at oBIX server, regularly updates elapsed time on it and listens to updates of "reset" parameter. If someone changes "reset" to true, then elapsed time is set to 0. Configuration file template can be found at res/example_timer_config.xml

Author:

Andrey Litvinov

Version:

1.1

Definition in file [example_timer.c](#).

5.2.2 Define Documentation

5.2.2.1 #define CONNECTION_ID 0

ID of the connection which is described in configuration file.

Definition at line 46 of file [example_timer.c](#).

Referenced by [main\(\)](#), and [timerTask\(\)](#).

5.2.3 Function Documentation

5.2.3.1 char* getDeviceData (char * deviceUri)

Generates device data in oBIX format.

Parameters:

deviceUri Address at the server where device will be stored to. This address will be written as the *href* attribute of the root object.

Returns:

Data which should be posted to the server.

Definition at line 215 of file example_timer.c.

References DEVICE_DATA.

Referenced by main().

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

Entry point of the Timer application.

It takes the name of the configuration file (use [example_timer_config.xml](#)).

See also:

[example_timer_config.xml](#)

Definition at line 233 of file example_timer.c.

References _taskThread, _timerTaskId, CONNECTION_ID, EXECUTE_INDEFINITE, getDeviceData(), obix_dispose(), obix_loadConfigFile(), obix_openConnection(), obix_registerDevice(), obix_registerListener(), OBIX_SUCCESS, ptask_cancel(), ptask_dispose(), ptask_init(), ptask_schedule(), resetListener(), timerTask(), and TRUE.

5.2.3.3 int resetListener (int connectionId, int deviceId, int listenerId, const char * newValue)

Handles changes of "reset" value.

The function implements [obix_update_listener\(\)](#) prototype and is registered as a listener of "reset" param using [obix_registerListener\(\)](#). If "reset" value is changed at oBIX server to "true" it will set it back to "false" and reset timer.

See also:

[obix_update_listener\(\)](#), [obix_registerListener\(\)](#).

Definition at line 76 of file example_timer.c.

References _taskThread, _time, _time_mutex, _timerTaskId, obix_batch_create(), obix_batch_send(), obix_batch_writeValue(), OBIX_SUCCESS, OBIX_T_BOOL, OBIX_T_RELTIME, ptask_reset(), and XML_FALSE.

Referenced by main().

5.2.3.4 void timerTask (void * arg)

Updates timer value and writes it to oBIX server.

Implements [periodic_task\(\)](#) prototype and is scheduled using [ptask_schedule\(\)](#). This method is executed in a separate thread that is why it uses [_time_mutex](#) for synchronization with [resetListener\(\)](#) which sets timer to 0.

See also:

[periodic_task\(\)](#), [ptask_schedule\(\)](#).

Parameters:

arg Assumes that a pointer to the device ID is passed here. Device ID is used for updating time value at the server.

Definition at line 182 of file example_timer.c.

References `_time`, `_time_mutex`, `CONNECTION_ID`, `obix_reltime_fromLong()`, `OBIX_SUCCESS`, `OBIX_T_RELTIME`, and `obix_writeValue()`.

Referenced by `main()`.

5.2.4 Variable Documentation

5.2.4.1 Task_Thread* _taskThread

Separate thread for timer value updating.

Definition at line 63 of file example_timer.c.

Referenced by `main()`, and `resetListener()`.

5.2.4.2 long _time

Elapsed time is stored here.

Definition at line 55 of file example_timer.c.

Referenced by `resetListener()`, and `timerTask()`.

5.2.4.3 int _timerTaskId

ID of the task which increases time every second.

Definition at line 65 of file example_timer.c.

Referenced by `main()`, and `resetListener()`.

5.2.4.4 const char* DEVICE_DATA

Initial value:

```
"<obj name=\"ExampleTimer\" displayName=\"Example Timer\" href=\"%s\">\r\n"
"  <reltime name=\"time\" displayName=\"Elapsed Time\" href=\"%stime\" val=\"PT0S\" writable=\"true\"/"
"  <bool name=\"reset\" displayName=\"Reset Timer\" href=\"%sreset\" val=\"false\" writable=\"true\"/"
"</obj>"
```

Data which is posted to oBIX server.

Definition at line 48 of file example_timer.c.

Referenced by `getDeviceData()`.

5.3 ixml_ext.h File Reference

Defines utility methods for work with XML DOM structure.

```
#include <upnp/ixml.h>
```

Defines

- `#define BOOL_H_`
ixml.h already contains the definition of `BOOL` type, thus we consider that [bool.h](#) is already added.

Functions

- `IXML_Element * ixmlAttr_getOwnerElement (IXML_Attr *attr)`
Returns element which the provided attribute belongs to.
- `IXML_Element * ixmlDocument_getElementByAttrValue (IXML_Document *doc, const char *attrName, const char *attrValue)`
Returns first element in the documents with provided attribute value.
- `IXML_Element * ixmlDocument_getRootElement (IXML_Document *doc)`
Returns root element (root tag) of the XML document.
- `IXML_Element * ixmlElement_cloneWithLog (IXML_Element *source)`
Duplicates provided element.
- `int ixmlElement_copyAttributeWithLog (IXML_Element *source, IXML_Element *target, const char *attrName, BOOL obligatory)`
Copies attribute value from one element to another.
- `void ixmlElement_freeOwnerDocument (IXML_Element *element)`
Frees the IXML_Document which the provided element belongs to.
- `IXML_Element * ixmlElement_parseBuffer (const char *data)`
Parses an XML text buffer and returns the parent element of the generated DOM structure.
- `int ixmlElement_removeAttributeWithLog (IXML_Element *element, const char *attrName)`
Removes attribute from the provided element.
- `int ixmlElement_setAttributeWithLog (IXML_Element *element, const char *attrName, const char *attrValue)`
Adds new attribute to the element.
- `void ixmlNode_freeOwnerDocument (IXML_Node *node)`
Frees the IXML_Document which the provided node belongs to.
- `IXML_Node * ixmlNode_parseBuffer (const char *data)`
Parses an XML text buffer and returns the parent node of the generated DOM structure.

XML node types conversion

- IXML_Node * [ixmlAttr_getNode](#) (IXML_Attr *attr)
Returns node which represents provided attribute.
- IXML_Node * [ixmlDocument_getNode](#) (IXML_Document *doc)
Returns node which represents provided document.
- IXML_Node * [ixmlElement_getNode](#) (IXML_Element *element)
Returns node which represents provided element.
- IXML_Attr * [ixmlNode_convertToAttr](#) (IXML_Node *node)
Converts node to the attribute.
- IXML_Element * [ixmlNode_convertToElement](#) (IXML_Node *node)
Converts node to the element.

5.3.1 Detailed Description

Defines utility methods for work with XML DOM structure.

Expands functionality of *ixml* library which provides DOM XML parser functionality. *ixml* is distributed as a part of *libupnp* (<http://pupnp.sourceforge.net/>).

Definition in file [ixml_ext.h](#).

5.3.2 Function Documentation

5.3.2.1 IXML_Node* [ixmlAttr_getNode](#) (IXML_Attr * attr)

Returns node which represents provided attribute.

Parameters:

attr Attribute whose node representation is needed.

Returns:

Node corresponding to the provided attribute.

5.3.2.2 IXML_Element* [ixmlAttr_getOwnerElement](#) (IXML_Attr * attr)

Returns element which the provided attribute belongs to.

Parameters:

attr Attribute whose owner element should be returned.

Returns:

Owner element.

5.3.2.3 IXML_Element* ixmlDocument_getElementByAttrValue (IXML_Document * *doc*, const char * *attrName*, const char * *attrValue*)

Returns first element in the documents with provided attribute value.

Parameters:

doc Document where to search.
attrName Name of the attribute to check.
attrValue Attribute value which should be found

Returns:

A pointer to the element with matching attribute; *NULL* if no such element found.

5.3.2.4 IXML_Node* ixmlDocument_getNode (IXML_Document * *doc*)

Returns node which represents provided document.

Parameters:

doc Document whose node representation is needed.

Returns:

Node corresponding to the provided document.

5.3.2.5 IXML_Element* ixmlDocument_getRootElement (IXML_Document * *doc*)

Returns root element (root tag) of the XML document.

Parameters:

doc Document, whose root element should be retrieved.

Returns:

Root element or *NULL* if the document is empty or other error occurred.

5.3.2.6 IXML_Element* ixmlElement_cloneWithLog (IXML_Element * *source*)

Duplicates provided element.

Creates new instance of *IXML_Document* and copies entire element including all its children to that document. Also writes message to log (using [log_utils.h](#)) on error.

Note:

Don't forget to free owner document of the clone after usage.

See also:

ixmlNode_getOwnerDocument() at *ixml.h*

Parameters:

source Element to be copied.

Returns:

NULL on error, otherwise a pointer to the new copy of the source element.

5.3.2.7 `int ixmlElement_copyAttributeWithLog (IXML_Element * source, IXML_Element * target, const char * attrName, BOOL obligatory)`

Copies attribute value from one element to another.

If the attribute with provided name doesn't exist in the target node, it is created. Method also writes error messages using [log_utils.h](#) facilities.

Parameters:

source Element where the attribute will be copied from.

target Element which the attribute will be copied to.

attrName Name of the attribute to be copied.

obligatory Tells whether the attribute should necessarily present in the source tag. If *TRUE* and the attribute is missing than the error message will be logged. If it is *False* than only an error code will be returned in the same situation.

Returns:

IXML_SUCCESS if everything went well, or one of *ixml* error codes.

5.3.2.8 `void ixmlElement_freeOwnerDocument (IXML_Element * element)`

Frees the *IXML_Document* which the provided element belongs to.

Note:

As long as the whole document is freed, all other nodes which belongs to the same document are also freed.

Parameters:

element Element which should be freed together with it's owner document.

5.3.2.9 `IXML_Node* ixmlElement_getNode (IXML_Element * element)`

Returns node which represents provided element.

Parameters:

element Element whose node representation is needed.

Returns:

Node corresponding to the provided element.

5.3.2.10 IXML_Element* ixmlElement_parseBuffer (const char * *data*)

Parses an XML text buffer and returns the parent element of the generated DOM structure.

Note:

Don't forget to free memory allocated for the parsed document, not only the element (e.g. using [ixmlElement_freeOwnerDocument\(\)](#)).

Parameters:

data Text buffer to be parsed.

Returns:

Element representing parent tag of the parsed XML.

5.3.2.11 int ixmlElement_removeAttributeWithLog (IXML_Element * *element*, const char * *attrName*)

Removes attribute from the provided element.

Unlike [ixmlElement_removeAttribute\(\)](#) the attribute node is removed totally, not only value. Also writes warning message to log (using [log_utils.h](#)) on error.

Parameters:

element Element from which the attribute should be removed.

attrName Name of the attribute to be removed.

Returns:

0 on success or 1 on error.

5.3.2.12 int ixmlElement_setAttributeWithLog (IXML_Element * *element*, const char * *attrName*, const char * *attrValue*)

Adds new attribute to the element.

If attribute with the same name already exists, it's value will be updated. Writes warning message to log (using [log_utils.h](#)) on error.

Parameters:

element Element to which the attribute should be added.

attrName Name of the attribute to be added.

attrValue Value of the attribute.

Returns:

0 on success or 1 on error.

5.3.2.13 IXML_Attr* ixmlNode_convertToAttr (IXML_Node * node)

Converts node to the attribute.

Parameters:

node Node which should be converted.

Returns:

NULL if node is not an attribute.

5.3.2.14 IXML_Element* ixmlNode_convertToElement (IXML_Node * node)

Converts node to the element.

Parameters:

node Node which should be converted.

Returns:

NULL if node is not an element (i.e. tag).

5.3.2.15 void ixmlNode_freeOwnerDocument (IXML_Node * node)

Frees the *IXML_Document* which the provided node belongs to.

Note:

As long as the whole document is freed, all other nodes which belongs to the same document are also freed.

Parameters:

node Node which should be freed together with it's owner document.

5.3.2.16 IXML_Node* ixmlNode_parseBuffer (const char * data)

Parses an XML text buffer and returns the parent node of the generated DOM structure.

Note:

Don't forget to free memory allocated for the parsed document, not only the node (e.g. using [ixmlElement_freeOwnerDocument\(\)](#)).

Parameters:

data Text buffer to be parsed.

Returns:

Node representing parent tag of the parsed XML.

5.4 log_utils.h File Reference

Definitions of logging tools.

Defines

Logging utilities

- `#define log_debug(fmt,...)`
Prints debug message to the configured output.
- `#define log_error(fmt,...)`
Prints error message to the configured output.
- `#define log_warning(fmt,...)`
Prints warning message to the configured output.

Typedefs

- `typedef void(* log_function)(char *fmt,...)`
This is a prototype of log handler function.

Enumerations

- `enum LOG_LEVEL { LOG_LEVEL_DEBUG, LOG_LEVEL_WARNING, LOG_LEVEL_ERROR, LOG_LEVEL_NO }`
Defines possible log levels.

Functions

- `void log_setLevel (LOG_LEVEL level)`
Sets the minimum priority level of the messages which will be processed.
- `void log_usePrintf ()`
Switches library to use printf for handling messages.
- `void log_useSyslog (int facility)`
Switches library to use syslog for handling messages.

Variables

Log handlers

Contain links to the current log handlers.

Normally these links should not be used directly.

- `log_function log_debugHandler`
Contains link to the current handler of debug log messages.
- `log_function log_errorHandler`
Contains link to the current handler of error log messages.

- [log_function log_warningHandler](#)

Contains link to the current handler of warning log messages.

5.4.1 Detailed Description

Definitions of logging tools.

Log system has two modes:

- *syslog* - when all log messages are forwarded to syslog;
- *printf* - all messages are printed using *printf* utility, thus in most cases are shown on a console.

The default mode is *printf*, but it can be switched at any time.

Library provides three simple methods for logging messages with different priority levels:

- [log_debug\(\)](#)
- [log_warning\(\)](#)
- [log_error\(\)](#)

Author:

Andrey Litvinov

Version:

2.0

Definition in file [log_utils.h](#).

5.4.2 Define Documentation

5.4.2.1 #define log_debug(fmt, ...)

Value:

```
(*log_debugHandler) ("%s(%d): " fmt, __FILE__, \
__LINE__, ## __VA_ARGS__)
```

Prints *debug* message to the configured output.

Automatically adds filename and string number of the place from where the log was written.

Parameters:

fmt Message format (used in the same way as with *printf()*).

Definition at line 85 of file log_utils.h.

5.4.2.2 #define log_error(fmt, ...)

Value:

```
(*log_errorHandler) ("%s(%d): " fmt, __FILE__, \
__LINE__, ## __VA_ARGS__)
```

Prints *error* message to the configured output.

Automatically adds filename and string number of the place from where the log was written.

Parameters:

fmt Message format (used in the same way as with *printf()*).

Definition at line 105 of file log_utils.h.

5.4.2.3 #define log_warning(fmt, ...)

Value:

```
(*log_warningHandler) ("%s(%d): " fmt, __FILE__, \
__LINE__, ## __VA_ARGS__)
```

Prints *warning* message to the configured output.

Automatically adds filename and string number of the place from where the log was written.

Parameters:

fmt Message format (used in the same way as with *printf()*).

Definition at line 95 of file log_utils.h.

5.4.3 Typedef Documentation

5.4.3.1 typedef void(* log_function)(char *fmt,...)

This is a prototype of log handler function.

Parameters:

fmt Message format (used in the same way as with *printf()*).

Definition at line 49 of file log_utils.h.

5.4.4 Enumeration Type Documentation

5.4.4.1 enum LOG_LEVEL

Defines possible log levels.

Enumerator:

LOG_LEVEL_DEBUG Debug log level.

LOG_LEVEL_WARNING Warning log level.

LOG_LEVEL_ERROR Error log level.

LOG_LEVEL_NO 'No' log level.

Definition at line 112 of file log_utils.h.

5.4.5 Function Documentation

5.4.5.1 void log_setLevel (LOG_LEVEL *level*)

Sets the minimum priority level of the messages which will be processed.

Parameters:

level Priority level:

- [LOG_LEVEL_DEBUG](#) - All messages will be printed;
- [LOG_LEVEL_WARNING](#) - Only warning and error messages will be printed;
- [LOG_LEVEL_ERROR](#) - Only error messages are printed;
- [LOG_LEVEL_NO](#) - Nothing is printed at all.

5.4.5.2 void log_useSyslog (int *facility*)

Switches library to use *syslog* for handling messages.

Parameters:

facility Facility tells syslog who issued the message. See documentation of *syslog* for more information.

5.4.6 Variable Documentation

5.4.6.1 log_function log_debugHandler

Contains link to the current handler of *debug* log messages.

Normally [log_debug\(\)](#) should be used instead.

5.4.6.2 log_function log_errorHandler

Contains link to the current handler of *error* log messages.

Normally [log_error\(\)](#) should be used instead.

5.4.6.3 log_function log_warningHandler

Contains link to the current handler of *warning* log messages.

Normally [log_warning\(\)](#) should be used instead.

5.5 mainpage.h File Reference

Main page of the documentation generated by Doxygen.

5.5.1 Detailed Description

Main page of the documentation generated by Doxygen.

Contains source for [oBIX Client Library Documentation](#)

Definition in file [mainpage.h](#).

5.6 obix_client.h File Reference

oBIX Client API.

```
#include <ixml_ext.h>
```

Data Structures

- struct [_oBIX_BatchResult](#)
Contains outputs of the command, which was executed in a [Batch](#).

Typedefs

- typedef struct [_oBIX_Batch](#) [oBIX_Batch](#)
Represents a Batch object.
- typedef struct [_oBIX_BatchResult](#) [oBIX_BatchResult](#)
Contains outputs of the command, which was executed in a [Batch](#).
- typedef int(* [obix_update_listener](#))(int connectionId, int deviceId, int listenerId, const char *newValue)
Callback function, which is invoked when subscribed value is changed at the oBIX server.

Enumerations

- enum [OBIX_DATA_TYPE](#) {
[OBIX_T_BOOL](#), [OBIX_T_INT](#), [OBIX_T_REAL](#), [OBIX_T_STR](#),
[OBIX_T_ENUM](#), [OBIX_T_ABSTIME](#), [OBIX_T_RELTIME](#), [OBIX_T_URI](#) }
Standard oBIX data types.
- enum [OBIX_ERRORCODE](#) {
[OBIX_SUCCESS](#) = 0, [OBIX_ERR_INVALID_ARGUMENT](#) = -1, [OBIX_ERR_NO_MEMORY](#) = -2, [OBIX_ERR_INVALID_STATE](#) = -3,
[OBIX_ERR_LIMIT_REACHED](#) = -4, [OBIX_ERR_BAD_CONNECTION](#) = -5, [OBIX_ERR_UNKNOWN_BUG](#) = -100, [OBIX_ERR_HTTP_LIB](#) = -6,
[OBIX_ERR_SERVER_ERROR](#) = -7 }
Error codes which are returned by library functions.

Functions

- int [obix_closeConnection](#) (int connectionId)
Closes specified connection releasing all used resources.
- int [obix_dispose](#) ()
Releases all resources allocated by library.
- int [obix_loadConfig](#) (IXML_Element *config)
Initializes library and loads connection setting from provided DOM structure.
- int [obix_loadConfigFile](#) (const char *fileName)
Initializes library and loads connection setting from XML file.
- int [obix_openConnection](#) (int connectionId)
Opens connection to the oBIX server.
- int [obix_read](#) (int connectionId, int deviceId, const char *paramUri, IXML_Element **output)
Reads the whole oBIX object from the server and returns it as a DOM structure.
- int [obix_readValue](#) (int connectionId, int deviceId, const char *paramUri, char **output)
Reads value of the specified parameter from the oBIX server.
- int [obix_registerDevice](#) (int connectionId, const char *obixData)
Posts the provided device information to the oBIX server.
- int [obix_registerListener](#) (int connectionId, int deviceId, const char *paramUri, [obix_update_listener](#) listener)
Registers listener for device parameter updates.
- int [obix_unregisterDevice](#) (int connectionId, int deviceId)
Removes device record from the oBIX server.
- int [obix_unregisterListener](#) (int connectionId, int deviceId, int listenerId)
Unregisters listener of device parameter updates.
- int [obix_writeValue](#) (int connectionId, int deviceId, const char *paramUri, const char *newValue, [OBIX_DATA_TYPE](#) dataType)
Overwrites value of the specified device parameter at the oBIX server.

oBIX Batch operations

- [oBIX_Batch](#) * [obix_batch_create](#) (int connectionId)
Creates a new Batch instance.
- void [obix_batch_free](#) ([oBIX_Batch](#) *batch)
Releases memory allocated for the provided Batch object.
- const [oBIX_BatchResult](#) * [obix_batch_getResult](#) ([oBIX_Batch](#) *batch, int commandId)
Returns execution results of the command from specified Batch object.

- int [obix_batch_read](#) ([oBIX_Batch](#) *batch, int deviceId, const char *paramUri)
Adds read operation to the provided Batch.
- int [obix_batch_readValue](#) ([oBIX_Batch](#) *batch, int deviceId, const char *paramUri)
Adds readValue operation to the provided Batch.
- int [obix_batch_removeCommand](#) ([oBIX_Batch](#) *batch, int commandId)
Removes command with specified ID from the Batch object.
- int [obix_batch_send](#) ([oBIX_Batch](#) *batch)
Sends the Batch object to the oBIX server.
- int [obix_batch_writeValue](#) ([oBIX_Batch](#) *batch, int deviceId, const char *paramUri, const char *newValue, [OBIX_DATA_TYPE](#) dataType)
Adds writeValue operation to the provided Batch.

5.6.1 Detailed Description

oBIX Client API.

oBIX Client API simplifies implementing oBIX client applications, such as device drivers. Library hides all network calls and allows accessing data at the oBIX server without dealing with oBIX request formats. There is also a possibility to subscribe for data updates on the server, which is performed by library using oBIX Watch engine.

5.6.2 Usage

The typical usage of the library in device adapter (see example at [example_timer.c](#)):

- Include [obix_client.h](#) header.
- Initialize library during program startup. It can be done either by calling [obix_loadConfigFile\(\)](#) which will load settings from configuration file, or by [obix_loadConfig\(\)](#) with own generated XML settings structure.
- Open configured connection to oBIX server by calling [obix_openConnection\(\)](#).
- Generate an oBIX object for each device (for instance, one adapter can handle several devices of the same type) and register them at the server by calling [obix_registerDevice\(\)](#).
- If oBIX object, generated for the device, contains controlling values which can be changed outside (e.g. enabling/disabling boolean switch), register listener for these values by calling [obix_registerListener\(\)](#). Library starts polling changes of subscribed values and calls corresponding [obix_update_listener\(\)](#) every time when receives a new value.
- When some update of state variable is received by the driver from device, post the new value to the oBIX server by calling [obix_writeValue\(\)](#).
- Call [obix_unregisterDevice\(\)](#) when driver detects that the device is not available any more (unplugged, connection broken, etc.).

- When device driver is going down call [obix_dispose\(\)](#) in order to close all connections and release resources reserved for communication with oBIX server(s).

Note:

In order to register a new device (using [obix_registerDevice](#)), server should support *signUp* feature which is not in the oBIX specification. Currently *signUp* operation is supported by C oBIX Server included into this distribution and oFMS (<http://www.stok.fi/eng/ofms/index.html>). All other functions should work with any proper oBIX server implementation. If not, please report the found error to the author of this distribution.

Author:

Andrey Litvinov

Version:

0.0.0

Definition in file [obix_client.h](#).

5.6.3 Typedef Documentation

5.6.3.1 typedef struct _oBIX_Batch oBIX_Batch

Represents a Batch object.

oBIX Batch allows combining several commands in one request to the server, thus reducing response time and network load.

General Usage:

- Create new Batch instance using [obix_batch_create\(\)](#);
- Add commands to batch using [obix_batch_read\(\)](#), [obix_batch_readValue\(\)](#) or [obix_batch_writeValue\(\)](#);
- Send Batch object to the server by calling [obix_batch_send\(\)](#);
- An instance of [oBIX_BatchResult](#) will be generated for each command in Batch, containing execution results. These results can be obtained using [obix_batch_getResult\(\)](#);
- Free Batch object with [obix_batch_free\(\)](#).

Definition at line 461 of file [obix_client.h](#).

5.6.3.2 typedef int(* obix_update_listener)(int connectionId, int deviceId, int listenerId, const char *newValue)

Callback function, which is invoked when subscribed value is changed at the oBIX server.

ID arguments of the listener can be used to define which parameter was updated in case when one function is registered to handle updates of several parameters.

See also:

[obix_registerListener\(\)](#)

Parameters:

connectionId ID of the connection from which the update is received.

deviceId ID of the device whose parameter was changed. If the parameter doesn't belong to any device which was registered by current client, than 0 will be passed.

listenerId ID of the listener which receives the event.

newValue New value of the parameter.

Returns:

The listener should return [OBIX_SUCCESS](#) if the event was handled properly. Any other returned value will be considered by library as an error.

Definition at line 155 of file obix_client.h.

5.6.4 Enumeration Type Documentation**5.6.4.1 enum OBIX_DATA_TYPE**

Standard oBIX data types.

Used in [obix_writeValue\(\)](#).

Enumerator:

OBIX_T_BOOL Boolean data type (bool).

Possible values: "true" or "false".

OBIX_T_INT Integer data type (int).

Possible values are defined by *xs:long*.

OBIX_T_REAL Real data type (real).

Possible values are defined by *xs:double*.

OBIX_T_STR String data type (str).

OBIX_T_ENUM Enumeration data type (enum).

Possible values are defined by associated *range* object.

OBIX_T_ABSTIME Time data type (abstime).

Represents an absolute point in time. Value format is defined by *xs:dateTime*.

OBIX_T_RELTIME Time data type (reltime).

Represents time interval. Value format is defined by *xs:duration*.

OBIX_T_URI URI data type (uri).

Almost like a string, but contains valid URI.

Definition at line 112 of file obix_client.h.

5.6.4.2 enum OBIX_ERRORCODE

Error codes which are returned by library functions.

Enumerator:

OBIX_SUCCESS Operation is completed successfully.

OBIX_ERR_INVALID_ARGUMENT Function received wrong input argument.

OBIX_ERR_NO_MEMORY Not enough memory to complete the operation.

OBIX_ERR_INVALID_STATE Library has invalid state.

OBIX_ERR_LIMIT_REACHED Allocated buffer for devices or listeners is full.

OBIX_ERR_BAD_CONNECTION Error in communication with server.

OBIX_ERR_UNKNOWN_BUG Reserved for uncaught errors.

If such error occurs, this is a bug.

OBIX_ERR_HTTP_LIB Error inside HTTP communication module.

OBIX_ERR_SERVER_ERROR oBIX server returned an error object.

Definition at line 86 of file obix_client.h.

5.6.5 Function Documentation

5.6.5.1 oBIX_Batch* obix_batch_create (int *connectionId*)

Creates a new Batch instance.

oBIX Batch contains several operations which can be executed by one request to the server. Commands are executed at the server in the order they were added to the Batch.

Parameters:

connectionId ID of the connection for which batch is created.

Returns:

New Batch instance or *NULL* on error.

Referenced by resetListener().

5.6.5.2 void obix_batch_free (oBIX_Batch * *batch*)

Releases memory allocated for the provided Batch object.

Batch object will become unusable after calling this method.

Parameters:

batch Batch object to be deleted.

5.6.5.3 const oBIX_BatchResult* obix_batch_getResult (oBIX_Batch * *batch*, int *commandId*)

Returns execution results of the command from specified Batch object.

Parameters:

batch Batch object in which the command was executed.

commandId ID of the command whose results should be returned.

Returns:

An instance of oBIX_BatchResult, containing returned error code of the executed command and other return values if any.

Note:

Do not free the returned values. They will be freed automatically by next execution of [obix_batch_send\(\)](#) or [obix_batch_free\(\)](#).

5.6.5.4 int obix_batch_read (oBIX_Batch * batch, int deviceId, const char * paramUri)

Adds read operation to the provided Batch.

When Batch is executed, this operation will act like [obix_read\(\)](#). Read object will be stored at the corresponding [oBIX_BatchResult::obj](#).

Parameters:

batch Batch object where read operation should be added to.

deviceId ID of the device whose parameter should be read.

paramUri Uri of the parameter which should be read.

Returns:

- >0 - ID of the added command. IDs are assigned according to the order of adding commands to the Batch. Thus ID of the first added command will be 1, ID of the second - 2, and so on.
- <0 - Error code indicating that adding command to the Batch failed. **Note** that this is not the return code of the execution of read command: It will be stored in the corresponding [oBIX_BatchResult::status](#) after the whole Batch is executed.

Note:

Results of the previous execution of the Batch will become unavailable after calling this method.

See also:

[obix_read\(\)](#)

5.6.5.5 int obix_batch_readValue (oBIX_Batch * batch, int deviceId, const char * paramUri)

Adds readValue operation to the provided Batch.

When Batch is executed, this operation will act like [obix_readValue\(\)](#). Read value will be stored at the corresponding [oBIX_BatchResult::value](#).

Parameters:

batch Batch object where readValue operation should be added to.

deviceId ID of the device whose parameter should be read.

paramUri Uri of the parameter which should be read.

Returns:

- >0 - ID of the added command. IDs are assigned according to the order of adding commands to the Batch. Thus ID of the first added command will be 1, ID of the second - 2, and so on.
- <0 - Error code indicating that adding command to the Batch failed. **Note** that this is not the return code of the execution of read command: It will be stored in the corresponding [oBIX_BatchResult::status](#) after the whole Batch is executed.

Note:

Results of the previous execution of the Batch will become unavailable after calling this method.

See also:

[obix_readValue\(\)](#)

5.6.5.6 int obix_batch_removeCommand (oBIX_Batch * *batch*, int *commandId*)

Removes command with specified ID from the Batch object.

Parameters:

batch Batch object, from which the command should be removed.

commandId ID of the command which should be removed.

Returns:

- [OBIX_SUCCESS](#) if the command is removed successfully;
- [OBIX_ERR_INVALID_ARGUMENT](#) if batch is *NULL*;
- [OBIX_ERR_INVALID_STATE](#) if no command with specified ID is found in the Batch.

Note:

Results of the previous execution of the Batch will become unavailable after calling this method.

5.6.5.7 int obix_batch_send (oBIX_Batch * *batch*)

Sends the Batch object to the oBIX server.

After successful execution, a set of [oBIX_BatchResult](#) objects will be generated containing execution results for each command in Batch. Same Batch object can be sent to the server several times. In that case previous results will be freed and new one will be generated.

Parameters:

batch Batch object to be sent.

Returns:

[OBIX_SUCCESS](#) if the Batch object is sent successfully and server returned no errors. Error code is returned in case if at least one command in batch caused error.

Referenced by [resetListener\(\)](#).

5.6.5.8 int obix_batch_writeValue (oBIX_Batch * *batch*, int *deviceId*, const char * *paramUri*, const char * *newValue*, OBIX_DATA_TYPE *dataType*)

Adds writeValue operation to the provided Batch.

When Batch is executed, this operation will act like [obix_writeValue\(\)](#).

Parameters:

batch Batch object where writeValue operation should be added to.

deviceId ID of the device whose parameter's value should be written.

paramUri Uri of the parameter which should be written.

newValue Text representation of the new value to be written. It should be a new value for the *val* attribute of the oBIX object on the server, not the whole object.

dataType Type of data which is written to the server.

Returns:

- >0 - ID of the added command. IDs are assigned according to the order of adding commands to the Batch. Thus ID of the first added command will be 1, ID of the second - 2, and so on.
- <0 - Error code indicating that adding command to the Batch failed. **Note** that this is not the return code of the execution of write command: It will be stored in the corresponding `oBIX_BatchResult::status` after the whole Batch is executed.

Note:

Only value of an object (*val* attribute) can be written using this method. It's not possible to overwrite a whole oBIX object on the server.

Results of the previous execution of the Batch will become unavailable after calling this method.

See also:

[obix_writeValue\(\)](#)

Referenced by `resetListener()`.

5.6.5.9 int obix_closeConnection (int *connectionId*)

Closes specified connection releasing all used resources.

Also unregisters all devices and listeners which were registered using this connection.

Parameters:

connectionId ID of the connection which should be closed.

Returns:

[OBIX_SUCCESS](#) on success, error code otherwise.

5.6.5.10 int obix_dispose ()

Releases all resources allocated by library.

All registered listeners and devices are unregistered, all open connections are closed.

Returns:

[OBIX_SUCCESS](#) if operation is completed successfully, error code otherwise.

Referenced by `main()`.

5.6.5.11 int obix_loadConfig (IXML_Element * *config*)

Initializes library and loads connection setting from provided DOM structure.

Unlike [obix_loadConfigFile\(\)](#) it doesn't configure log system of the library. It can be configured manually using [config_log\(\)](#) or [log_utils.h](#) functions. By default, all messages (including debug ones) are written to *stdout*. The format of the configuration file can be found at [example_timer_config.xml](#)

Parameters:

config DOM structure representing a configuration XML.

Returns:

[OBIX_SUCCESS](#) if the library is initialized successfully, error code otherwise.

5.6.5.12 int obix_loadConfigFile (const char * *fileName*)

Initializes library and loads connection setting from XML file.

Also sets up the logging system of the library. The format of the configuration file can be found at [example_timer_config.xml](#)

Parameters:

fileName Name of the configuration file.

Returns:

[OBIX_SUCCESS](#) if the library initialized successfully, error code otherwise.

Referenced by [main\(\)](#).

5.6.5.13 int obix_openConnection (int *connectionId*)

Opens connection to the oBIX server.

Parameters:

connectionId Connection ID which was specified in the loaded configuration file.

Returns:

[OBIX_SUCCESS](#) if connection is opened successfully, error code otherwise.

Referenced by [main\(\)](#).

5.6.5.14 int obix_read (int *connectionId*, int *deviceId*, const char * *paramUri*, IXML_Element ** *output*)

Reads the whole oBIX object from the server and returns it as a DOM structure.

This function can be used to read any object on the server. It can be some object which was registered either by the same client, or by any other device driver. It can be also any server's own object.

Example: Server contains the following information:

```
<obj name="device1" href="/obix/device1/">
  <obj name="configure" href="conf/" />
  <str name="name" href="name" val="My Device"/>
</obj>
</obj>
```

There are two options to read the "*configure*" object of "*device1*":

- If this device was registered by the same client earlier, than the client should use device ID assigned to this device and provide parameter URI relative to the root of device data (in current example it is "*conf*").
- In case if that device was registered by someone else, than 0 should be used instead device ID + full URI of the required object (in this example it is "*/obix/device1/conf*").

If the whole object "*device1*" should be read and it was previously published by the same client, then the client should provide device ID assigned to this device and *NULL* as paramUri.

Note:

Although there is no need for this function in the normal workflow (see [Usage](#)) it still can be used, for instance, during initialization phase for obtaining some data from the server. Usage of this function for periodical reading of some object is not efficient and should be avoided. Use [obix_registerListener](#) instead.

Parameters:

connectionId ID of the connection which should be used.

deviceId ID of the device whose data should be read or 0 if the object doesn't belong to devices registered by this client.

paramUri URI of the object. It should be either relative to the device record like it was provided during device registration, or relative to the server root if the object doesn't belong to devices registered by this client.

output If read command executed successfully the DOM representation of the read object is stored here.

Returns:

[OBIX_SUCCESS](#) on success, negative error code otherwise.

5.6.5.15 int obix_readValue (int connectionId, int deviceId, const char * paramUri, char ** output)

Reads value of the specified parameter from the oBIX server.

This function can be used to read value of any object on the server: It can be some parameter of device which was registered either by the same client, or by any other device driver. It can be also any value of the server settings.

Example: Server contains the following information:

```
<obj name="device1" href="/obix/device1/">
  <obj name="configure" href="conf/" />
  <str name="name" href="name" val="My Device"/>
</obj>
</obj>
```

There are two ways to read the value of *"name"* string of *"device1"*:

- If this device was registered by the same client earlier, than the client should use device ID assigned to this device and provide parameter URI relative to the root of device data (in current example it is *"conf/name"*).
- In case if that device was registered by someone else, than *0* should be used instead device ID + full URI of the required parameter (in this example it is *"/obix/device1/conf/name"*).

Note:

This method is used to read only *val* attribute of some object, but not the whole object itself. If you need to read the whole oBIX object then use [obix_read](#) instead.

Although there is no need for this function in the normal workflow (see [Usage](#)), it still can be used, for instance, during initialization phase for obtaining some data from the server. Usage of this function for periodical reading of some object is not efficient and should be avoided. Use [obix_registerListener](#) instead.

Parameters:

connectionId ID of the connection which should be used.

deviceId ID of the device whose parameter should be read or *0* if the parameter doesn't belong to devices registered by this client.

paramUri URI of the parameter. It should be either relative to the device record like it was provided during device registration, or relative to the server root if the parameter doesn't belong to devices registered by this client.

output If read command executed successfully the attribute's value is stored here.

Returns:

[OBIX_SUCCESS](#) on success, negative error code otherwise.

5.6.5.16 int obix_registerDevice (int connectionId, const char * obixData)

Posts the provided device information to the oBIX server.

Note:

Input data is not tested to conform with oBIX specification. It is strongly recommended to provide *displayName* attribute to every object. Also attributes *href* and *writable* are obligatory for all device parameters which are going to be changed by the device driver or external oBIX server users:

- *writable* attribute should be set to *true*.
- *href* attribute of the parent object should be a valid URI, relative to the server root (start with *"/"*).
- *href* attributes of all child objects should have a valid URIs, relative to the parent object.

Example:

```
<obj name="light" href="/kitchen/light/" displayName="Kitchen lights">
  <bool name="switch"
    href="switch/"
    displayName="Switch"
    is="obix:bool"
    val="true"
    writable="true"/>
</obj>
```


Note:

Parent object should specify *href* attribute but the oBIX server is free to modify it (for instance, add prefix of the device storage), thus the URI can't be used to refer to the device record. Use the assigned device ID instead.

Parameters:

connectionId ID of the connection which should be used.

obixData oBIX object representing a new device.

Returns:

- >0 ID of the created device record;
- <0 error code.

Referenced by `main()`.

5.6.5.17 int obix_registerListener (int *connectionId*, int *deviceId*, const char * *paramUri*, obix_update_listener *listener*)

Registers listener for device parameter updates.

Overwrites existing listener if it is called twice for the same parameter.

This method can be also used to subscribe for the updates of any other objects stored at the oBIX server. In that case 0 should be provided as *deviceId* and *paramUri* should be relative to the server root.

Parameters:

connectionId ID of the connection which should be used.

deviceId ID of the device whose parameter should be monitored or 0 if the parameter doesn't belong to devices registered by this client.

paramUri URI of the parameter which should be monitored. It should be either relative to the device record like it was provided during device registration, or relative to the server root if the parameter doesn't belong to devices registered by this client.

listener Pointer to the listener function which would be invoked every time when the subscribed parameter is changed.

Note:

listener method should be quick. Slow listener (especially if it waits for some resource) will block subsequent calls to all listeners.

Returns:

- >=0 ID of the created listener;
- <0 error code.

Referenced by `main()`.

5.6.5.18 int obix_unregisterDevice (int *connectionId*, int *deviceId*)

Removes device record from the oBIX server.

Also removes all listeners which were registered for this device.

Parameters:

connectionId ID of the connection which should be used.

deviceId ID of the device which should be removed.

Returns:

[OBIX_SUCCESS](#) if the device record is removed successfully, error code otherwise.

5.6.5.19 int obix_unregisterListener (int *connectionId*, int *deviceId*, int *listenerId*)

Unregisters listener of device parameter updates.

Parameters:

connectionId ID of the connection which should be used.

deviceId ID of the device whose parameter is now monitored or 0 if the parameter doesn't belong to devices registered by this client.

listenerId ID of listener to be removed.

Returns:

[OBIX_SUCCESS](#) if the listener is removed successfully, error code otherwise.

5.6.5.20 int obix_writeValue (int *connectionId*, int *deviceId*, const char * *paramUri*, const char * *newValue*, OBIX_DATA_TYPE *dataType*)

Overwrites value of the specified device parameter at the oBIX server.

This function can be also used to change a value of any *writable* object at the oBIX server.

Parameters:

connectionId ID of the connection which should be used.

deviceId ID of the device whose parameter should be changed or 0 if the parameter doesn't belong to devices registered by this client.

paramUri URI of the parameter. It should be either relative to the device record like it was provided during device registration, or relative to the server root if changing parameter doesn't belong to devices registered by this client.

newValue Text representation of the new value to be written. It should be a new value for the *val* attribute of the oBIX object on the server, not the whole object.

Note:

Only value of an object (*val* attribute) can be written using this method. It's not possible to overwrite a whole oBIX object on the server.

Parameters:

dataType Type of data which is written to the server.

Returns:

[OBIX_SUCCESS](#) on success, negative error code otherwise.

See also:

[obix_readValue\(\)](#) for the usage example.

Referenced by `timerTask()`.

5.7 obix_utils.h File Reference

Contains oBIX keywords (object names, contracts, facets, etc) and some utility functions.

```
#include <ixml_ext.h>
```

Enumerations

- enum [RELTIME_FORMAT](#) {
RELTIME_SEC, RELTIME_MIN, RELTIME_HOUR, RELTIME_DAY,
RELTIME_MONTH, RELTIME_YEAR }

Specifies a format of reltime value, generated by [obix_retime_fromLong](#).

Functions

- [BOOL obix_obj_implementsContract](#) (IXML_Element *obj, const char *contract)
Checks whether oBIX object implements specified contract.
- char * [obix_retime_fromLong](#) (long duration, [RELTIME_FORMAT](#) format)
Generates reltime value from the provided time in milliseconds.
- int [obix_retime_parseToLong](#) (const char *str, long *duration)
Parses string value of reltime object and returns corresponding time in milliseconds.

Variables

- const char * [OBIX_OBJ_NULL_TEMPLATE](#)
String which represents oBIX NULL object.
- const char * [XML_FALSE](#)
String representation of boolean false value.
- const char * [XML_TRUE](#)
String representation of boolean true value.

oBIX Object Attributes and Facets

- const char * [OBIX_ATTR_DISPLAY](#)
oBIX facet "display".
- const char * [OBIX_ATTR_DISPLAY_NAME](#)

oBIX facet "displayName".

- const char * [OBIX_ATTR_HREF](#)
Object attribute "href".
- const char * [OBIX_ATTR_IS](#)
Object attribute "is".
- const char * [OBIX_ATTR_NAME](#)
Object attribute "name".
- const char * [OBIX_ATTR_NULL](#)
Object attribute "null".
- const char * [OBIX_ATTR_VAL](#)
Object attribute "val".
- const char * [OBIX_ATTR_WRITABLE](#)
oBIX facet "writable".

oBIX Error Contracts' URIs

Can be used to define the error type returned by an oBIX server.

- const char * [OBIX_CONTRACT_ERR_BAD_URI](#)
URI of the BadUriErr error contract.
- const char * [OBIX_CONTRACT_ERR_PERMISSION](#)
URI of the PermissionErr error contract.
- const char * [OBIX_CONTRACT_ERR_UNSUPPORTED](#)
URI of the UnsupportedErr error contract.

oBIX Object Names

Object names which are used in oBIX contracts.

- const char * [OBIX_NAME_BATCH](#)
Name of batch operation in the Lobby object.
- const char * [OBIX_NAME_SIGN_UP](#)
Name of signUp operation in the Lobby object.
- const char * [OBIX_NAME_WATCH_ADD](#)
Name of the Watch.add operation.
- const char * [OBIX_NAME_WATCH_DELETE](#)
Name of the Watch.delete operation.
- const char * [OBIX_NAME_WATCH_LEASE](#)
Name of the Watch.lease parameter.
- const char * [OBIX_NAME_WATCH_POLL_WAIT_INTERVAL](#)
Name of the Watch.pollWaitInterval object.

- const char * [OBIX_NAME_WATCH_POLL_WAIT_INTERVAL_MAX](#)
Name of the Watch.pollWaitInterval.max parameter.
- const char * [OBIX_NAME_WATCH_POLL_WAIT_INTERVAL_MIN](#)
Name of the Watch.pollWaitInterval.min parameter.
- const char * [OBIX_NAME_WATCH_POLLCHANGES](#)
Name of the Watch.pollChanges operation.
- const char * [OBIX_NAME_WATCH_POLLREFRESH](#)
Name of the Watch.pollRefresh operation.
- const char * [OBIX_NAME_WATCH_REMOVE](#)
Name of the Watch.remove operation.
- const char * [OBIX_NAME_WATCH_SERVICE](#)
Name of the Watch Service in the Lobby object.
- const char * [OBIX_NAME_WATCH_SERVICE_MAKE](#)
Name of the watchService.make operation.

oBIX Object Types (XML Element Types)

- const char * [OBIX_OBJ](#)
oBIX Object (obj)
- const char * [OBIX_OBJ_ABSTIME](#)
oBIX Absolute Time (abstime)
- const char * [OBIX_OBJ_BOOL](#)
oBIX Boolean (bool)
- const char * [OBIX_OBJ_ENUM](#)
oBIX Enumeration (enum)
- const char * [OBIX_OBJ_ERR](#)
oBIX Error (err)
- const char * [OBIX_OBJ_FEED](#)
oBIX Feed (feed)
- const char * [OBIX_OBJ_INT](#)
oBIX Integer (int)
- const char * [OBIX_OBJ_LIST](#)
oBIX List (list)
- const char * [OBIX_OBJ_OP](#)
oBIX Operation (op)
- const char * [OBIX_OBJ_REAL](#)
oBIX Real (real)

- const char * [OBIX_OBJ_REF](#)
oBIX Reference (ref)
- const char * [OBIX_OBJ_RELTIME](#)
oBIX Relative Duration of Time (reltime)
- const char * [OBIX_OBJ_STR](#)
oBIX String (str)
- const char * [OBIX_OBJ_URI](#)
oBIX URI (uri)

5.7.1 Detailed Description

Contains oBIX keywords (object names, contracts, facets, etc) and some utility functions.

The list of keywords is not complete and should be expanded every time when something new is needed. In most cases a common oBIX client application doesn't need to use any of those, because [oBIX Client API](#) hides all oBIX syntax (except case with registering new device data).

Read more about oBIX from <http://obix.org/>

Definition in file [obix_utils.h](#).

5.7.2 Function Documentation

5.7.2.1 **BOOL obix_obj_implementsContract (IXML_Element * *obj*, const char * *contract*)**

Checks whether oBIX object implements specified contract.

Object implements a contract when contract's URI is listed in object's *is* attribute.

Parameters:

- obj* XML DOM structure representing an oBIX object.
- contract* URI of the contract which should be checked.

Returns:

[TRUE](#) if the object implements specified contract, [FALSE](#) otherwise.

5.7.2.2 **char* obix_reftime_fromLong (long *duration*, RELTIME_FORMAT *format*)**

Generates *reltime* value from the provided time in milliseconds.

Parameters:

- duration* Time in milliseconds which should be converted.
- format* Format of the generated *reltime* value. Specifies the maximum time component for the output value. For example, converting 2 minutes with *RELTIME_MIN* will result in "PT2M"; *RELTIME_SEC* - "PT120S".

Returns:

String, which represents provided time in *xs:duration* format, or *NULL* if memory allocation failed.

Referenced by `timerTask()`.

5.7.2.3 int obix_retime_parseToLong (const char * *str*, long * *duration*)

Parses string value of *retime* object and returns corresponding time in milliseconds.

Follows *xs:duration* format.

Note:

Durations which can overload *long* variable are not parsed.

Parameters:

str String value of a *retime* object which should be parsed.

duration If parsing is successful then the parsed value will be written there.

Returns:

- 0 - Operation completed successfully;
- -1 - Parsing error (provided string has bad format);
- -2 - Provided *retime* value is bigger than or equal to 24 days (The maximum possible value is "P23DT23H59M59.999S"). Also this error code is returned when the input value is not normalized: If some of time components (e.g. hours) presents, then all smaller components (minutes and seconds) should represent less time than previous component. For example "PT60M" and "PT60S" are allowed, but "PT1H60M" and "PT1H60S" are not.

5.7.3 Variable Documentation

5.7.3.1 const char* OBIX_ATTR_DISPLAY

oBIX facet "display".

5.7.3.2 const char* OBIX_ATTR_DISPLAY_NAME

oBIX facet "displayName".

5.7.3.3 const char* OBIX_ATTR_HREF

Object attribute "href".

5.7.3.4 const char* OBIX_ATTR_IS

Object attribute "is".

5.7.3.5 const char* OBIX_ATTR_NAME

Object attribute "name".

5.7.3.6 const char* OBIX_ATTR_NULL

Object attribute "null".

5.7.3.7 const char* OBIX_ATTR_VAL

Object attribute "val".

5.7.3.8 const char* OBIX_ATTR_WRITABLE

oBIX facet "*writable*".

5.7.3.9 const char* OBIX_CONTRACT_ERR_BAD_URI

URI of the *BadUriErr* error contract.

5.7.3.10 const char* OBIX_CONTRACT_ERR_PERMISSION

URI of the *PermissionErr* error contract.

5.7.3.11 const char* OBIX_CONTRACT_ERR_UNSUPPORTED

URI of the *UnsupportedErr* error contract.

5.7.3.12 const char* OBIX_NAME_BATCH

Name of *batch* operation in the Lobby object.

5.7.3.13 const char* OBIX_NAME_SIGN_UP

Name of *signUp* operation in the Lobby object.

5.7.3.14 const char* OBIX_NAME_WATCH_ADD

Name of the *Watch.add* operation.

5.7.3.15 const char* OBIX_NAME_WATCH_DELETE

Name of the *Watch.delete* operation.

5.7.3.16 const char* OBIX_NAME_WATCH_LEASE

Name of the *Watch.lease* parameter.

5.7.3.17 const char* OBIX_NAME_WATCH_POLL_WAIT_INTERVAL

Name of the *Watch.pollWaitInterval* object.

5.7.3.18 const char* OBIX_NAME_WATCH_POLL_WAIT_INTERVAL_MAX

Name of the *Watch.pollWaitInterval.max* parameter.

5.7.3.19 const char* OBIX_NAME_WATCH_POLL_WAIT_INTERVAL_MIN

Name of the *Watch.pollWaitInterval.min* parameter.

5.7.3.20 const char* OBIX_NAME_WATCH_POLLCHANGES

Name of the *Watch.pollChanges* operation.

5.7.3.21 `const char* OBIX_NAME_WATCH_POLLREFRESH`

Name of the *Watch.pollRefresh* operation.

5.7.3.22 `const char* OBIX_NAME_WATCH_REMOVE`

Name of the *Watch.remove* operation.

5.7.3.23 `const char* OBIX_NAME_WATCH_SERVICE`

Name of the Watch Service in the Lobby object.

5.7.3.24 `const char* OBIX_NAME_WATCH_SERVICE_MAKE`

Name of the *watchService.make* operation.

5.7.3.25 `const char* OBIX_OBJ_NULL_TEMPLATE`

String which represents oBIX *NULL* object.

5.7.3.26 `const char* XML_FALSE`

String representation of boolean *false* value.

Referenced by `resetListener()`.

5.7.3.27 `const char* XML_TRUE`

String representation of boolean *true* value.

5.8 `ptask.h` File Reference

Periodic Task - tool for asynchronous task execution.

```
#include "bool.h"
```

Defines

- `#define EXECUTE_INDEFINITE -1`
Specifies that the task should be executed indefinite number of times (until *`ptask_cancel()`* is called).

Typedefs

- `typedef void(* periodic_task)(void *arg)`
Prototype of the function which can be scheduled.
- `typedef struct _Task_Thread Task_Thread`
Represents a separate thread which can be used to schedule tasks.

Functions

- `int ptask_cancel (Task_Thread *thread, int taskId, BOOL wait)`
Removes task from the scheduled list.
- `int ptask_dispose (Task_Thread *thread, BOOL wait)`
Releases resources allocated for the provided Task_Thread instance.
- `Task_Thread * ptask_init ()`
Creates new instance of Task_Thread.
- `BOOL ptask_isScheduled (Task_Thread *thread, int taskId)`
Checks whether the task with provided id is scheduled for execution in the thread.
- `int ptask_reschedule (Task_Thread *thread, int taskId, long period, int executeTimes, BOOL add)`
Sets new execution period for the specified task.
- `int ptask_reset (Task_Thread *thread, int taskId)`
Resets time until the next execution of the specified task.
- `int ptask_schedule (Task_Thread *thread, periodic_task task, void *arg, long period, int executeTimes)`
Schedules new task for execution.

5.8.1 Detailed Description

Periodic Task - tool for asynchronous task execution.

Periodic Task utility can be used to schedule some function(s) to be invoked periodically in a separate thread. A function can be scheduled to be invoked either a defined number of times or indefinitely (until it is canceled).

5.8.2 Usage

The following piece of code will schedule a function `foo()` to be executed every second in a separate thread:

```
// prints greetings
int foo(void* args)
{
    printf(Greetings from %s!, (char*) args);
}

...

// initialize a new thread
Task_Thread* thread = ptask_init();
if (thread == NULL)
{
    // initialization failed
    return -1;
}

long period = 1000;    // interval between executions in milliseconds
int timesToExecute = EXECUTE_INDEFINITE;    // how many times to execute
```

```
char* arg = (char*) malloc(7); // It will be passed to the foo() as argument
strcpy(arg, "Andrey");

// schedule foo() method to be executed indefinitely once is a second
int taskId = ptask_schedule(thread, &foo, arg, period, timesToExecute);
```

Note:

foo() function should match [periodic_task](#) prototype.

The variable which is passed to the [ptask_schedule\(\)](#) function as argument for *foo()* shouldn't be locally defined. Otherwise it can appear that it doesn't exist when the *foo()* is executed.

The returned task id can be then used to change execution period, or cancel the task.

One task thread can be used to schedule several tasks, but scheduled functions must be quick enough in order not to block other tasks to be executed in time.

At the end of application all initialized task thread should be freed using [ptask_dispose\(\)](#).

Author:

Andrey Litvinov

Version:

1.1

Definition in file [ptask.h](#).

5.8.3 Typedef Documentation

5.8.3.1 `typedef void(* periodic_task)(void *arg)`

Prototype of the function which can be scheduled.

Parameters:

arg Argument which is passed to the function when it is invoked.

Definition at line 95 of file `ptask.h`.

5.8.4 Function Documentation

5.8.4.1 `int ptask_cancel (Task_Thread * thread, int taskId, BOOL wait)`

Removes task from the scheduled list.

Parameters:

thread Thread in which task is scheduled.

taskId ID of the task to be removed.

wait When task is being executed it can be canceled only after execution is completed. This parameter defines whether the function should wait until the task is really canceled, or it can just mark the task as canceled, which guarantees that the task will be removed as soon as the current execution is completed. In case when this function is called while the task is not executed *wait* argument makes no difference.

Returns:

- 0 on success;
- -1 if task with provided ID is not found.

Referenced by main().

5.8.4.2 int ptask_dispose (Task_Thread * thread, BOOL wait)

Releases resources allocated for the provided Task_Thread instance.

All scheduled tasks are canceled.

Parameters:

thread Pointer to the Task_Thread to be freed.

wait If TRUE than the method will block and wait until specified thread is really disposed. Otherwise, method will only schedule asynchronous disposing of the thread.

Returns:

0 on success, negative error code otherwise.

Referenced by main().

5.8.4.3 Task_Thread* ptask_init ()

Creates new instance of Task_Thread.

Returns:

Pointer to the new instance of Task_Thread, or NULL if some error occurred.

Referenced by main().

5.8.4.4 BOOL ptask_isScheduled (Task_Thread * thread, int taskId)

Checks whether the task with provided id is scheduled for execution in the thread.

Parameters:

thread Thread where the task should be searched for.

taskId Task id which is searched for.

Returns:

TRUE if the task with specified taskId is scheduled, FALSE otherwise.

5.8.4.5 int ptask_reschedule (Task_Thread * thread, int taskId, long period, int executeTimes, BOOL add)

Sets new execution period for the specified task.

Parameters:

thread Thread in which the task is scheduled.

taskId Id of the scheduled task.

period New time interval in milliseconds (or time which will be added to the current task period).

executeTimes Defines how many times (min. 1) the task should be executed. If `EXECUTE_INDEFINITE` is provided than the task is executed until `ptask_cancel()` is called with corresponding task ID.

add Defines whether time provided in *period* argument will be used as new execution period, or will be added to the current one.

Note:

When *add* is set to `TRUE`, *period* will be also added to the next execution time, but when *add* is `FALSE` the next execution will be (current time + *period*).

Returns:

0 on success, negative error code otherwise.

5.8.4.6 `int ptask_reset (Task_Thread * thread, int taskId)`

Resets time until the next execution of the specified task.

The next execution time will be current time + *period* provided when the task was scheduled. If the *period* needs to be changed than use `ptask_reschedule()` instead.

Parameters:

thread Thread where the task is scheduled.

taskId Id of the task whose execution time should be reset.

Returns:

0 on success, negative error code otherwise.

Referenced by `resetListener()`.

5.8.4.7 `int ptask_schedule (Task_Thread * thread, periodic_task task, void * arg, long period, int executeTimes)`

Schedules new task for execution.

Parameters:

thread Thread in which the task will be executed.

task Task which should be scheduled.

arg Argument which will be passed to the task function each time when it is invoked.

period Time interval in milliseconds, which defines how often the task will be executed.

executeTimes Defines how many times (min. 1) the task should be executed. If `EXECUTE_INDEFINITE` is provided than the task is executed until `ptask_cancel()` with corresponding task ID is called.

Returns:

- >0 - ID of the scheduled task.
- <0 - Error code.

Referenced by `main()`.

5.9 xml_config.h File Reference

Declares configuration API.

```
#include <ixml_ext.h>
```

Functions

- void [config_finishInit](#) (IXML_Element *conf, [BOOL](#) successful)
Releases resources allocated for settings parsing.
- IXML_Element * [config_getChildTag](#) (IXML_Element *conf, const char *tagName, [BOOL](#) obligatory)
Returns child tag of the provided element with the specified name.
- const char * [config_getChildTagValue](#) (IXML_Element *conf, const char *tagName, [BOOL](#) obligatory)
Returns value of the child tag with specified name.
- char * [config_getResFullPath](#) (const char *filename)
Returns the address of the resource file by adding resource folder path to the filename.
- int [config_getTagAttrBoolValue](#) (IXML_Element *tag, const char *attrName, [BOOL](#) obligatory)
Returns boolean value of the specified tag attribute.
- const char * [config_getTagAttributeValue](#) (IXML_Element *tag, const char *attrName, [BOOL](#) obligatory)
Returns value of the specified tag attribute.
- int [config_getTagAttrIntValue](#) (IXML_Element *tag, const char *attrName, [BOOL](#) obligatory, int defaultValue)
Returns integer value of the specified tag attribute.
- long [config_getTagAttrLongValue](#) (IXML_Element *tag, const char *attrName, [BOOL](#) obligatory, long defaultValue)
Returns long integer value of the specified tag attribute.
- IXML_Element * [config_loadFile](#) (const char *filename)
Opens the configuration file and performs it's initial parsing.
- int [config_log](#) (IXML_Element *configTag)
Configures log system using parameters passed in XML format.
- void [config_setResourceDir](#) (char *path)
Sets the address of the resource folder where configuration file is stored.

Variables

Common configuration file keywords

The following acronyms are used in constants' names:

- CT means Configuration Tag;
- CTA - Configuration Tag's Attribute;
- CTAV - Configuration Tag's Attribute's Value.
- const char * **CT_CONFIG**
Main configuration tag's name.
- const char * **CTA_VALUE**
Most commonly used tag attribute: 'val'.

Log configuration keywords

These variables contain names of all log configuration tags, their attributes and possible attributes' values.

- const char * **CT_LOG**
Parent log configuration tag, containing all settings.
- const char * **CT_LOG_LEVEL**
Log level tag.
- const char * **CT_LOG_USE_SYSLOG**
Defines whether syslog should be used for logging.
- const char * **CTA_LOG_FACILITY**
Attribute which defines syslog facility.
- const char * **CTAV_LOG_FACILITY_DAEMON**
Log facility 'daemon'.
- const char * **CTAV_LOG_FACILITY_LOCAL0**
Log facility 'local0'.
- const char * **CTAV_LOG_FACILITY_USER**
Log facility 'user'.
- const char * **CTAV_LOG_LEVEL_DEBUG**
Log level debug.
- const char * **CTAV_LOG_LEVEL_ERROR**
Log level error.
- const char * **CTAV_LOG_LEVEL_NO**
Log level no.
- const char * **CTAV_LOG_LEVEL_WARNING**
Log level warning.

5.9.1 Detailed Description

Declares configuration API.

Configuration API allows loading settings from XML file which contains `<config>` element. This header defines required functions which simplify parsing of XML settings.

The API is tightly integrated with the logging system ([log_utils.h](#)). All functions extensively use [log_utils.h](#) utilities for logging warning and error messages occurred during parsing. There is also a possibility to load log system configuration from an XML file.

Author:

Andrey Litvinov

Version:

1.1

Definition in file [xml_config.h](#).

5.9.2 Function Documentation

5.9.2.1 void config_finishInit (IXML_Element * *conf*, BOOL *successful*)

Releases resources allocated for settings parsing.

Should be called once after all settings are loaded (or failed to load). Also writes message to log, telling that initialization is completed. Depending on *successful* parameter the log message tells that initialization completed or failed.

Parameters:

conf Settings which should be freed. It can be the link returned by [config_loadFile\(\)](#) or a link to any child tag.

successful Tells whether application initialized successfully or not.

5.9.2.2 IXML_Element* config_getChildTag (IXML_Element * *conf*, const char * *tagName*, BOOL *obligatory*)

Returns child tag of the provided element with the specified name.

Parameters:

conf Configuration tag in which search should be performed.

tagName Name of the tag to be searched for.

obligatory If [TRUE](#) than an error message will be logged when the tag is not found.

Returns:

Found child tag, or *NULL* if nothing is found.

5.9.2.3 `const char* config_getChildTagValue (IXML_Element * conf, const char * tagName, BOOL obligatory)`

Returns value of the child tag with specified name.

Tag value is the value of it's `val` attribute.

Parameters:

conf Configuration tag in which search should be performed.

tagName Name of the tag to be searched for.

obligatory If `TRUE` than an error message will be logged when the tag is not found, or it doesn't have any value.

Returns:

Value of the found child tag or `NULL` if the tag is not found or the found one doesn't have any value.

5.9.2.4 `char* config_getResFullPath (const char * filename)`

Returns the address of the resource file by adding resource folder path to the filename.

Note:

Do not forget to release memory allocated for returned string.

Parameters:

filename Name of the file.

Returns:

Full path to the resource file, or `NULL` on error.

5.9.2.5 `int config_getTagAttrBoolValue (IXML_Element * tag, const char * attrName, BOOL obligatory)`

Returns boolean value of the specified tag attribute.

Parameters:

tag Configuration tag whose attribute should be parsed.

attrName Name of the attribute to read.

obligatory if `TRUE` than an error message will be written when the attribute is not found.

Returns:

- `0` - Attribute's value is `"false"`;
- `1` - Attribute's value is `"true"`;
- `-1` - Parsing error.

5.9.2.6 const char* config_getTagAttributeValue (IXML_Element * tag, const char * attrName, BOOL obligatory)

Returns value of the specified tag attribute.

Parameters:

tag Configuration tag whose attribute should be returned.

attrName Name of the attribute to read.

obligatory if **TRUE** than an error message will be written when the attribute is not found.

Returns:

Attribute's value or *NULL* if the attribute is not found.

5.9.2.7 int config_getTagAttrIntValue (IXML_Element * tag, const char * attrName, BOOL obligatory, int defaultValue)

Returns integer value of the specified tag attribute.

Can be used only for parsing positive values.

Parameters:

tag Configuration tag whose attribute should be parsed.

attrName Name of the attribute to read.

obligatory if **TRUE** than an error message will be written when the attribute is not found.

defaultValue if *obligatory* is **FALSE** than this value will be returned instead of the error code.

Returns:

- ≥ 0 - Parsed integer attribute value;
- -1 - Error code. Error description will be logged.

5.9.2.8 long config_getTagAttrLongValue (IXML_Element * tag, const char * attrName, BOOL obligatory, long defaultValue)

Returns long integer value of the specified tag attribute.

Can be used only for parsing positive values.

Parameters:

tag Configuration tag whose attribute should be parsed.

attrName Name of the attribute to read.

obligatory if **TRUE** than an error message will be written when the attribute is not found.

defaultValue if *obligatory* is **FALSE** than this value will be returned instead of the error code.

Returns:

- ≥ 0 - Parsed long integer attribute value;
- -1 - Error code. Error description will be logged.

5.9.2.9 IXML_Element* config_loadFile (const char *filename)

Opens the configuration file and performs it's initial parsing.

Note:

Returned XML DOM structure should not be freed manually. Instead, use [config_finishInit\(\)](#) after parsing all required settings.

Parameters:

filename Name of the configuration file.

Returns:

Reference to the <config> tag, which can be used to load application settings.

5.9.2.10 int config_log (IXML_Element * configTag)

Configures log system using parameters passed in XML format.

Passed XML should contain at least <log> tag with child <level>, which specifies log level. For example:

```
<log>
  <level val="debug" />
</log>
```

There are other optional configuration tags. Please refer to the <log> element in [example_timer_config.xml](#) for the full description of possible tags.

Parameters:

configTag XML configuration document, which contains log configuration tags. Document can contain other elements which are ignored.

Returns:

0 on success; -1 on error.

5.9.2.11 void config_setResourceDir (char * path)

Sets the address of the resource folder where configuration file is stored.

The general idea is to keep all resource files in one place which is defined only once in the application. After that all resources (including configuration file) can be reached using [config_getResFullPath\(\)](#).

Parameters:

path Path to the application's resource folder.

5.9.3 Variable Documentation

5.9.3.1 const char* CT_CONFIG

Main configuration tag's name.

5.9.3.2 const char* CT_LOG

Parent log configuration tag, containing all settings.

5.9.3.3 const char* CT_LOG_LEVEL

Log level tag.

Defines which messages are logged.

5.9.3.4 const char* CT_LOG_USE_SYSLOG

Defines whether *syslog* should be used for logging.

If tag with such name is not present, than all messages are printed to *stdout*.

5.9.3.5 const char* CTA_LOG_FACILITY

Attribute which defines syslog facility.

5.9.3.6 const char* CTAV_LOG_FACILITY_DAEMON

Log facility 'daemon'.

5.9.3.7 const char* CTAV_LOG_FACILITY_LOCAL0

Log facility 'local0'.

5.9.3.8 const char* CTAV_LOG_FACILITY_USER

Log facility 'user'.

5.9.3.9 const char* CTAV_LOG_LEVEL_DEBUG

Log level *debug*.

All messages are logged.

5.9.3.10 const char* CTAV_LOG_LEVEL_ERROR

Log level *error*.

Only *error* messages are logged.

5.9.3.11 const char* CTAV_LOG_LEVEL_NO

Log level *no*.

Nothing is logged at all.

5.9.3.12 const char* CTAV_LOG_LEVEL_WARNING

Log level *warning*.

Only *warning* and *error* messages are logged.

Index

- [_oBIX_BatchResult](#), [2](#)
 - [obj](#), [3](#)
 - [status](#), [3](#)
 - [value](#), [3](#)
- [_taskThread](#)
 - [example_timer.c](#), [7](#)
- [_time](#)
 - [example_timer.c](#), [7](#)
- [_timerTaskId](#)
 - [example_timer.c](#), [7](#)
- [BOOL](#)
 - [bool.h](#), [4](#)
- [bool.h](#), [3](#)
 - [BOOL](#), [4](#)
 - [FALSE](#), [4](#)
 - [TRUE](#), [4](#)
- [config_finishInit](#)
 - [xml_config.h](#), [45](#)
- [config_getChildTag](#)
 - [xml_config.h](#), [45](#)
- [config_getChildTagValue](#)
 - [xml_config.h](#), [45](#)
- [config_getResFullPath](#)
 - [xml_config.h](#), [46](#)
- [config_getTagAttrBoolValue](#)
 - [xml_config.h](#), [46](#)
- [config_getTagAttributeValue](#)
 - [xml_config.h](#), [46](#)
- [config_getTagAttrIntValue](#)
 - [xml_config.h](#), [47](#)
- [config_getTagAttrLongValue](#)
 - [xml_config.h](#), [47](#)
- [config_loadFile](#)
 - [xml_config.h](#), [47](#)
- [config_log](#)
 - [xml_config.h](#), [48](#)
- [config_setResourceDir](#)
 - [xml_config.h](#), [48](#)
- [CONNECTION_ID](#)
 - [example_timer.c](#), [6](#)
- [CT_CONFIG](#)
 - [xml_config.h](#), [48](#)
- [CT_LOG](#)
 - [xml_config.h](#), [48](#)
- [CT_LOG_LEVEL](#)
 - [xml_config.h](#), [49](#)
- [CT_LOG_USE_SYSLOG](#)
 - [xml_config.h](#), [49](#)
- [CTA_LOG_FACILITY](#)
 - [xml_config.h](#), [49](#)
- [CTAV_LOG_FACILITY_DAEMON](#)
 - [xml_config.h](#), [49](#)
- [CTAV_LOG_FACILITY_LOCAL0](#)
 - [xml_config.h](#), [49](#)
- [CTAV_LOG_FACILITY_USER](#)
 - [xml_config.h](#), [49](#)
- [CTAV_LOG_LEVEL_DEBUG](#)
 - [xml_config.h](#), [49](#)
- [CTAV_LOG_LEVEL_ERROR](#)
 - [xml_config.h](#), [49](#)
- [CTAV_LOG_LEVEL_NO](#)
 - [xml_config.h](#), [49](#)
- [CTAV_LOG_LEVEL_WARNING](#)
 - [xml_config.h](#), [49](#)
- [DEVICE_DATA](#)
 - [example_timer.c](#), [8](#)
- [example_timer.c](#), [4](#)
 - [_taskThread](#), [7](#)
 - [_time](#), [7](#)
 - [_timerTaskId](#), [7](#)
 - [CONNECTION_ID](#), [6](#)
 - [DEVICE_DATA](#), [8](#)
 - [getDeviceData](#), [6](#)
 - [main](#), [6](#)
 - [resetListener](#), [6](#)
 - [timerTask](#), [7](#)
- [FALSE](#)
 - [bool.h](#), [4](#)
- [getDeviceData](#)
 - [example_timer.c](#), [6](#)
- [ixml_ext.h](#), [8](#)
 - [ixmlAttr_getNode](#), [9](#)
 - [ixmlAttr_getOwnerElement](#), [10](#)
 - [ixmlDocument_getElementByAttrValue](#), [10](#)
 - [ixmlDocument_getNode](#), [10](#)
 - [ixmlDocument_getRootElement](#), [10](#)
 - [ixmlElement_cloneWithLog](#), [10](#)
 - [ixmlElement_copyAttributeWithLog](#), [11](#)
 - [ixmlElement_freeOwnerDocument](#), [11](#)
 - [ixmlElement_getNode](#), [11](#)
 - [ixmlElement_parseBuffer](#), [12](#)
 - [ixmlElement_removeAttributeWithLog](#), [12](#)
 - [ixmlElement_setAttributeWithLog](#), [12](#)
 - [ixmlNode_convertToAttr](#), [13](#)
 - [ixmlNode_convertToElement](#), [13](#)
 - [ixmlNode_freeOwnerDocument](#), [13](#)

- ixmlNode_parseBuffer, 13
- ixmlAttr_getNode
 - ixml_ext.h, 9
- ixmlAttr_getOwnerElement
 - ixml_ext.h, 10
- ixmlDocument_getElementByAttrValue
 - ixml_ext.h, 10
- ixmlDocument_getNode
 - ixml_ext.h, 10
- ixmlDocument_getRootElement
 - ixml_ext.h, 10
- ixmlElement_cloneWithLog
 - ixml_ext.h, 10
- ixmlElement_copyAttributeWithLog
 - ixml_ext.h, 11
- ixmlElement_freeOwnerDocument
 - ixml_ext.h, 11
- ixmlElement_getNode
 - ixml_ext.h, 11
- ixmlElement_parseBuffer
 - ixml_ext.h, 12
- ixmlElement_removeAttributeWithLog
 - ixml_ext.h, 12
- ixmlElement_setAttributeWithLog
 - ixml_ext.h, 12
- ixmlNode_convertToAttr
 - ixml_ext.h, 13
- ixmlNode_convertToElement
 - ixml_ext.h, 13
- ixmlNode_freeOwnerDocument
 - ixml_ext.h, 13
- ixmlNode_parseBuffer
 - ixml_ext.h, 13
- LOG_LEVEL_DEBUG
 - log_utils.h, 16
- LOG_LEVEL_ERROR
 - log_utils.h, 17
- LOG_LEVEL_NO
 - log_utils.h, 17
- LOG_LEVEL_WARNING
 - log_utils.h, 16
- log_utils.h
 - LOG_LEVEL_DEBUG, 16
 - LOG_LEVEL_ERROR, 17
 - LOG_LEVEL_NO, 17
 - LOG_LEVEL_WARNING, 16
- log_debug
 - log_utils.h, 15
- log_debugHandler
 - log_utils.h, 17
- log_error
 - log_utils.h, 15
- log_errorHandler
 - log_utils.h, 17
- log_function
 - log_utils.h, 16
- LOG_LEVEL
 - log_utils.h, 16
- log_setLevel
 - log_utils.h, 17
- log_useSyslog
 - log_utils.h, 17
- log_utils.h, 14
 - log_debug, 15
 - log_debugHandler, 17
 - log_error, 15
 - log_errorHandler, 17
 - log_function, 16
 - LOG_LEVEL, 16
 - log_setLevel, 17
 - log_useSyslog, 17
 - log_warning, 16
 - log_warningHandler, 17
- log_warning
 - log_utils.h, 16
- log_warningHandler
 - log_utils.h, 17
- main
 - example_timer.c, 6
- mainpage.h, 17
- obix_client.h
 - OBIX_ERR_BAD_CONNECTION, 23
 - OBIX_ERR_HTTP_LIB, 23
 - OBIX_ERR_INVALID_ARGUMENT, 22
 - OBIX_ERR_INVALID_STATE, 23
 - OBIX_ERR_LIMIT_REACHED, 23
 - OBIX_ERR_NO_MEMORY, 22
 - OBIX_ERR_SERVER_ERROR, 23
 - OBIX_ERR_UNKNOWN_BUG, 23
 - OBIX_SUCCESS, 22
 - OBIX_T_ABSTIME, 22
 - OBIX_T_BOOL, 22
 - OBIX_T_ENUM, 22
 - OBIX_T_INT, 22
 - OBIX_T_REAL, 22
 - OBIX_T_RELTIME, 22
 - OBIX_T_STR, 22
 - OBIX_T_URI, 22
- OBIX_ERR_BAD_CONNECTION
 - obix_client.h, 23
- OBIX_ERR_HTTP_LIB
 - obix_client.h, 23
- OBIX_ERR_INVALID_ARGUMENT
 - obix_client.h, 22
- OBIX_ERR_INVALID_STATE

- obix_client.h, 23
- OBIX_ERR_LIMIT_REACHED
 - obix_client.h, 23
- OBIX_ERR_NO_MEMORY
 - obix_client.h, 22
- OBIX_ERR_SERVER_ERROR
 - obix_client.h, 23
- OBIX_ERR_UNKNOWN_BUG
 - obix_client.h, 23
- OBIX_SUCCESS
 - obix_client.h, 22
- OBIX_T_ABSTIME
 - obix_client.h, 22
- OBIX_T_BOOL
 - obix_client.h, 22
- OBIX_T_ENUM
 - obix_client.h, 22
- OBIX_T_INT
 - obix_client.h, 22
- OBIX_T_REAL
 - obix_client.h, 22
- OBIX_T_RELTIME
 - obix_client.h, 22
- OBIX_T_STR
 - obix_client.h, 22
- OBIX_T_URI
 - obix_client.h, 22
- OBIX_ATTR_DISPLAY
 - obix_utils.h, 36
- OBIX_ATTR_DISPLAY_NAME
 - obix_utils.h, 36
- OBIX_ATTR_HREF
 - obix_utils.h, 36
- OBIX_ATTR_IS
 - obix_utils.h, 36
- OBIX_ATTR_NAME
 - obix_utils.h, 36
- OBIX_ATTR_NULL
 - obix_utils.h, 36
- OBIX_ATTR_VAL
 - obix_utils.h, 36
- OBIX_ATTR_WRITABLE
 - obix_utils.h, 36
- oBIX_Batch
 - obix_client.h, 21
- obix_batch_create
 - obix_client.h, 23
- obix_batch_free
 - obix_client.h, 23
- obix_batch_getResult
 - obix_client.h, 23
- obix_batch_read
 - obix_client.h, 24
- obix_batch_readValue
 - obix_client.h, 24
- obix_batch_removeCommand
 - obix_client.h, 25
- obix_batch_send
 - obix_client.h, 25
- obix_batch_writeValue
 - obix_client.h, 25
- obix_client.h, 18
 - oBIX_Batch, 21
 - obix_batch_create, 23
 - obix_batch_free, 23
 - obix_batch_getResult, 23
 - obix_batch_read, 24
 - obix_batch_readValue, 24
 - obix_batch_removeCommand, 25
 - obix_batch_send, 25
 - obix_batch_writeValue, 25
 - obix_closeConnection, 26
 - OBIX_DATA_TYPE, 22
 - obix_dispose, 26
 - OBIX_ERRORCODE, 22
 - obix_loadConfig, 26
 - obix_loadConfigFile, 27
 - obix_openConnection, 27
 - obix_read, 27
 - obix_readValue, 28
 - obix_registerDevice, 29
 - obix_registerListener, 30
 - obix_unregisterDevice, 30
 - obix_unregisterListener, 31
 - obix_update_listener, 21
 - obix_writeValue, 31
- obix_closeConnection
 - obix_client.h, 26
- OBIX_CONTRACT_ERR_BAD_URI
 - obix_utils.h, 37
- OBIX_CONTRACT_ERR_PERMISSION
 - obix_utils.h, 37
- OBIX_CONTRACT_ERR_UNSUPPORTED
 - obix_utils.h, 37
- OBIX_DATA_TYPE
 - obix_client.h, 22
- obix_dispose
 - obix_client.h, 26
- OBIX_ERRORCODE
 - obix_client.h, 22
- obix_loadConfig
 - obix_client.h, 26
- obix_loadConfigFile
 - obix_client.h, 27
- OBIX_NAME_BATCH
 - obix_utils.h, 37
- OBIX_NAME_SIGN_UP
 - obix_utils.h, 37

- OBIX_NAME_WATCH_ADD
 - obix_utils.h, 37
- OBIX_NAME_WATCH_DELETE
 - obix_utils.h, 37
- OBIX_NAME_WATCH_LEASE
 - obix_utils.h, 37
- OBIX_NAME_WATCH_POLL_WAIT_-INTERVAL
 - obix_utils.h, 37
- OBIX_NAME_WATCH_POLL_WAIT_-INTERVAL_MAX
 - obix_utils.h, 37
- OBIX_NAME_WATCH_POLL_WAIT_-INTERVAL_MIN
 - obix_utils.h, 37
- OBIX_NAME_WATCH_POLLCHANGES
 - obix_utils.h, 37
- OBIX_NAME_WATCH_POLLREFRESH
 - obix_utils.h, 37
- OBIX_NAME_WATCH_REMOVE
 - obix_utils.h, 38
- OBIX_NAME_WATCH_SERVICE
 - obix_utils.h, 38
- OBIX_NAME_WATCH_SERVICE_MAKE
 - obix_utils.h, 38
- obix_obj_implementsContract
 - obix_utils.h, 35
- OBIX_OBJ_NULL_TEMPLATE
 - obix_utils.h, 38
- obix_openConnection
 - obix_client.h, 27
- obix_read
 - obix_client.h, 27
- obix_readValue
 - obix_client.h, 28
- obix_registerDevice
 - obix_client.h, 29
- obix_registerListener
 - obix_client.h, 30
- obix_reftime_fromLong
 - obix_utils.h, 35
- obix_reftime_parseToLong
 - obix_utils.h, 35
- obix_unregisterDevice
 - obix_client.h, 30
- obix_unregisterListener
 - obix_client.h, 31
- obix_update_listener
 - obix_client.h, 21
- obix_utils.h, 32
 - OBIX_ATTR_DISPLAY, 36
 - OBIX_ATTR_DISPLAY_NAME, 36
 - OBIX_ATTR_HREF, 36
 - OBIX_ATTR_IS, 36
 - OBIX_ATTR_NAME, 36
 - OBIX_ATTR_NULL, 36
 - OBIX_ATTR_VAL, 36
 - OBIX_ATTR_WRITABLE, 36
 - OBIX_CONTRACT_ERR_BAD_URI, 37
 - OBIX_CONTRACT_ERR_PERMISSION, 37
 - OBIX_CONTRACT_ERR_UNSUPPORTED, 37
 - OBIX_NAME_BATCH, 37
 - OBIX_NAME_SIGN_UP, 37
 - OBIX_NAME_WATCH_ADD, 37
 - OBIX_NAME_WATCH_DELETE, 37
 - OBIX_NAME_WATCH_LEASE, 37
 - OBIX_NAME_WATCH_POLL_WAIT_-INTERVAL, 37
 - OBIX_NAME_WATCH_POLL_WAIT_-INTERVAL_MAX, 37
 - OBIX_NAME_WATCH_POLL_WAIT_-INTERVAL_MIN, 37
 - OBIX_NAME_WATCH_POLLCHANGES, 37
 - OBIX_NAME_WATCH_POLLREFRESH, 37
 - OBIX_NAME_WATCH_REMOVE, 38
 - OBIX_NAME_WATCH_SERVICE, 38
 - OBIX_NAME_WATCH_SERVICE_MAKE, 38
 - obix_obj_implementsContract, 35
 - OBIX_OBJ_NULL_TEMPLATE, 38
 - obix_reftime_fromLong, 35
 - obix_reftime_parseToLong, 35
 - XML_FALSE, 38
 - XML_TRUE, 38
- obix_writeValue
 - obix_client.h, 31
- obj
 - _oBIX_BatchResult, 3
- periodic_task
 - ptask.h, 40
- ptask.h, 38
 - periodic_task, 40
 - ptask_cancel, 40
 - ptask_dispose, 41
 - ptask_init, 41
 - ptask_isScheduled, 41
 - ptask_reschedule, 41
 - ptask_reset, 42
 - ptask_schedule, 42
- ptask_cancel
 - ptask.h, 40
- ptask_dispose
 - ptask.h, 41
- ptask_init
 - ptask.h, 41

- ptask_isScheduled
 - ptask.h, [41](#)
- ptask_reschedule
 - ptask.h, [41](#)
- ptask_reset
 - ptask.h, [42](#)
- ptask_schedule
 - ptask.h, [42](#)
- resetListener
 - example_timer.c, [6](#)
- status
 - _oBIX_BatchResult, [3](#)
- timerTask
 - example_timer.c, [7](#)
- TRUE
 - bool.h, [4](#)
- value
 - _oBIX_BatchResult, [3](#)
- xml_config.h, [43](#)
 - config_finishInit, [45](#)
 - config_getChildTag, [45](#)
 - config_getChildTagValue, [45](#)
 - config_getResFullPath, [46](#)
 - config_getTagAttrBoolValue, [46](#)
 - config_getTagAttributeValue, [46](#)
 - config_getTagAttrIntValue, [47](#)
 - config_getTagAttrLongValue, [47](#)
 - config_loadFile, [47](#)
 - config_log, [48](#)
 - config_setResourceDir, [48](#)
 - CT_CONFIG, [48](#)
 - CT_LOG, [48](#)
 - CT_LOG_LEVEL, [49](#)
 - CT_LOG_USE_SYSLOG, [49](#)
 - CTA_LOG_FACILITY, [49](#)
 - CTAV_LOG_FACILITY_DAEMON, [49](#)
 - CTAV_LOG_FACILITY_LOCAL0, [49](#)
 - CTAV_LOG_FACILITY_USER, [49](#)
 - CTAV_LOG_LEVEL_DEBUG, [49](#)
 - CTAV_LOG_LEVEL_ERROR, [49](#)
 - CTAV_LOG_LEVEL_NO, [49](#)
 - CTAV_LOG_LEVEL_WARNING, [49](#)
- XML_FALSE
 - obix_utils.h, [38](#)
- XML_TRUE
 - obix_utils.h, [38](#)