Kylone MicroCMS Endpoint API User Guide

Version 2.0.81

Revision April 19, 2018

CONTENTS

1. Introduction

Kylone provides an Endpoint API for developers to help creating a custom application to use with Kylone MicroCMS server including accessing and utilising several types of content related data located on server and ready to use functions while working with the server.

Beside accessing and using such data easily while doing development of custom endpoint applications, the main purpose of the API is to maintain functionalities on both server and endpoint side and to be sure about the consistency of whole framework.

Kylone Endpoint API will support following platforms;

Starting from MicroCMS v2.0.81	Linux, aarch64-linux-gnu, Linaro GCC 4.9-2014.09 Linux, arm-linux-gnueabi, Linaro GCC 4.9-2014.08, armv7 Linux , x86_64-linux-gnu, Linaro GCC 4.9-2014.08 MacOS, x86_64-apple-darwin15.6.0, Apple LLVM version 8.0.0

This document includes guidelines for connecting and fetching data from server, utilising content related data and some guidelines about usage of functionalities on server side to maintain same functionalities on endpoint side. Therefore, the API and this document will not cover topics like user specific tasks should be taken (such as checking network connectivity before trying to connect to target host, running on WiFi or Ethernet based networks only) or user or platform specific software development tasks (such as developing user interface or media players). Instead, provides ready to use methods for collecting components from server and organising them to use easily with custom application.

2. Using the Endpoint API

There are two main components of the Endpoint API;

- `src/cLshcapi.h`

  C header file contains type definitions and prototypes to access functions implemented in shared-object (dynamic library).

- `lib/cLshcapi.so`

  Shared object to link with custom application.

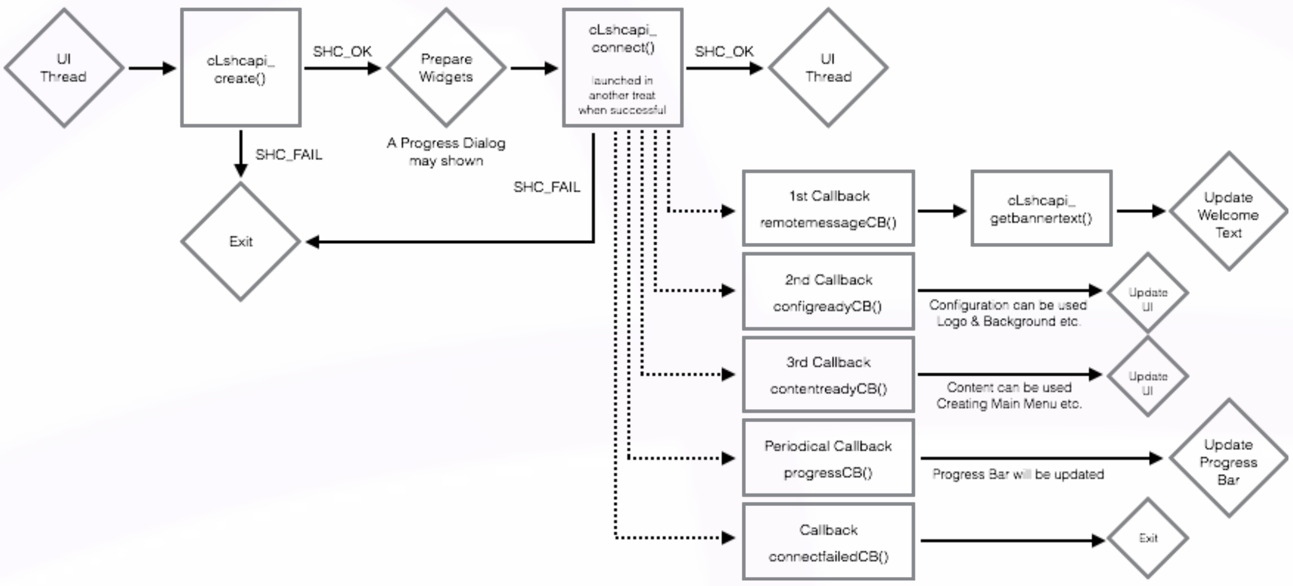
Custom applications will use the API by calling implemented functions in conjunction with predefined call-back functions. Basically, the API consists of following items;

- Definitions of some constants and types
- Functions exported by shared object.
- Call-back functions which should be implemented by the custom application
- Some helper methods implemented to use by custom application

2.1. Workflow

Functions exported from shared object should be used in a workflow shown below. All other helper functions can be used instantly whenever they are required.

In order to start using API, the essential method “`cLshcapi_create()`” should be called first. This function will return `SHC_OK` once successful. The second essential method is the “`cLshcapi_connect`()”. This method starts running the API and triggers required callback functions accordingly.


UI Thread
cLshcapi_ create()
SHC_OK
SHC_FAIL
Exit
Prepare Widgets
A Progress Dialog may shown
cLshcapi_ connect()
launched in another treat when successful
SHC_FAIL
SHC_OK
UI Thread
1st Callback remotemessageCB()
cLshcapi_ getbannertext()
Update Welcome Text
2nd Callback configreadyCB()
Configuration can be used Logo & Background etc.
Update UI
3rd Callback contentreadyCB()
Content can be used Creating Main Menu etc.
Update UI
Periodical Callback progressCB()
Progress Bar will be updated
Update Progress Bar
Callback connectfailedCB()
Exit

2.2. API Calls

`const char *cLshcapi_getapiversion_alloc()`

This function returns the pointer of an allocated memory which contains API version and build number. Custom application should release this allocated resource after use. Typical use of this functions is shown below;

```
const char *version = cLshcapi_getapiversion_alloc();
printf("> Using API version %s\n", version);
free((void *)version);
```

`int cLshcapi_create(int argc, char **argv, const char *cachedir)`

The function which prepares environment and starts threads inside the API. It will called with standard command-line arguments along with single option which for setting the cache directory inside the API. The most common usage of this method is shown below;

```
int main(int argc, char** argv)
{
   ...
   if (cLshcapi_create(argc, argv, "/tmp") != SHC_OK) {
      cLshcapi_delete();
      return 1;
   }
   ...
   ...
   return 0;
}
```

When `SHC_OPT_DOCACHE` option provided while calling `cLshcapi_connect()` function, there will be a new folder called `".shcache"` created under given folder. This `“.shcache"` folder should be exists while the application is running. It can be freely deleted if caching will not be used or while termination of the application.

`int cLshcapi_delete()`

The function which needs to be called before terminating the custom application. Since there are threads running inside the API, this function will stop them all and cleans the environment when called, so custom application can be terminated safely.

`int cLshcapi_setcallback(void *uptr, int cbid, ...)`

The custom application should set call-back functions to get benefit of some extra features of the API before calling `cLshcapi_connect()` explained below. There are three arguments should be provided while calling the function;

`void *uptr`
User defined pointer which will be passed to calling function as it is.

`int cbid`
The call-back identifier which is an integer. Possible values are defined in the header file like below;

```
#define CLSHC_INVOKE_CONFIGREADY    0
#define CLSHC_INVOKE_CONTENTREADY   1
#define CLSHC_INVOKE_CONNECTFAILED  2
#define CLSHC_INVOKE_PROGRESS       3
#define CLSHC_INVOKE_REMOTEMESSAGE  4
```

`... (variadic argument)`
Statically defined call-back function. Function prototypes for possible call-back functions are defined in header file like below;

```
typedef void (*fcLshcapi_configreadyCB)(const char *, void *);
typedef void (*fcLshcapi_contentreadyCB)(const char *, void *);
typedef void (*fcLshcapi_connectfailedCB)(int, void *);
typedef void (*fcLshcapi_progressCB)(int, void *);
typedef void (*fcLshcapi_remotemessageCB)(const char *, void *);
```

Custom application should provide call-back functions according to above prototypes. Each call-back function will be explained in next section.

Typical usage of the function is shown below;

```
static void app_progressCB(int progress, void *uptr)
{
   printf("~ callback: progress: %d%%\n", progress);
}

int app_set_callback_functions(void *uptr)
{
   ...
   if (cLshcapi_setcallback(uptr, CLSHC_INVOKE_PROGRESS, app_progressCB) != SHC_OK) {
      printf("! unable to set callback: app_progressCB\n");
      return 1;
   }
   ...
   return 0;
}
```

`int cLshcapi_connect(const char *targethost, int cache)`

Custom application will start communicating with the server when calling this function.

It will be good idea to check network connectivity before calling it. It is also recommended to use `cLshcapi_serverisready()` function explained below.

There are two options needs to be provided while calling the function: IP address or domain name of the server and the caching option. The most common usage of the function is shown below;

```
if (cLshcapi_connect(targethost, SHC_OPT_DOCACHE) != SHC_OK) {
   printf("! could not connect to %s\n", targethost);
   cLshcapi_delete();
   ...
}
```

Even if the function returns immediately with `SHC_OK` code, it will create a new thread and keeps running inside it till `fcLshcapi_contentreadyCB()` triggered. Required callback methods explained next section should be implemented to handle such tasks by the custom application.

`int cLshcapi_isconnecting()`

The thread-safe function which can be used to check whether the thread launched by `cLshcapi_connect()` is still running or not. It may useful for single-threaded custom applications. The function will return `SHC_OK` if the connection is still in progress. Typical usage is show below;

```
while (cLshcapi_isconnecting() == SHC_OK)
   cLshcapi_util_delaymillisecond(1000);
printf("> connect thread finished\n");
```

`int cLshcapi_serverisready(const char *targethost, int timeout)`

This function checks server status by performing tests in application layer and it can be used to see whether the server is functioning or not. Custom application may perform this test before trying to connect to server and it may give up after doing few more tests. For example, server may be doing boot at that time. On the other hand, this test will give more control to understand issues while doing troubleshooting etc.

Custom application may use this function before calling the `cLshcapi_connect()` function. It returns `SHC_OK` upon successful and means that the server is ready to connect, otherwise returns `SHC_FAIL`.

There are two options needs to be provided while calling the function: IP address or domain name of the server and the timeout value in seconds. The most common usage of the method is shown below;

```
int app_wait_server_if_not_ready(const char *targethost, int timeout, int maxtry)
{
   int trycount = 0;
   while (cLshcapi_serverisready(targethost, timeout) != SHC_OK) {
      if (++trycount >= maxtry) {
         printf("! server is not ready, giving up\n");
         return 1;
      }
      printf("! waiting server to be ready(%d)\n", trycount);
      cLshcapi_util_delaymillisecond(1000);
   }
   printf("  server is ready\n");
   return 0;
}
```

2.3. Callbacks

Following callbacks should be implemented by the custom application to handle required tasks properly.

```
void (*fcLshcapi_remotemessageCB)(const char *, void *)
```

The function which will be called automatically when the server sends a message. The first argument contains message which is explained in next chapter (Definitions and Types). The second argument is user defined pointer given while calling the `cLshcapi_setcallback()` function.

```
void (*fcLshcapi_progressCB)(int, void *)
```

The function to handle progress callback generated by the `cLshcapi_connect()` while communicating with the server. It's first argument is the progress value in percent (in range of 0-100). A ProgressDialog may closed when the value reaches to 100 or when the method `cbContentReady()` triggered. The second argument is the user defined pointer.

```
void (*fcLshcapi_configreadyCB)(const char *, void *)
```

The function to handle interim update while `cLshcapi_connect()` is running and the content is downloading from the server. This callback will be triggered when the "configuration" is downloaded and ready to use by the custom application. For example, application may set background, logo and welcome text immediately when this callback is triggered while the content downloading phase is still in progress.

The fist argument contains configuration data in XML format. Custom application may process it by it's own methods instead of using provided API functions. The second argument is the user defined pointer.

```
void (*fcLshcapi_contentreadyCB)(const char *, void *)
```

The function to handle final update of `cLshcapi_connect()`. It means that `cLshcapi_connect()` is finished running, all data-types are created and they are ready to use by the custom application. When this call-back is triggered, user application can start to use all data-structures and API functions related with content data.

The fist argument contains content data in XML format. Custom application may process it by it's own methods instead of using provided API functions. The second argument is the user defined pointer.

```
void (*fcLshcapi_connectfailedCB)(int, void *)
```

The function to handle errors while `cLshcapi_connect()` function is running. Once triggered, it means that the `cLshcapi_connect()` stops running and there will no further callbacks be triggered. Error code will be set into first calling argument accordingly. The second argument is the user defined pointer.

3. Definitions and Types

There are some other definitions in the header file for error codes (resulting value of some of API functions), option codes while calling some of API function and for definitions for remote message keywords.

3.1. Error Codes

Most of the functions implemented in API will return with an integer value explained below;

```
#define SHC_OK                         0 // calling function successful
#define SHC_FAIL                      -1 // calling function get failed
```

The method `cLshcapi_connect` may return following values in addition to above codes and `fcLshcapi_connectfailedCB()` callback method will be triggered accordingly.

```
#define SHC_UNSUPPORTED_PLATFORM 1 // device or platform is not supported
#define SHC_L3_DENIED            2 // access through routed networks not allowed
#define SHC_DEVICE_DISABLED      3 // device status is disabled on server side
```

3.2. Calling Arguments

There are only two options defined to use with the API calls, which are;

```
#define SHC_OPT_DOCACHE          1 // Store remote content as cache
#define SHC_OPT_DONOTCACHE       2 // Disable caching
```

API will cache media objects such as JPEG or PNG files into local storage in advance to access them quickly whenever it possible. Since the cached objects live in local file system, API will check whether required object is already cached or not, if so, it will be used from the local quickly, otherwise object will be downloaded from the server.

If caching is disabled, media files will not be cached and they will be downloaded from server every time.

The cache folder will be determined by the custom application. This folder can be removed freely before the API calls when necessary.

3.3. Remote Messages

When server sends a message, API triggers a callback method with `const char *` message argument. Following message keywords can be sent by the server;

- `import` Update Message (Banner Text)
- `commit` Reload the content or Restart the Application (Commit launched on server side)
- `kill` Suspend (app may get suspended or quits, "commit" should restart it)
- `reboot` Reboot the system
- `clean` Update the firmware (custom application will update itself, cleans the cache etc.)

Above string constants are defined also in the header file like below;

```
#define CLSHC_MESSAGE_BANNERTEXT    "import" // Update Message
#define CLSHC_MESSAGE_RESTART       "commit" // Restart Application
#define CLSHC_MESSAGE_SUSPEND       "kill"   // Suspend
#define CLSHC_MESSAGE_REBOOTSYSTEM  "reboot" // Reboot System
#define CLSHC_MESSAGE_FWUPDATE      "clean"  // Update Firmware
```

3.4. Types

There is only single type defined in header file which is `shcapi_mediaitem`. It can be used to load remote or local media files specified as URL into memory.

```
struct shcapi_mediaitem {
   const char *url;
   const unsigned char *data;
   unsigned long size;
};
```

- The `url` member specifies the location of media object. It should be set before calling the `cLshcapi_media_loaditem()` function.
- The `data` member is the pointer of media-data which stored in to memory.
- The `size` member holds size of the media in bytes.

`struct shcapi_mediaitem *cLshcapi_media_newitem()`

This API function can be used to allocate a new media-item type object. It should be released after use. Resulting pointer will have `null` definitions for all of its members.

`int cLshcapi_media_loaditem(struct shcapi_mediaitem *item)`

This API function can be used to load remote or local media file into given object. The `url` property of the object should be set before calling this function. Typical usage is shown below;

```
struct shcapi_mediaitem *item = cLshcapi_media_newitem();
item->url = “http://targethost/path/to/mediafile.extension";
// or a local file
item->url = "file:///path/to/mediafile.extension";
if (cLshcapi_media_loaditem(item) == SHC_OK) {
   printf("  item loaded:\n    url: %s\n    sze: %lu\n", item->url, item->size);
} else {
   printf("! unable to load media from url(%s)\n", url);
}
...
cLshcapi_media_releaseitem(item);
```

4. Configuration

Kylone Endpoint API provides functions to access configuration data which set on server side. Custom application may use such values to maintain compatibility with server.

Custom application will have a callback when the configuration-data is arrived and it will use following function to access one of the configuration data;

`const char *cLshcapi_getconfigval(const char *tag)`

Its only argument is the “key” or “tag” which for identifying requested configuration data. The function always return a pointer to data and it shouldn’t be released or freed.

Following configuration properties are defined on server side and values can be accessible using “key” name with `cLshcapi_getconfigval()` function. Using such values to make UI configurable through Kylone server is totally up to custom applications.

Key	Sample Value	Description

`i18n`	`en`	UI Language. This can be one of the following values;
		
		`"en", English`
		`"de", German`
		`"fr", French`
		`"ru", Russian`
		`"zh", Chinese`
		`"th", Thai`
		`"tr", Turkish`
		`"ar", Arabic`
		`"el", Greek`

`logo`	`[resource]`	Main logo (PNG image) resource (URL).
`bgnd`	`[resource]`	Main background (PNG or JPEG image) resource (URL)
`opac`	`0`	Main background transparency in percent (0-100).
`strm`	`kylonev2`	Streaming engine (currently it is Kylone v2).

`surl`	`m3u`	Streaming engine access model. This can be one of the following values;
		
		`"m3u", Resources are configured to use m3u playlists`
		`"raw", Resources are configured to use RAW HTTP socket`

`mmtype`	`csr`	Main Menu Type or Theme. This can be one of following values;
		
		`"def", Classic`
		`"pre", Facility` (most used by hospitality organisations)
		`"csr", Consumer Style`
		
		Custom application may have a few different types of UI design (Theme) and may display them according to selection above which configured on server side.

`xfsfast`	`yes`	Effects/Animations on UI can be configured as enabled/disabled on server side:
		
		`"yes", Disable effects (fast UI, for all hardwares)`
		`"no" , Enable effects (strong hardware may needed)`
		
		Custom application may obey on this value and enable/disable its UI effects accordingly to be compatible with infrastructure (i.e. only for hardware utilised as STB).

`mmshdcl`	`black`	Shadow (or shade) colour. Generally used to suppress whatever is on the screen and putting something on top of it.

`mmstyle`	`rec`	Button style of main menu items. This can be one of following values;
		
		`"rec", Menu buttons created using vector drawings.`
		`“img", Menu buttons created using images.`

`mmitatc`	`orange`	Text colour of active (selected or highlighted) menu item.
`mmitasc`	`black`	Style colour of text on active menu item.
`mitxtcl`	`white`	Default text colour of menu item.
`mitxtsc`	`silver`	Default style colour of text of menu items.
`mmitidl`	`[resource]`	Background image (URL) of passive (default) menu item.
`mmitact`	`[resource]`	Background image (URL) of active menu item.

`langbut`	`[resource]`	Background image (URL) of language selection button.
`appclose`	`[resource]`	Background image (URL) of Close/Home/Back button.
`appcatico`	`[resource]`	Indicator image (URL). It can be used to indicated selected category or playing item etc.
`appbusypng`	`[resource]`	Busy indicator image (URL).
`mcbnostar`	`[resource]`	An “empty” “star” image (URL) to be used to show rating.
`mcbstar`	`[resource]`	An “filled” “star” image (URL) to be used to show rating.
`mcbspk0`	`[resource]`	An image (URL) to be used to show volume level as 0%.
`mcbspk1`	`[resource]`	An image (URL) to be used to show volume level as 33%
`mcbspk2`	`[resource]`	An image (URL) to be used to show volume level as 66%
`mcbspk3`	`[resource]`	An image (URL) to be used to show volume level as 100%
`mcbaspect`	`[resource]`	An image (URL) to be used to put “aspect ratio” button.
`mcbasprhv`	`[resource]`	An image (URL) to be used to put “cover-all-screen” button.
`mcbdock`	`[resource]`	An image (URL) to be used to put “dock” button.
`mcbundock`	`[resource]`	An image (URL) to be used to put “undock” button.
`mcbgrip`	`[resource]`	An image (URL) to be used to put “grip” button.
`mcbmove`	`[resource]`	An image (URL) to be used to put “move” button.
`mcbplay`	`[resource]`	An image (URL) to be used to put “play” button.
`mcbpause`	`[resource]`	An image (URL) to be used to put “pause” button.
`mcbstop`	`[resource]`	An image (URL) to be used to put “stop” button.
`mcbprev`	`[resource]`	An image (URL) to be used to put “prev” button.
`mcbnext`	`[resource]`	An image (URL) to be used to put “next” button.
`mcbback`	`[resource]`	An image (URL) to be used to put “back” button.
`mcbbufb`	`8`	Minimum buffer value for manual buffering. This value is dependent on media-player backend of hardware utilised. The media-player may use this value as minimum value to play stream.
`mcbbufm`	`16`	Medium buffer value for manual buffering. The media-player may use this value as medium value to start buffering again.
`mcbbuft`	`24`	Maximum buffer value for manual buffering. The media-player may use this value as maximum value to stop buffering.

`mcbbufa`	`1`	Auto Buffering (ignore manual buffering values above).
		`“0", Disable auto-buffering (use manual buffering)`
		`“1", Enable auto-buffering (ignore manual buffering)`

`mcbavse`	`1`	Audio/Video time synchronisation while playing streams.
		`“0”, Auto (media-player will handle everting)`
		`“1”, Enabled (custom application will manage everything)`

mcbavss	1	Stream start-playing method. “1”, Auto “2”, Show still image and wait time-sync “3”, Show blank screen and wait time-sync
mcbavsm	2	Audio/Video time synchronisation handling (customisation requirements): “1”, Use System Default value (on advanced settings) “2”, Use Channel Configuration (override sys-default) “3”, Always use “Video PID” as time source “4”, Always use “Audio PID” as time source “5”, Always use “PCR PID” as time source
mcbavco	1	Channel Override for Audio/Video time synchronisation “1”, No “2”, Use System Default “3”, Use “Video PID” as time source “4”, Use “Audio PID” as time source “5”, Use “PCR PID” as time source
mcbavpo	1	PCR Override for Audio/Video time synchronisation “1”, No “2”, Use “Video PID” as sync source “3”, Use “Audio PID” as sync source
mcbavst	65	Audio/Video time synchronisation threshold (milliseconds).
mcbchsd	350	Channel switch delay (zap) in milliseconds.
mcbtout	10	Timeout while playing: Connection timeout while playing live stream (seconds). Since live-streams may goes offline time by time, custom application may stop playing after waiting given amount of time.
mcbtrcv	7	Recovery: Connection timeout while starting to play live stream (seconds). Since live-streams may temporarily offline, custom application may try to play same stream after given value again and again.
mcbdoct	20	Undocking timeout (seconds).
mcbmenu	3000	OSD timeout (milliseconds).
mcbrnum		Sort and renumber channels (starting from 1) in every category alphabetically according to their names. “true”, sort and renumber empty of “false”, don’t take any action.
mcbsort	true	Sort and renumber channels (starting from 1) alphabetically according to their names (covering all categories). “true”, sort and renumber empty of “false”, don’t take any action.
mcstart	1	Starts playing given live-tv channel (with it's number) immediately when the custom application starts running.

`mcpltsd`	`2`	TS Demuxing method
		`"1", software`
		`"2", hardware (accelerated)`

`mcpmpfs`	`2`	Fast switch (one of live stream resource to another).
		`"1", enabled: don't waste so much time to analyse input stream, start as soon as possible.`
		`"2", disabled: analyse input stream correctly and start playing.`

`mcphtor`	`3`	Number of HTTP Open retry before give up (Live stream).
`mcphtrr`	`2`	Number of HTTP Read retry before give up (Live stream).
`mcpxato`	`300`	Live TS analyse timeout (milliseconds).
`mcbbuf0`	`[resource]`	An image (URL) to be used to show buffer status as "empty" when starting to play stream. It may used when the live-stream is not connected yet also.
`mcbbuf1`	`[resource]`	An image (URL) to be used to show connection status as "connected" when starting to play stream.
`mcbbuf2`	`[resource]`	An image (URL) to be used to show buffer status as "buffering" when starting to play stream.
`mcbbuf3`	`[resource]`	An image (URL) to be used to show buffer status as "buffering" when stream stops playing (when buffer gets empty while playing).
`mcbbuf4`	`[resource]`	not used
`mcbbuf5`	`[resource]`	An image (URL) to be used to show buffer status as "full" enough. It may used also when the live-stream starts playing.
`txtncol`	`white`	General text colour to be used on UI.
`txtscol`	`gray`	General text style colour to be used on UI.
`txtszm`	`40`	General text size to be used on UI.
`txtszt`	`36`	General text size for "titles" to be used on UI.
`txtszi`	`45`	General text size for "item-titles" to be used on UI.
`navblen`	`6`	Border size (in pixels) of Navigation cursor.
`navbcol`	`orange`	Border colour of Navigation cursor.
`navatcl`	`black`	Text colour to be used with Navigation cursor.
`navatst`	`silver`	Text style colour to be used with Navigation cursor.

`navstyle`	`rec`	Navigation cursor style
		`"rec", rectangles (vector drawing)`
		`"img", pixmaps or images`

`catvnbgc`	`transparent`	Background colour of a "Category Item"
`catvnopa`	`0.50`	Background colour transparency of a "Category Item" (float, 0-1)
`catvnrad`	`5`	Border radius of "Category Item"

`catvngr2`	`black`	Background gradient-start colour of "Category Item"
`catvngr1`	`purple`	Background gradient-stop colour of "Category Item"
`catvnbor`	`gray`	Border colour of “Category Item"
`catvnact`	`white`	Border colour of active or highlited “Category Item"
`catvntcl`	`white`	Text colour of “Category Item"
`catvntsc`	`gray`	Text style-colour of “Category Item"
`prename`		Header text to display on top of the screen - for “Facility" theme
`premimg`	`[resource]`	Picture (URL) to display on main screen for “Facility" theme
`prelogo`	`[resource]`	Logo image (URL) to display as logo for “Facility" theme
`prehdbg`	`[resource]`	Mid-header background image (URL) for “Facility" theme
`pretbgi`	`[resource]`	Top-header background image (URL) for “Facility" theme
`prewfic`	`[resource]`	WiFi icon image (URL)
`prewftx`		WiFi description text (SSID, password etc) for “Facility" theme
`csrmfc`	`#c53329`	Navigation cursor border colour of “C-Style” theme
`csrmfa`	`#ff4c3e`	Navigation cursor border colour for active or highlighted items of “C-Style” theme
`csrsbt`	`[resource]`	Settings button image (URL) for “C-Style" theme
`csrlogo`	`[resource]`	Logo image (URL) to display as logo for “C-Style” theme
`csrback`	`[resource]`	Background image (URL) for “C-Style" theme.

5. Content

Kylone Endpoint API will provide its own functions to use with structured data sent by the server. In order to understand such data-types and it’s related functions well, we will have a look into data coming from the server first.

Kylone server will always send data in XML format. After doing request to server, resulting XML data will be processed and helper functions will be ready to use it easily with other UI components.

```
<content id="item">...</content>
<content id="prefs">...</content>
<content id="movie">...</content>
<content id="music">...</content>
<content id="clip">...</content>
<content id="tv">...</content>
<content id="radio">...</content>
...
<content id="locs">...</content>
```

In the basic form of XML document seen above, there are `content` sections which are defining content-groups with their `id` property.

```
<content id=“music" rank="">
   <category id="28">
```

```
        <item>...</item>
        <item>...</item>
        ...
     </category>
     <category id=“21" rank="">
        <item>...</item>
        <item>...</item>
        ...
     </category>
     <category id=“18" rank="">
        <item>...</item>
        <item>...</item>
        ...
     </category>
     ...
  </content>
  ...
```

Each `content` section has it’s own `category` list and each `category` has an `id` which holds the category value and it’s own `item` list inside.

The category value defined as `id` is a custom information set by the server admin.

Each `item` section holds a list of values which we will call them as `attributes`. Following is an example about attributes of an `item` which belongs to “`music`” content group;

```
  <item>
     <attr id=“src”>record name of song</attr>
     <attr id=“txt”>display name of the song</attr>
     <attr id=“title”>track name of the song</attr>
     <attr id=“artist">artist of band name</attr>
     <attr id=“album">album name</attr>
     <attr id=“year">year value</attr>
     <attr id=“type”>genre of the song</attr>
     <attr id=“track">track number of the song in album</attr>
     <attr id="cat">18</attr>
  </item>
```

Each content-group may have items inside it with different attribute `id` values according to their functions. For example a movie item may have description or a poster image but a music item will have a cover art image etc.

Custom applications may directly use such resulting XML data instead of using ready to use functions when necessary.

5.1. Nodes

API has required functions to process XML document and `nodes` represents each tag which can be found in the document.

```
  <content id=“music">
     <category id="28">
       <item>
          <attr id=“src”>record name of song</attr>
          ...
       </item>
       ...
```

```
        </category>
        ...
    </content>
    ...
```

All main tags shown above which are `<content>`, `<category>`, `<item>`, `<attr>` will be called as `node` in general.

`void *cLshcapi_node_next(void *item)`

The function which returns the next node of given `node` pointer. This function returns its pointer if there is next node in same level of the tree. Otherwise returns `null` if there is no next node available.

`void *cLshcapi_node_childfirst(void *item)`

The function which returns the first child-node of given `node` pointer. Following example can be used to get linked-list of nodes of content-groups;

```
void *root = cLshcapi_content_root()
void *contentgroup = cLshcapi_node_childfirst(root);
while (contentgroup != (void *) 0) {
   ...
   contentgroup = cLshcapi_node_next(contentgroup);
}
```

`int cLshcapi_node_childcount(void *item)`

The function which returns the number of child-nodes of given `node` pointer. It is possible to use this function to do iterations like below;

```
int numchilds = cLshcapi_node_childcount(item);
for (int i = 0; i < numchilds; i++) {
    ...
    item = cLshcapi_node_next(item);
}
```

`void *cLshcapi_content_root()`

This function will return the root-node pointer. The root-node is the single node which holds other nodes within the XML document. In other word, document can be accessible using the child-nodes of the root-node. Please check `cLshcapi_node_childfirst()` example to see its usage.

5.2. Attributes

Attributes are key/value pairs in the content. Required functions are provided to access such information.

```
<item>
    <attr id="title">track name of the song</attr>
    <attr id="artist">artist or band name</attr>
    <attr id="album">album name</attr>
</item>
```

All attributes are referenced by its parent `node` `<item>` while using helper functions.

`const char *cLshcapi_item_attribute(void *item, const char *tag)`

Returns the value of attribute of item `node` with given tag name. In above example data, calling this function with `“title”` **`tag`** argument will return `“track name of the song”` as value.

`const char *cLshcapi_node_attribute(void *node, const char *tag)`

Returns the value of property of `node` with given name. In above example data, calling this function with `“id”` argument will return `“title”` as value for the first `node`.

`const char *cLshcapi_node_value(void *node)`

Returns the value of given `node`. In above example data, calling this function will return `“track name of the song”` as value for the first `node`.

5.3. Items

This represents the linked-list of items within the category. Required functions are provided to access such information.

```
<item>
   <attr id=“txt”>item 0</attr>
    ...
</item>
<item>
   <attr id=“txt”>item 1</attr>
   <attr id=“title”>title value of item 1</attr>
    ...
</item>
<item>
   <attr id=“txt”>item 2</attr>
    ...
</item>
<item>
   <attr id=“txt”>item 3</attr>
    ...
</item>
```

An item doesn’t has identifier but one of it’s attributes can be used for identification. For example, the `“txt”` attribute is the most used attribute to identify an item.

`void *cLshcapi_content_items(const char *rootid, const char *category)`

This function can be used to get linked `node` list of given category and content-group (`rootid`).

```
void *node = cLshcapi_content_items("item", "0");
```

In above example it will return a pointer to linked-list of nodes of main-menu items.

`void *cLshcapi_content_item(const char *rootid, const char *category, const char *tag, const char *tagval)`

It is possible to find an item `node` by giving one of it’s attribute key/value besides category and content-group (`rootid`). Once an item `node` is obtained, it can be used to access its attributes using functions explained in previous section. Typical usage is show below;

```
void *node = cLshcapi_content_item("item", "0", "txt", "Television");
const char *logo_url = cLshcapi_item_attribute(node, "logo");
```

In above example, resulting `node` pointer will be used in second call to access its property. If the first call will return `null`, then the second call will return `null` too since the `node` is undefined.

5.4. Categories

This represents the linked-list of categories within the content-group. Required functions are provided to access such information.

```
<category id="28" rank="1">...</category>
<category id="21" rank="3">...</category>
   <item>...</item>
   <item>...</item>
    ...
<category id="18" rank="2">...</category>
```

`void *cLshcapi_content_categories(const char *rootid)`

This function can be used to get linked `node` list of given content-group (`rootid`).

```
printf("> categories in main-menu;\n");
void *node = cLshcapi_content_categories("item");
while (node != (void *) 0) {
   const char *catid = cLshcapi_node_attribute(node, “id");
   const char *rank = cLshcapi_node_attribute(node, "rank");

   node = cLshcapi_node_next(node);
}
printf("- done\n");
```

In above example it is possible to do iteration on list of categories gathered by this function. There is also another example of usage to get properties such as “`id`” and “`rank`” of the node.

5.5. Content Groups

Content-Groups represent the linked-list of main items within the document. Required functions are provided to access such information.

```
<content id="movie">...</content>
<content id="music">...</content>
   <category id="28" rank="">...</category>
   <category id="21" rank="">...</category>
   ...
<content id="tv">...</content>
<content id="radio">...</content>
```

Each content-group `node` is identified by its `“id”` property.

This may called as "*rootitems*" also. It is different form the menu-items. Please check next chapter for more details about menu-items.

`void *cLshcapi_content_rootitem(const char *rootid)`

The function which can be used to get `node` of a certain content-group. According to above data-set, the calling argument `rootid` can be one of `movie`, `music`, `tv` or `radio`.

6. Using the Content

Attribute keys for different items in different content-groups will have different set of definitions. Possible content-groups and its usage will be explained in next chapters.

6.1. Menu Items

There is content-group called `"item"` holds menu-items. Their `“src”` attribute is same with the content-group IDs in general.

```
<content id="item">
   <category id="0" rank="">
      <item>...</item>
      <item>
         <attr id="src">music</attr>
         <attr id="txt">Music</attr>
         <attr id="ena">1</attr>
         <attr id="ord">3</attr>
         <attr id="logo">...</attr>
         <attr id="bgnd">...</attr>
         <attr id="opac">60</attr>
         <attr id="sub">0</attr>
      </item>
      <item>...</item>
      <item>...</item>
   </category>
</content>
```

As shown above example, there is category with ID zero (`"0"`) even if there is no need to use categories to maintain compatibility with our organised data structure. The category with ID zero (`"0"`) may used to represent all items in particular content-group also.

In order to find menu-items to show on UI of the custom application, items under category(`"0"`) of content-group(`"item"`) should be used. When a particular menu-item is selected, its `“src”` attribute will be used to access content-groups to find related content shown below.

```
<content id=“music">...</content>
   <category id="28" rank=“">...</category>
   <category id="21" rank="">...</category>
   ...
</content>
```

It is possible to get list of menu-items like below;

```
 void *item = cLshcapi_content_items("item", "0");
```

Following example can be used to get a linked-list of "*rootitems*" and to print name of each;

```
printf("> listing root items;\n");
void *item = cLshcapi_node_childfirst(cLshcapi_content_root());
while (item != (void *) 0) {
   const char *rootid = cLshcapi_node_attribute(item, "id");
   printf("    %s\n", rootid);
   item = cLshcapi_node_next(item);
}
printf("- done\n");
```

Above example will produce following output;

```
> listing root items;
    item
    prefs
    movie
    music
    clip
    tv
    radio
    locs
    tube
- done
```

Following example can be used to get list of “Main Menu Items” configured on server side and to print name of each;

```
printf("> listing main-menu items in category '0';\n");
void *item = cLshcapi_content_items("item", "0");
while (item != (void *) 0) {
   const char *menuid = cLshcapi_item_attribute(item, "src");
   const char *menutext = cLshcapi_item_attribute(item, "txt");
   printf("    %s \t: %s\n", menuid, menutext);
   item = cLshcapi_node_next(item);
}
printf("- done\n");
```

Above example will produce following output;

```
> listing main-menu items in category '0';
    weather: Weather
    tv     : Television
    radio  : Radio
    prefs  : Settings
    music  : Music
    movie  : Movie
    info   : Information
    clip   : Video Clip
- done
```

Another example shown below will print all attributes of “tv” which is from “Main Menu Item” list.

```
printf("> listing attributes of menu-item 'tv';\n");
void *item = cLshcapi_content_item("item", "0", "src", "tv");
void *attr = cLshcapi_node_childfirst(item);
while (attr != (void *) 0) {
   const char *attrid = cLshcapi_node_attribute(attr, "id");
   const char *attrval = cLshcapi_node_value(attr);
   printf("    %s \t: %s\n", (attrid ? attrid : "<>"), (attrval ? attrval : "<>"));
   attr = cLshcapi_node_next(attr);
}
printf("- done\n");
```

Above example will produce following output;

```
> listing attributes of menu-item 'tv';
    src    : tv
    txt    : Television
    ena    : 1
    ord    : 0
    logo   : ...
```

```
            bgnd   : ...
            opac   : 60
            sub    : 0
            ptxt   :
            cat    : 0
        - done
```

Key	Sample Value	Description
`src`	`tv`	Attribute that match to content-group ID.
`txt`	`Television`	Display name of the item.

`ena`	`1`	Attribute that holds status of this item
		`“0”, Disabled, item will not functioning`
		`“1”, Enabled`

`ord`	`0`	Sequence number of the item. Items may sorted using this attribute while placing to UI.
`logo`	`[resource]`	An image (URL) to be used as button or background image of menu item.
`bgnd`	`[resource]`	An image (URL) to be used as background image of the new page when an item is selected. It may used for any other purpose also.
`opac`	`60`	Transparency of the background image (percent, 0-100).

`sub`	`0`	Attribute that holds sub-menu flag of this item. If the item is sub-menu, then it a new set of menu will be displayed on UI using the items’s `src` attribute.
		`“0”, there is no sub-menu for this item.`
		`“1”, there is sub-menu defined for this item.`

`ptxt`		A text to be used as description of the item. It may used for any other purpose also.
`cat`	`0`	Category ID of this item (copied from the parent category id).

6.2. Sub Menu Items

Sub-menus are defined in same way with the main-menu items. The difference is that the sub-menus are located into a category with id called sub-menu name.

Please check following example.

```
<content id="item">
   <category id="0" rank="">
      <item>...</item>
      <item>
         <attr id="src">info</attr>
         <attr id="sub">1</attr>
      </item>
      <item>...</item>
   </category>
   <category id="info" rank="">
         <item>...</item>
         <item>...</item>
         ...
   </category>
</content>
```

In above example, since there is “sub” flag set as “1”, the “info” menu item is defined as sub-menu. In this situation, the custom application should look into another category with id “info” which is the name of the item too. So, items of sub-menu will be used from that category.

Please keep in mind that there may nested sub-menu definitions.

6.3. Preferences

This content group called as “prefs” and holds the preferences menu of the application.

```
<content id="prefs">
```

Following example will get all items in this content-group.

```
void *item = cLshcapi_content_items("prefs", "0");
```

Key	Sample Value	Description
src	language	not used
txt	Language	Display name of the item.
ena	1	Attribute that holds status of this item “0”, Disabled, item will not functioning “1”, Enabled
logo	[resource]	An image (URL) to be used as button or background image of the item.
cat	0	Category ID of this item (copied from the parent category id).

6.4. Movie

This content group called as “movie” and holds the Video On Demand content.

```
<content id="movie">
```

Following example will get all items in this content-group.

```
void *item = cLshcapi_content_items("movie", "0");
```

Key	Sample Value	Description
src	[resource]	URL to be used to play movie
txt		Title or display name of the movie
itxt		A brief description of the movie

idir		A text to be used to display “directors” of the movie
star		A text to be used to display “cast” of the movie
irate	7.4	A float number (0.0-10.0) to be used to display “rating” of the movie.
kind	Advanture	A text to be used to display “kind” of the movie
dur	2h 22min	A text to be used to display “duration” of the movie
year	2018	A text to be used to display “year” of the movie
logo	[resource]	An image (URL) to be used as poster image of the movie.
cat	0	Category ID of this item (copied from the parent category id).

6.5. Music

This content group called as “`music`” and holds the Video On Demand content.

```
<content id="music">
```

There will be no category with `id “0”` for “`music`”. Following example will get all categories this content-group.

```
void *cat = cLshcapi_content_categories("music");
```

After using above command, you can use node pointer (`cat`) to access items in categories like below;

```
while (cat != (void *) 0) {
   const char *catid = cLshcapi_node_attribute(cat, “id");
   ...
   void *item = cLshcapi_node_childfirst(cat);
   while (item != (void *) 0) {
      const char *keyval = cLshcapi_item_attribute(item, "artist");
      ...
      item = cLshcapi_node_next(item);
   }
   cat = cLshcapi_node_next(cat);
}
```

Key	Sample Value	Description
src	[resource]	URL to be used to play song
fbp		File base path of the song on the server side.
txt		Title or display name of the movie
title		A text to be used to display “name” of the song
artist		A text to be used to display “singer” or “band" of the song
album		A text to be used to display “album” of the song
year	2018	A text to be used to display “year” of the album or song
type	20	not used

track	4/15	A text to be used to display “track” information in album of the song
cat	0	Category ID of this item (copied from the parent category id).

Most of the attributes of the music item will be determined from the music file itself using ID3 tags including cover-art picture.

Cover-art can be fetched from the server using an API call with file-base-path (fbp) option. Following is the function definition in the API;

```
struct shcapi_mediaitem *cLshcapi_getcoverart(const char *targethost, const
char *mediabasepath)
```

Following is an example use of such function to fetch cover-art from the server;

```
const char *mediabasepath = cLshcapi_item_attribute(item, "fbp");
struct shcapi_mediaitem *coverart = cLshcapi_getcoverart(targethost, mediabasepath);
if (coverart != (struct shcapi_mediaitem *) 0) {
   const char *title = cLshcapi_item_attribute(item, "txt");
   printf("    cover-art image of item '%s' : %lu byte(s)\n", title, coverart->size);
   cLshcapi_media_releaseitem(coverart);
}
```

6.6. Music Video

This content group called as “clip” and holds the Music Video content.

```
<content id="clip">
```

Following example will get all items in this content-group.

```
void *item = cLshcapi_content_items("clip", "0");
```

Key	Sample Value	Description
src	[resource]	URL to be used to play music video.
txt		Title or display name of the music video.
star		A text to be used to display “singer” or “band" of the music video.
logo	[resource]	An image (URL) to be used as cover-art of the music video.
cat	0	Category ID of this item (copied from the parent category id).

6.7. Live TV

This content group called as “tv” and holds the Live TV content.

```
<content id="tv">
```

Following example will get all items in this content-group.

```
void *item = cLshcapi_content_items("tv", "0");
```

Key	Sample Value	Description
`src`		not used
`txt`		Title or display name of the live tv channel.

`syn`		Attribute that holds A/V sync source of the live tv channel.
		`“sys”, System default (use configuration parameter)`
		`“aid”, force to use Audio PID`
		`“vid”, force to use Video PID`
		`“pcr”, force to use PCR PID`

`logo`	`[resource]`	An image (URL) to be used as logo of the live tv channel.
`rrc`	`[resource]`	URL to be used to play live tv channel for STBs mostly based on Linux/ffmpeg.
`arc`	`[resource]`	URL to be used to play live tv channel for Android based systems.
`xrc`	`[resource]`	URL to be used to play live tv channel for iOS based systems.
`raw`	`[resource]`	URL to be used to play live tv channel using RAW HTTP socket.
`rpc`	`[resource]`	URL to be used to play live tv channel for Desktop PCs mostly based on ffmpeg.
`xpc`	`[resource]`	URL to be used to play live tv channel for other Desktop PCs (playlist based).
`chn`	`1`	Sequence number of the live tv channel.
`cat`	`0`	Category ID of this item (copied from the parent category id).

6.8. Live Radio

This content group called as “`radio`” and holds the Live Radio content.

```
<content id="radio">
```

Following example will get all items in this content-group.

```
void *item = cLshcapi_content_items("radio", "0");
```

Attributes and usage of Live Radio content group is same with the Live TV.

7. Helper Functions

`const char *cLshcapi_getbannertext()`

The header or welcome text set on server side can be fetched using this function. It is available to use this function only after the `fcLshcapi_configreadyCB` call-back is triggered.

`void cLshcapi_util_delaymillisecond(long ms)`

An helper function to create delays in proper way. Calling argument should be a number in milliseconds.

8. Compiling and Linking Options

There is a sample Makefile provided with API package which may used as a reference also.

8.1. Linux x86_64

```
CFLAGS   -fPIC
LIBS     -lstdc++ -lz -ldl -lpthread
```

8.2. Linux aarch64

```
CFLAGS   -fPIC
LIBS     -lstdc++ -lz -lrt -lnsl -lm -ldl -lpthread
```

8.3. Linux armv7

```
CFLAGS   -march=armv7 -ffunction-sections -fdata-sections -mfloat-abi=soft -fPIC
LIBS     -lstdc++ -lz -lrt -lnsl -lm -ldl -lpthread
```

8.4. MacOS x86_64

```
CFLAGS
LIBS     -ldl -lpthread -framework Foundation -framework CoreFoundation
```

9. Example Application

Following example will show how to use API to create custom application. You can find a complete running example application within the API distribution package.

```c
/*
 * shcapitest.c
 * Authors: Gokhan Poyraz <gokhan@kylone.com>
 *
 * Kylone Endpoint API for C implementation
 * Copyright (c) 2018, Kylone Technology International Ltd.
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston MA 02110-1301, USA.
*/

#include <stdio.h>  //printf
#include <stdlib.h> //free
#include <string.h> //strcmp

#include <cLshcapi.h>

struct shcapi_mediaitem *app_new_media_item(const char *url)
{
   struct shcapi_mediaitem *item = cLshcapi_media_newitem();
   item->url = url;
   if (cLshcapi_media_loaditem(item) == SHC_OK) {
      printf("  item loaded:\n    url: %s\n    sze: %lu\n", item->url, item->size);
   } else {
      printf("! unable to load media from url(%s)\n", url);
   }
   return item;
}

void app_set_main_items()
{
   const char *url = (const char *) 0;
   struct shcapi_mediaitem *item = (struct shcapi_mediaitem *) 0;

   printf("~ setting-up: logo\n");
   url = cLshcapi_getconfigval("logo");
   item = app_new_media_item(url);
   if (item->size) {
      // consume item->data
      printf("  main logo set\n");
   }
   cLshcapi_media_releaseitem(item);

   printf("~ setting-up: background\n");
   url = cLshcapi_getconfigval("bgnd");
   item = app_new_media_item(url);
   if (item->size) {
      // consume item->data
      printf("  main background set\n");
   }
   cLshcapi_media_releaseitem(item);
}

static void app_configreadyCB(const char *xml, void *userptr)
{
   printf("~ callback: config is ready\n");
   app_set_main_items();
}

static void app_contentreadyCB(const char *xml, void *userptr)
{
   void *rootnode, *rootitem, *category, *item, *attr;
   int itemnum;

   printf("~ callback: content is ready\n");

   printf("> walking on content;\n");
   rootnode = cLshcapi_content_root();
   printf("  number of item: %d\n", (rootnode != (void *) 0) ? cLshcapi_node_childcount(rootnode) : 0);
   rootitem = cLshcapi_node_childfirst(cLshcapi_content_root());
   while (rootitem != (void *) 0) {
      const char *rootid = cLshcapi_node_attribute(rootitem, "id");
      printf("    [%s], number of category: %d\n", (rootid ? rootid : "<>"), cLshcapi_node_childcount(rootitem));
      category = cLshcapi_node_childfirst(rootitem);
      while (category != (void *) 0) {
         const char *catid = cLshcapi_node_attribute(category, "id");
         const char *catrank = cLshcapi_node_attribute(category, "rank");
         printf("      category: [%s], rank: %s, number of items: %d\n”,
            (catid ? catid : "<>"), (catrank ? catrank : "<>"), cLshcapi_node_childcount(category)
         );
         item = cLshcapi_node_childfirst(category);
         itemnum = 0;
         while (item != (void *) 0) {
            printf("         [%d], number of attributes: %d\n", itemnum, cLshcapi_node_childcount(item));
            attr = cLshcapi_node_childfirst(item);
            while (attr != (void *) 0) {
               const char *attrid = cLshcapi_node_attribute(attr, "id");
               const char *attrval = cLshcapi_node_value(attr);
               printf("            %s \t: %s\n", (attrid ? attrid : "<>"), (attrval ? attrval : "<>"));
               attr = cLshcapi_node_next(attr);
            }
            item = cLshcapi_node_next(item);
            ++itemnum;
         }
         category = cLshcapi_node_next(category);
      }
      rootitem = cLshcapi_node_next(rootitem);
   }
   printf("- done\n");
}

static void app_connectfailedCB(int errcode, void *userptr)
{
   printf("~ callback: connect failed with reason(%d)\n", errcode);
}

static void app_progressCB(int progress, void *userptr)
{
   printf("~ callback: progress: %d%%\n", progress);
}

static void app_remotemessageCB(const char *msg, void *userptr)
```

```
{
   printf("~ got remote message: %s\n", msg);
   if (strcmp(msg, CLSHC_MESSAGE_BANNERTEXT) == 0) {
      printf("  update header text: %s\n", cLshcapi_getbannertext());
   } else
   if (strcmp(msg, CLSHC_MESSAGE_RESTART) == 0) {
      printf("  app restart\n");
   } else
   if (strcmp(msg, CLSHC_MESSAGE_SUSPEND) == 0) {
      printf("  app suspend\n");
   } else
   if (strcmp(msg, CLSHC_MESSAGE_REBOOTSYSTEM) == 0) {
      printf("  sys reboot\n");
   } else
   if (strcmp(msg, CLSHC_MESSAGE_FWUPDATE) == 0) {
      printf("  sys firmware or app update\n");
   } else {
      printf("! unknown message\n");
   }
}
void app_example_usage_of_content(const char *targethost)
{
   void *cat, *item, *attr;
   const char *pstr;
   struct shcapi_mediaitem *coverart;

   printf("> listing root items;\n");
   item = cLshcapi_node_childfirst(cLshcapi_content_root());
   while (item != (void *) 0) {
      const char *rootid = cLshcapi_node_attribute(item, "id");
      printf("    %s\n", (rootid ? rootid : "<>"));
      item = cLshcapi_node_next(item);
   }
   printf("- done\n");

   printf("> listing categories in main-menu;\n");
   item = cLshcapi_content_categories("item");
   while (item != (void *) 0) {
      const char *catid = cLshcapi_node_attribute(item, "id");
      printf("    %s\n", (catid ? catid : "<>"));
      item = cLshcapi_node_next(item);
   }
   printf("- done\n");

   printf("> listing main-menu items in category '0';\n");
   item = cLshcapi_content_items("item", "0");
   while (item != (void *) 0) {
      const char *menuid = cLshcapi_item_attribute(item, "src");
      const char *menutext = cLshcapi_item_attribute(item, "txt");
      printf("    %s \t: %s\n", (menuid ? menuid : "<>"), (menutext ? menutext : "<>"));
      item = cLshcapi_node_next(item);
   }
   printf("- done\n");

   printf("> listing main-menu items in category 'info';\n");
   item = cLshcapi_content_items("item", "info");
   while (item != (void *) 0) {
      const char *menuid = cLshcapi_item_attribute(item, "src");
      const char *menutext = cLshcapi_item_attribute(item, "txt");
      printf("    %s \t: %s\n", (menuid ? menuid : "<>"), (menutext ? menutext : "<>"));
      item = cLshcapi_node_next(item);
   }
   printf("- done\n");

   printf("> listing attributes of menu-item 'tv';\n");
   item = cLshcapi_content_item("item", "0", "src", "tv");
   attr = cLshcapi_node_childfirst(item);
   while (attr != (void *) 0) {
      const char *attrid = cLshcapi_node_attribute(attr, "id");
      const char *attrval = cLshcapi_node_value(attr);
      printf("    %s \t: %s\n", (attrid ? attrid : "<>"), (attrval ? attrval : "<>"));
      attr = cLshcapi_node_next(attr);
   }
   printf("- done\n");

   printf("> getting only txt attribute of menu-item 'tv';\n");
   item = cLshcapi_content_item("item", "0", "src", "tv");
   pstr = cLshcapi_item_attribute(item, "txt");
   printf("    txt attribute of menu-item 'tv' is '%s'\n", (pstr ? pstr : "<>"));
   printf("- done\n");

   printf("> fetching cover-art image of first item in first music category;\n");
   cat = cLshcapi_content_categories("music");
   while (cat != (void *) 0) {
      item = cLshcapi_node_childfirst(cat);
      while (item != (void *) 0) {
         const char *mediabasepath = cLshcapi_item_attribute(item, "fbp");
         pstr = cLshcapi_item_attribute(item, "txt");
         coverart = cLshcapi_getcoverart(targethost, mediabasepath);
         if (coverart != (struct shcapi_mediaitem *) 0) {
            printf("    cover-art image of item '%s' : %lu byte(s)\n", pstr, coverart->size);
            cLshcapi_media_releaseitem(coverart);
            // only fist item is processed.
            break;
         } else {
            printf("  ! unable to fetch cover-art of item '%s'\n", pstr);
         }
         // continue to process next item
         item = cLshcapi_node_next(item);
      }
      // only fist category is processed.
      break;
      cat = cLshcapi_node_next(cat);
   }
   printf("- done\n");
}

int app_set_callback_functions()
{
   if (cLshcapi_setcallback(0, CLSHC_INVOKE_CONFIGREADY, app_configreadyCB) != SHC_OK) {
      printf("! unable to set callback: app_configreadyCB\n\n");
      return 1;
   }
   if (cLshcapi_setcallback(0, CLSHC_INVOKE_CONTENTREADY, app_contentreadyCB) != SHC_OK) {
      printf("! unable to set callback: app_contentreadyCB\n\n");
      return 1;
   }
   if (cLshcapi_setcallback(0, CLSHC_INVOKE_CONNECTFAILED, app_connectfailedCB) != SHC_OK) {
      printf("! unable to set callback: app_connectfailedCB\n\n");
      return 1;
   }
   if (cLshcapi_setcallback(0, CLSHC_INVOKE_PROGRESS, app_progressCB) != SHC_OK) {
      printf("! unable to set callback: app_progressCB\n\n");
      return 1;
   }
   if (cLshcapi_setcallback(0, CLSHC_INVOKE_REMOTEMESSAGE, app_remotemessageCB) != SHC_OK) {
      printf("! unable to set callback: app_remotemessageCB\n\n");
      return 1;
   }
   return 0;
}

int app_wait_server_if_not_ready(const char *targethost, int timeout, int maxtry)
{
   int trycount = 0;
   while (cLshcapi_serverisready(targethost, timeout) != SHC_OK) {
      if (++trycount >= maxtry) {
         printf("! server is not ready, giving up\n\n");
         return 1;
      }
      printf("! waiting server to be ready(%d)\n", trycount);
      cLshcapi_util_delaymillisecond(1000);
   }
   printf("  server is ready\n");
   return 0;
}

int app_wait_connect_thread()
{
   while (cLshcapi_isconnecting() == SHC_OK)
      cLshcapi_util_delaymillisecond(1000);
   printf("> connect thread finished\n");
   return 0;
}

void app_print_api_version()
{
   const char *version = cLshcapi_getapiversion_alloc();
   printf("> Using API version %s\n", version);
   free((void *)version);
}

int app_terminate(int retcode)
{
   printf("> destroying API\n");
   cLshcapi_delete();
   printf("> done\n");
   return retcode;
}

int main(int argc, char** argv)
{
   const char *targethost = "10.72.0.2";

   app_print_api_version();

   /*
    * given cache folder should be exists, otherwise cLshcapi_create will return SHC_FAIL
```

```
     */
    printf("> creating API\n");
    if (cLshcapi_create(argc, argv, "/tmp") != SHC_OK) {
       printf("! unable to create API\n\n");
       return app_terminate(1);
    }

    printf("> setting-up: callback functions\n");
    if (app_set_callback_functions() != 0)
       return app_terminate(1);

    printf("> checking server availability\n");
    if (app_wait_server_if_not_ready(targethost, 2, 10) != 0)
       return app_terminate(1);

    printf("> connecting...\n");
    if (cLshcapi_connect(targethost, SHC_OPT_DONOTCACHE) != SHC_OK) {
       printf("! could not connect to %s\n\n", targethost);
       return app_terminate(1);
    }

    printf("> connect thread launched, waiting to be finished\n");
    app_wait_connect_thread();

    app_example_usage_of_content(targethost);

    return app_terminate(0);
}
```

Kylone Technology International Limited
610, 6/F, B.36, Chentian Industrial Park, Baotian 1st RD.
518055, Xixiang Town, Bao'an District, Shenzhen, China
Phone: +86 (755) 26501345
Fax: +86 (755) 26549326
http://kylone.com/ , sales@kylone.com

TBS Technologies International Limited
610, 6/F, B.36, Chentian Industrial Park, Baotian 1st RD.
518055, Xixiang Town, Bao'an District, Shenzhen, China
Phone: +86 (755) 26501345
Fax: +86 (755) 26549326
sales@tbsdtv.com http://www.tbsiptv.com/