Kylone MicroCMS XML API User Guide

Version 2.0.64

Revision November 15, 2017

CONTENTS

1. Introduction

In addition to the web interface, MicroCMS provides an XML API to manage all functions available on the system which includes functions related with Content, Media Server, STBs and some OS components. The API allows access to several types of data on the system for integration with and use in other systems. The API is provided as a web service that is implemented using HTTP requests and responses.

The structure of the URI for the API requests is shown below:

Starting from MicroCMS v2.0.64	`http(s)://hostname/portal/`

The hostname is the device's IP address or Domain name. There is no need to use parameters-values pairs within the URI. The keywords for all the parameters should be sent within XML document and all are described here. The values should be keywords or data-values in XML format. The response data is always in XML format. When using the API with a command line tool such as cURL or wget, HTTP POST method is supported only.

2. Using the XML API

There is currently single type of API request which is accessed via the `request` parameter along with other arguments including `act` and `key` arguments when necessary. Basic tasks are as follows;

- Login
- Inquiry on Single Entities
- Doing actions on Single Entity using `act` parameter
- Doing actions on List of Objects using `act` and `key` parameter
- Inquiry on List of Objects using `srt`, `dir`, `src` and `dsb` parameters
- Inquiry for particular object in the list using `act` and `key` parameters
- Logout

The XML document (the request) should be sent in "xml" form element with POST method. Arguments sent in URI will be discarded. Other form elements in POST request will be treated as attributes of `args` section within the XML document to easy use of the API.
.

2.1. Request

Basic XML document body is shown below for all kind of requests. It consists of two main parts: `operation` and `data`;

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">session id which gathered via login request</atr>
         </cookies>
      </operation>
      <data model="struct">
         <request>target function keyword of this request</request>
         <args>
            <atr name="name or mapped keyword of parameter">value of parameter</atr>
            <atr name="name or mapped keyword of parameter">value of parameter</atr>
         </args>
      </data>
   </container>
</cLst>
```

The XML document has to include `type` tag with static value `request` in the `operation` section for all type of requests;

```
         <type>request</type>
```

The `cookies` section should be provided with its attribute `"s"` except login request to maintain the session;

```
         <cookies>
            <atr name="s">session id which gathered via login request</atr>
         </cookies>
```

The `model` attribute of the `data` tag should always be `struct` since there will several types of information may need to sent to the service;

```
      <data model="struct">
```

The target function of the request should be provided in `request` tag as a keyword under `data` section:

```
         <request>target function keyword of this request</request>
```

A repeated-list of parameters related with the target function which you are doing inquiry should be provided under `args` section as attributes:

```
         <args>
            <atr name="name or mapped keyword of parameter">value of parameter</atr>
            <atr name="name or mapped keyword of parameter">value of parameter</atr>
         </args>
```

2.2. Response

The structure of the XML document which sent as response of a service call is very similar to request document. It consists of two main parts too as `operation` and `data`. There are some additional tags in the document such as `fieldmap`, `options` and `keyfield` to organise data returned by the service.

```
<cLst>
   <container>
      <operation>
         <type>response</type>
         <status>ok</status>
         <fieldmap>
            <atr name="keyword" must="true" ro="false" type="str">title</atr>
            <atr name="keyword" must="false" ro="false" type="timestamp">title</atr>
            ...
         </fieldmap>
         <options>
            <opt name="keyword">
               <atr val="value to use with requests">value to display</atr>
               ...
            </opt>
            <opt name="keyword">
              ...
            </opt>
            ...
         </options>
         <keyfield>keyword</keyfield>
      </operation>
      <data model="tree">
         <elm>
            <atr n="keyword">value</atr>
            ...
         </elm>
         <elm>
           ...
         </elm>
         ...
     </data>
   </container>
</cLst>
```

Document includes the `type` tag which holds static value `response` and the `status` tag which hold the result code of the operation performed. Both located under `operation` tag:

```
         <type>response</type>
         <status>ok</status>
```

Status will always be "`ok`" when the operation was successful. Otherwise it will always be "`failed`" when the operation was failed.

2.2.1. Field Mapping and Field Properties

Response document may include a repeated-list of `fieldmap` attributes under `operation` section. This information can be used conjunction with other requests and should be included into request document as an attribute under `args` section while working on Singe Entities or List of Objects.

```
<fieldmap>
   <atr name="k0" must="true" ro="true" type="str">ID Number</atr>
   <atr name="k1" must="true" ro="false" type="str">Mobile</atr>
   <atr name="k2" must="false" ro="false" type="str">E-Mail</atr>
</fieldmap>
```

Client application should use the value of `name` flag for that particular field while doing further operations. An example is shown below to modify only "Mobile" field with sending relevant attribute (`k1`) in the `args` list;

```
<args>
   <atr name="k1">123456789</atr>
</args>
```

Client application may use the content of attributes in this `fieldmap` list to display it to end-user as field descriptor or title (in above example they are ID Number, Mobile, E-Mail).

There are also some other useful information inside attributes that client application should take care of, such as;

- `must` : The parameter that mapped to attribute which has `must="true"` flag should be provided during object creation and can not be left blank while updating (or while doing modification on) an object.
- `ro` : The parameter that mapped to attribute which has `ro="true"` (read-only) can be provided while object creation but can not be modified in the future.
- `type` : The content of the parameter should be processed by the client application according to value of this `type` field in its mapped attribute.

A cropped response document against NTP service settings inquiry (request=ntp) is shown below as an example of field mapping;

```
<operation>
   <type>response</type>
   <fieldmap>
      <atr name="k0" must="false" ro="true" type="section">Settings</atr>
      <atr name="k1" must="false" ro="false" type="str">Allowed Networks</atr>
      <atr name="k2" must="false" ro="true" type="section">Service</atr>
      <atr name="k3" must="true" ro="false" type="str">Status</atr>
   </fieldmap>
</operation>
<data model="tree">
   <elm>
      <atr n="k1">172.22.22.0/255.255.255.128</atr>
      <atr n="k3">1</atr>
   </elm>
</data>
```

On above example, the `type` information indicated as `section` within attributes are just for informing the client application that some of the fields are grouped into that section. Thus, those are not parameters and will not be included into the `data` set.

2.2.2.Option Lists

Response document may include valid values for particular field as a list of attributes under `operation` section. When creating an object or doing modifications on it, the `options` list should

be utilised if necessary. On the other hand, client application may use the value in `options` list while presenting information to end-users instead of using the field value.

There is `type="option"` flag will be given in the attributes of `fieldmap` to identify which fields will use `options` list.

A cropped response document against System Date/Time settings inquiry (request=timezone) is shown below as an example of option lists;

```
<operation>
   <type>response</type>
   <fieldmap>
      <atr name="k4" must="true" ro="false" type="option">Time Zone</atr>
   </fieldmap>
   <options>
      <opt name="k4">
         <atr val="UTC">UTC</atr>
         ...
         <atr val="EET">Eastern European Time</atr>
         ...
     </opt>
   </options>
</operation>
<data model="tree">
   <elm>
      <atr n="k4">UTC</atr>
   </elm>
</data>
```

When doing modifications, the value of `val` flag should be passed as the value of field instead of any other value. If we want to change `k4` which is `Time Zone` to "`Eastern Europe Time`" we should send below parameter in `args` list;

```
<args>
   <atr name="k4">EET</atr>
</args>
```

2.3. Data Handling

There may `data` section exists or not within response document depending on the target function. For example, there will be no `data` section as response of the login request since the session ID which is the only information already sent in `cookies` area.

The `model` attribute of the `data` tag will be determined by the calling function. Client application should take care of `model` keyword in order to proper processing of data sent by the service. Possible keywords of `model` value shown as follows;

- `tree` : A repeated-list of information included which all item have same structure.
- `struct` : Several type of information included depending on the calling function.
- `text` : A single body of text included.
- `json` : A single body of a JSON document included.

You can check another cropped example below to see `status` code and `data` section in response of a failed request;

```
<operation>
   <type>response</type>
   <status>failed</status>
</operation>
<data model="text">job failed</data>
```

2.3.1.List of Objects

When you call a function without any parameter and if the returning response document has a list of objects in `data` set then it means you are working on List of Objects.

Object list is a kind of container which you can;

- search objects within the list,
- edit (modify or update) particular object in list,
- delete an object from the list,
- add an object into list,

by calling the same function name with a different set of parameters.

A cropped response of Video/TV categories (request=catmov) list is shown below as an example;

```
<operation>
   <type>response</type>
   <fieldmap>
      <atr nanme="k1" type="str">Display Name</atr>
      <atr nanme="k2" type="str">Record Name</atr>
      <atr nanme="k3" type="str">Category ID</atr>
      <atr nanme="k4" type="str">Rank</atr>
   </fieldmap>
   <keyfield>k2</keyfield>
</operation>
<data model="tree">
   <elm>
      <atr n="k1">All</atr>
      <atr n="k2">all</atr>
      <atr n="k3">0</atr>
      <atr n="k4">99</atr>
   </elm>
   ...
   <elm>
      <atr n="k1">Other</atr>
      <atr n="k2">other</atr>
      <atr n="k3">24</atr>
      <atr n="k4">0</atr>
   </elm>
</data>
```

The list in the returning `data` set may not contain all fields of an object but it will return only a few of them. You can access full information of particular object by calling service with additional `act` keyword which will be explained later.

2.3.1.1.Sorting the List

There are two parameters which we need to use to have a sorted list of objects in result document. Those are named as `srt` and `dir`.

`srt` : Name of the field to be sorted against it.
`dir` : The direction of the sorting. Valid values are “`a`” for ascending and “`d`” for descending.

Please check the following request to have a sorted object list with descending order against “Rank” field of Video/TV Categories;

```
<operation>
   <type>request</type>
</operation>
<data model="struct">
   <request>catmov</request>
   <args>
      <atr name="srt">k4</atr>
      <atr name="dir">d</atr>
   </args>
</data>
```

2.3.1.2.Searching Objects

There are two parameters which we need to use to have a filtered list of objects in result document. Those are named as `dsb` and `src`.

`dsb` : Name of the field to be filtered against it.
`src` : The value which will be used for filtering.

Please check the following request to have a filtered object list according to “ID” field of Set Top Boxes;

```
<operation>
   <type>request</type>
</operation>
<data model="struct">
   <request>stbs</request>
   <args>
      <atr name="dsb">k2</atr>
      <atr name="src">aabbccddeeff</atr>
   </args>
</data>
```

The result set may not have a data section if there is no record found. There might be a single object in the list or a list of matched objects returned.

2.3.1.3.Actions and The Key Field

Response document of List of Objects will have a `keyfield` tag inside the `operation` block. This is another differentiation of Single Entities vs. List of Objects. This tag will be used to work on particular object in the list, so, Single Entities don't need such information.

The `keyfield` is the attribute name which will be used to determine unique value of objects in the list while doing object specific operations on same function. For example, if you want to delete particular object in the list, you need to provide the value of attribute which has the same name indicated in the `keyfield` tag.

In previous example, if the value of `keyfield` is `k2` then you need to provide the value of attribute which has `k2` as value in its name property of the object you want to delete.

A cropped version of request document for accessing full details of object “`other`” in the Video/TV category list and the cropped response of that inquiry is shown below;

```
<operation>
   <type>request</type>
   <cookies>
      <atr name="s">b75c...73ef</atr>
   </cookies>
</operation>
<data model="struct">
   <request>catmov</request>
   <args>
      <atr name="act">view</atr>
      <atr name="key">other</atr>
   </args>
</data>
```

In example above, the action is indicated as "`view`" with `act` attribute and the target object is indicated as "`other`" (which is the `keyfield` value) with `key` attribute within the `args` section of the request.

```
<operation>
   <type>response</type>
   <fieldmap>
      <atr name="k0" must="false" ro="true" type="section">Record Details</atr>
      <atr name="k1" must="true" ro="true" type="str">Record Name</atr>
      <atr name="k2" must="true" ro="true" type="str">Display Name</atr>
      <atr name="k3" must="true" ro="true" type="str">Category ID</atr>
      <atr name="k4" must="true" ro="false" type="str">Rank</atr>
   </fieldmap>
   <keyfield>k1</keyfield>
</operation>
<data model="tree">
   <elm>
      <atr n="k1">other</atr>
      <atr n="k2">Other</atr>
      <atr n="k3">24</atr>
      <atr n="k4">0</atr>
   </elm>
</data>
```

Response will be returned with different set of fields which is the full details of particular object indicated at the request.

Possible values of `act` tag used as parameter within the request document is shown below;

- `view` : accessing full details of particular object
- `edit` : fetching the list of parameters and possible option lists to modify an object
- `save` : modifying or updating an object with parameters fetched by edit
- `add` : creating an adding new object into the list
- `del` : deleting an object from the list.

Please remember that all action keys above should be used together with `keyfield` to perform actions on targeted objects.

There are also two kind of action keywords which will be explained in next chapter:

- `res` : calling an actionable resource of the object using field index
- `sres` : calling an actionable resource of the object using field index

2.3.2. Single Entities

Single Entities are predefined objects and can not be created or deleted. You can only do modifications on them or call actionable resources of its particular attributes which will be explained in next chapter.

It is very simple that if there is no `keyfield` presented within the response document and there is single record within `data` set after an inquiry, this means you are working on Single Entity.

It is valid to use actions with `act` key against Single Entities. It is also valid to use actionable-resources combining with `res` keyword. The only difference is that you don't need to provide `keyfield` with `key` tag in `data` section while doing requests since you are already working on Single Entity which is an object actually.

2.3.3. Resources

Response document may have results for fields as a value or a resource. This will be indicated with `type` flag in the `fieldmap`. All others should be treated as regular value except `res` and `sres`.

Those field types indicates that the field value is not statically recorded but it is gathered dynamically. For example the system data/time information should be dynamically gathered and it will be placed in the `data` section.

Please check below cropped response document of System Date/Time (request=timezone) inquiry as an example to see usage of resources;

```
<operation>
   <type>response</type>
   <fieldmap>
      <atr name="k1" must="false" ro="true" type="res">Date</atr>
      <atr name="k2" must="false" ro="true" type="res">Clock</atr>
   </fieldmap>
</operation>
<data model="tree">
   <elm>
      <atr n="k1">14 November 2017 Tuesday</atr>
      <atr n="k2">05:15:52</atr>
   </elm>
</data>
```

As shown in above example, information in `data` section consists of dynamic values and they are indicated as `res` with `type` flag in `fieldmap` section.

2.3.4. Actionable Resources

It may be possible to do trigger on resource for performing an action depending on its behaviour. Thus we call them as actionable resources.

In order to perform action for particular resource, its name value should be used in request conjunction with `act` and `res` keywords. To explain it with an example please check below cropped response document for System NTP Settings (request=ntp) inquiry;

```
<operation>
   <type>response</type>
   <fieldmap>
      <atr name="k4" must="false" ro="true" type="res">NTP Server</atr>
   </fieldmap>
</operation>
<data model="tree">
   <elm>
      <atr n="k4">running</atr>
   </elm>
</data>
```

As you can see in above response document, there is an attribute which has `type="res"` flag within the `fieldmap` section which is a resource. The current status of the NTP service is indicated with the `k4` attribute in the `data` section also and its value is `running`.

In order to call same function for a resource field, we should use the name of that resource field. In above example, if we want to change the current status of the NTP Service (to make it stopped), we just need to call same function against `k4` attribute and we need to send name value using `act` parameter and the value of `act` should be combining with `res` keyword like below;

```
<operation>
   <type>request</type>
</operation>
<data model="struct">
   <request>ntp</request>
   <args>
      <atr name="act">res,k4</atr>
   </args>
</data>
```

You may also provide additional parameters while calling the actionable resources. For example, for calling “Commit Changes” resource of Single Entity “`revs`”, you may provide `rstb` as `true` to restart all STBs and may provide `rsrv` as `true` to restart Media Server along with the request like below;

```
<operation>
   <type>request</type>
</operation>
<data model="struct">
   <request>revs</request>
   <args>
      <atr name="act">res,k11</atr>
      <atr name="rstb">true</atr>
      <atr name="rsrv">true</atr>
   </args>
</data>
```

3. Examples

All examples shown below are in request-response pairs and performed using Session ID which gathered by Login request. Subsequent examples will use values from previous responses. Please use your own values in your real environment.

3.1. Login

```
<cLst>
   <container>
      <operation>
         <type>request</type>
      </operation>
      <data model="struct">
         <request>login</request>
         <args>
            <atr name="username">admin</atr>
            <atr name="password">kylone</atr>
         </args>
      </data>
   </container>
</cLst>
```

```
<cLst>
   <container>
      <operation>
         <type>response</type>
         <status>ok</status>
         <cookies>
            <atr n="s">9ab4...c09d</atr>
         </cookies>
      </operation>
   </container>
</cLst>
```

3.2. List of Set Top Boxes

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">9ab4...c09d</atr>
         </cookies>
      </operation>
      <data model="struct">
         <request>stbs</request>
      </data>
   </container>
</cLst>
```

```
<cLst>
   <container>
      <operation>
         <type>response</type>
         <status>ok</status>
         <fieldmap>
            <atr nanme="k1" type="str">Model</atr>
            <atr nanme="k2" type="str">ID</atr>
            <atr nanme="k3" type="str">IP Address</atr>
            <atr nanme="k4" type="str">Name</atr>
            <atr nanme="k5" type="str">Full Name</atr>
            <atr nanme="k6" type="str">Platform</atr>
            <atr nanme="k7" type="str">Version</atr>
            <atr nanme="k8" type="str">Local</atr>
            <atr nanme="k9" type="str">Current State</atr>
            <atr nanme="k10" type="timestamp">First Seen</atr>
            <atr nanme="k11" type="timestamp">Last Seen</atr>
```

```
         </fieldmap>
         <keyfield>k2</keyfield>
      </operation>
      <data model="tree">
         <elm>
            <atr n="k1">TBS3700</atr>
            <atr n="k2">112233445566</atr>
            <atr n="k3">172.22.22.2</atr>
            <atr n="k4">Room 101</atr>
            <atr n="k5">Room 101, Building C</atr>
            <atr n="k6">Linux</atr>
            <atr n="k7">v6.1.70</atr>
            <atr n="k8">Yes</atr>
            <atr n="k9">Active</atr>
            <atr n="k10">1510404686</atr>
            <atr n="k11">1510404903</atr>
         </elm>
         <elm>
            <atr n="k1">TBS3700</atr>
            <atr n="k2">aabbccddeeff</atr>
            <atr n="k3”>172.22.22.5</atr>
            <atr n="k4">Room 102</atr>
            <atr n="k5">Room 102, Building C</atr>
            <atr n="k6">Linux</atr>
            <atr n="k7”>v6.1.54</atr>
            <atr n="k8">Yes</atr>
            <atr n="k9">Active</atr>
            <atr n="k10">1510407951</atr>
            <atr n="k11">1510409365</atr>
         </elm>
      </data>
   </container>
</cLst>
```

3.3. Full Details of the STB

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">9ab4...c09d</atr>
         </cookies>
      </operation>
      <data model="struct">
         <request>stbs</request>
         <args>
            <atr name="act">view</atr>
            <atr name="key">112233445566</atr>
         </args>
      </data>
   </container>
</cLst>

<cLst>
   <container>
      <operation>
         <type>response</type>
         <status>ok</status>
         <fieldmap>
            <atr name="k0" must="false" ro="true" type="section">Record Details</atr>
            <atr name="k1" must="true" ro="false" type="str">ID</atr>
            <atr name="k2" must="false" ro="false" type="str">Name</atr>
```

```
      <atr name="k3" must="false" ro="false" type="str">Full Name</atr>
      <atr name="k4" must="true" ro="false" type="option">Current State</atr>
      <atr name="k5" must="false" ro="false" type="longtext">Description</atr>
      <atr name="k6" must="false" ro="true" type="section">Customisations</atr>
      <atr name="k7" must="false" ro="false" type="option">GUI Language</atr>
      <atr name="k8" must="true" ro="false" type="option">Display Mode</atr>
      <atr name="k9" must="false" ro="false" type="option">AP Profile</atr>
      <atr name="k10" must="false" ro="false" type="str">SSID</atr>
      <atr name="k11" must="false" ro="false" type="str">Passphrase</atr>
      <atr name="k12" must="true" ro="false" type="option">Operating Mode</atr>
      <atr name="k13" must="false" ro="false" type="option">Channel</atr>
      <atr name="k14" must="false" ro="false" type="option">Country</atr>
      <atr name="k15" must="false" ro="true" type="section">Actions</atr>
      <atr name="k16" must="false" ro="false" type="longtext">Message</atr>
      <atr name="k17" must="false" ro="true" type="res">Update Message</atr>
      <atr name="k18" must="false" ro="true" type="res">Restart Application</atr>
      <atr name="k19" must="false" ro="true" type="res">Suspend</atr>
      <atr name="k20" must="false" ro="true" type="res">Reboot System</atr>
      <atr name="k21" must="false" ro="true" type="res">Update firmware</atr>
      <atr name="k22" must="false" ro="true" type="section">Last Update</atr>
      <atr name="k23" must="false" ro="true" type="str">IP Address</atr>
      <atr name="k24" must="false" ro="true" type="str">Version</atr>
      <atr name="k25" must="false" ro="true" type="str">Build Number</atr>
      <atr name="k26" must="false" ro="true" type="str">Operating System</atr>
      <atr name="k27" must="false" ro="true" type="str">Device ID</atr>
      <atr name="k28" must="false" ro="true" type="str">Unique ID</atr>
      <atr name="k29" must="false" ro="true" type="option">Set Top Box</atr>
      <atr name="k30" must="false" ro="true" type="option">Local Device</atr>
      <atr name="k31" must="false" ro="true" type="timestamp">First seen on</atr>
      <atr name="k32" must="false" ro="true" type="timestamp">Last seen on</atr>
      <atr name="k33" must="false" ro="true" type="res">Screenshot</atr>
   </fieldmap>
   <keyfield>k1</keyfield>
   <options>
      <opt name="k4">
         <atr val="1">Active</atr>
         <atr val="0">Disabled</atr>
      </opt>
      <opt name="k7">
         <atr val="default">System Default</atr>
         <atr val="en">English</atr>
         <atr val="ar">Arabic</atr>
         <atr val="de">German</atr>
         <atr val="fr">French</atr>
         <atr val="tr">Turkish</atr>
         <atr val="zh">Chinese</atr>
      </opt>
      <opt name="k8">
         <atr val="default">System Default</atr>
         <atr val="1080p50hz">1080p50hz (1920x1080)</atr>
         <atr val="1080i50hz">1080i50hz (1920x1080)</atr>
         <atr val="1080p60hz">1080p60hz (1920x1080)</atr>
         ...
      </opt>
      <opt name="k9">
         <atr val="opensecurity">Open Security (Kylone)</atr>
         <atr val="wpa2secure">WPA2 Secure (Kylone)</atr>
      </opt>
      <opt name="k12">
         <atr val="-">Profile Default</atr>
         <atr val="ap">Access Point</atr>
         <atr val="cl">Client</atr>
      </opt>
      <opt name="k13">
```

```
            <atr val=" "/>
            <atr val="6">Auto</atr>
            <atr val="1">1</atr>
            <atr val="2">2</atr>
            <atr val="3">3</atr>
            ...
         </opt>
         <opt name="k14">
            <atr val=" "/>
            <atr val="US">Auto</atr>
            <atr val="AD">Andorra</atr>
            <atr val="AE">United Arab Emirates</atr>
            ...
         </opt>
         <opt name="k29">
            <atr val="1">Yes</atr>
            <atr val="2">No</atr>
         </opt>
         <opt name="k30">
            <atr val="0">Yes</atr>
            <atr val="1">No</atr>
         </opt>
      </options>
   </operation>
   <data model="tree">
      <elm>
         <atr n="k1">112233445566</atr>
         <atr n="k2">Room 101</atr>
         <atr n="k3">Room 101, Building C</atr>
         <atr n="k4">1</atr>
         <atr n="k5"/>
         <atr n="k7">default</atr>
         <atr n="k8">default</atr>
         <atr n="k9"/>
         <atr n="k10"/>
         <atr n="k11"/>
         <atr n="k12">-</atr>
         <atr n="k13"></atr>
         <atr n="k14"></atr>
         <atr n="k16"/>
         <atr n="k17">resource</atr>
         <atr n="k18">resource</atr>
         <atr n="k19">resource</atr>
         <atr n="k20">resource</atr>
         <atr n="k21">resource</atr>
         <atr n="k23">172.22.22.2</atr>
         <atr n="k24">v6.1.70</atr>
         <atr n="k25">2017110522</atr>
         <atr n="k26">Linux</atr>
         <atr n="k27">TBS3700</atr>
         <atr n="k28">9344...f7ca1</atr>
         <atr n="k29">1</atr>
         <atr n="k30">0</atr>
         <atr n="k31">11 November 2017 Saturday, 14:51:26</atr>
         <atr n="k32">11 November 2017 Saturday, 14:55:03</atr>
         <atr n="k33">resource</atr>
      </elm>
   </data>
</container>
</cLst>
```

3.4. Disabling the STB

There is only request shown below. Parameters are determined from the response in Example-3.3. Response will be the same as “3.3 Full Details of an STB” shown above but with the modified values.

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">9ab4...c09d</atr>
         </cookies>
      </operation>
      <data model="struct">
         <request>stbs</request>
         <args>
            <atr name="act">save</atr>
            <atr name="key">112233445566</atr>
            <atr name="k4">0</atr>
         </args>
      </data>
   </container>
</cLst>
```

3.5. Enabling the STB

There is only request shown below. Parameters are determined from the response in Example-3.3. Response will be the same as “3.3 Full Details of an STB” above but with the modified values.

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">9ab4...c09d</atr>
         </cookies>
      </operation>
      <data model="struct">
         <request>stbs</request>
         <args>
            <atr name="act">save</atr>
            <atr name="key">112233445566</atr>
            <atr name="k4">1</atr>
         </args>
      </data>
   </container>
</cLst>
```

3.6. Modifying Message value of the STB

There is only request shown below. Parameters are determined from the response in Example-3.3. Response will be the same as “3.3 Full Details of an STB” above but with the modified values.

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">9ab4...c09d</atr>
         </cookies>
      </operation>
      <data model="struct">
```

```
         <request>stbs</request>
         <args>
            <atr name="act">save</atr>
            <atr name="key">112233445566</atr>
            <atr name="k16">Welcome to Kylone Multimedia Framework</atr>
         </args>
      </data>
   </container>
</cLst>
```

3.7. Update Message text of the STB

This is an actionable resource call which will push message to targeted online STB. If the STB is in offline state then there is no need to perform this action since the STB will get message automatically when it gets online. Parameters are determined from the response in Example-3.3.

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">9ab4...c09d</atr>
         </cookies>
      </operation>
      <data model="struct">
         <request>stbs</request>
         <args>
            <atr name="act">res,k17</atr>
            <atr name="key">112233445566</atr>
         </args>
      </data>
   </container>
</cLst>
```

```
<cLst>
   <container>
      <operation>
         <type>response</type>
         <status>ok</status>
      </operation>
   </container>
</cLst>
```

3.8. Update Firmware of the STB

This is an actionable resource call which will force targeted online STB to update its firmware. Parameters are determined from the response in Example-3.3.

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">9ab4...c09d</atr>
         </cookies>
      </operation>
      <data model="struct">
         <request>stbs</request>
         <args>
            <atr name="act">res,k21</atr>
            <atr name="key">112233445566</atr>
```



```
         </args>
      </data>
   </container>
</cLst>

<cLst>
   <container>
      <operation>
         <type>response</type>
         <status>ok</status>
      </operation>
   </container>
</cLst>
```

3.9. Committing Changes

In order to perform such action it is needed to know which actionable resource we need to use. If you already know and you are sure about the version of the software is not changed, you can keep using the field ID you discovered before.

As an example here is the request-response pair to get full details against “`revs`” function to determine `key` for performing “Commit Changes”;

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">9ab4...c09d</atr>
         </cookies>
      </operation>
      <data model="struct">
         <request>revs</request>
      </data>
   </container>
</cLst>

<cLst>
   <container>
      <operation>
         <type>response</type>
         <status>ok</status>
         <fieldmap>
            <atr name="k0" must="false" ro="true" type="section">Configuration</atr>
            <atr name="k1" must="true" ro="false" type="option">GUI Language</atr>
            <atr name="k2" must="true" ro="false" type="str">Share Password</atr>
            <atr name="k3" must="false" ro="true" type="section">Visuals</atr>
            <atr name="k4" must="false" ro="false" type="option">Main Menu Type</atr>
            <atr name="k5" must="false" ro="false" type="longtext">Banner Text</atr>
            <atr name="k6" must="false" ro="false" type="image">Logo</atr>
            <atr name="k7" must="false" ro="false" type="image">Background</atr>
            <atr name="k8" must="false" ro="false" type="str">Transparency</atr>
            <atr name="k9" must="false" ro="true" type="section">Update Information</atr>
            <atr name="k10" must="true" ro="true" type="timestamp">Last Commit</atr>
            <atr name="k11" must="false" ro="true" type="res">Commit Changes</atr>
         </fieldmap>
         <options>
            <opt name="k1">
               <atr val="en">English</atr>
               <atr val="de">German</atr>
               <atr val="fr">French</atr>
```

```
            <atr val="ru">Russian</atr>
            <atr val="zh">Chinese</atr>
            <atr val="th">Thai</atr>
            <atr val="tr">Turkish</atr>
            <atr val="ar">Arabic</atr>
         </opt>
         <opt name="k4">
            <atr val="def">Classic</atr>
            <atr val="pre">Facility</atr>
            <atr val="csr">CStyle Red</atr>
         </opt>
      </options>
   </operation>
   <data model="tree">
      <elm>
         <atr n="k1">en</atr>
         <atr n="k2">kylone</atr>
         <atr n="k4">def</atr>
         <atr n="k5"/>
         <atr n="k6">...AAElFTkSuQ</atr>
         <atr n="k7">...zhvPnydoku</atr>
         <atr n="k8">0</atr>
         <atr n="k10">14 November 2017 Tuesday, 05:59:00</atr>
         <atr n="k11">resource</atr>
      </elm>
   </data>
</container>
</cLst>
```

From the response, we will grab `key` value as `k11` as you can see above. We will call same function with value `k11` by pairing `res` keyword.

You may also provide additional parameters while calling the function which are;

`rstb` : To restart all STBs if the value passed as “`true`”
`rsrv` : To restart Media Server if the value passed as “`true`”

Here is the request-response pair to perform action while restarting Media Server and all online STBs;

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">9ab4...c09d</atr>
         </cookies>
      </operation>
      <data model="struct">
         <request>revs</request>
         <args>
            <atr name="act">res,k11</atr>
            <atr name="rstb">true</atr>
            <atr name="rsrv">true</atr>
         </args>
      </data>
   </container>
</cLst>

<cLst>
   <container>
      <operation>
```

```
         <type>response</type>
         <status>ok</status>
      </operation>
   </container>
</cLst>
```

3.10.List of Available Functions

It is possible to fetch all available functions defined in the system. You can call “`apicalls`” request to have a list of them as shown below;

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">9ab4...c09d</atr>
         </cookies>
      </operation>
      <data model="struct">
         <request>apicalls</request>
      </data>
   </container>
</cLst>

<cLst>
   <container>
      <operation>
         <type>response</type>
         <status>ok</status>
      </operation>
      <data model="tree">
         <elm>
            <atr n="login">Sign in</atr>
            <atr n="logout">Sign out</atr>
            ...
         </elm>
      </data>
   </container>
</cLst>
```

3.11.Logout

It is not necessary technically but it is required for security to do logout after doing calls to the system as shown below;

```
<cLst>
   <container>
      <operation>
         <type>request</type>
         <cookies>
            <atr name="s">9ab4...c09d</atr>
         </cookies>
      </operation>
      <data model="struct">
         <request>logout</request>
      </data>
   </container>
</cLst>

<cLst>
```

```
   <container>
      <operation>
         <type>response</type>
         <status>ok</status>
      </operation>
   </container>
</cLst>
```

After performing logout, the session ID will not be valid anymore.

4. Code Examples

4.1. PHP Code Example

An example code written in PHP is given below. There are two required modules which are php-xml and php-curl and they should be installed on your system in order to use the code.

You can save below code snippets to file as “apicall.php” and can run with below parameters to check if your environment working or not;

```
php ./apicall.php ip_address username password cpustat
```

Please cut starting from next line to have php code.

```
<?php
/*
 * Example PHP application to make calls on
 * Kylone MicroCMS XML API, v2.0.64
 * Revision 14 November, 2017
 */

// posts data with cURL and get XML document as response
function doc_post($url, $doc)
{
   $ct = curl_init();
   curl_setopt($ct, CURLOPT_RETURNTRANSFER, TRUE);
   curl_setopt($ct, CURLOPT_FOLLOWLOCATION, TRUE);
   curl_setopt($ct, CURLOPT_AUTOREFERER, TRUE);
   curl_setopt($ct, CURLOPT_CONNECTTIMEOUT, 30);
   curl_setopt($ct, CURLOPT_TIMEOUT, 20);
   curl_setopt($ct, CURLOPT_MAXREDIRS, 2);
   curl_setopt($ct, CURLOPT_URL, $url);
   curl_setopt($ct, CURLOPT_POST, 1);
   curl_setopt($ct, CURLOPT_SAFE_UPLOAD, false);
   curl_setopt($ct, CURLOPT_POSTFIELDS, array("xml" => $doc));
   curl_setopt($ct, CURLOPT_USERAGENT, "API Client, MicroCMS-XML-API/v2.0.64”);
   curl_setopt($ct, CURLOPT_HTTPHEADER, array('Content-Encoding: UTF-8'));
   curl_setopt($ct, CURLOPT_ENCODING , "gzip");
   $res = curl_exec($ct);
   curl_close($ct);
   return $res;
}

// converts 'act=val&key=val'
// to '<args><atr name="act">val</atr><atr name="key">val</atr></args>'
function arg_construct($argstr)
{
   $l = "";
   $v = explode("&", str_replace(array("\\", "0x"), "", $argstr), 20);
   $c = count($v);
   for ($i = 0; $i < $c; $i++) {
      if ($v[$i] != "") {
        $x = explode("=", $v[$i], 2);
        $l .= '<atr name="'.htmlspecialchars($x[0], ENT_QUOTES).'">';
```

```
        $l .= htmlspecialchars($x[1], ENT_NOQUOTES);
        $l .= '</atr>';
      }
   }
   return '<args>'.$l.'</args>';
}

// creates XML document with target function name, parameters list
// and with session ID if given
function doc_construct($fname, $argxml, $ssnid = "", $xfile = false)
{
   $s = ($ssnid != "") ? '<atr name="s">'.$ssnid.'</atr>' : '';
   $x = '<?xml version="1.0"'.'?'.'>';
   $x .= '
<cLst>
  <container>
    <operation>
       <type>request</type>
       <cookies>'.$s.'</cookies>
    </operation>
    <data model="struct">
      <request>'.htmlspecialchars($fname, ENT_QUOTES).'</request>
      '.arg_construct($argxml).'
    </data>
  </container>
</cLst>
';
   if ($xfile !== false)
      file_put_contents($xfile."_".$fname."_request.xml", $x);
   return $x;
}

// performs inquiry and returns response as it is (text XML document)
function do_query_and_get_doc($url, $doc)
{
   return doc_post($url, $doc);
}

// performs inquiry and returns response as php-object
// after doing some sanitiy checks
function do_query_and_get_obj($url, $doc, $xfile)
{
   $resp = doc_post($url, $doc);
   if ($xfile !== false)
      file_put_contents($xfile."_login_response.xml", $resp);
   $flags = LIBXML_COMPACT | LIBXML_NOBLANKS | LIBXML_NOCDATA | LIBXML_NOEMPTYTAG;
   $flags |= LIBXML_NONET | LIBXML_PEDANTIC | LIBXML_PARSEHUGE;
   $xobj = simplexml_load_string($resp, "SimpleXMLElement", $flags);
   if (!isset($xobj->container->operation->type))
      return false;
   if ((string)$xobj->container->operation->type != "response")
      return false;
   if (!isset($xobj->container->operation->status))
      return false;
   if ((string)$xobj->container->operation->status != "ok")
      return false;
   return $xobj;
}

// creates login document and gets sessinid with inqury
function do_login($url, $uname, $pass, $xfile)
{
   $logindoc = doc_construct("login", "username=".$uname."&password=".$pass, "", $xfile);
   $response = do_query_and_get_obj($url, $logindoc, $xfile);
   if ($response === false)
      return false;
   if (!isset($response->container->operation->cookies))
      return false;
   $cvals = false;
   foreach ($response->container->operation->cookies->children() as $node) {
      $n = $node['n'];
```

```
      $cvals["$n"] = (string)$node;
   }
   return (isset($cvals["s"]) ? $cvals["s"] : false);
}

// performs login, apicall and logout
function do_apicall($url, $uname, $pass, $fname, $argstr, $xfile)
{
// performs login and gets sessinid if possible
   $ssnid = do_login($url, $uname, $pass, $xfile);
   if ($ssnid === false)
      return false;

// performs apicall for target function with parametes and sessionid
   $calldoc = doc_construct($fname, $argstr, $ssnid, $xfile);
   $callres = do_query_and_get_doc($url, $calldoc);
   if ($xfile !== false)
      file_put_contents($xfile."_".$fname."_response.xml", $callres);

// performs logout without considering the previous result
   $logoutdoc = doc_construct("logout", "", $ssnid, $xfile);
   $logoutres= do_query_and_get_doc($url, $logoutdoc);
   if ($xfile !== false)
      file_put_contents($xfile."_logout_response.xml", $logoutres);

// returns request and response document in array for the target function
   return array($calldoc, $callres);
}

if (!isset($argv[4])) {
   echo "Usage: ".$argv[0];
   echo " <host> <username> <password> <function> <argstring> [export_name]";
   echo "\n";
   echo "php ".$argv[0];
   echo " 10.47.48.1 admin kylone cpustat \"arg1=val1&arg2=val2\" cpustat_log";
   echo "\n\n";
   exit();
}

$v = do_apicall(
        "http://".$argv[1]."/portal/",      // URL
        $argv[2],                           // Username
        $argv[3],                           // Password
        $argv[4],                           // Function name
        (isset($argv[5]) ? $argv[5]: ""),   // Parameters String
        (isset($argv[6]) ? $argv[6]: false) // export each doucments to file
     );

if (isset($argv[6])) {
   echo "All requests and responses are exported to ".$argv[6]."_*.xml\n";
} else {
   echo "Request:\n".$v[0]."\nResponse:\n".$v[1]."\n";
}

?>
```

Please cut until to previous line.

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/