

API settings and best practice for API connection with Plexus CRP 6.x system

Contents

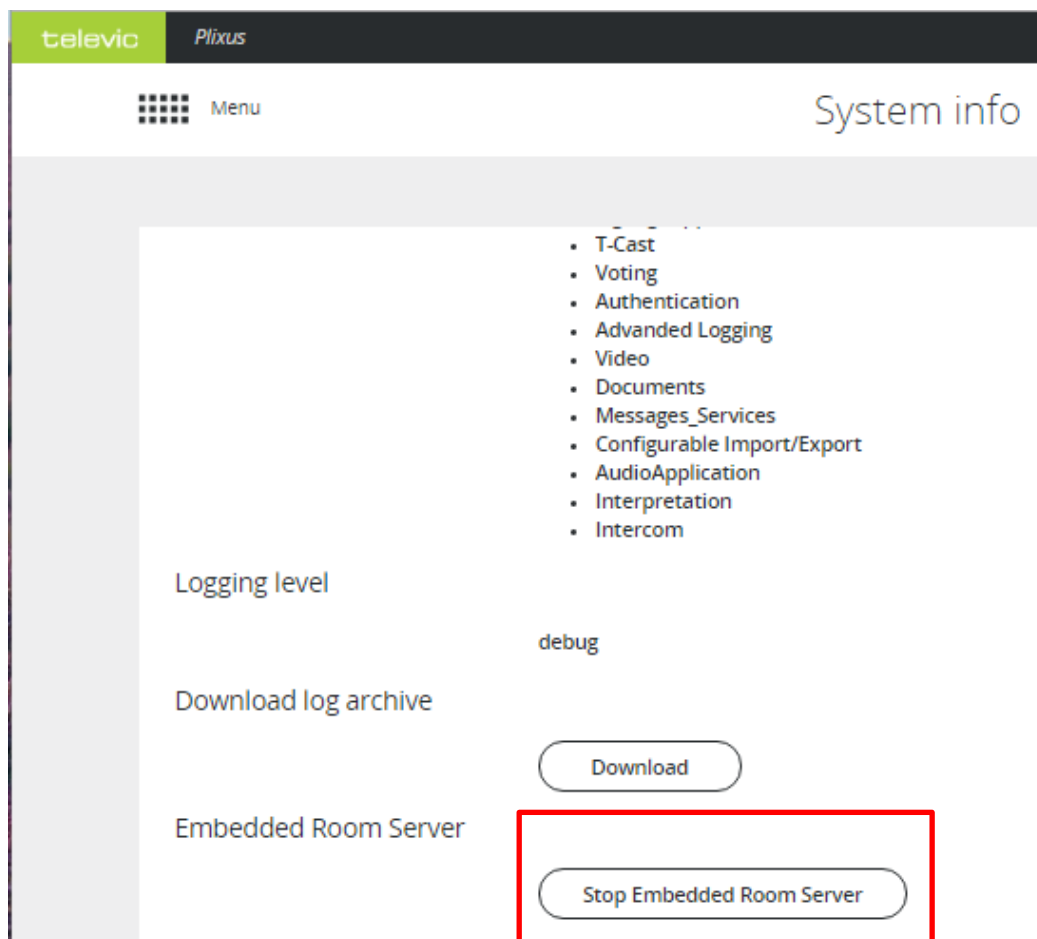
1.	What settings to be used for API ?.....	3
1.1.	Plixus Engine in ERS/CORE mode.....	3
1.2.	Plixus Engine in Non-ERS/CORE mode	5
2.	API compatibility	5
3.	Best practice for use of API	6
3.1.	Setting up and operating the CoCon API connection	6
3.1.1.	General	6
3.1.2.	Wireshark log screenshot of API connection	7
3.2.	Advised way of working.....	8
3.3.	To be avoided.....	9

1. What settings to be used for API ?

1.1. Plixus Engine in ERS/CORE mode

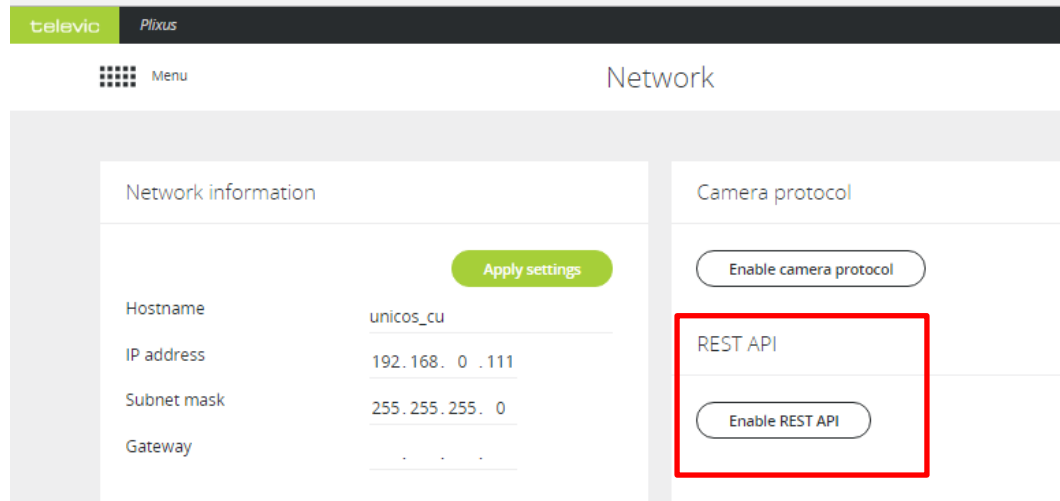
E(mbedded) R(oom) S(erver) (=CORE) can be enabled/disabled via the Plixus Engine webserver > system Info

In ERS/CORE mode , API possibilities are limited , see Cocon API document for more detailed info



The screenshot shows the 'System info' page of the Plixus Engine webserver. The page has a dark header with the 'televic' logo and 'Plixus' text. Below the header is a 'Menu' icon and the title 'System info'. The main content area is divided into two columns. The right column contains a list of services: T-Cast, Voting, Authentication, Advanced Logging, Video, Documents, Messages_Services, Configurable Import/Export, AudioApplication, Interpretation, and Intercom. The left column contains 'Logging level' and 'Download log archive'. Below 'Download log archive' is a 'Download' button. Below 'Embedded Room Server' is a 'Stop Embedded Room Server' button, which is highlighted with a red box.

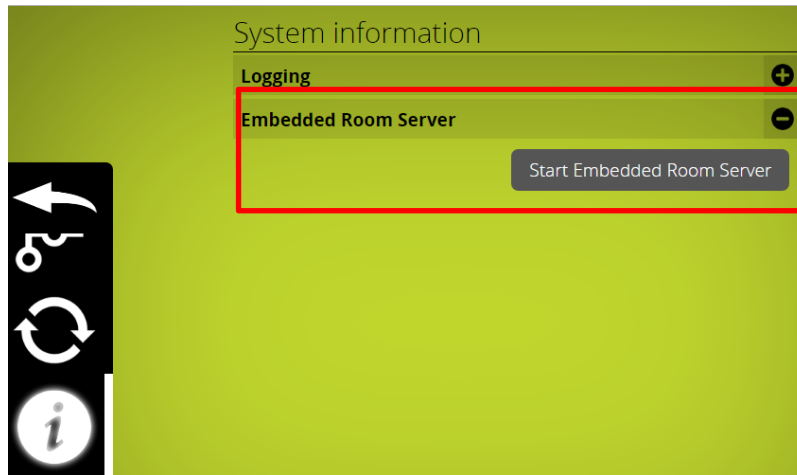
When using API , REST API needs to be enabled on the Plixus webserver and port 8890 is to be used



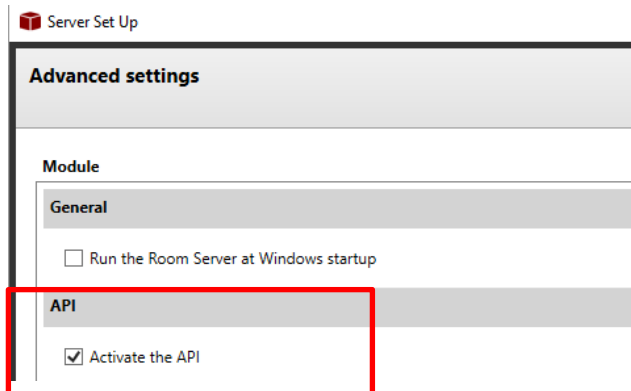
In ERS/CORE mode Cocon may be used for additional features (e.g. use of operator application) , but then use Cocon Core must be used

1.2. Plixus Engine in Non-ERS/CORE mode

E(mbedded) R(oom) S(erver)/CORE can be enabled/disabled via the Plixus Engine webservice
> system Info



In that case API connection needs to be made to the Cocon server ip address – port 8990
, API needs to be enabled in the Cocon server configuration wizard as well



2. API compatibility

The API document indicates what commands are compatible with Coon for Plixus (= Cocon classic) ,
Cocon CORE and directly to Plixus engine

1.3 API module compatibility

This table shows which API modules are available in the different systems.

API Module	CoCon for Plixus	Cocon for Plixus Core	Plixus Core
<u>Server to Client</u>			
3.2.1 Room			
3.2.1.1 InitializationState	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3.2.1.2 UnitsAdded	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3.2.1.3 UnitsChanged	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3.2.1.4 UnitStateChanged	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3.2.1.5 DefaultVolumeChangedForRoom	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3.2.1.6 CurrentVolumeChangedForRoom	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3.2.1.7 DefaultMicrophoneModeUpdate	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3.2.1.8 VolumeChangedForRoom	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
3.2.1.9 UnitError	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
.....			
.....			

3. Best practice for use of API

3.1. Setting up and operating the CoCon API connection

3.1.1. General

The CoCon API connection consists of a number of URI's (Uniform Resource Identifier) which are exchanged between the API client and the CoCon Server over a TCP/IP-connection.

Basically, the CoCon API uses two (or more) parallel HTTP-connections. These are the following:

- A "notification" connection that the client and Server keep open in case any information need to be transferred from the Server to the client.
- Any other connection that is used to transfer information from the client to the Server.

The notification connection should be initialized by the client, and is kept open by both sides. Then one of two things can happen:

- The Server has information to send to the client and uses the notification connection for this. The information is transferred and the connection is closed (as per the HTTP standard).

After this, the client **needs to create a new notification connection**. The Server will guarantee that no information is lost during this short period where the client has no connection to the Server.

Note that the new notification connection needs to be created within 30 seconds. If this does not happen, the Server will discard the client data.

- The HTTP-connection times out before the Server has any information to send to the client. At this point, the client **needs to create a new notification connection**. Again, the Server will guarantee that no information is lost.

3.1.2. Wireshark log screenshot of API connection

In below screenshot an example of communication between API client and Cocon/ERS server is shown : this illustrates the following

- 1) Setup connection between server and client ; a connection ID is returned
- 2) Client sends a notification request with this connection ID to request information
- 3) After some time , If there is no information to be send , server will return "request time out" , which has to be replied immediately by a new notification request
- 4)
- 5) Each time information was send by the server (e.g. mic button event) this has to be followed by a new notification request by the client

+	332	3.129278	192.168.0.108	192.168.0.249	HTTP	111 GET /CoCon/Connect HTTP/1.1
	337	3.179482	192.168.0.249	192.168.0.108	TCP	60 8890 → 61487 [ACK] Seq=1 Ack=58 Win=256 Len=0
-	5006	43.851748	192.168.0.249	192.168.0.108	HTTP	173 HTTP/1.1 408 Request Timeout
	5008	43.863568	192.168.0.249	192.168.0.108	HTTP	276 HTTP/1.1 200 OK (application/json)
	5009	43.863812	192.168.0.108	192.168.0.249	TCP	54 61487 → 8890 [ACK] Seq=58 Ack=342 Win=254 Len=0
	5010	43.864428	192.168.0.108	192.168.0.249	HTTP	156 GET /CoCon/Notification/id=c8e399ce-a509-495f-b16a-b5067461e616 HTTP/1.1
	5015	43.890506	192.168.0.249	192.168.0.108	HTTP	173 HTTP/1.1 408 Request Timeout
	5016	43.891204	192.168.0.108	192.168.0.249	HTTP	156 GET /CoCon/Notification/id=c8e399ce-a509-495f-b16a-b5067461e616 HTTP/1.1
	5017	43.916006	192.168.0.249	192.168.0.108	TCP	60 8890 → 61487 [ACK] Seq=342 Ack=160 Win=255 Len=0
	5020	43.941920	192.168.0.249	192.168.0.108	TCP	60 8890 → 61656 [ACK] Seq=120 Ack=103 Win=251 Len=0
	12458	103.889076	192.168.0.249	192.168.0.108	HTTP	173 HTTP/1.1 408 Request Timeout
	12459	103.889873	192.168.0.108	192.168.0.249	HTTP	156 GET /CoCon/Notification/id=c8e399ce-a509-495f-b16a-b5067461e616 HTTP/1.1
	12461	103.901198	192.168.0.249	192.168.0.108	HTTP	173 HTTP/1.1 408 Request Timeout
	12462	103.901653	192.168.0.108	192.168.0.249	HTTP	156 GET /CoCon/Notification/id=c8e399ce-a509-495f-b16a-b5067461e616 HTTP/1.1
	12467	103.940580	192.168.0.249	192.168.0.108	TCP	60 8890 → 61487 [ACK] Seq=461 Ack=262 Win=255 Len=0
	12468	103.952522	192.168.0.249	192.168.0.108	TCP	60 8890 → 61656 [ACK] Seq=239 Ack=205 Win=256 Len=0
	13323	111.461271	192.168.0.249	192.168.0.108	HTTP	264 HTTP/1.1 200 OK (application/json)
	13324	111.461992	192.168.0.108	192.168.0.249	HTTP	156 GET /CoCon/Notification/id=c8e399ce-a509-495f-b16a-b5067461e616 HTTP/1.1
	13331	111.504840	192.168.0.249	192.168.0.108	HTTP	266 HTTP/1.1 200 OK (application/json)
	13332	111.505533	192.168.0.108	192.168.0.249	HTTP	156 GET /CoCon/Notification/id=c8e399ce-a509-495f-b16a-b5067461e616 HTTP/1.1
	13333	111.506350	192.168.0.249	192.168.0.108	HTTP	557 HTTP/1.1 200 OK (application/json)
	13334	111.506775	192.168.0.108	192.168.0.249	HTTP	156 GET /CoCon/Notification/id=c8e399ce-a509-495f-b16a-b5067461e616 HTTP/1.1
	13343	111.561223	192.168.0.249	192.168.0.108	TCP	60 8890 → 61656 [ACK] Seq=1164 Ack=511 Win=255 Len=0
	19379	163.911423	192.168.0.249	192.168.0.108	HTTP	173 HTTP/1.1 408 Request Timeout
	19380	163.912200	192.168.0.108	192.168.0.249	HTTP	156 GET /CoCon/Notification/id=c8e399ce-a509-495f-b16a-b5067461e616 HTTP/1.1
-	19391	163.962781	192.168.0.249	192.168.0.108	TCP	60 8890 → 61487 [ACK] Seq=580 Ack=364 Win=254 Len=0

3.2. Advised way of working

In order to optimize the performance of the external developed REST API client application we propose the following:

- Make sure that only a single connect is done.
- The id returned by the connect call must be used to make a quickly as possible a notification call. This notification call can:
 - Return immediately if the id is invalid (HTTP 400) → In this case a new connection should be opened. This can e.g. take place if Plixus CU is restarted.
 - Return after a period of time (30sec) when no message is send (HTTP 408). → In this case a new notification call should be opened as quick as possible.
 - Return a message before time-out → In this case the message should be send/processed and a new notification call should be openend as quick as possible.
- It is strongly advised that for quick handling of messages, there is a separate thread for handling notifications. It should be avoided that there are gaps in open notification calls
- In case REST API client program also involved business logic and/or UI, this is best placed in separate thread with quick communication from the notification thread.

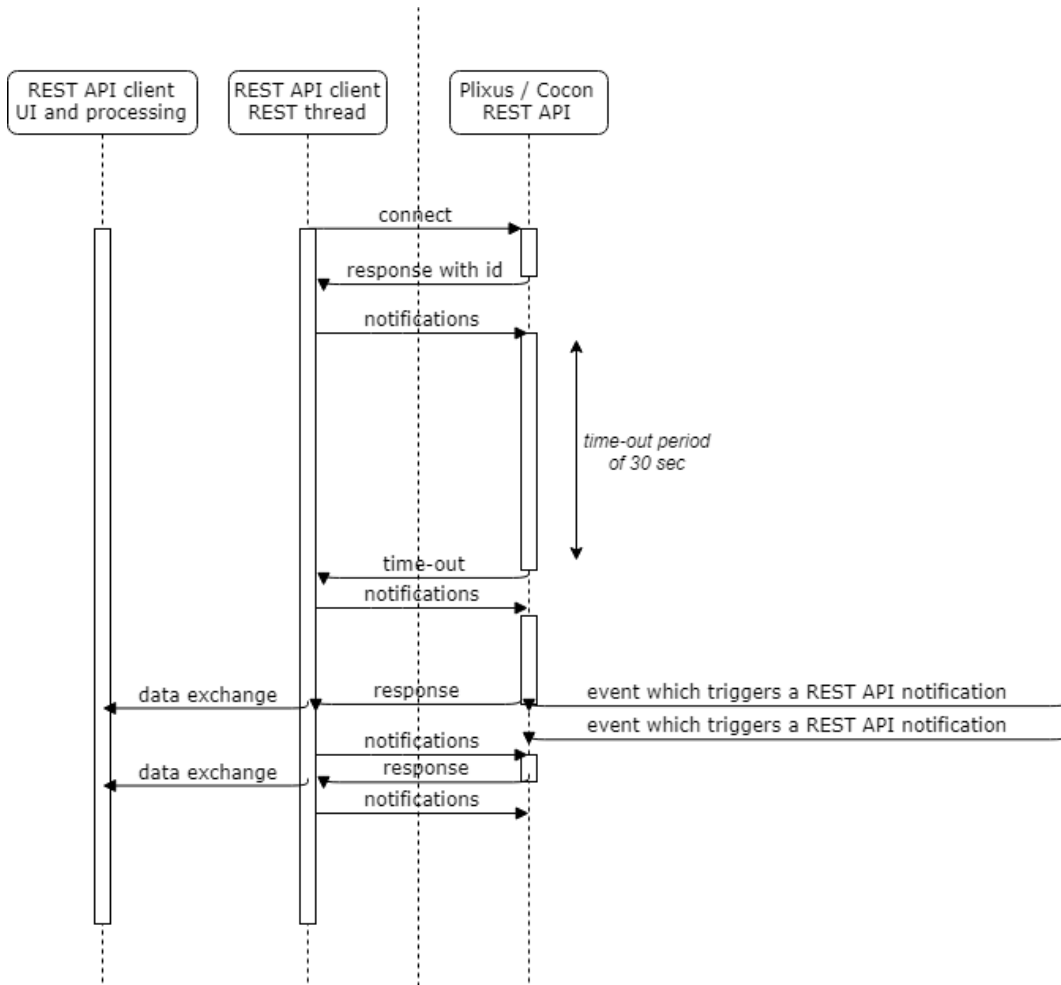
The above way of working will guarantee quick and proper handling of REST API notfications.

A schematic representation is given in the picture on next page.

Important remark:

When opening a connection, by default notification from ALL modules will be received. One can unsubscribe to specific modules. The advantages of unsubscribing to modules for which the notification are not needed/desired are the following:

- **Plixus CU will not be processing and sending out unwanted messages**
- **Client API program will not spend any time receiving and processing packets which are of no interest for the application.**



3.3. To be avoided

Common mistake is that with very short intervals , new connections are established by the API client : this creates **lots of unnecessary** traffic , which is to be avoided. So through notification requests , the ongoing connection needs to be kept open.