
ckSCADA Documentation

Release 0.1a

ckSCADA

Aug 28, 2020

OVERVIEW:

1	Admin Client	1
2	Server Overview	3
2.1	ckAgent	3
2.2	ckAgentDevice	3
2.3	config.json	4
3	Viewer Overview	5
4	Client Installation	7
4.1	Viewer Only	7
5	Server Installation	9
6	Command Line Tools	11
6.1	Create IO Device	11
7	Admin-Client	13
8	Admin-Server	15
9	Server	17
10	Todo list	19
11	Glossary	21
11.1	I/O Server	21
11.2	Kafka broker	21
11.3	IO Device	21
12	Indices and tables	23
	Index	25

ADMIN CLIENT

The admin client is a web based interface for you to modify/view system configuration. This includes:-

- IO Device configuration
- Point/Object/Class Configuration
- Kafka Monitoring

SERVER OVERVIEW

The server package provides the basic functionality of getting data from the IO devices into Kafka, monitoring the IO device processes and providing an api for the admin client to query the configuration.

2.1 ckAgent

The ckAgent process is the main process to be run on each of the *I/O Server*. It creates the *IO Device* processes as they are requested. Once the IO device process is created then it is left to the IO device to handle redundancy, io polling, enabling/disabling of the device.

Todo: ckAgent should be able to open the configuration file and load the server config

Todo: ckAgent should be able to poll other online ckAgent process to get a copy of the configuration.

The ckagent process can be run using the command

```
python3 ckagent.py --configfile ../../config/'config.json'
```

This will create the ckAgent process and have it wait for messages.

2.2 ckAgentDevice

Currently there is only the simulation IO device type available with the plan to include an OPC DA, OPC UA as well as a Modbus device as soon as the base logic has been tested and has proven itself.

Todo: This is by design to limit the out of the box usefulness until it is safer to be used in production

When an IO device is created you are able to specify the replication factor for that device. The ckAgent process creates the IO device and monitors the others servers for similar online devices. The replication factor defines how many active instances of this IO device should be available.

If there is less instances available then what has been configured a new IO device will come online if available.

The main interface to modify IO devices is through the *Admin Client* however there are also *Command Line Tools* available to programatically modify them.

2.3 config.json

The config.json file is used to store the projects configuration. It can be stored anywhere. An example is included in the ckscada-client/config and ckscada-servers/config folders.

The file is written in JSON. The following structures can be defined in the file.

brokers - contains a dictionary of Kafka brokers. This is to bootstrap the connection between the client or server and the Kafka cluster. Once a connection has been established a list of available brokers in the cluster is downloaded.

The **nodeId** is the id assigned to the node within the Kafka Configuration

The **host** is the ip address or server name. When using a server name make sure it is configured in your hosts file so you don't rely on a DNS.

The **port** is the port that is configured within the Kafka Configuration on the server.

```
"brokers": [  
  {  
    "nodeId": <nodeid>,  
    "host": "<ip address>",  
    "port": <port>  
  }  
]
```

Todo: Add additional configuration to the config file to be used as the base configuration when starting the system. This file should be able to be exported from a running Kafka System via the Admin Client.

VIEWER OVERVIEW

ckViewer is the operators interface to the running ckSCADA system. It is based around an embedded web browser. This allows for the flexibility of using open web standards to display information to the operator such as HTML5 and SVG. It also allows animation frameworks to be used such as Anime.

During development inkscape has been used to develop SVG graphics. This seems to work well as it provides a professional drawing package which is very flexible. There are some attributes and elements such as the **cktag**, **ckparameter** and **ckscript** classes that are required to be added manually so that ckSCADA points can be shown on the screen. This can be done within the inkscape xml editor window.

When a page is loaded the viewer will first search for any **<image>** elements which also have a class attribute of **ckobject**. It will then use **name** attribute and replace all references of **\$Object** within a copy of the linked file with the object in the **name** attribute. This copy then gets embedded within the page so the external file doesn't need to be read each time the tag is updated.

```
<image width="50px" height="50px" x="100px" y="100px" class="ckobject" xlink:href=
↳ "svg/Motor.svg" object="Simulated.Flow.Transmitter"/>
```

It will then search for elements with the **cktag** attribute and create a list of tags that are used on the screen. It will then subscribe to the Kafka topics for each tag.

```
<text class="cktag" name="Simulated.Flow.Transmitter.Scaled!eu" style="fill: 'black';
↳ font-family: &quot;Roboto Slab&quot;; font-size: 12px; white-space: pre;" x="420" y=
↳ "300"><title>Test</title>####</text>
```

When a message is received on one of these topics, the elements associated with that topic are updated. When using the **cktag** class attribute the text content of that element is updated with the new value.

When specifying the **cktag** class the **name** attribute specifies the tag. The tag is a combination of the Kafka topic name and the attribute from received message, such as **Simulated.Flow.Transmitter.Raw!value** when **Simulated.Flow.Transmitter.Raw** is the topic and **value** is the attribute. This allows for several attributes to be sent within the same message.

The **ckscript** element can also be used to include script behaviour which is run when the included **ckparameter** topics are updated.

Todo: Add the ability to include multiple **ckparameter** element per script.

```
<text class="cktag" name="Simulated.Flow.Transmitter.Stored!value" style="fill: 'black'
↳ "; font-family: &quot;Roboto Slab&quot;; font-size: 12px; white-space: pre;" x="500
↳ y="300">
  <title>Test</title>
  <g class="ckparameter">Simulated.Flow.Transmitter.Raw!value</g>
```

(continues on next page)

(continued from previous page)

```
<g class="ckscript">if (this.v > 50) {  
    this.e.style.fill = "red";  
} else {  
    this.e.style.fill = "green";  
};  
</g>  
####  
</text>
```

This allows for updating specific attributes of the element when tags are updated such as background colour, position, etc..

As these scripts are run on every update of the tag these scripts should be kept as short as possible with the tag update rate being taken into account. If the tag is updated every 10ms then it doesn't leave much time for the script to run. If it is updated every 1-2 seconds the script can be made longer.

CLIENT INSTALLATION

4.1 Viewer Only

To install the viewer download the Node.js package from nodejs.org for your operating system.

For Windows you will need to install node.js manually as well as python 3. You can then install kafka-python with pip. Then manually install node.js dependencies using

```
cd ckscada-client  
npm install .
```

For Debian/Ubuntu use apt to install it

```
sudo apt install npm  
sudo npm install -g npm-cache
```

Following this, download the ckscada package and build the npm packages.

```
cd ckSCADA  
make
```

Edit the config.json file in the config folder. Include the nodeId, ip address and the port of one of your Kafka brokers.

Assuming you have already setup your server you should then be able to run the viewer.

```
cd ckscada-client  
npm start
```

This will open the welcome.svg screen and then open the Sample-Screen.svg file. This is just an example page using built-in tags from the standard server install. For production you will need to create your own HMI pages.

SERVER INSTALLATION

To install the server components.

Firstly make sure your Kafka broker/clustr has been setup already. The server components should be able to be run on the Kafka broker.

For Windows you will need to install node.js manually as well as python 3. You can then install kafka-python with pip and manually install node.js dependancies using

```
npm install .
```

in the ckscada-client and ckscada-server/admin-client folders.

For Debian/Ubuntu use apt to install it

```
sudo apt install npm python3
sudo npm install -g npm-cache
pip3 install kafka-python
```

Following this, download the ckscada package and build the npm packages.

```
cd ckSCADA
make
```

Edit the config.json file in the config folder. Include the nodeId, ip address and the port of one of your Kafka brokers.

We will next run the server components.

```
cd ckscada-server/server/src
python3 ckagent.py --config ../../config/config.json

cd ckscada-server/admin-server/src
python3 points.py --config ../../config/test.json

cd ckscada-server/admin-client
npm start .
```

There is a helper script to setup a few tags on simulation device and start publishing them the sample page on the client uses these.

```
cd ckscada-server/server/src
./simulate_plant.sh
```


COMMAND LINE TOOLS

6.1 Create IO Device

An I/O server is the machine that hosts the main ckAgent process. Only one ckAgent process can be run per server.

Note: IO servers can be any server and need not be the same server the kafka broker is on.

ADMIN-CLIENT

getTopicList (*topic, filter, callback, setProgress*)

Get list from server using a string filter. It also updates the status bar.

class App (*props*)

Main React Class

Main constructor

App.addCopyEventListener ()

Listens for the browser copy command triggered by either Ctrl+C or from the browser menu.

Copies the current filtered list to the clipboard.

App.convertListtoTabDelimited (*list*)

Convert a json list to a tab delimited string.

Converts flat json file to a tab delimited string.

Arguments

- **list** – json string

App.displayPage (*topic*)

Shows a specific page within the main area

This sets the show state within the List React Component. This then doesn't render the page.

Arguments

- **topic** (*string*) – name of the page to display e.g. topics, points, devices.

App.getCurrentPage ()

Gets the reference to the currently shown page.

Checks the show state of each page and returns the reference to the List Component.

Returns ObjectListComponent –

App.queryFilteredList (*ref*)

Requests an update of the currently displayed list from the servers.

A request is sent to the web server which then broadcasts a request to update the list. This is then passed back and stored within an object list.

Arguments

- **ref** (*event*) – reference to the event that triggered the update. Expecting an on key press event.

ADMIN-SERVER

ckadmincommon.**log**(s)

SERVER

The server package provides the basic functionality of getting data from the IO devices, monitoring the IO device processes and providing an api for the admin server to query the configuration.

It initially opens a Kafka consumer process and waits for configuration messages on the default admin topic ‘*_admin*’.

The ckAgent listens for messages on the ‘*_admin*’ topic. The following messages will be acted upon:-

- add - This adds an IO device.
- del - This deletes an IO device.
- get - This gets the agents configuration and publishes it on the same topic.

TODO LIST

Todo: ckAgent should be able to open the configuration file and load the server config

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/ckscada/checkouts/latest/docs/source/about/server.rs line 18.)

Todo: ckAgent should be able to poll other online ckAgent process to get a copy of the configuration.

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/ckscada/checkouts/latest/docs/source/about/server.rs line 22.)

Todo: This is by design to limit the out of the box usefulness until it is safer to be used in production

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/ckscada/checkouts/latest/docs/source/about/server.rs line 41.)

Todo: Add additional configuration to the config file to be used as the base configuration when starting the system. This file should be able to be exported from a running Kafka System via the Admin Client.

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/ckscada/checkouts/latest/docs/source/about/server.rs line 91.)

Todo: Add the ability to include multiple ckparameter element per script.

(The [original entry](#) is located in /home/docs/checkouts/readthedocs.org/user_builds/ckscada/checkouts/latest/docs/source/about/viewer.rs line 48.)

GLOSSARY

11.1 I/O Server

An I/O server is the machine that hosts the main ckAgent process. Only one ckAgent process can be run per server.

Note: IO servers can be any server and need not be the same server the kafka broker is on.

11.2 Kafka broker

A Kafka broker is a machine running Apache Kafka. This may be part of a larger Kafka cluster or be a standalone broker.

11.3 IO Device

An IO device is an abstract class that reads data from a physical or simulated device and sends this data to Kafka topics. The implementation left to then specific IO device.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

INDEX

A

`App()` (*class*), [13](#)
`App.addCopyEventListener()` (*App method*), [13](#)
`App.convertListtoTabDelimited()` (*App method*), [13](#)
`App.displayPage()` (*App method*), [13](#)
`App.getCurrentPage()` (*App method*), [13](#)
`App.queryFilteredList()` (*App method*), [13](#)

G

`getTopicList()` (*built-in function*), [13](#)