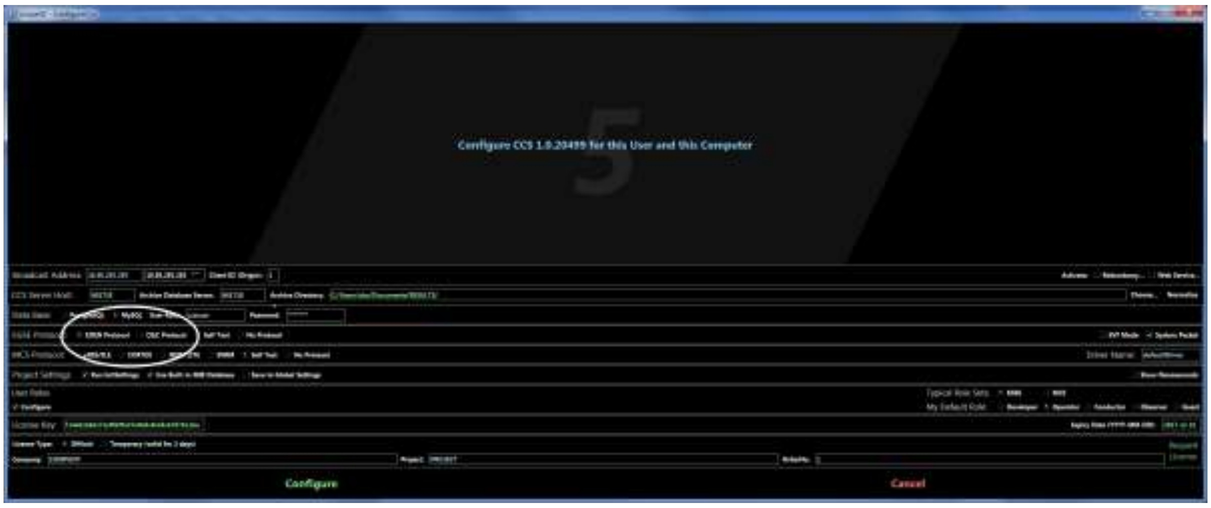
Migrating from TSC to CCS5

This note is provided to help users migrate from TSC to CCS5. You should find that your TOPE scripts are completely portable, but you need to be aware of the differences in architecture between TSC and CCS5.

Configuring the SCOE protocol

In CCS5 you start by configuring your system, and at that time you select your SCOE protocol, typically this will be EDEN or CNC.



This configures the plugin your system will use. The main difference from TSC to CCS5 is that the SCOE protocol plugin is embedded in **IFMGR**, instead of directly in the client console you are using.

Access to the SCOE Protocol plugin

Here are some of the plugin commands and status that you might be accustomed to using:

EDEN	CNC

`::EDEN::connect SCOE1`	`::CncProto::connect SCOE1_TM`
	`::CncProto::connect SCOE1_TC`

`::EDEN::sendCmdExec SCOE1 "transfer remote"`	`::CncProto::sendcnc SCOE1 "transfer remote"`
`$::EDEN::Status(SCOE1_Connected)`	`$::CncProto::Status(SCOE1_Connected)`

CCS5 arranges for plugin commands and accesses to plugin Status to work from the CCS client in the same way as they did in TSC. It does this by passing plugin commands and status accesses to IFMGR.

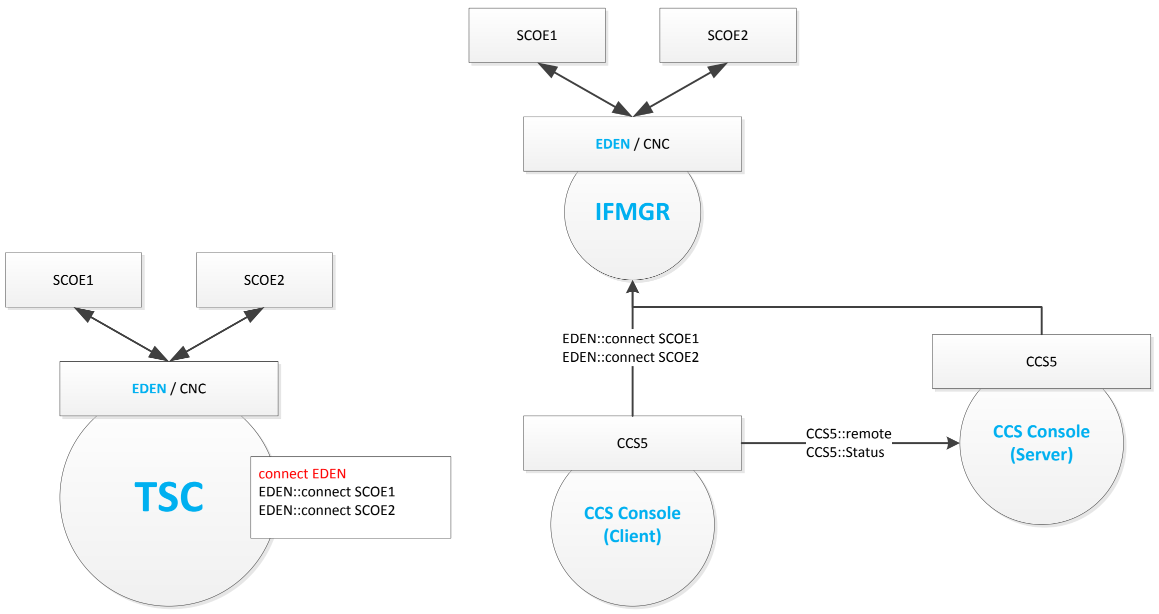
In CCS5 you are using the CCS5 plugin. This plugin has its own status and commands, for example:

`::CCS5::remote CCS-S "syslog HI"`	Run the given command on the CCS server
`$::CCS5::Status(CONNECTED)`	Is the CCS5 plugin connected to the CCS?

In addition, CCS5 plugin arranges that the chosen IFMGR plugin status and commands are also available from the CCS console. Accesses to commands or status in the plugin namespace result in an exchange with IFMGR to run the command remotely, to get, or set a plugin Status value. So the examples for EDEN or CNC shown above will work **as well**.

Accessing the SCOE protocol plugin will be a little slower in CCS5 than it was in TSC because it has to be done remotely. In TSC, access to the plugin is direct.

The picture below shows the difference in architecture (slightly simplified), assuming that EDEN was chosen.

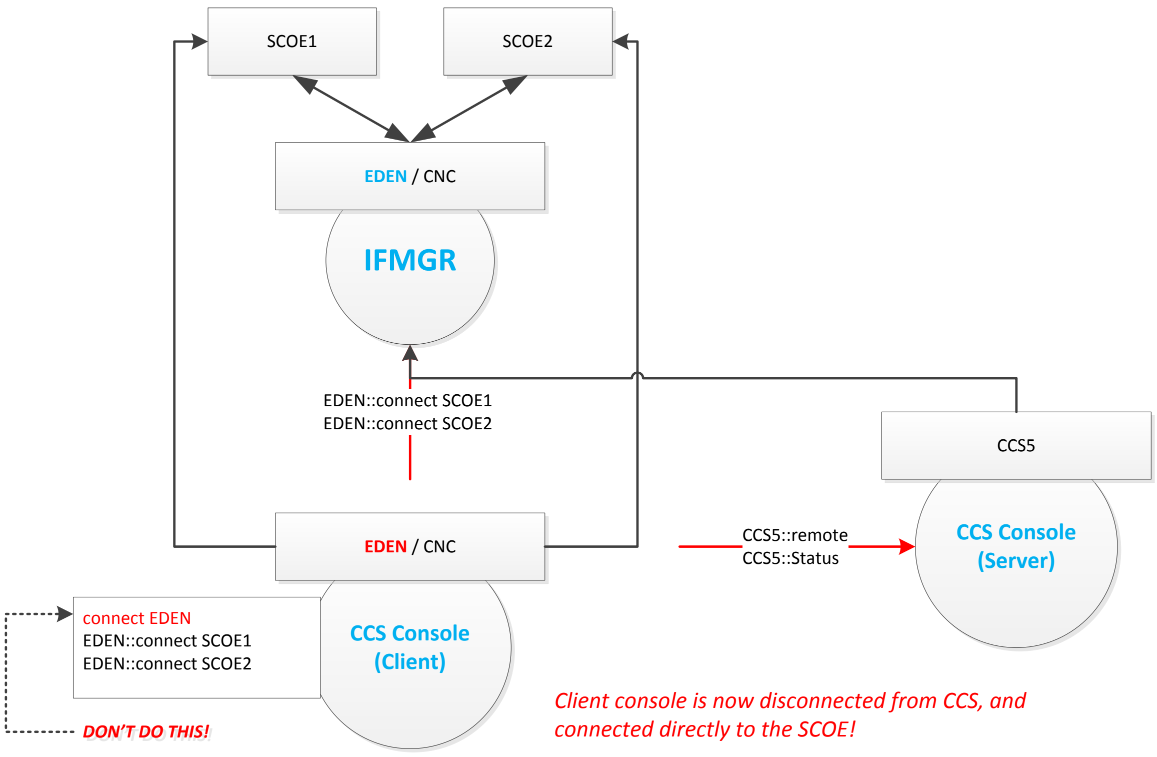


Do not try to connect the protocol explicitly!

A typical problem seen when users migrate from TSC to CCS5 is that they explicitly select and connect the protocol plugin as they did in TSC. In the picture above, the problematic command is shown in red. If this is run in a CCS console (client or server), it will select the EDEN plugin instead of CCS5 plugin. This disconnects the CCS console from the CCS, and makes it use EDEN directly. In newer releases of CCS, this will raise an error, while older releases permit it, and can leave the system in a strange state.

The subsequent connections to SCOE will very probably succeed, but will result in the console being connected directly to the SCOE, instead of via IFMGR. If this happens, you are using a CCS client console directly as a TSC. You would typically see the CCS5 launch pad showing disconnected status.

If you connect the protocol explicitly, you would get an erroneous situation like this:



The only thing you have to do is **omit** the explicit protocol connection:

```
# don't do this in CCS5!!
connect EDEN
```

If you want your script to be portable, you could connect your SCOE like this:

```
# TSC selects the EDEN plugin directly
# CCS has been configured so that IFMGR connects EDEN
switch -exact $::utope::platform(tool) {
TSC {connect EDEN}
CCS {}
}
# EDEN plugin uses this command to connect a specific SCOE
EDEN::connect SCOE1
```

This is a very simple approach, you can obviously arrange connecting the SCOE in different ways . Generally, for portability of scripts, it would be recommended to separate connecting to SCOE from the logic of the script, for example you could put it inside a library procedure.

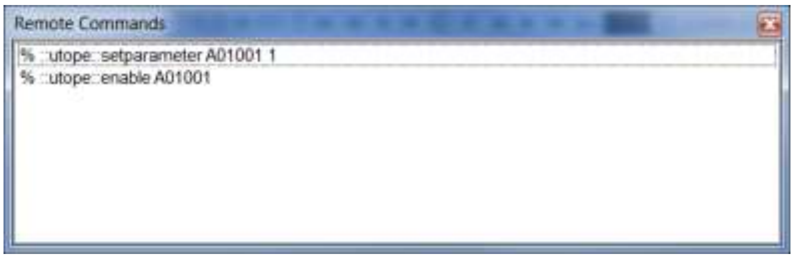
Remote commands take longer

CCS5 is a distributed system, and obviously all clients on the hosts must show the same telemetry values or telecommand status. The system arranges to distribute all this information, but it takes slightly longer to do this; you cannot expect different workstations attached to a CCS to be instantaneously synchronised with each other.

Many TOPE commands are automatically run remotely, generally this applies to any command that would have a global effect on the TM or TC status, for example:

```
setparameter A01001 1
enable A01001
```

If you emit these commands from a TOPE script, then under-the-hood they are automatically run remotely across the whole system. Each CCS client (and the CCS server) is listening for remote commands that were emitted by other workstations connected to the system. You can see the remote commands received by your client in the “Remote Commands” tab.



```
% ::utope::setparameter A01001 1
% ::utope::enable A01001
```

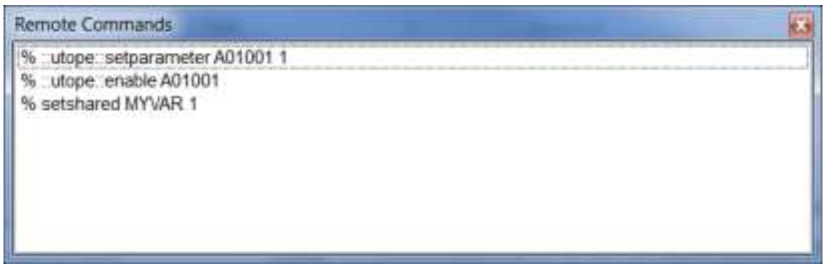
Other commands are not automatically distributed, for example:

```
# this only sets a shared variable locally
setshared MYVAR 1
# the sequence ID must be running locally
tellsequence 1 TERMINATE
```

It is worthwhile to be aware when commands only have a local effect. Let’s say we wanted to set a shared variable across the whole system, or on a specific host; you can do this by explicitly running the command remotely:

```
 # this sets the shared variable across the whole system
CCS5::remote * "setshared MYVAR 1"

# this sets the shared variable on the CCS server
CCS5::remote CCS-S "setshared MYVAR 1"
```

After running this sequence, the CCS client console shows the remote command once, but the CCS server console shows it twice. This screenshot shows the Remote Commands received on a CCS client.



```
% ::utope::setparameter A01001 1
% ::utope::enable A01001
% setshared MYVAR 1
```

Note: it would be unwise to design a suite of TOPE scripts that depend on remote commands being instantaneously effective; remote command distribution should reliably complete within one CCS heartbeat, i.e. < 0.5s.

Settings have multiple applications on CCS5

In TSC, you may be familiar with reading and writing settings like this:

```
set ep $::utope::settings(TM/epoch)
⇨ 1999-08-22T00:00:00
```

In CCS5, the system comprises more than one application, and many of these settings have to be consistent across the whole system. For example the TM epoch must be consistent on all clients as well as the server.

If you want to configure the TM epoch for the whole CCS5 system, you have to configure it for the applications CCS client and CCS server, giving the application name and a colon separator:

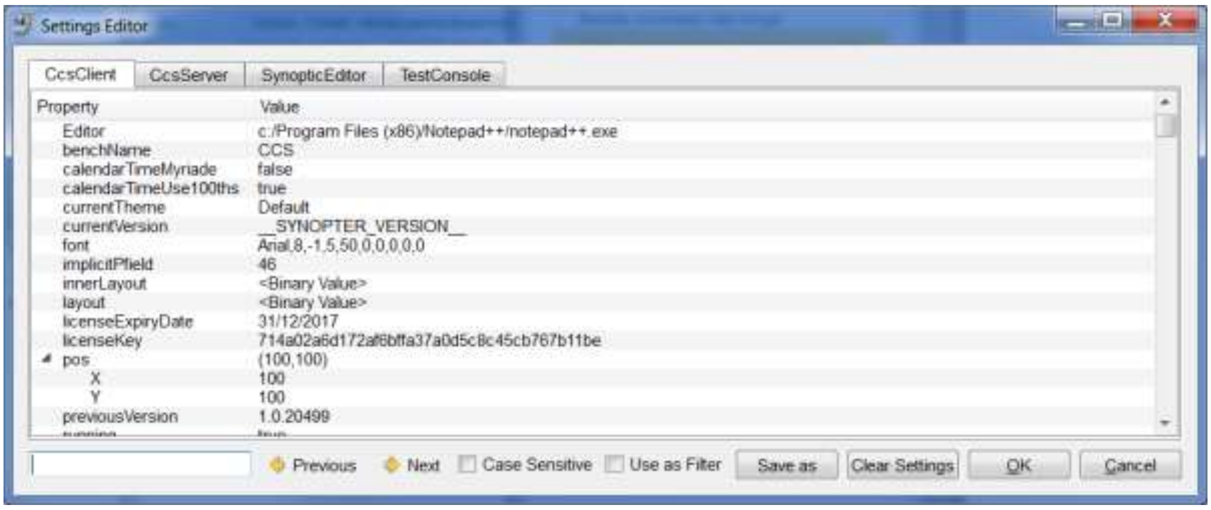
```
# set the default TM epoch on TSC (or the current client)
# there is only one application, i.e. the current one
set ::utope::settings(TM/epoch) 1999-08-22T00:00:00

# set the default TM epoch on CCS5
# there are multiple applications,…
set ::utope::settings(CcsClient:TM/epoch) 1999-08-22T00:00:00
set ::utope::settings(CcsServer:TM/epoch) 1999-08-22T00:00:00
set ::utope::settings(SynopticEditor:TM/epoch) 1999-08-22T00:00:00

# or alternatively…
foreach app {CcsClient CcsServer SynopticEditor} {
      set ::utope::settings(${app}:TM/epoch) 1999-08-22T00:00:00
}
```

If you read or write settings without a colon, then it means the current application by default.

If your TOPE scripts written for TSC read settings, they should work just fine on CCS, because the CCS client settings should be consistent with the TSC ones. In fact, if you want to write portable scripts then it is better to omit the application name, because that means “look up the setting value for whatever application I am running in”

However if you are configuring the whole system (e.g. InitSettings.tcl), then you have to take care that all applications in the CCS have been configured consistently, and to do this, you will need to explicitly give the application name, e.g. CcsClient, CcsServer, SynopticEditor.

It is not feasible to describe here all the settings, but in CCS5 you get a separate settings editor application which shows the different applications in different tabs:



Property	Value
Editor	c:/Program Files (x86)/Notepad++/notepad++.exe
benchName	CCS
calendarTimeMyriade	false
calendarTimeUse100ths	true
currentTheme	Default
currentVersion	__SYNOPTER_VERSION__
font	Arial,8,-1,5,50,0,0,0,0,0
implicitPfield	46
innerLayout	<Binary Value>
layout	<Binary Value>
licenseExpiryDate	31/12/2017
licenseKey	714a02a6d172af6bffa37a0d5c8c45cb767b11be
pos	(100,100)
X	100
Y	100
previousVersion	1.0.20499

Notice that you see the TestConsole tab too: that means you can configure TSC as well.

The settings editor should appear as a desktop icon but you can also start it on Linux from the command line like this:

```
/opt/CCS/bin/settings.sh
```

It is worth mentioning at this point that the latest CCS releases also allow a "Global Settings" mechanism, which you can activate by clicking the check box when configuring CCS (see screenshot above). This ensures that all clients and users attached to this CCS will use the same settings automatically. When you start setting up a system with multiple workstations, no doubt you will find this useful to keep the settings consistent between all workstations and users, and can save you a lot of time. You should nevertheless read the online help documentation about it.