

OPENDCS 6

DCP Monitor Maintenance Guide

Document Revision 3

January, 2018

This Document is part of the OpenDCS Software Suite for environmental data acquisition and processing. The project home is:

<https://github.com/opendcs/opendcs>

See INTENT.md at the project home for information on licensing.

Table of Contents

1	Overview.....	1
1.1	Glossary and List of Acronyms.....	2
2	DCP Monitor Database Schema.....	3
3	DCP Monitor Daemon.....	4
3.1	Enumerations.....	4
3.2	The DCP Monitor 'Process' Record.....	5
3.3	DCP Monitor Routing Spec.....	7
3.4	Running the Daemon.....	8
3.5	Monitoring and Stopping the Daemon.....	8
4	Apache Tomcat.....	10
5	DCP Monitor Web Archive.....	11

1 Overview

The DCP Monitor Web Application is compatible with OpenDCS 6.1. It has the following components:

- DCP Monitor Database Schema – part of the standard OpenDCS schema. Compatible with PostgreSQL and Oracle. The DCP Monitor Tables are *not* included with the CWMS CCP/DECODES schema.
- DCP Monitor Daemon – functions like a routing spec. Runs in the background continually collecting data from DCPs that you specify via channels and/or network lists. Raw messages are stored in the SQL database.
- DCP Monitor Routing Spec – this record in your database is used by the Routing Spec Daemon to control several aspects of its operation.
- Apache Tomcat Container – Tomcat is available as a free download for all modern operating systems.
- DCP Monitor Web Archive (WAR) – You must first customize the WAR for your database, page headers, logos, etc., and then rebuild the WAR (Web Archive) as described below. Once deployed, the WAR runs as an application under Tomcat, providing views into the database via web pages.

The following sections describe how to install, configure, and deploy each of the components.

1.1 Glossary and List of Acronyms

CP	Computation Processor – the background program that executes computations as new data arrives.
CCP	CWMS Computation Processor – i.e. the CP configured for CWMS.
CWMS	Corps Water Management System (pronounced ‘swims’) - A system for hydrologic data storage and analysis used by USACE.
DAS	Data Acquisition Server – responsible for collecting raw DCP messages via a variety of satellite and internet links.
DBMS	Database Management System
DCP	Data Collection Platform – equipment in the field that collects and transmits raw environmental measurements.
DCS	Data Collection System
DECODES	DeviceCONversion and DELivery System – A collection of software for decoding raw environmental data, and converting it to a time-series in a variety of formats.
ERD	Entity Relationship Diagram
GUI	Graphical User Interface
HDB	Hydrologic Database – A system for hydrologic data storage and analysis used by USBR.
LRGS	Local Readout Ground Station – This is synonymous with DAS. It is the legacy name for a Data Acquisition Server.
NWIS	National Water Information System - A system for hydrologic data storage and analysis used by USGS.
SDI	Site Data-type ID. In HDB this is used to denote a particular parameter at a particular site. It is stored as a numeric ID.
SQL	(a.k.a. “sequel”) Structured Query Language
TSDB	Time Series Database
USACE	U. S. Army Corps of Engineers
USBR	U. S. Bureau of Reclamation
USGS	U. S. Geological Survey
XML	Extensible Markup Language

2 DCP Monitor Database Schema

The DCP Monitor requires the Database Schema corresponding to OpenDCS Release 6.1.

To determine your database version, run SQL (psql or PgAdmin3 for PostgreSQL; sqlplus or Oracle Developer for Oracle), and issue the following query:

```
select * from DecodesDatabaseVersion;
```

The table contains a single row. The version_num value must be 11 or greater.

If you are running an OpenDCS 6.0 database the version_num will be 10. In this case you can run the utility “dbupdate”. It will ask you for the database owner username and password. It will update the schema in place. You can examine the file “dbupdate.log” after running the utility to see if any problems were encountered.

If your version_num is less than 10, you cannot run DCP Monitor on this database. Please install a fresh database from the schema provided with the OpenDCS 6.1 release.

3 DCP Monitor Daemon

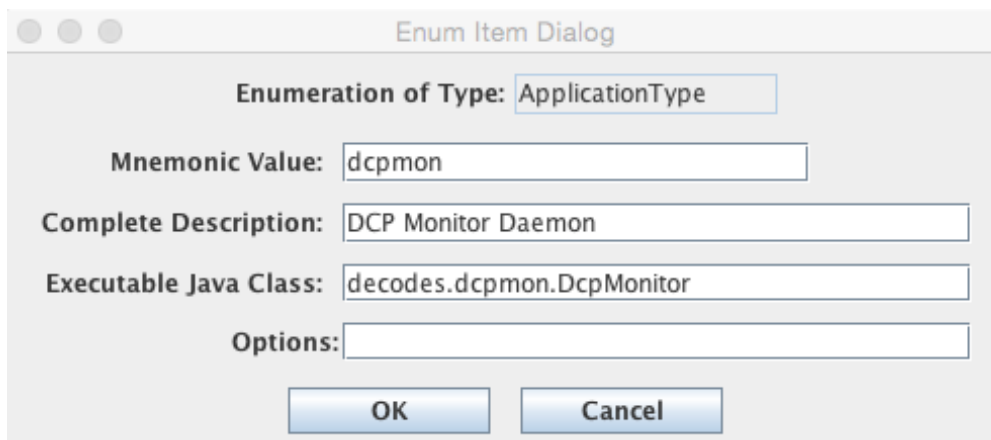
The DCP Monitor Daemon is included with OpenDCS 6.1 and later releases. This section will describe how to configure and run the daemon.

3.1 Enumerations

If you have upgraded from OpenDCS 6.0 you may need to create some enumeration values before continuing.

Run the “rledit” command and select the Enumerations tab.

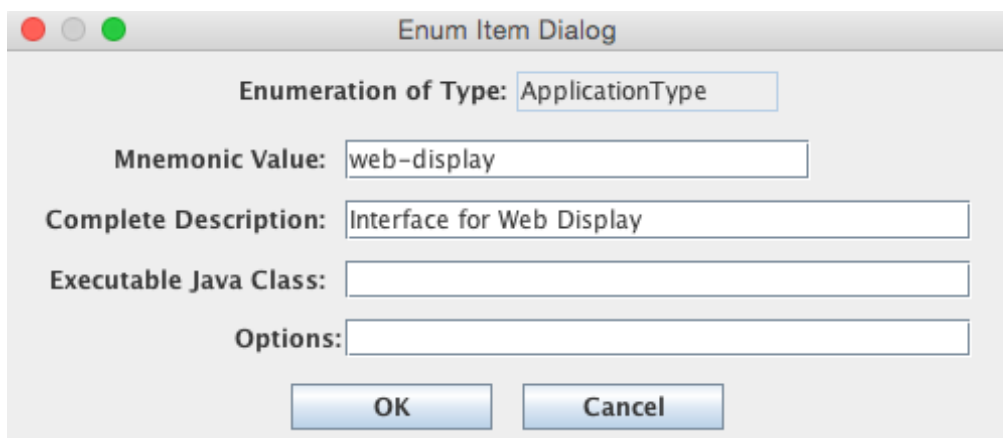
Select the “Application Type” enumeration. Verify that a record exists with name “dcpmon”. Verify that the Executable Java Class is exactly as shown:



The screenshot shows a dialog box titled "Enum Item Dialog". It contains the following fields and buttons:

- Enumeration of Type:** A dropdown menu showing "ApplicationType".
- Mnemonic Value:** A text field containing "dcpmon".
- Complete Description:** A text field containing "DCP Monitor Daemon".
- Executable Java Class:** A text field containing "decodes.dcpmon.DcpMonitor".
- Options:** An empty text field.
- Buttons:** "OK" and "Cancel" buttons at the bottom.

Also create a separate record as follows for the web-application components. For web-display, Java Class is not relevant:



The screenshot shows a dialog box titled "Enum Item Dialog". It contains the following fields and buttons:

- Enumeration of Type:** A dropdown menu showing "ApplicationType".
- Mnemonic Value:** A text field containing "web-display".
- Complete Description:** A text field containing "Interface for Web Display".
- Executable Java Class:** An empty text field.
- Options:** An empty text field.
- Buttons:** "OK" and "Cancel" buttons at the bottom.

If you have modified anything, select File – Save to DB before exiting.

3.2 The DCP Monitor ‘Process’ Record

Earlier implementations of DCP Monitor were configured in a text file called “dcpmon.conf”. This was problematic because configuration variables must be shared by the daemon and web components. The new implementation solves this problem by storing all configuration variables in the database.

Start the computations editor in one of two ways:

- Select Computations from the Launcher
- Use the command line “compedit”.

Select the Processes tab.

If one doesn’t already exist, create a new process and name it “dcpmon”.

For “Process Type” select “dcpmon”. If this is not a selection in the pull-down list, revisit the section above on Enumerations. After selecting the “dcpmon” process type, the Applications Properties area at the right should fill in with numerous property names:

Computation Editor

Algorithms | Computations | **Processes**

List dcpmon

Process Name: dcpmon

Process ID: 22

Process Type: dcpmon

☐ Manual Edit App

Name	Value
LastModified	20141226173849
cdtLocalFile	/Users/mmaloney/test/AESRD-PG-61/cdt
cdtUrl	-
numDaysStorage	2
nwsXrefLocalFile	/Users/mmaloney/test/AESRD-PG-61/nwsxref
nwsXrefUrl	-
pdtLocalFile	/Users/mmaloney/test/AESRD-PG-61/pdt
pdtUrl	-
hadsUrl	-
grp:<netlist-name>	
controlDistSuffix	
dcpmonNameType	
redMsgTime	
yellowMsgTime	
webUsesRoutingSpecNetlists	
allChannels	
mergeDir	

Comments

DCP Monitor Web Application

Commit Close

If you hover the mouse pointer over a property name, a tool-tip will appear explaining the property. If you double click, a dialog will pop up with the help text and an area to enter the property value.

Allowable properties are explained in the table below:

Property Name	Default	Description
monitor	false	Set to 'true' to allow monitoring the DCP Monitor Daemon from the GUI (recommended.)
eventPort	(none)	Works when monitor=true. Assign a unique port number greater than 1024 to each process that you want to monitor.
numDaysStorage	5	Number of days (including current day) that DCP Monitor will keep data for. Data older than this is flushed periodically.
omitFailureCodes	(empty string)	Failure codes are single character values that usually indicate some kind of problem condition. If you do <i>not</i> want to see certain codes, add them to this string.
redMsgTime	0	Number of seconds. Messages that are within this many seconds outside the assigned time window are marked red.
yellowMsgTime	2	Number of seconds. Messages that are within this many seconds of the assigned time window are marked yellow.
redFailureCodes	M	If a message has any of these failure codes it will be displayed red.
yellowFailureCodes	?UT	If a message has any of these failure codes it will be displayed yellow.
redFreqOffset	6	If a message has a frequency offset with a magnitude greater than this, it will be displayed red.
yellowFreqOffset	6	If a message has a frequency offset with a magnitude greater than this, it will be displayed yellow.
redSignalStrength	30	If a message has a signal strength less than this, it is flagged as red.
yellowSignalStrength		If a message has a signal strength less than this, it is flagged as yellow.
redBattery	9.0	A battery voltage less than this is flagged as red.
yellowBattery	11.0	A battery voltage less than this is flagged as yellow.
maxCarrierMS	2500	Maximum length of pre-message carrier (in milliseconds). Messages with more carrier than this before start of message will be flagged with result code 'C'.
allChanel	false	Set to true to receive all GOES channels. The default is to receive only channels used by the DCPs in your groups.
grp: NetlistName	N/A	For each group that you want to appear in the pull-down lists on the web app, specify a property with the prefix 'grp:' followed by the name to appear on the web. The property value should then be the name of the network list in your database.
webUsesRoutingSpecNetlists	true	As an alternative to naming your groups explicitly with the 'grp:' prefix, you can set this to 'true'. All of the network lists used by the "dcpmon" routing spec will also appear as groups for selection in the webapp.
pdtLocalFile		This specifies the location of the local, merged, copy of the PDT. This is used for names by the webapp, particularly for

		DCPs that are not in your database. Default = /tmp/pdt
pdtUrl	(none)	If you leave this blank, then no download will be attempted. If you specify an URL, then the daemon will periodically download a new file from NOAA and merge it into the local copy. The default URL is: https://dcs1.noaa.gov/pdts_compressed.txt
cdtLocalFile		Default = /tmp/chans_by_baud.txt
cdtUrl		Default: https://dcs1.noaa.gov/chans_by_baud.txt
nwsXrefLocalFile		Default = /tmp/nwsxref.txt
nwsXrefUrl		Default: http://www.nws.noaa.gov/oh/hads/USGS/ALL_USGS-HADS_SITES.txt
dcpmonNameType	(none)	The site name type to prefer for use on the web application.
rtstatURL	URL	The URL to the dynamic HTML file being updated by an LRGS daemon. Set this to a single hyphen '-' to disable this feature.
statusErrorThreshold		Number of seconds. Devices and platforms with an error within this amount of time are shown in red on the appropriate status pages.

3.3 DCP Monitor Routing Spec

Now create the “dcpmon” routing spec through the DECODES Database Editor. Start the editor either by clicking “DECODES Database Editor” from the launcher, or with the “dbedit” command. Then click the “Routing” tab. Open the routing spec named “dcpmon” if one exists. If not, create one.

- Specify the Data Source name that you want the DCP Monitor to collect data from. This must be an LRGS-type data source, or an LRGS Hot Backup Group data source.
 - The data source should contain a “username” property. DCP Monitor uses this when connecting to the LRGS.
- It doesn’t matter what you specify for Destination, output format, or timezone. The DCP Monitor application will override these setting to store transmission records in your SQL database.
- It also does not matter what timezone you specify. DCP Monitor will maintain where it left off so as not to lose data and maintain a real time stream.
- Enter the network lists representing the groups that you want to appear on the web.

After specifying data source and network lists, save the routing spec and exit the editor.

3.4 Running the Daemon

We recommend creating a script to start the DCP Monitor Daemon. Here is a script appropriate for the Unix/Linux BASH shell. It assumes that we have a directory set up called “dcpmon” under the \$DCSTOOL_HOME directory:

```
#!/bin/bash

DCPMON_DIR=$DCSTOOL_HOME/dcpmon
LOG=-l dcpmon.log
DBG=-d2

PATH=$DCSTOOL_HOME/bin:$PATH

cd $DCPMON_DIR
nohup decj decodes.dcpmon.DcpMonitor $DBG $LOG >dcpmon.nohup 2>&1 &
```

Note the LOG and DBG arguments are set near the top of the script. The script CD's to the directory we have set up for DCP Monitor and then runs the java program. On the command line:

- **nohup** means to run in the background, divorced from any terminal associations.
- **>dcpmon.nohup** means to redirect any standard output to the file “dcpmon.nohup”. There should normally be no output unless unexpected errors occur.
- **2>&1** is the bash directive telling it to send any error output to the same place as standard output, that is, the dcpmon.nohup file.
- **&** at the end of the line means to run in the background.

You can run this script at startup, but be aware that it assumes that \$DCSTOOL_HOME has been set prior.

3.5 Monitoring and Stopping the Daemon

The DCP Monitor Daemon functions like many of the other background processes in the DECODES/LRGS/CCP suite.

It creates a lock in the database. You can view locks with the command:

```
complocklist
```

The response will be a list of running daemons, one per line. The line for dcpmon will look something like:

```
appId=22, pid=9930, host=unknown, heartbeat=Fri Jan 16 09:18:01 EST 2015,
status=run=1, wait=0, cmpl=0, name=dcpmon
```

The heartbeat should be within 10 seconds of the current time. The pid is the Unix/Linux process ID which can be referenced in the Unix ‘ps’ or ‘jps’ programs.

To stop a running daemon, use this command:

```
stopcomp -a dcpmon
```

This removes the daemon's lock. After a maximum of 10 seconds, the daemon will notice that its lock has disappeared and will exit. If for any reason the process will not stop, you can kill it with the process ID that appears in `complocklist` or `jps`.

The daemon is continually writing to a log file with whatever verbosity you specified. In our script we specified `-d2`, which is quite verbose (0 = default meaning only INFO, WARNING, and FAILURE messages go to the log, 3 is the most verbose).

Once a log file reaches 10MB in size, it is renamed with a ".old" suffix, and a new file is opened. Thus you always have at least the most recent 10MB of log messages.

You can view new log messages with the command:

```
tail -f dcpmon.log
```

In order to monitor your processes (Routing Specs, Scheduler, CompProc, and DCP Monitor) in the GUI, you have to set two properties: `monitor`, and `eventPort`. See descriptions of these properties in the table above.

On the launcher bar select "Process Status". You see the status of all of your daemons at the top. This is the same information in the 'complocklist' utility. You can check the 'Events' box to have the events from that process appear in a scrolling list at the bottom.

The screenshot shows a window titled "Process Monitor" with a sub-header "Process Status Monitor". It contains a table with the following columns: App ID, App Name, Host, PID, Heartbeat (UTC), Status, and Events?. The table lists four processes: StaleDataChecker (PID N/A, Not Running), dcpmon (PID 10404, running since Jan-16-2015 14:52:09, Status: run=1, wait=0, cmpl=0), ScadaConvert (PID N/A, Not Running), and RoutingScheduler (PID N/A, Not Running). The dcpmon process has its Events? checkbox checked. Below the table is a scrolling log area showing various database and network messages, including "XmitRecordDao Saving new msg" and "lrgsds-0 DDS Connection" requests. At the bottom left, there is a checkbox labeled "Snap to End" which is also checked.

App ID	App Name	Host	PID	Heartbeat (UTC)	Status	Events?
14	StaleDataChecker	N/A	N/A	-	Not Running	<input type="checkbox"/>
22	dcpmon	localhost	10404	Jan-16-2015 14:52:09	run=1, wait=0, cmpl=0	<input checked="" type="checkbox"/>
23	ScadaConvert	N/A	N/A	-	Not Running	<input type="checkbox"/>
24	RoutingScheduler	N/A	N/A	-	Not Running	<input type="checkbox"/>

DBG2 dcpmon 01/16/15 14:52:11 XRWriteThread.processQueue qsize=39
DBG1 dcpmon 01/16/15 14:52:11 XmitRecordDao Saving new msg with dcp addr=CE50CD5C at time 2015/01/16-14
DBG1 dcpmon 01/16/15 14:52:11 XmitRecordDao Saving new msg with dcp addr=CA8040E8 at time 2015/01/16-14
DBG1 dcpmon 01/16/15 14:52:11 XmitRecordDao Saving new msg with dcp addr=327797E0 at time 2015/01/16-14
DBG1 dcpmon 01/16/15 14:52:11 XmitRecordDao Saving new msg with dcp addr=190400DE at time 2015/01/16-14
DBG2 dcpmon 01/16/15 14:52:11 lrgsds-0 DDS Connection (www.sutronwin.com:16003) Requesting DCP Message
DBG2 dcpmon 01/16/15 14:52:12 XRWriteThread.processQueue qsize=35
DBG2 dcpmon 01/16/15 14:52:12 lrgsds-0 DDS Connection (www.sutronwin.com:16003) Requesting DCP Message
DBG2 dcpmon 01/16/15 14:52:12 DcpMonitorConsumer.startMessage() msg=DDAB523215016145112G43+0NN167EFF001
DBG2 dcpmon 01/16/15 14:52:12 XRWriteThread.enqueue: DDAB523215016145112G43+0NN167EFF00157 queue.size=3
DBG2 dcpmon 01/16/15 14:52:12 DcpMonitorConsumer.startMessage() msg=9439805C15016145114G50-1NN137EFF000
DBG2 dcpmon 01/16/15 14:52:12 XRWriteThread.enqueue: 9439805C15016145114G50-1NN137EFF00089 queue.size=3
DBG2 dcpmon 01/16/15 14:52:12 lrgsds-0 DDS Connection (www.sutronwin.com:16003) Requesting DCP Message

☒ Snap to End

4 Apache Tomcat

Install Apache Tomcat according to the instructions from Apache for your platform.

Set an environment variable `$CATALINA_HOME` to the installation directory. The directory `$CATALINA_HOME/webapps` is where the dcp monitor webapp will be deployed.

5 DCP Monitor Web Archive

The DCP Monitor Web is available from Cove Software, LLC as a single compressed tarfile called `dcpmon-VERSION.tar.gz`, where VERSION corresponds to the OpenDCS delivery. It is not included in the OpenDCS installation. For example:

`dcpmon-6.1RC01.tar.gz`

Unpack the distribution on your development machine with the commands:

```
gunzip dcpmon-6.1RC01.tar.gz
tar xvf dcpmon-6.1RC01.tar
```

This will create the following directory hierarchy:

- `dcpmon-build`
 - `war-distro` - *A tree of many files and directories under here. These are the stock files included with the webapp.*
 - `war-custom` – *Initially empty directories. You will customize files and place them here. When you build the webapp these file will override the stock versions.*
 - `WEB-INF`
 - `lib`
 - `META-INF`
 - `makeDcpmon.sh` – *A shell script to combine the stock files and overrides, and build the deployable WAR file.*

We assume that each site that wants to deploy DCP Monitor will want to override certain files to:

- Point to your own database (required)
- Add you agency's page headers & footers.
- Modify the CSS with your own look and feel.
- Include additional pages
- Exclude some of the stock pages that come with the distro.

Customizing usually involves copying the stock file from the `war-distro` tree to the same location in the `war-custom` tree, and then modifying the copy in the `war-custom` tree.

You will certainly need to override the file that specifies your database:

```
cp war-distro/META-INF/context.html war-custom/META-INF
vi war-custom/META-INF/context.html (or whatever your favorite editor is)
```

In this file you will want to modify the arguments for:

- `url`: Set this to point to your actual database. This should be the same as the setting for DECODES Settings – Database Location.

- **username:** The database user that dcpmon will use when connecting to the database
- **password:** corresponds to username
- **driverClassName:** the fully-qualified Java class name for the JDBC driver.

If you use a customized JDBC driver class, place it in the war-custom/WEB-INF/lib directory.

If this is a HDB database, also add the following to the context.xml:

```
<Parameter name="appDbIoClass" value="decodes.hdb.HdbSqlDatabaseIO" />
```

You can customize *any* file that appears in the distro by this method. Files you will likely want to customize include:

- dcpmon-footer.html – HTML footer included on every page
- dcpmon-header.html – HTML header included on every page
- css/dcpmon.css – Defines styles, fonts, etc.
- index.html – Top level page. Modify this to include/exclude additional links and content.

To build the deployable WAR, CD to the dcpmon-build directory and run:

```
./makeDcpmon.sh
```

This will combine the stock files with your overrides and create a file called “dcpmon.war”. To deploy, simply copy this file to Tomcat’s webapps directory in the normal fashion.