SDK Installation

Release 1.6.0

Software and Controls Group

Jul 03, 2019

Contents

CHAPTER

ONE

DEVELOPMENT PLATFORM

The development platform is no longer distributed as a standalone ISO file. Instead, the following guide is provided to assist with hardware, operating system and third-party software configuration required for running the OCS SDK. Some key areas, such as disk partitioning, user authentication and NFS mounting of home directories, depend on individual factors at each partner institution and should therefore be considered examples only.

Benefits of this approach, versus distributing the complete ISO, include the ability to support different network, hardware and software platforms used for software development by different partner institutions, as well as allowing for separate upgrade schedules for the Development Platform and Software Development Kit without affecting ongoing subsystem software development.

Note: At GMTO, these instructions are formalized using Kickstart files and installed via the network using tools such as Cobbler. The GMTO DevOps team can provide assistance in setting up a similar system for development at partner institutions, if required.

The Observatory Control System (OCS) is designed to be a distributed system with device control components running on real-time computers, connected to common services and user interface components via the control network.

For device control systems, the following operating systems are supported:

- Fedora server

For user interfaces, the following operating systems are supported:

- MacOS

Future versions of the SDK could include support for CentOS, RHEL or Scientific Linux. Fedora has a very short release and support cycle (6 months and 18 months respectively), which is not ideal for a platform that requires significant stability over long periods of time.

CHAPTER

TWO

SERVER CONFIGURATION

Servers are used for developing, running and testing device control software and core services. When real-time communication with hardware is required, the real-time kernel should be installed and configured. The following guidelines for creating a server should be tailored according to its intended purpose.

2.1 Required Hardware

Minimum hardware requirements for development machines:

- Intel Pentium 4 or higher, 2 GHz CPU
- 1 GB Memory
- 20 GB Hard drive

Typical GMT OCS development machine specs:

- Intel Xeon E3 or higher, 3.2 GHz CPU
- 4 - 8 GB Memory
- 120 - 250 GB Hard drive

2.2 Operating System

Install the Operating System using these instructions:

2.3 Repository Configuration

Some required RPMs are built by GMTO and need to be downloaded from the GMTO Yum Repository, currently hosted on Amazon Web Services.

To add the GMT repositories:

1. Add the file `/etc/yum.repos.d/gmt.repo` with the following content:

```
[gmt]
name=GMT $releasever - $basearch
baseurl=http://52.52.46.32/srv/gmt/yum/stable/$releasever/
```

```
gpgcheck=0
enabled=1
```

2. Add the file `/etc/yum.repos.d/gmt-updates.repo` with the following content:

```
[gmt-updates]
name=GMT $releasever - $basearch - Updates
baseurl=http://52.52.46.32/srv/gmt/yum/updates/$releasever/
gpgcheck=0
enabled=1
```

2.4 Advanced System Configuration (optional)

In a distributed computing environment, used by multiple developers, it is very convenient to use a centralized LDAP server for User Authentication and automatically mount /home directories from a network drive. The LDAP and NFS server configuration is network-dependent. The following instructions can be used as guidelines when configuring individual development machines to make use these services, if available.

Note: This configuration is currently optional. The alternative is to add users manually and manage permissions locally on each development machine.

2.4.1 User Authentication using LDAP

Managing user credentials in a centralized server allows developers to log into any development machine without the need to duplicate the configuration for each new machine.

Official Fedora Documentation for setting up a Directory Server using OpenLDAP can be found here:

https://docs.fedoraproject.org/en-US/fedora/f28/system-administrators-guide/servers/Directory_Servers/index.html

The following instructions are for configuring **development machines** to use the LDAP server on the network and should be seen as an example to be tailored to the actual network it's running on.

1. Install the necessary packages as root

```
$ dnf install -y openldap-clients sssd
```

2. Edit `/etc/openldap/ldap.conf` and set the following options:

```
URI ldap://<ldap_server_ip>/
BASE dc=gmto,dc=org
```

where `<ldap_server_ip>` is the IP Address of the LDAP server and `dc=gmto,dc=org` should reflect the configuration in the LDAP server.

3. Create file `/etc/sssd/sssd.conf` and set the following options:

```
[domain/default]
id_provider = ldap
autofs_provider = ldap
auth_provider = ldap
chpass_provider = ldap
```

```
ldap_uri = ldap://<ldap_server_id>/
ldap_search_base = dc=gmto,dc=org
ldap_id_use_start_tls = False
ldap_tls_cacertdir = /etc/openldap/certs
cache_credentials = True
ldap_tls_reqcert = allow

[sssd]
services = nss, pam, autofs
domains = default

[nss]
homedir_substring = /home
```

where `<ldap_server_ip>` is the IP Address of the LDAP server and `dc=gmto,dc=org` should reflect the configuration in the LDAP server.

4. Enable sssd

```
$ chmod 600 /etc/sssd/sssd.conf
$ systemctl restart sssd oddjobd
$ systemctl enable sssd oddjobd
```

5. Optionally, use LDAP to grant sudo access to developers within a group. This needs to be configured correctly on the LDAP server. For example, if a group has been defined in the LDAP server with name %dev, containing all the developers that may access this machine, add the following line to the `/etc/sudoers` file via `visudo`. This command needs to be run as root.

```
%dev ALL=(ALL)   ALL
```

Alternatively, configure sudo access for specific users only.

[back to top]

2.4.2 Mounting Home Directories using NFS

In a distributed computing environment used by multiple developers, it is useful to be able to automatically mount /home directories from a network drive so that development files can be easily transferred from one machine to another.

The following instructions are for configuring **development machines** to use available NFS directories on the network and should be seen as an example to be tailored to the actual network it's running on.

1. Install the necessary packages

```
$ sudo dnf install -y nfs-utils
```

2. Create an export directory

```
$ mkdir /export
```

3. Add the following line to the `/etc/fstab` file:

```
<nfs_ip>:/export/home /home nfs noauto,nofail,x-systemd.automount,x-
↪systemd.requires=network-online.target,x-systemd.device-timeout=1,
↪timeo=14,noacl 0 0
```

where `<nfs_ip>` is the IP Address of the server where the home directories are stored.

[back to top]

2.5 Package List

The following RPM packages should be installed by an Administrative user for use in the development environment:

1. Install Common OS Utilities

```
$ sudo dnf install -y xorg-x11-xauth urw-fonts wget net-tools pciutils
$ sudo dnf install -y strace rpl bash-completion sed
```

2. Install Development Tools

```
$ sudo dnf install -y autoconf automake cmake elfutils gcc gdb libtool make
$ sudo dnf install -y cpp cscope ctags gc gcc-c++ gcc-gdb-plugin glibc-devel
$ sudo dnf install -y glibc-headers kernel-headers libstdc++-devel
$ sudo dnf install -y flex git libcurl-devel
$ sudo dnf install -y python3-sphinx python3-sphinx_rtd_theme
```

3. Install OCS Dependencies

```
$ sudo dnf install -y rdma librdmacm-devel boost-devel
```

2.6 Node Installation

1. Download and install **Node version 10**

```
$ curl -sL https://rpm.nodesource.com/setup_10.x | sudo bash -
$ sudo dnf install -y nodejs
```

2. Install necessary node packages:

```
$ sudo npm install -g coffeescript webpack webpack-cli raw-loader
```

2.7 MongoDB Configuration

1. Install the necessary packages:

```
$ sudo dnf install -y mongodb mongodb-server
```

2. Configure the firewall

```
$ sudo firewall-offline-cmd --direct --add-rule ipv4 filter INPUT 0 -p tcp --
↪dport 27017 -j ACCEPT
```

2. Enable the MongoDB service

```
$ sudo systemctl enable mongod
$ sudo systemctl start mongod
```

3. Check that the MongoDB service is up

```
$ sudo systemctl status -l mongod
```

2.8 Infiniband Configuration (optional)

Infiniband is a low-latency networking communications protocol that requires specialized hardware. The following configuration steps should be used as a guide when configuring Infiniband communications.

1. Install the neccessary packages

   ```
   $ sudo dnf install -y infiniband-diags opensm libmlx4
   ```

2. Edit `/etc/rdma/mlx4.conf` and add the following line:

   ```
   01:00.0 auto auto
   ```

3. Find the interface used for Infiniband and edit the corresponding configuration file (for example `/etc/sysconfig/network-scripts/ifcfg-ib0`) to set the following options:

   ```
   DEVICE=ib0
   ONBOOT=yes
   TYPE=Infiniband
   BOOTPROTO=none
   IPADDR=<ib_ip_address>
   NETMASK=<ib_netmask>
   ```

   where `<ib_ip_address>` is the static IP Address associated with the Infiniband network interface and `<ib_netmask>` is the netmask used for the infiniband subnet.

4. Enable Infiniband Services

   ```
   $ sudo systemctl enable opensm
   $ sudo systemctl enable rdma
   ```

2.9 Ethercat Configuration

EtherCAT is a high-speed fieldbus communication system used for real-time control. The following configuration steps should be used as a guide when configuring EtherCAT communications.

1. Install the real-time kernel and relevant packages

   ```
   $ sudo dnf install -y --nogpgcheck kernel-3.14.73-rt78.x86_64 ethercat-devel

   .. warning::
     This Linux kernel requires the root partition to be an **ext4** file␣
    ↪system. Otherwise, your machine will not boot.
   ```

2. Select the Ethernet interface to be used for EtherCAT communication (e.g. enp4s0) and edit the corresponding configuration file (e.g. `/etc/sysconfig/network-scripts/ifcfg-enp4s0`) to set the following options:

   ```
   BOOTPROTO=none
   ONBOOT=no
   ```

3. Check the Hardware Address of the selected EtherCAT network interface

```
$ ifconfig
```

4. Edit `/etc/ethercat.conf` and set the following configuration options:

```
MASTER0_DEVICE="<mac_address_1>"
MASTER0_BACKUP="<mac_address_2>"  # optional line
```

   where `<mac_address_1>` and `<mac_address_2>` are the two hardware addresses associated with the Ethercat network interface communicating with the Ethercat ring (redundant topology). If you you prefer using a linear topology (non redundant), comment or remove the second line (MASTER0_BACKUP="<mac_address_2>").

5. Edit `/usr/lib/systemd/system/ethercat.service` and uncomment the following line:

```
Before=network.service
```

6. Reboot into the RT Kernel, if you're not in it already.

7. Enable the Ethercat service

```
$ sudo systemctl enable ethercat
$ sudo systemctl start ethercat
```

8. Edit `/etc/security/limits.d/99-realtime.conf` and add the following options:

```
@realtime - rtprio 99
@realtime - memlock unlimited
```

9. Add a new group and add the "gmto" user to it.

```
$ sudo groupadd -f -g 2001 realtime
$ sudo usermod --groups realtime gmto
```

8. Test the Ethercat configuration

```
$ ethercat master
$ ethercat slaves
```

If the "ethercat master" command does not produce the correct output, ensure that you're currently running the real-time kernel. If the "ethercat slaves" command produces no output, check that the ethernet cable is connected to the correct port as configured above.

2.10 Network Time Protocol Configuration

For general network timekeeping, use NTP, unless Precision Time Protocol is required.

1. Install the necessary packages:

```
$ sudo dnf install -y chrony
```

2. Enable the NTP Service

```
$ sudo systemctl enable chronyd
```

2.11 Precision Time Protocol Configuration (optional)

1. Install the necessary packages:

```
$ sudo dnf install -y linuxptp
```

2. Edit `/etc/ptp4l.conf` and add the following options:

```
[global]
slaveOnly        1
verbose          1
time_stamping    software
summary_interval 6
[enp3s0]
```

where `[enp3s0]` should be set to the interface to use for PTP.

3. Edit `/etc/sysconfig/phc2sys` and add the following options:

```
OPTIONS="-a -r -u 60"
```

4. Edit `/etc/sysconfig/ptp4l` and add the following options:

```
OPTIONS="-f /etc/ptp4l.conf -i enp3s0"
```

5. Configure access through the firewall

```
$ sudo firewall-offline-cmd --direct --add-rule ipv4 filter INPUT 0 -p udp --
↪dport 319:320 -j ACCEPT
```

6. Enable the ptp service

```
$ sudo systemctl enable ptp4l
```

CHAPTER THREE

OPERATIONS WORKSTATION CONFIGURATION

The OCS User Interface needs to be run on a system with sufficient graphical rendering capability. At the moment, the Real-time kernel used for device control systems running EtherCAT does not contain the graphics modules necessary to support the user interface. It is recommended to run the user interface in a Mac, connected to the DCS via the network. Future releases will include support for Linux workstations.

3.1 Operating System

Apple Mac systems have the operating system already installed. The User Interface has been tested on the following versions of MacOS:

- MacOS High Sierra
- MacOS Mojave

3.2 Packages

There are very few external packages that are not already installed in MacOS. The application Homebrew can be used to install these:

1. Install Homebrew

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/
↪install/master/install)"
```

   More information can be found on the Homebrew website: <https://brew.sh/>

2. Install common utilities

```
$ brew install wget
```

3.3 Python Installation

1. Check whether Python3 is installed

```
$ python3 --version
```

2. Install Python3, if not already installed

```
$ brew install python3
```

3.4 Node Installation

1. Install Node

Follow the instructions here https://nodesource.com/blog/installing-nodejs-tutorial-mac-os-x/

CHAPTER

FOUR

SOFTWARE DEVELOPMENT KIT (SDK)

The Software Development Kit is distributed as a TAR file and can be downloaded from the GMTO release server.

The SDK should be installed in a **Global GMT Software Location**, defined by the GMT_GLOBAL environment variable (default value: /opt/gmt). A **Local Working Directory**, defined by the GMT_LOCAL variable, is used as a unique workspace for individual developers. The local working directory typically resides underneath the /home/<username> directory.

1. Download the latest SDK distribution:

   ```
   $ sudo wget http://52.52.46.32/srv/gmt/releases/sdk/linux/gmt-sdk-1.6.0.tar.
   ↪gz
   ```

2. Extract the TAR file in the /opt directory, into a new folder for the latest release:

   ```
   $ sudo mkdir /opt/gmt_release_1.6.0
   $ sudo tar -xzvf <gmt-tar.gz> -C /opt/gmt_release_1.6.0
   ```

   where <gmt-tar.gz> is the file downloaded in step 1.

3. Create a symbolic link from the **Global GMT Software Location** to the latest release:

   ```
   $ sudo ln -sfn gmt_release_1.6.0 /opt/gmt
   ```

4. Create a **Local Working Directory**

   ```
   $ mkdir <local_working_dir>
   ```

   where `<local_working_dir>` is in the current users' home directory, for example ~/work. The GMT software modules developed by the user are created in this folder.

5. Add the following lines to your .bash_profile (or .kshrc or .bashrc depending on your preferred shell)

   ```
   $ export GMT_GLOBAL=/opt/gmt
   $ export GMT_LOCAL=<local_working_dir>
   $ source $GMT_GLOBAL/bin/gmt_env.sh
   ```

   This will ensure that the environment variables are correctly configured when opening a new terminal. Please log out and back in for the changes to take effect. To configure the environment for the current shell, run the commands manually.

6. Check the values of the environment variables:

   ```
   $ gmt_env
   ```

7. Install Node Modules

```
$ cd $GMT_GLOBAL
$ npm install

$ cd $GMT_LOCAL
$ cp $GMT_GLOBAL/package.json ./
$ npm install
```

Install global node modules for *Webpack* and *Coffeescript*.

```
$ sudo npm install -g coffeescript webpack webpack-cli coffee-loader
```

8. Initialize the Development Environment:

```
$ cd $GMT_LOCAL
$ gds init
```

The correct folders will be created in the $GMT_LOCAL directory for use when compiling and running modules.

Create a **local javascript library folder** in order to create built bundles for your model files. This folder is also used to install upgraded version of the library.

```
$ mkdir -p $GMT_LOCAL/lib/js
```

9. Create a **modules** directory in $GMT_LOCAL

```
$ cd $GMT_LOCAL
$ mkdir modules
```

10. Clone the HDK and isample modules

This step is relevant for any module that the developer will be working on. It is recommended to fork the central repository in GitHub and cloning your personal fork, instead of working with the GMTO repositories. Any modifications should be submitted through a Pull Request, to be approved and merged after peer review.

```
$ cd $GMT_LOCAL/modules
$ git clone https://github.com/<username>/ocs_hdk_dcs
$ git clone https://github.com/<username>/ocs_isample_dcs
```

Where <username> is your GitHub username, assuming you've forked from the GMTO repository.

Alternatively, use `git clone https://github.com/GMTO/ocs_hdk_dcs` to clone from the central repository.

11. Create the **bundles.coffee** and **ocs_local_bundle.coffee** files, defining the local modules under development

These files may be copied from $GMT_GLOBAL and then edited to reflect the developer's configuration.

```
$ mkdir $GMT_LOCAL/etc/bundles
$ cp $GMT_GLOBAL/etc/bundles/bundles.coffee $GMT_LOCAL/etc/bundles/
$ cp $GMT_GLOBAL/etc/bundles/ocs_local_bundle.coffee $GMT_LOCAL/etc/bundles/
```

Edit **bundles.coffee** to point to the ocs_local_bundle.coffee file

```
module.exports =
    ocs_local_bundle:   {scope: "local",  desc: "GMT iSample and HDK bundle"}
```

Edit **ocs_local_bundle.coffee** to include the isample and HDK modules, or other modules that you are working on

```
module.exports =
name:       "local"
desc:       "List of local development modules"
elements:
    isample_dcs: { active: true, test: false, developer: 'gmto', domain:
↪'idcs' }
    hdk_dcs:     { active: true, test: false, developer: 'gmto', domain:
↪'idcs' }
```

12. Systems that run the User Interface require compiled model files to be used by the Navigator application.

    Build all model files from modules in your ocs_local_bundles definition using webpack. For example:

```
$ cd $GMT_LOCAL/modules/ocs_hdk_dcs/model
$ webpack
$ cd $GMT_LOCAL/modules/ocs_isample_dcs/model
$ webpack
```

CHAPTER FIVE

NEXT STEPS

To start using the SDK with the Hardware Development Kit (HDK), instructions can be found here: hdk_example.

To run the Navigator application and start using the UI framework, see the UI Framework guide: ui_fwk.

[back to top]