

The DDS User Manual v1.0

The DDS User Manual v1.0

Table of Contents

1. Introduction	1
1.1. The Dynamic Deployment System	1
1.2. Features	1
2. Requirements	2
2.1. Server/UI	2
2.2. Workers	2
3. Download	3
3.1. Download location	3
3.2. DDS Version Number Scheme	3
4. Installation	4
4.1. Step #1: Get the source	4
4.1.1. from DDS git repository	4
4.1.2. from DDS source tarball	4
4.2. Step #2: Configure the source	4
4.3. Step #3: Build and install	5
4.4. Step #4: DDS Environment	5
4.5. Step #5: DDS shared Installation	6
5. Configuration	7
6. Quick Start	8
7. Topology	9
7.1. Topology file	9
7.2. Topology file example	9
7.3. Topology XML tag reference	12
8. How to Start	18
8.1. Environment	18
8.2. Server	18
8.3. Deploy Agents	18
8.3.1. Deploy-Agents using: SSH plug-in	18
8.4. Check availability of Agents	18
8.5. Set Topology	18
8.6. Activate Agents	19
9. How to Test	20
9.1. First Section	20
10. Tutorials	21
10.1. Tutorial 1	21
10.1.1. Usage	21
10.1.2. Result	21
10.2. Tutorial 2	21
10.2.1. Usage	22
10.2.2. Result	22
11. Command-line interface	23
dds-server	24
dds-commander	25
dds-user-defaults	26
dds-submit	27
dds-info	28
dds-test	29
dds-topology	30
dds-agent-cmd	31
12. SSH plug-in	32
12.1. Resource definition	32

List of Tables

4.1. DDS configuration variables	4
7.1. Topology XML tags	12
7.2. Topology XML attributes	16
12.1. DDS's SSH plug-in configuration fields	32

List of Examples

7.1. A topology file example	9
12.1. An example of an SSH plug-in configuration file	32

1. Introduction

1.1. The Dynamic Deployment System

1.2. Features

2. Requirements

2.1. Server/UI

DDS UI/Server/WN run on Linux and Mac OS X.

General requirements:

- Incoming connection on dds-commander port (configurable)
- a C++11 compiler
- [cmake](#) 2.8.11 or higher
- [BOOST](#) 1.54 or higher (built by a C++11 compiler, with C++11 enabled)
- shell: [BASH](#)

Additional requirements for SSH plug-in:

- A public key access (or password less, via ssh-agent, for example) to destination worker nodes.

2.2. Workers

General requirements:

- Outgoing connection on dds-commander's port (configurable). This is required by dds-agent to be able to connect to DDS commander server
- shell: [BASH](#)

3. Download

3.1. Download location

Please, use DDS's [Download](#) page to get the latest version and all other versions of DDS.

3.2. DDS Version Number Scheme

DDS version has a form of MAJOR.MINOR(.PATCH), where:

- MAJOR - the major number is increased when there are significant jumps in functionality.
- MINOR - the minor number is incremented when only minor features or significant fixes have been added.
- PATCH - represents a number of commits (patches) to a current major.minor pair.



Note

The DDS's version scheme reflects the fact that DDS is both a production system and a research project. DDS uses odd minor version numbers to denote development releases and even minor version numbers to denote stable releases.

4. Installation

DDS supports Private and Shared installations.

A Private Installation - it is when a user installs DDS for individual usage in his/her local folder. Any Private Installation can be used by other users as well. It's just a matter of file privileges.

A Shared Installation - it is when a site administrator installs DDS in some central location, so it can be shared by many users. This type of installation may be convenient for some users, since they don't need to install DDS by their own. In case of a shared Installation you need to execute one additional step, see [Section 4.5, "Step #5: DDS shared Installation"](#). All the rest is the same as with Private Installations.

Be advised, that in both cases DDS acts identically and always provides private clusters, one for each user. In case of a shared installation, users share only binaries and configurations, but each user gets its own DDS instance and can't disturb other users. Each user can tune DDS by changing the DDS user defaults configuration in `$HOME/.DDS/DDS.cfg`.

4.1. Step #1: Get the source

4.1.1. from DDS git repository

```
git clone https://github.com/FairRootGroup/DDS.git DDS-master
```

4.1.2. from DDS source tarball

Unpack DDS tarball:

```
tar -xzvf DDS-X.Y.Z-Source.tar.gz
```

Tar will create a new directory `./DDS-X.Y.Z-Source`, where `X.Y.Z` represents a version of DDS.

4.2. Step #2: Configure the source

Change to the DDS source directory:

```
cd ./DDS-X.Y.Z-Source
```

You can adjust some configuration settings in the `BuildSetup.cmake` bootstrap file. The following is a list of variables:

Table 4.1. DDS configuration variables

Variable	Description
<code>CMAKE_INSTALL_PREFIX</code>	Install path prefix, prepended onto install directories. (default <code>\$HOME/DDS/[DDS_Version]</code>)
<code>CMAKE_BUILD_TYPE</code>	Set cmake build type. Possible options are: None, Debug, Release, RelWithDebInfo, MinSizeRel (default Release)
<code>BUILD_DOCUMENTATION</code>	Build source code documentation. Possible options are: ON/OFF (default OFF)

Variable	Description
BUILD_TESTS	Build DDS tests. Possible options are: ON/OFF (default OFF)

Now, prepare a build directory for an out-of-source build and configure the source:

```
mkdir build
cd build
cmake -C ../BuildSetup.cmake ..
```



Tip

If for some reason, for example a missing dependency, configuration failed. After you get the issue fixed, right before starting the **cmake** command it is recommended to delete everything in the build directory recursively. This will guaranty a clean build every time the source configuration is restarted.

4.3. Step #3: Build and install

Issue the following commands to build and install DDS:

```
make -j
make install
```



Installation Prefix

Please note, that by default DDS will be installed in $\$HOME/DDS/X.Y.Z$, where X.Y.Z is a version of DDS. However users can change this behavior by setting the install prefix path in the bootstrap script `BuildSetup.cmake`. Just uncomment the setting of `CMAKE_INSTALL_PREFIX` variable and change dummy `MY_PATH_HERE` to a desired path.



WN package

Users have a possibility to additionally build DDS worker package for the local platform. In case if you have same OS types on all of the target machines and don't want to use WN packages from the DDS binary repository, just issue:

```
make -j wn_bin
make install
```

the commands will build and install a DDS worker package for the given platform.

4.4. Step #4: DDS Environment

In order to enable DDS's environment you need to source the `DDS_env.sh` script. Change to your newly installed DDS directory and issue:

```
cd [DDS INSTALL DIRECTORY]
source DDS_env.sh
```

You need to source this script every time before using DDS in a new system shell. Simplify it by sourcing the script in your bash profile.

Now the installation is done. But if you were preparing a shared installation, then please see the [Section 4.5, “Step #5: DDS shared Installation”](#) as well.

4.5. Step #5: DDS shared Installation

TODO

5. Configuration

TODO

6. Quick Start

- [Download DDS source tarball.](#)
- [Install DDS from source.](#)

```
cd [DDS INSTALLATION]
source DDS_env.sh
dds-server start
dds-submit --rms localhost
dds-info -n
dds-info -l
dds-topology --set $DDS_LOCATION\tests\tutorial1_topo.xml
dds-topology --activate
```

Enable DDS environment.

Start DDS commander server.

Deploy DDS agents on the localhost.

Use dds-info to find out a number of agents, which are online.

Use dds-info to check detailed information about agents.

Set a topology.

Activate (run) the previously defined topology.

7. Topology

The definition of the topology by the user has to be simple and powerful at the same time. Therefore a simple and powerful so called topology language has been developed.

The basic building block of the system is a *task*. Namely, a task is a user defined executable or a shell script, which will be deployed and executed by DDS on a given Resource Management System.

In order to describe dependencies between tasks in a topology we use *properties*. In run-time properties will be turned into simple key-value pairs. DDS uses its key-value propagation engine to make sure, that once property is set by one task, it will be propagated to other depended tasks. DDS treats values of properties as simple strings and doesn't do any special treatment/preprocessing on them. So, basically tasks can write anything into the values of properties (256 char max). Any of depended tasks can set properties. Anytime property is set it will be propagated to other depended tasks. (see for details TODO:"key-value propagation").



Tip

For example, if one task needs to connect with another task they can have the same property. A "server" task can store its TCP/IP port and host in the property. Once the property set, DDS will notice that and propagate it to other tasks.

Tasks can be grouped into *collections* and *groups*. Both collections and groups can be used to group several tasks. The main difference between collections and groups is that a collection requests from DDS to execute its tasks on the same physical machine, if resource allow that. This is useful if tasks suppose to communicate a lot or they want to access the same shared memory. A set of tasks and task collections can be also grouped into task groups. Another difference between groups and collection is that only groups can define multiplication factor for all its child elements.

Main group defines the entry point for task execution. Only main group can contain other groups.

7.1. Topology file

At the moment we use an XML based file to store topologies. XML is chosen because it can be validated against XSD schema. DDS's XSD schema file can be found in \$DDS_LOCATION/share/topology.xsd.

```
<topology id="myTopology">
  [... Definition of tasks, properties, and collections ...]
  <main name="main">
    [... Definition of the topology itself, where also groups can be defined ...]
  </main>
</topology>
```

The file is basically divided on two parts: declaration and main part.

All properties, tasks and collections should be defined in the declaration part of the file. Users can define any number of topology entities in that block, even some, which are not going to be used in the main block.

In the main block the topology itself is defined. Groups and multiplication factors are also defined in main block.

7.2. Topology file example

Example 7.1. A topology file example

```
<topology id="myTopology">
  <var id="appNameVar" value="app1 -l -n --taskIndex %taskIndex% --collectionIndex %co
```

```
<var id="nofGroups" value="10" />

<property id="property1" />
<property id="property2" />

<declrequirement id="requirement1">
  <hostPattern type="hostname" value="+.gsi.de"/>
</declrequirement>

<decltask id="task1">
  <requirement>requirement1</requirement>
  <exe reachable="true">${appNameVar}</exe>
  <env reachable="false">env1</env>
  <properties>
    <id access="read">property1</id>
    <id access="readwrite">property2</id>
  </properties>
</decltask>
<decltask id="task2">
  <exe>app2</exe>
  <properties>
    <id access="write">property1</id>
  </properties>
</decltask>

<declcollection id="collection1">
  <requirement>requirement1</requirement>
  <tasks>
    <id>task1</id>
    <id>task2</id>
    <id>task2</id>
  </tasks>
</declcollection>

<declcollection id="collection2">
  <tasks>
    <id>task1</id>
    <id>task1</id>
  </tasks>
</declcollection>

<main id="main">
  <task>task1</task>
  <collection>collection1</collection>
  <group id="group1" n="${nofGroups}">
    <task>task1</task>
    <collection>collection1</collection>
    <collection>collection2</collection>
  </group>
  <group id="group2" n="15">
    <collection>collection1</collection>
  </group>
</main>

</topology>
```

DDS allows to define variables which later can be used inside the topology file. During the preprocessing all variable are replaced with their values. Variables are defined using the `var` tag which has two attributes `id` and

value. Inside the file variable can be used as follows $\${variable_name}$. In the above example we define two variables $\${appNameVar}$ and $\${nofGroups}$.

When a particular task or collection is multiplied, sometimes it is necessary for the user to get the index of the task or collection instance. This can be done in two different ways. In the definition of the executable path one can use special tags $\%taskIndex\%$ and $\%collectionIndex\%$ to get the task and collection index respectively. Before the task execution these tags are replaced with real values. The second possibility is to get task and collection index from environment. Two environment variables are defined for each task $\$DDS_TASK_INDEX$ and $\$DDS_COLLECTION_INDEX$.

For each user task a set of environment variables is populated.

Populated environment variables

- $\$DDS_TASK_PATH$ - full path to the user task, for example, *main/group1/collection_12/task_3*
- $\$DDS_GROUP_NAME$ - ID of the parent group.
- $\$DDS_COLLECTION_NAME$ - ID of the parent collection if any.
- $\$DDS_TASK_NAME$ - ID of the task.
- $\$DDS_TASK_INDEX$ - index of the task.
- $\$DDS_COLLECTION_INDEX$ - index of the collection.

In the example above we define 2 properties - *property1* and *property2*. As you can see the `property` tag is used to define properties. *id* attribute is required and has to be unique for all properties.

Requirements is another nice feature of DDS. Requirements is a way to tell the DDS that a task or collection has to be deployed to a particular computing node. As of now only host name or worker node name which is defined in the SSH configuration file requirements are supported. Requirements are defined using `declrequirement` tag. *id* attribute is required and has to be unique for all requirements. Pattern of the host name is defined using `hostPattern` tag which attribute value can be either a full host name or a regular expression which matches the required host name. `type` defines the type of the `hostPattern`. Can be either *hostname* if the requirement is defined based on the host name or *wname* if the requirement is defined based on the SSH worker node name.

In the next block we define tasks. For this the `decltask` tag is used. A task must also have the *id* attribute which is required and has to be unique for all declared tasks. The `requirement` element is optional and specifies the already declared requirement for the task. The `exe` element defines path to executable. The path can include program options, even options with quotes. DDS will automatically parse the path and extractor program options in runtime. The `exe` tag has an optional attribute `reachable`, which defines whether executable is available on worker nodes. If it is not available, then DDS will take care of delivering it to an assigned worker in run-time.

In case when there is a script, that, for example sets environment, has to be executed prior to main executable one can specify it using the `env` element. The `env` tag also have `reachable` attribute.

If a task depends on some properties this can be specified using the `properties` tag together with a list of *id* elements which specify ID of already declared properties. Each property has an optional `access` attribute which defines whether user task will read (`read`), write (`write`) or both read and write (`readwrite`) a property. Default is `readwrite`.

Collections are declared using the `declcollection` tag. It contains a list of `task` tags with IDs which specified already declared tasks. Task has to be declared before it can be used in the collection. As for the task collection has an optional `requirement` element which is used to specify the requirement for the collection. If the requirement defined for both task and collection than collection requirement has higher priority and is used for deployment.

The `main` tag declares the topology itself. In the example our main block consists of one task (*task1*), one collection (*collection1*) and two groups (*group1* and *group2*).

A group is declared using the `group` tag. It has a required attribute *id*, which is used to uniquely identify the group and optional attribute *n*, which defines multiplication factor for the group. In the example *group1* consists of one task (*task1*) and two collections (*collection1* and *collection2*). *group2* consists of one collection (*collection1*).

7.3. Topology XML tag reference

Table 7.1. Topology XML tags

Tag	Description
topology	<p><i>Parents:</i> No</p> <p><i>Children:</i> property, task, collection, main</p> <p><i>Attributes:</i> id</p> <p><i>Description:</i> Declares a topology.</p> <pre><topology id="myTopology"> [... Definition of tasks, properties, collections and groups ...] </topology></pre>
var	<p><i>Parents:</i> topology</p> <p><i>Children:</i> No</p> <p><i>Attributes:</i> id, value</p> <p><i>Description:</i> Declares a variable which can be used inside the topology file as $\\${variable_name}$.</p> <pre><var id="var1" value="value1"/> <var id="var2" value="value2"/></pre>
property	<p><i>Parents:</i> topology</p> <p><i>Children:</i> No</p> <p><i>Attributes:</i> id</p> <p><i>Description:</i> Declares a property.</p> <pre><property id="property1"/> <property id="property2"/></pre>
declrequirement	<p><i>Parents:</i> topology</p> <p><i>Children:</i> hostPattern</p> <p><i>Attributes:</i> id</p> <p><i>Description:</i> Declares a requirement for tasks and collections.</p>

Tag	Description
	<pre><declrequirement id="requirement1"> <hostPattern value="+.gsi.de"/> </declrequirement></pre>
hostPattern	<p><i>Parents:</i> declrequirement</p> <p><i>Children:</i> no</p> <p><i>Attributes:</i> value, type</p> <p><i>Description:</i></p> <p>Declares a pattern of the host name.</p> <pre><hostPattern type="hostname" value="+.gsi.de"/></pre>
decltask	<p><i>Parents:</i> topology</p> <p><i>Children:</i> exe, env, requirement, properties</p> <p><i>Attributes:</i> id</p> <p><i>Description:</i></p> <p>Declares a task.</p> <pre><decltask id="task1"> <exe reachable="true">appl -l -n</exe> <env reachable="false">env1</env> <requirement>requirement1</requirement> <properties> <id access="read">property1</id> <id access="readwrite">property2</id> </properties> </decltask></pre>
declcollection	<p><i>Parents:</i> topology</p> <p><i>Children:</i> task</p> <p><i>Attributes:</i> id</p> <p><i>Description:</i></p> <p>Declares a collection.</p> <pre><declcollection id="collection1"> <task>task1</task> <task>task1</task> </declcollection></pre>
task	<p><i>Parents:</i> collection, group</p> <p><i>Children:</i> No</p>

Tag	Description
	<p><i>Attributes:</i> No</p> <p><i>Description:</i></p> <p>Specifies the unique ID of the already defined task.</p> <pre><task>task1</task></pre>
collection	<p><i>Parents:</i> group</p> <p><i>Children:</i> No</p> <p><i>Attributes:</i> No</p> <p><i>Description:</i></p> <p>Specifies the unique ID of the already defined collection.</p> <pre><collection>collection1</collection></pre>
group	<p><i>Parents:</i> main</p> <p><i>Children:</i> task, collection</p> <p><i>Attributes:</i> id, n</p> <p><i>Description:</i></p> <p>Declares a group.</p> <pre><group id="group1" n="10"> <task>task1</task> <collection>collection1</collection> <collection>collection2</collection> </group></pre>
main	<p><i>Parents:</i> topology</p> <p><i>Children:</i> task, collection, group</p> <p><i>Attributes:</i> id</p> <p><i>Description:</i></p> <p>Declares a main group.</p> <pre><main id="main"> <task>task1</task> <collection>collection1</collection> <group id="group1" n="10"> <task>task1</task> <collection>collection1</collection> <collection>collection2</collection> </group></pre>

Tag	Description
	<code></main></code>
exe (required)	<p><i>Parents:</i> decltask</p> <p><i>Children:</i> No</p> <p><i>Attributes:</i> reachable</p> <p><i>Description:</i></p> <p>Defines path to the executable or script for the task.</p>
	<code><exe reachable="true">app1 -l -n</exe></code>
env (optional)	<p><i>Parents:</i> decltask</p> <p><i>Children:</i> No</p> <p><i>Attributes:</i> reachable</p> <p><i>Description:</i></p> <p>Defines the path to script that has to be executed prior to main executable.</p>
	<code><env reachable="false">setEnv.sh</env></code>
properties (optional)	<p><i>Parents:</i> decltask</p> <p><i>Children:</i> id</p> <p><i>Attributes:</i> No</p> <p><i>Description:</i></p> <p>Defines a list of dependent properties.</p>
	<pre><properties> <id>property1</id> <id>property2</id> </properties></pre>
id (required)	<p><i>Parents:</i> properties</p> <p><i>Children:</i> No</p> <p><i>Attributes:</i> access</p> <p><i>Description:</i></p> <p>Defines an ID of the already declared property.</p>
	<code><id>property1</id></code>

Table 7.2. Topology XML attributes

Attribute	Description
id	<p><i>Use:</i> required</p> <p><i>Default:</i> No</p> <p><i>Tags:</i> topology, property, decltask, declcollection, group, main</p> <p><i>Restrictions:</i></p> <p>String with minimum length of 1 character.</p> <p><i>Description:</i></p> <p>Defines identifier (ID) for topology, property, task, collection and group. ID has to be unique within its scope, i.e. ID for tasks has to be unique only for tasks.</p> <pre><topology id="myTopology"></pre>
reachable	<p><i>Use:</i> optional</p> <p><i>Default:</i> true</p> <p><i>Tags:</i> exe, env</p> <p><i>Restrictions:</i> true false</p> <p><i>Description:</i></p> <p>Defines if executable or script is available on the worker node.</p> <pre><exe reachable="true">app -l</exe> <env>env1</env></pre>
n	<p><i>Use:</i> optional</p> <p><i>Default:</i> 1</p> <p><i>Tags:</i> group</p> <p><i>Restrictions:</i> unsigned integer 32-bit which is more or equal to 1</p> <p><i>Description:</i></p> <p>Defines multiplication factor for group.</p> <pre><exe reachable="true">app -l</exe> <env>env1</env></pre>
access	<p><i>Use:</i> optional</p> <p><i>Default:</i> readwrite</p> <p><i>Tags:</i> id</p> <p><i>Restrictions:</i> read write readwrite</p>

Attribute	Description
	<p><i>Description:</i></p> <p>Defines access type from user task to properties.</p> <pre data-bbox="386 360 1401 450"><id access="read">property1</id></pre>
type	<p><i>Use:</i> required</p> <p><i>Tags:</i> hostPattern</p> <p><i>Restrictions:</i> hostname wnname</p> <p><i>Description:</i></p> <p>Defines the type of the host pattern.</p> <pre data-bbox="386 757 1401 842"><hostPattern type="wnname" value="wn2" /></pre>

8. How to Start

8.1. Environment

In order to enable DDS environment you need to source the `DDS_env.sh` script. The script is located in the directory where you installed PoD.

```
cd [DDS INSTALLATION]
source DDS_env.sh
```

8.2. Server

Use the `dds-server` command to *start/stop/status* DDS servers.

```
dds-server start
```

8.3. Deploy Agents

In order to deploy agents you can use different DDS plug-ins.

8.3.1. Deploy-Agents using: SSH plug-in

DDS's [SSH plug-in](#) is the best and the fastest way to deploy DDS agents. When you don't have an RMS or you want to use a Cloud based system or even if you want just to use resources around you, like computers of your colleagues, then the plug-in is the best way to go.

First of all you need to [define resources](#).

Then use `dds-submit` to deploy DDS agents on the given resources:

```
dds-submit --rms ssh --ssh-rms-cfg FULL_PATH_TO_YOUR_SSHPLUGIN_RESOURCE_FILE
```

8.4. Check availability of Agents

Using `dds-info` you can query different kinds of information from DDS. For example you can check how many agents are already online:

```
dds-info -n
```

or query more detailed info about agents:

```
dds-info -l
```

8.5. Set Topology

To assign a topology to your deployment use:

```
dds-topology --set FULL_PATH_TO_YOUR_TOPOLOGY_FILE
```

8.6. Activate Agents

Once you get enough online agents, you can activate them. Activation of agents means, that DDS will use the given topology to distribute user tasks across available resources (agents):

```
dds-topology --activate
```

DDS will automatically check whether available resources are actually sufficient to execute the given topology.

9. How to Test

xxxx

9.1. First Section

xxxx

10. Tutorials

10.1. Tutorial 1

This tutorial demonstrates how to deploy a simple topology of 2 types of tasks (TaskTypeOne and TaskTypeTwo). By default, there will be deployed one instance of TaskTypeTwo and 5 instances of TaskTypeOne. Additionally TaskTypeTwo subscribes on key-value property from TaskTypeOne, which name is TaskIndexProperty. Once TaskTypeTwo receives values of TaskIndexProperty from all TaskTypeOne, it will set the ReplyProperty property. Number of instances can be changed in the topology file (tutorial1_topo.xml) using --instances option of TaskTypeOne. Please note that number of worker nodes in the SSH-plugin configuration file (tutorial1_hosts.cfg) has to be changed accordingly.

After DDS is installed the tutorial can be found in `$DDS_LOCATION/tutorials/tutorial1`

The source code of tasks is located in "`DDS_SRC_DIR`" /`dds-tutorials/dds-tutorial1`

Files of the tutorial

- task-type-one: executable of the task TaskTypeOne
- task-type-two: executable of the task TaskTypeTwo
- tutorial1_topo.xml: a topology file
- tutorial1_hosts.cfg: a configuration file for DDS SSH plug-in

10.1.1. Usage

Before running the tutorial make sure that: 1) Default working directory `~/tmp/dds_wn_test` must exist before running the tutorial. The directory can be changed in `tutorial1_hosts.cfg`. 2) SSH passwordless access to the localhost is required.

```
cd $DDS_LOCATION/tutorials/tutorial1
dds-server start -s
dds-submit -r ssh --ssh-rms-cfg tutorial1_hosts.cfg
dds-topology --set tutorial1_topo.xml
dds-topology --activate
```

10.1.2. Result

To check the result, change to `~/tmp/dds_wn_test`. If the default setup was used, then there will be WN directories located: `wn`, `wn_1`, `wn_2`, `wn_3`, `wn_4`, `wn_5`.

DDS catches output of tasks and saves it in log files under names `[task_name]_[date_time]_out[err].log`. For example: `TaskTypeOne_2015-07-16-11-44-42_6255430612052815609_out.log`

10.2. Tutorial 2

This tutorial demonstrates how to use DDS custom commands for user task and for utility.

After DDS is installed the tutorial can be found in `$DDS_LOCATION/tutorials/tutorial2`

The source code of tasks is located in "`DDS_SRC_DIR`" /`dds-tutorials/dds-tutorial2`

Files of the tutorial

- `task-custom-cmd`: user task which receives and send DDS custom commands
- `ui-custom-cmd`: utility which connects to DDS commander and send custom commands to user tasks
- `tutorial2_topo.xml`: a topology file
- `tutorial2_hosts.cfg`: a configuration file for DDS SSH plug-in

10.2.1. Usage

Before running the tutorial make sure that: 1) Default working directory `~/tmp/dds_wn_test` must exist before running the tutorial. The directory can be changed in `tutorial1_hosts.cfg`. 2) SSH passwordless access to the localhost is required.

```
cd $DDS_LOCATION/tutorials/tutorial2
dds-server start -s
dds-submit -r ssh --ssh-rms-cfg tutorial2_hosts.cfg
dds-topology --set tutorial2_topo.xml
dds-topology --activate
ui-custom-command
```

10.2.2. Result

To check the result, change to `~/tmp/dds_wn_test`. If the default setup was used, then there will be WN directories located: `wn`, `wn_1`, `wn_2`, `wn_3`, `wn_4`, `wn_5`.

DDS catches output of tasks and saves it in log files under names `[task_name]_[date_time]_out[err].log`. For example: `TaskTypeOne_2015-07-16-11-44-42_6255430612052815609_out.log`

After executing **ui-custom-command** there will be an output to the console with receiving and sending custom commands. Also check output files of tasks.

11. Command-line interface

Name

dds-server — wraps to manage DDS commander server daemon
UNIX/Linux/OSX

Synopsis

```
dds-server {[[start] | [-s]] [[restart] | [-s]] | [stop] | [status]}
```

Description

Using this command users can *start/stop/restart/status* DDS commander server. The command is actually a wrapper for the [dds-commander](#) command (DDS commander server) which become a daemon process when started.

Options

start

Start DDS commander server.

At the server start DDS will detect availability of DDS WN bin. packages and download them from the DDS repository if they are missing. Users can provide an additional parameter *-s* (the parameter can be specified with *start* and *restart*). When the parameter is provided, DDS will check availability of a binary package compatible with the local system only.

To build a binary package for the local system, just issue:

```
make -j wn_bin
make -j install
```

restart

Restart DDS commander server.

stop

Stop DDS commander server.

status

Request the status information. It will show process id of the DDS commander server daemon and the TCP port it listens on.

Name

dds-commander — manages DDS facility
UNIX/Linux/OSX

Synopsis

```
dds-commander [[-h, --help] | [-v, --version]] {[start] | [stop]}
```

Description



Warning

The command must not be used directly. Please use the [dds-server](#) command instead.

Name

`dds-user-defaults` — get and set global DDS options
UNIX/Linux/OSX

Synopsis

```
dds-user-defaults [[-h, --help] | [-v, --version] | [-V, --verbose] | [-p, --path] | [-d, --default]] [-c, --config arg] [-f, --force] [[--key arg] | [--wrkpkg] | [--wrkscript] | [--rms-sandbox-dir] | [--user-env-script] | [--server-info-file]]
```

Description

The **dds-user-defaults** command can be used to get and set global DDS options. It also can be used to get different static settings, related to the current deployment.

Options

- `-h, --help`
Shows usage options.
- `-v, --version`
Shows version information.
- `-V, --verbose`
Causes the command to verbose additional information and error messages.
- `-p, --path`
Shows default DDS user defaults config file path.
- `-d, --default`
Generates a default DDS configuration file.
- `-f, --force`
If the destination file exists, removes it and creates a new file, without prompting for confirmation. Can only be used with the `-d, --default` options.
- `-c, --config arg`
This options can be used together with other options to specify non-default location of the DDS configuration file. By default the command uses `~/ .DDS/DDS.cfg`.
- `--key arg`
Gets a value for the given key from the DDS user defaults.
- `--wrkpkg`
Shows the full path of the worker package. The path must be evaluated before use.
- `--wrkscript`
Shows the full path of the worker script. The path must be evaluated before use.
- `--rms-sandbox-dir`
Shows the full path of the RMS sandbox directory. It returns `server.sandbox_dir` if it is not empty, otherwise `server.work_dir` is returned. The path must be evaluated before use.
- `--user-env-script`
Shows the full path of user's environment script for workers (if present). The path must be evaluated before use.
- `--server-info-file`
Shows the full path of the DDS server info file. The path must be evaluated before use.

Name

dds-submit — submits and activates DDS agents
UNIX/Linux/OSX

Synopsis

```
dds-submit [[-h, --help] | [-v, --version] | [-c, --config arg] | [-r, --rms arg] | [--ssh-rms-cfg arg] | [-n, --number arg]
```

Description

The command is used to submit DDS agents to allocate resources for user tasks. Once enough agents are online use the [dds-topology](#) command to activate the agents - i.e. distribute user tasks across agents and start them.

Options

-h, --help

Shows usage options.

-v, --version

Shows version information.

-c, --config *arg*

Only for advanced users. This options can be used to specify the location of the **dds-submit** configuration file.

--r, --rms *arg*

Defines a destination resource management system. At the moment only the SSH plug-in is supported, therefore *arg* should be equal to: ssh or localhost

--ssh-rms-cfg *arg*

Specifies an SSH plug-in resource definition file.

-n, --number *arg*

It can only be used with "localhost". It defines how many agents to spawn. If omitted, then the default value will be used, which is equal to a number of logical cores of a given host.

Name

dds-info — can be used to query different kinds of information from DDS commander server
UNIX/Linux/OSX

Synopsis

```
dds-info [[-h, --help] | [-v, --version]] [--commander-pid] | [--status] | [-n, --agents-number] | [-l, --agents-list]]
```

Description

The command can be used to query different kinds of information from DDS commander server.

Options

- `--commander-pid`
Return the pid of the commander server
- `--status`
Query current status of DDS commander server
- `-n, --agents-number`
Returns a number of online agents
- `-l, --agents-list`
Show detailed info about all online agents

Name

dds-test — allows to test the running system
UNIX/Linux/OSX

Synopsis

```
dds-test {[-t, --transport]}
```

Description

This command allows test the system after run.

Options

-t, --transport
Performs transport test.

Name

dds-topology — topology related commands
UNIX/Linux/OSX

Synopsis

```
dds-topology [[-h, --help] | [-v, --version] | [-V, --verbose] | [--set arg] | [--disable-validation] | [--activate] | [--stop] | [--validate arg]]
```

Description

This command allows to perform topology related tasks.

Options

- h, --help
Shows usage options.
- v, --version
Shows version information.
- V, --verbose
Causes the command to verbose additional information and error messages.
- set *arg*
Sets the given topology for the currently running DDS session.
- disable-validation
Switches off topology validation.
- activate
Requests DDS to activate agents, i.e. distribute and start user tasks.
- stop
Requests DDS to stop execution of user tasks. Stop the active topology.
- validate *arg*
Validates topology file against DDS's XSD schema.

Name

dds-agent-cmd — send commands to agent
UNIX/Linux/OSX

Synopsis

```
dds-agent-cmd [[-h, --help] | [-v, --version] | [command, --command arg]] {[getlog arg] | [-a, --all]} | [update-key arg] {[--key arg] | [--value arg]}
```

Description

This command allows to send commands to DDS agents.

Options

getlog *arg*

Download log files from all active workers.

-a, --all

Download all log files.

update-key *arg*

TODO

--key

TODO

--value

TODO

12. SSH plug-in

12.1. Resource definition

DDS's SSH plug-in is capable to deploy DDS agents on any resource machine available for password-less access (public key, ssh agent, etc.) To define resources for the SSH plug-in we use a comma-separated values (CSV) configuration file. Fields are normally separated by commas. If you want to put a comma in a field, you need to put quotes around it. Also 3 escape sequences are supported.

Table 12.1. DDS's SSH plug-in configuration fields

1	2	3	4	5
id (must be any unique string). This id string is used just to distinguish different DDS workers in the plug-in.	a host name with or without a login, in a form: login@host.fqdn	additional SSH params (could be empty)	a remote working directory	RESERVED

Example 12.1. An example of an SSH plug-in configuration file

```
r1, anar@lxg0527.gsi.de, -p24, /tmp/test, 0
# this is a comment
r2, user@lxi001.gsi.de, , /home/user/dds, 0
125, user2@host, , /tmp/test,
```