

CTL in 5 minutes

Table of contents

| | |
|-------------------------|---|
| 1 1. Define | 2 |
| 2 2. Build..... | 3 |
| 3 3. Run..... | 4 |
| 4 Parting thoughts..... | 5 |

This document describes how to use CTL to define your own commands and assumes you have already installed it. If not, first [download CTL](#) and follow the installation instructions. After it is installed, type the following command at your shell:

```
ctl
```

You should see a listing of stock modules and their commands.

1. 1. Define

To create a new module of commands, you will use the ProjectBuilder `create-type` command.

Run the `create-type` command accepting the defaults (input what is shown in bold text):

```
ctl -m ProjectBuilder -c create-type
Base directory where module source files reside [/tmp/ctier/ctl/src]
Target directory where build files are generated [/tmp/ctier/target]
Name of supertype: [Managed-Entity]
Name of type:
poly
Description of type:
Says hello
Creating module definition files in directory: /tmp/ctier/ctl/src ...
Initializing type module from template dir:
/tmp/ctier/ctl/depots/default/modules/ProjectBuilder/templates/boilerplate
...
Created dir: /tmp/ctier/ctl/src/modules/poly
Copying 1 file to /tmp/ctier/ctl/src/modules/poly
Creating directory structure...
Created dir: /tmp/ctier/ctl/src/modules/poly/bin
Created dir: /tmp/ctier/ctl/src/modules/poly/commands
Created dir: /tmp/ctier/ctl/src/modules/poly/objects
Created dir: /tmp/ctier/ctl/src/modules/poly/templates
Copying
/tmp/ctier/ctl/depots/default/modules/ProjectBuilder/templates/types/Deployment
to /tmp/ctier/ctl/src/modules/poly ...
Initializing type module from template dir:
/tmp/ctier/ctl/depots/default/modules/ProjectBuilder/templates/types/Deployment
...
Copying 2 files to /tmp/ctier/ctl/src/modules/poly
Define commands and attributes in this file:
/tmp/ctier/ctl/src/modules/poly/type.xml
```

You will notice the `create-type` command created a standard directory structure and some initial files.

```
poly
```

```
--bin
--commands
--lib
  `--command.xml
--objects
--templates
--type.xml
```

The `type.xml` file will be used to define your new commands.

Open the `$CTL_BASE/src/modules/poly/type.xml` in your preferred text editor and inside the `<commands></commands>` tags paste in the following XML fragment:

```
<command name="hello" description="say hello with Ant."
  command-type="AntCommand" is-static="true">
  <implementation>
    <!--
      ** ant tasks here
    -->
    <echo message="hello world!"/>
  </implementation>
</command>
```

The fragment above, defines a command named "hello".

Two immediate things to notice. First, the `command-type` attribute has been set to "AntCommand". This tells ProjectBuilder that you are defining a command with an Ant implementation. Second, are the `<implementation></implementation>` tags. Between these tags you are free to type in any ant code that would normally type inside an Ant target. In this example, the Ant echo task was used to produce a "hello world" message.

By default, CTL has already defined its own Ant tasks but has also ensured the ant-contrib tasks are defined, too.

2. 2. Build

The next step in the process is to build your module. Run ProjectBuilder's `build-type` command to generate the module files for your new command (accept the defaults again):

```
ctl -m ProjectBuilder -c build-type -- -type poly -deploy
Base directory where module source files reside [/tmp/ctier/ctl/src]
Target directory where build files are generated [/tmp/ctier/target]
Building type using the buildmodule.xml via classloader
```

```

converting type.xml for module: poly
generating handlers...
packaging module: poly
Copying 1 file to /tmp/ctier/ctl/src/modules/poly
Moving 1 file to /tmp/ctier/ctl/src/modules/poly
Building jar: /tmp/ctier/target/modules/poly-1.jar
deploying new build of poly module to local installation ...
Extracting /tmp/ctier/target/modules/poly-1.jar to:
/tmp/ctier/ctl/depots/default/modules/poly
Expanding: /tmp/ctier/target/modules/poly-1.jar into
/tmp/ctier/ctl/depots/default/modules/poly

```

What did build-type do?

The build-type command reads the type.xml file and generates a number of files to make it a runnable module.

The listing below shows four new files that now exist in the module structure (new ones are marked with an asterisk). The properties files contain various bits of metadata used by the framework. The "hello.xml" is the command handler for the hello command you defined in the type.xml.

```

poly
|--commands.properties*
|--module.properties*
|--type.properties*
|--bin
|--commands
|   |--hello.xml*
|--lib
|   |--command.xml
|--objects
|--templates
|--type.xml

```

Because these files are re-generated each time you run build-type do not edit them manually as you will lose your changes.

Finally, the -deploy flag stipulates to also install the module into the framework making it ready for use.

3.3. Run

With the module built and installed its commands are ready to run. Because it was installed, you will see it in a CTL listing. Run `ctl` without arguments

```
ctl
poly: Says hello
commands: [hello]
-- snip --
```

Run the command:

```
ctl -m poly -c hello
hello world!
```

That's it!. Follow the same three steps for each new command you wish to add.

4. Parting thoughts

We hope this brief introduction has made you feel more comfortable with the module and command definition process. As you have seen, the process is typical of most development processes: *edit* the type.xml, *build* it with `build-type`, and *run* it via `ctl`. Note this has been a very brief tutorial. From here you will benefit from more in depth explanations in the general documentation sections.

Being an Ant user, you'll also want to acquaint yourself with the Ant tasks and types CTL provides, as well as the remainder of this section.

[Next: CTL for Ant by Example#](#)