

# Managed-Entity

**Represents an abstract manageable object.**

## Table of contents

1 Overview.....	2
2 Design.....	2
3 Constraints.....	3
4 Attributes.....	3
4.1 Defaults for Imported Attributes.....	3
5 Commands.....	3
5.1 Get-Properties.....	3
5.2 Get-Supers.....	5
5.3 Install-Module.....	5
5.4 Install.....	6
5.5 Properties.....	7
5.6 Purge.....	8

## 1. Overview

**Managed-Entity:** *Represents an abstract manageable object.*

A Managed-Entity represents any resource that can benefit from or require a management interface. No assumption is made about how or where Managed-Entity is applied but its effective use has been found for a wide variety of administration tasks.

The Managed-Entity type defines an abstraction useful for automating tasks such as control, configuration and measurement.

- *control:* Control operations are defined as "commands". Managed-Entity establishes several basic commands useful for bootstrapping and other basic administration needed by the CTL framework. Users can override the base commands and or define additional commands.
- *configuration:* Configuration information falls into two general categories: attributes and dependencies. *Attributes* : Characteristics about the Managed-Entity itself are described as a named property called an attribute. An attribute can have a default value defined at the type level but overridden at the instance level. *Dependencies* : A Managed-Entity can aggregate other Managed-Entities to form composition hierarchies. Dependency relationships are characterized as either parent and child.
- *measurement:* The managed object may have statistical data and metrics available at run time or accumulate over time.

### Control Hierarchy

Composed Managed-Entities represent a control hierarchy. Under this arrangement, a top level Managed-Entity dispatches actions to its child Managed-Entity dependencies and each child Managed-Entity in turn dispatches that same action to their own children. This dispatching pattern is easiest when it is the same command names and arguments passed down the hierarchy (eg, polymorphic).

### Model

The Managed-Entity can also be used as a low-level conceptual model from which higher order models can be defined. Higher order models are established by defining subtypes of Managed-Entity that are domain specific and work together via pre-defined control workflows.

## 2. Design

**Super Type**  
Assembly

Role	<b>Abstract.</b> (Objects cannot be created.)
Instance Names	<b>Unique</b>
Notification	<b>false</b>
Template Directory	<code>\${module.dir}/templates</code>
Data View	Children, proximity: <b>1</b>
Logger Name	Managed-Entity

### 3. Constraints

No constraints

### 4. Attributes

Managed-Entity declares several basic attributes.

#### 4.1. Defaults for Imported Attributes

Name	Default	Description
<code>dirlist</code>	<code>bin,logs,var</code>	List of subdirectories to create

### 5. Commands

**Note:**

Commandline options displayed in square brackets "[ ]" are optional. If an option expects arguments, then angle brackets are shown after the option "<>". Any default value is shown within the brackets.

Managed-Entity is a base type establishing several administrative operations. Your subtypes will declare additional actions. See the [Developing a module](#) page for instructions on creating subtypes and defining commands.

#### 5.1. Get-Properties

*Gets the properties for the object.*

Retrieves the configuration data for the Managed-Entity object. This data is made accessible to your command execution context.

Access

Managed-Entity includes this base implementation for accessing and storing property data driven by the following two parameters.

- `entity.properties.file`: This property references the file where property data will be stored. By default it is stored as `${entity.instance.dir}/var/entity.properties`.
- `entity.properties.url`: This property references a URL that can be used to retrieve the file.

The locations for either property above can be modified via the framework configuration file, `${antdepo.base}/etc/entity.properties`.

For efficiency sake, this implementation uses a timestamp check to compare if the `${entity.properties.url}` is newer than `${entity.properties.file}`, and if so, will download a new copy.

#### Data Format

CTL uses Java properties as its own data format but you are not required to follow that convention. Since it is only your commands that load and interpret the data, you are free to choose alternative formats such as XML, YAML, CSV, etc.

If you do choose, another format, be sure to override the [Properties](#) command to provide a pretty format for it.

#### Usage

```
Get-Properties [-destfile <>] [-force] [-noexpand] [-print]
```

#### Options

Option	Description
<code>destfile</code>	<i>file to save property data</i> If the property is unset, the data will be saved to the value of <code>\${entity.properties.file}</code> .
<code>force</code>	<i>Remove old properties file first</i>
<code>noexpand</code>	<i>Do not expand the property values</i> Normally, the data is expanded to resolve any embeded property references. This is generally preferable but can be avoided if <code>-noexpand</code> flag is used.
<code>print</code>	<i>flag specifying Properties command should run after properties are saved</i>

## 5.2. Get-Supers

*Gets the supertype module.*

Looks up supertype module for the given module and for each one found, requests and installs it via Install-Module.

**Note:**

Since this command is only utilized by [Install-Module](#) and not normally called directly, it may be removed and made an internal function of Install-Module.

**Usage**

```
Get-Supers [-depot <>] [-module <>]
```

**Options**

Option	Description
depot	<i>project depot</i>
module	<i>module name</i>

## 5.3. Install-Module

*Installs the specified module.*

Installs the specified module. Downloads from the repository and extracts it.

Access

Managed-Entity includes this base implementation for retrieving and installing modules driven by the following two parameters.

- `modules.dir`: This property references the directory where the module archive will be extracted.
- `modules.url`: This property references a base URL that can be used to retrieve the modules archives from a repository.

The locations for either property above can be modified via the framework configuration file, `${antdepo.base}/etc/modules.properties`.

For efficiency sake, this implementation uses a timestamp check to compare if the module archive found in `${modules.url}` is newer than the locally installed one, and if so, will download a new copy and install the new one.

**Usage**

```
Install-Module [-depot <>] [-downloadaddr
<${framework.var.dir}/tmp/downloads/${context.depot}>]
[-module <>] [-modulesurl <>] [-version <head>]
```

## Options

Option	Description
depot	<i>project depot</i>
downloadaddr	<i>destination directory where module jars are downloaded</i>
module	<i>module name</i>
modulesurl	<i>base uri where published modules are staged</i> This command uses this URL base to construct a URL that includes the module name and version. The URL format is: <code>\${opts.modulesurl} + "/" + \${opts.module} + "-" + \${opts.version} + ".jar"</code>
version	<i>required module version</i> Modules have a version associated with them and in archive form have a version identifier. Typically, these are numeric but by CTL convention one may also use the version named "head" to represent the latest version.

## 5.4. Install

*Initializes object workspace.*

This command creates the directory referenced `${entity.instance.dir}`, calls [Install-Module](#), and then [Get-Properties](#).

Since it is a bootstrapping command it can be run outside of an object context if the the necessary flags are supplied.

*static:* This command can be run outside of an object context.

### Usage

```
Install [-basedir <>] [-depot <>] [-name <>] [-nodir]
[-nomodule] [-noproperties] [-subdirs <bin,conf,logs,var>]
[-type <>]
```

## Options

Option	Description
basedir	<i>base directory</i>
depot	<i>object depot</i> Not required unless run outside of object context for bootstrapping.
name	<i>object name</i> Not required unless run outside of object context for bootstrapping.
nodir	<i>do not create directories</i> If specified the <code>\${entity.instance.dir}</code> directory is not created.
nomodule	<i>do not update module</i> If specified the <a href="#">Install-Module</a> command is not executed.
noproperties	<i>do not update properties</i> If specified the <a href="#">Get-Properties</a> command is not executed.
subdirs	<i>directory list</i>
type	<i>object type</i> Not required unless run outside of object context for bootstrapping.

## 5.5. Properties

*Prints out entity properties.*

This command reads the information generated by [Get-Properties](#) command and prints it to the console using the specified format.

### Usage

```
Properties [-detail] [-file <>] [-format <pretty>]
```

### Options

Option	Description
detail	<i>print more detail</i>

	This flag works in conjunction with the <code>-format pretty</code> flag and signifies to show properties besides those of this entity.
<code>file</code>	<i>file containing property data</i>
<code>format</code>	<p><i>output format. pretty or plain</i></p> <p>Two formats are defined:</p> <ul style="list-style-type: none"> <li>• <code>pretty</code>: This format signifies to make the information suitable for human interpretation.</li> <li>• <code>plain</code>: When <code>plain</code> is specified the content of the file are printed directly without any adornment.</li> </ul>

## 5.6. Purge

*Removes the object.*

Moves the object `entity.instance.dir` from the depot and into the depot archive.

### **Usage**

Purge