MicoCCM Tools Manual

Frank Pilhofer

April 2002

Abstract

This document describes the tools that come with the MicoCCM distribution. Their usage and options are documented.

Contents

1 mico-ccm

1.1 Synopsis

```
mico-ccm [<options>] <file>
```

1.2 Description

mico-ccm is the C++ code generator for CORBA Components, similar to **idl**. It reads an IDL file and generates C++ code that must be compiled and linked with your component in addition to the output of the **idl** tool. **idl** generates stubs and marshalling code, **mico-ccm** generates component-specific skeletons that interface between the component implementation and the container.

Two output files are generated, a C++ header file and C++ code file. The base names of the output files are the base name of the IDL file plus the string **_ccm**.

1.3 Options

--session Generates code for session components (default).

--service Generates code for service components.

--monolithic Generates code for components that use the monolithic implementation strategy (default).

--locator Generates code for components that use the locator strategy.

--standalone Generates a main function, so that the component(s) can be linked into a standalone executable rather than into a loadable shared library. If this executable is executed, then it registers all homes in the Naming Service, using the home's name as the name of the binding. If the name of a home is given as a parameter, then only this home is deployed. In that case, the options **--ior <filename>** and **--ns <nsname>** are accepted to write that home's IOR to a file or to bind it into the Naming Service.

In addition, **mico-ccm** also accepts the options **-I**, **-D**, **--c++-suffix**, **--hh-suffix** and **--name** with the same meaning as **idl**.

2 mico-ccmd

2.1 Synopsis

```
mico-ccmd [<options>]
```

2.2 Description

mico-ccmd is the Mico CCM Daemon. Its purpose is to manage component installations and deployed components on a single host, and it needs to run on each host where you want to deploy components.[1] **mico-ccmd** implements the standard ComponentInstallation, AssemblyFactory and ServerActivator interfaces.

Via the ComponentInstallation interface, component implementations can be installed on the host where **mico-ccmd** is running. Via the AssemblyFactory and ServerActivator interfaces, assemblies and component servers can be started on demand - for these, mico-ccmd requires the separate **assembly** and

[1] Except standalone components (where the **--standalone** option of **mico-ccm** was used). **mico-ccmd** is not required to run standalone components.

componentserver tools.[2] These tools need to be in the search path so that **mico-ccmd** can execute them.

The deployment tool contacts **mico-ccmd** to first upload component implementations and then to start up componentservers to host the components as required.

mico-ccmd is a daemon that normally runs in perpetuity the background and that does not do anything visible. If signalled to exit (via SIGINT, usually Ctrl-C), then it in turn signals all assemblies and component servers to exit. Before finally exiting, it removes all component implementations that have been installed.

2.3 Options

--root <pkgdir> Gives the path name of a directory where installed component implementations are to be stored. Without this option, component implementations are stored in the current directory.[3]

--ior <filename> Stores the stringified object reference of the MicoCCM Daemon in the given file. If a single hyphen is given as file name, then the object reference is written to standard output.

-v Verbosity. Status messages, such as about the starting and stopping of component servers will be written to standard output.

The standard option of the Mico ORB **-ORBIIOPAddr** can be used to assign a specific port and interface to bind to, as in **-ORBIIOPAddr inet::1234**. Note that an object URL to **mico-ccmd** can be constructed easily if it is running on a known port. **mico-ccmd** uses an Object Key of **MicoCCMD**, so the complete object URL is

```
corbaloc::<host>:<port>/MicoCCMD
```

where **host** and **port** are the known host name and port number where **mico-ccmd** is running.

3 ccmload

3.1 Synopsis

```
ccmload [<options>] <home name> <library file>
```

3.2 Description

ccmload is a simple deployment tool that can deploy a single component (more specifically, a single home). **ccmload** contacts a running MicoCCM Daemon (**mico-ccmd**) to create a new component server to host the new component, causes the home **home name** from the shared library **library file** to be loaded into the container, and then optionally registers the home in the Naming Service. **home name** must be the absolute name of the home that is to be deployed, and **library file** must be the file name of a shared library that contains the implementation for **home name**. Note that ccmload does not upload this implemenation; the file name must be valid in the context of the MicoCCM Daemon.

3.3 Options

--ccmd <IOR> Gives the object reference of the MicoCCM Daemon.

--host <host>[:port] This is an alternate possibility to specify the address of the MicoCCM Daemon by host name and port number. If the port number is not given, then port 1234 is assumed.

[2] The **componentserver** tool is part of MicoCCM, the **assembly** tool is distributed as part of the MicoCCM Assembly and Deployment Toolkit.

[3] Note that **mico-ccmd** refuses to overwrite existing files, so if you run, e.g. during testing, **mico-ccm** in the same directory where the component resides that is to be deployed, then an error will be reported because of this.

`--ns <name>` Registers the deployed home in the Naming Service.

`--ior <filename>` Writes the object reference of the deployed home into the given file, or to standard output if a single hyphen is given as file name.

`-v` Causes some some status messages to be printed during deployment.

4 componentserver

4.1 Synopsis

```
componentserver [<options>]
```

4.2 Description

An instance of the **`componentserver`** provides the runtime environment for one or more homes and components. It implements the ComponentServer and Container interfaces and is able to load components from shared libraries. **`componentserver`** is usually not started manually, new instances are spawned by **`mico-ccmd`** on demand.

There are no command-line options to load shared libraries; this is done via the **`install_home`** method of the Container interface.

4.3 Options

`--ccmd <IOR>` Gives the object reference of the MicoCCM Daemon.

`--token <string>` Assings a name to this componentserver instance. This is required for communication with the MicoCCM Daemon, but not necessary for freestanding instances.

`--root <pkgdir>` Containers will search this directory for component implementations.

`--ior <filename>` Writes the object reference of the ComponentServer into the given file, or to standard output if a single hyphen is given as file name.

`-v` Causes some status messages to be printed.