

CTL Concepts

Table of contents

1 Framework.....	2
2 Project Depots.....	2
3 Nodes.....	2
4 Node Dispatch.....	2
5 Modules.....	3
6 Data Context.....	3
7 Repository.....	4
8 Glossary.....	5

This section describes core concepts that underly the CTL software and its use.

1. Framework

CTL contains a control-dispatching framework built using the CTL API, to execute specified actions. The framework also provides project depots for managing project configuration files, modules, objects and their instance data. Several utilities are included to execute commands and administer the framework.

Figure 1: CTL Software

2. Project Depots

The framework repository is organized into project depots. For each project depot, the framework provides a file structure. Within each depot, there is a space for configuration, a module library, and an object store.

3. Nodes

A machine hosting a CTL software instance is called a node. By convention a node can take on one of two roles, target node or admin node. A target node is one that receives commands from an admin and executes them. An admin node is a central server from which remote management of target nodes will be performed. Typically, the admin node hosts a [repository](#).

Nodes are described with metadata in the [nodes.properties](#) configuration file. This metadata is used by the node dispatching mechanism, and facilitates keyword based filtering.

Because remote actions are invoked via SSH, the target nodes will need to include the admin nodes in their `authorized_keys` file.

4. Node Dispatch

The internal control dispatch mechanism supports an execution strategy referred to as `nodedispatch`. The control dispatcher is able to lookup the node of the target, dispatching the command remotely or locally as appropriate. This feature provides network abstraction to command execution and makes management of server pools and clusters more convenient.

The CTL framework supports two styles of control dispatching. Ad-hoc commands can be executed via [ctl-exec](#). Commands defined in a module are executed via [ctl](#).

- `ctl-exec` will dispatch any command you would normally type at the shell to the nodes that match the filtering criteria.

- `ctl` will dispatch commands defined in a module to the locations registered in the [deployments.properties](#) file that match the filtering criteria.

The `nodedispatch` strategy uses SSH to invoke CTL actions on remote hosts.

5. Modules

A module is a packaged set of defined commands created to execute some set of procedures. Modules can be likened to a plugin to the framework. Once a module has been installed the framework dispatcher provides access to execute its commands.

Modules can be organized into an inheritance hierarchy of super- and sub-types to support generic/specific behavior and provide a means for reuse.

The figure below shows how a user executes a command via the CTL framework. The control-dispatcher receives the request to run a command, looks up the command's handler, and then, provides as input, a data context that can be managed within the framework.

Figure 2: CTL Module Dispatching

Module Structure

At a file level, a module has a directory structure that organizes files into several locations. Modules are defined in an XML file called [type.xml](#). The [ProjectBuilder](#) `build-type` command reads the `type.xml` file and generates an executable set of commands.

Figure 3: Module structure

Inside the `type.xml` file, you define one or more commands. Commands can be used to control life cycle actions but can also be used to control processes and interface with other tools. The implementation of a command is called its *handler*. Command handlers are XML files that read property files to establish a data context and execute the defined action.

Given the supplied parameters, the CTL control dispatcher knows how to lookup command handlers in the module library, generate a data context, and execute them.

Figure 4: Command dispatcher mechanism

A fundamental tenet in CTL is to "soft code" commands by stating important detail in data files in order to separate the procedural logic from environment detail. This leads to commands that are more flexible and reusable in different environments.

6. Data Context

As mentioned earlier, commands in CTL are driven by a data context. The CTL framework

assumes standard data files generated from the type.xml file but you are free to create and use data from any source for your commands.

Data you might store in the property files include:

- configuration data: app and environment settings
- dependency data: bindings to other objects
- any thing else your procedures need

The framework uses several configuration files where global configuration data is stored. Every command handler can access this global data as it is loaded by the dispatcher when the handler is loaded.

CTL uses Java property file format, key-value pairs, as its native format. Global configuration is stored in the [framework.properties](#) file.

Module Context

The module defines several standard property files:

1. [module.properties](#): This contains metadata about the module.
2. [type.properties](#): This contains attribute data
3. [commands.properties](#): This contains metadata about each command such as command parameters.

Modules load this property data when executed. Other data can be stored in other files and read in by the command handler.

Figure 5: Context Data Files

Object Context

When there is a lot of data, or when your commands need to use different sets of data depending on environment or location, the CTL framework provides the capability of defining objects to organize and discretely maintain the data. See the [CTL Objects](#) page for more information about defining and using objects.

The standard file for storing object specific data is called the [entity.properties](#) file. This file can contain anything you wish and it is obtained and updated via the [Get-Properties](#) command.

7. Repository

An important facility for managing a network of CTL nodes is a central repository. Once command modules have been developed they can be stored in this repository so that they can be downloaded by nodes that require them. The CTL repository can be a simple web server

or if one wishes to publish files via PUT, a server based on WebDAV.

The Repository uses a content structure divided into the following categories:

- **Configuration:** A set of configuration files that include information about nodes and their deployments. One such file, `nodes.properties`, describes the nodes in the environment to which you want to dispatch actions.
- **Modules:** A collection of packaged CTL control modules that can be deployed to CTL nodes.
- **Packages:** If you use CTL to manage deployment you can include a collection of software packages. These packages may be of various file types such as zip, tgz, rpm, war, jar, ear, etc.

Each project depot on the repository has a directory for each of these content categories:

8. Glossary

- Command - Defined procedure that can be executed via CTL
- Command handler - An implementation of a Command
- Context - Data context made available during the execution of a handler
- Deployment - A registered deployment of a module or object
- Framework - An instance of the CTL control dispatching mechanism and repository of command modules and management data.
- Module - A packaged set of Commands.
- Node - A registered host to which commands can be dispatched
- Object - A management entity that has its own workspace and an associated module.
- Project - A repository of modules and objects. Often referred to as a "depot".