Introduction to the Odyssey API 1

Welcome to the Odyssey API document. Odyssey is an agent system implemented as a set of Java class libraries that support agents and places. This document helps you get started creating applications with the Odyssey API by guiding you through a sample application.

The sample application referred to in this document is very simple. Based on user input, the application sends an agent to one or more destinations to carry out the same task and return the results. The application code is available at the General Magic Web site (http://www.genmagic.com/agents).

1.1 The Odyssey API

Odyssey adds support for developing distributed, mobile applications to the Java class libraries. The paradigm Odyssey supports for distributed, mobile applications includes agents, agent systems, and places.

1.1.1 Agents

An agent is a process that acts autonomously on behalf of a person or organization. Each agent has its own thread of execution so that it can perform tasks on its own initiative. Agents are created by subclassing the Odyssey Agent class or the Odyssey Worker class (a worker is a type of agent discussed later).

A mobile agent is an agent that is not bound to the system where it begins execution. It has the unique ability to transport itself from one system in a network to another. Odyssey agents are mobile agents. However, not every mobile agent is required to move during its execution.

For Odyssey processes, both agents and places, their live(int) method defines their behavior.

1.1.2 Workers

The Odyssey Worker class is a subclass of the Odyssey Agent class. A worker is structured as a set of tasks and a set of destinations. At each destination, the worker executes to completion the next task on its task list.

Structuring an application as a series of tasks, each to be completed at a specific host, can make it easier to create mobile agent applications. Associating a task with a host removes the necessity of checking whether the methods the task requires can run on the current host or not.

An Odyssey worker may manipulate its task list at any point during its travels. For example, a worker could go to a Yellow Pages place, find the top suppliers of rare books, and add these places as new destinations and tasks in the task list.

1.1.3 Agent Systems

An agent system is a platform that can create, manage, interpret, execute, transfer, and terminate agents. An agent system has the authority of the region or organization that it represents. The Odyssey agent system consists of a set of Java classes to support Odyssey agents and Odyssey places.

1.1.4 Places

A place is a context within an agent system in which agents execute. A mobile agent travels between places. This context can provide functions such as access control.

A place is the stationary part of the application you write. Because agents execute only within places, you add an instance of one or more places to each host running your application.

In Odyssey, the initial place created when the system is started is distinct from all other places. This place is derived from the class BootPlace. When the BootPlace exits, the Odyssey agent system shuts down, terminating all agents and places currently existing within that agent system.

The place provides the interface to the functionality that is local at each host running your application. This local functionality might include, the user interface code, and access to the local databases. In general, a place is the gateway between agents visiting a host and the host's resources.

1.1.5 The Odyssey Class Hierarchy

The Odyssey class hierarchy contains the classes that implement the Odyssey paradigm. There are two classes for constructing mobile agents: Agent and Worker. There is also a class, called Place, for the agent execution environments.

The hierarchy also includes classes that support agents, workers, and places. These classes include Ticket (specifies how and where an agent travels), Means (specifies how an agent travels), Petition (identifies who an agent wants to communicate with), and ProcessName (used to generate the unique names of all processes, including agents and places).

The class hierarchy also includes three interfaces: genmagic.odyssey.AgentSystem, genmagic.odyssey.Finder, and genmagic.odyssey.Transport. These interfaces allow a developer to customize the implementation of an Odyssey agent system.

1.2 The Sample Application

The sample application presents a window that lets a user enter one operating system command. Note that a command must be an executable (for example, chkdsk in Windows NT, or ls in Unix), because the application does not start up a shell.

Once the user finishes entering the command, the application spawns a worker to travel to each of the hosts the user specified, executing that command at each host.

1.2.1 Creating a Place

To use the sample application, you must create a place at each destination that the user can specify. Each place is an instance of the class CmdPlace, which extends the class BootPlace (refer to the class hierarchy link for information about this class). The code for the class CmdPlace is in the file CmdPlace.java.

The master place is the place in the application that is given the list of the other destinations at boot time. It also runs the application’s user interface.

1.2.2 Processing the User Commands

Once the places are created at each of the hosts a worker will visit, the application waits for a user command to process. The window contains:

- a box where the user can specify the command to execute
- a Do Command button, which causes the application to try executing the operating system command the user entered
- a menu containing two options: Clear Text, which clears the text in the command output box, and Exit, which exits the application and shuts down the Odyssey system.

Note that the user can also press Return to signal the application to execute the command. When the user either presses the Do Command button or Return, the doAgent() method is called.

1.2.3 Setting Up the Worker Tasks

The doAgent() method is the method that creates the task list for the worker, and spawns the worker. The worker in the sample application executes the user-specified command at one or more hosts, then reports the results of each command.

The user supplies the list of destinations as arguments when starting the Odyssey engine on a host. The engine, in turn creates a CmdPlace, providing the list of destinations as an array to the constructor.

To set up the tasks for the workers, the application:

- creates a list of destinations from the destination array
- turns the list of destinations into a list of destination/method name pairs that become the task list for the worker

The following code handles the first step, creating the list of destinations from the user input. This snippet is from the Demo() method in the file Demo.java.

```
//Set up the list of places to execute commands
destinations = new String [hosts.length];
for (int i = 0; i , hosts.length; i++) {
  try {
    destinations[i] =
      InetAddress.getByName(hosts[i]).getHostAddress() +
      “/RemoteCmdServer”;
  }catch (java.net.UnknownHostException.exc) {
    System.out.println(“Unable to determine host name for: “ +
                          hosts[i]);
    destinations[i] = hosts[i] + “/RemoteCmdServer”;
}
  System.out.println(“Destination[“+ i +”] => “ + destinations[i]);
}
```

The following code generates the list of tasks for the Worker from the list of destinations derived from the user input. It also initiates executing the tasks. This code is part of the method doAgent() in the file Demo.java.

```
//Turn the list of destinations into a task list for the worker
Task[] tasks = new Task [destinations.length + 1];
for (i = 0; i < destinations.length; i++) {
  tasks[i] = new Task (destinations[i], “doCommand”);
}
```

Task is an Odyssey class that describes a task that a worker will perform. Each task consists of a ticket to the place where the specified method should be executed, and the name of a method to execute on the worker.

Part of doAgent() (shown below) calls the method, reportResults(), that reports the outcome of the task at each host.

```
//In the end we will return here and report the results
tasks[i] = new Task (OdysseyProcess.place().returnTicket(),
  “reportResults”);
```

Note that the call to this method uses four of the Odyssey classes: Task, OdysseyProcess, Place, and Ticket.

1.2.4 Creating a Worker

The sample application creates a worker to carry out the tasks the user specified. As you recall, a worker is a type of mobile agent that travels from one destination to another, and does the next task on its list at each destination. Here is the code that spawns the worker:

```
//Spawn the worker to go execute the tasks
try{
  new CmdWorker(tasks, theCmd.getText()).start();
}catch (OccupancyDeniedException.exc) {
  String[] error = new String[1];

  error[0] = “Unable to spawn agent: “ + exc:
  the reportResults(error);
  }
}
```

The method CmdWorker() that creates the worker requires two parameters: a list of tasks, and a string containing the command to execute.

Note that the start() method actually initiates execution of the agent. Note also that the command specified in theCmd variable is executed by the doCommand() method.

1.2.5 Reporting the Results

When the agents return, they use the method reportResults(), to report the outcome of each task. As the following code demonstrates, the method appends the output of each task to the results buffer.

```
for (int i = 1; i<results.length; i++) {
      info.append (“====\n”);
      if (results[i] != null) {
         info.append(results[i]);
      } else {
         info.append(“results[“+ i +“] is null”);
      }
```