Hello World Example

This example:

Uses a null lease
Uses unicast discovery & join
Is derived from an example in Noel Enete’s Nuggets for Jini

```java
package whitney.jini.examples.hello;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface HelloInterface extends Remote {
```
public String sayHello() throws RemoteException;
}

HelloServer

package whitney.jini.examples.hello;
import net.jini.core.entry.Entry;
import net.jini.core.lookup.ServiceID;
import net.jini.core.lookup.Entry.Name;
import com.sun.jini.lookup.ServiceIDListener;
import com.sun.jini.lookup.JoinManager;
import com.sun.jini.lease.LeaseRenewalManager;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;
public class HelloServer extends UnicastRemoteObject
implements HelloInterface, ServiceIDListener
{
   private ServiceID myID;

   public HelloServer() throws RemoteException
   {
   }

   public String sayHello () throws RemoteException
   {
      return ("Hello World from Jini Hello server!");
   }

   public void serviceIDNotify (ServiceID uniqueID)
   {
      myID = uniqueID;
      System.out.println("server: ID set: " + myID );
   }

HelloServer Main

public static void main (String[] args) throws Exception
{
   System.setSecurityManager (new RMISecurityManager ());
   HelloServer myServer = new HelloServer ();
   Entry[] identityingAttributes = new Entry[1];
   identityingAttributes[0] = new Name("HelloServer");
   JoinManager myManager = new JoinManager
   {
      myServer,
      identityingAttributes,
      myServer,
      new LeaseRenewalManager ()
   );
   System.out.println ("Server has been Joined!");
}

HelloClient

package whitney.jini.examples.hello;
import net.jini.core.entry.Entry;
import net.jini.core.lookup.ServiceTemplate;
import net.jini.core.lookup.ServiceRegistrar;
import net.jini.core.discovery.LookupLocator;
import net.jini.lookup.entry.Name;
import java.rmi.RMISecurityManager;
public class HelloClient
{
    public static void main (String[] args) throws Exception
    {
        System.setSecurityManager (new RMISecurityManager ());
        LookupLocator lookup = new LookupLocator ("jini://eli.sdsu.edu");
        ServiceRegistrar registrar = lookup.getRegistrar();

        Entry[] serverAttributes = new Entry[1];
        serverAttributes[0] = new Name ("HelloServer");
        ServiceTemplate template =
            new ServiceTemplate (null, null, serverAttributes);
        HelloInterface myServerInterface =
            (HelloInterface) registrar.lookup (template);
        System.out.println ( myServerInterface.sayHello () );
    }
}
**Interface net.jini.core.lookup.ServiceRegistrar**

**Methods**

- Information about available services
  - `Class[] getEntryClasses(ServiceTemplate tmpl)`
  - `Object[] getFieldValues(ServiceTemplate, int setIndex, String field)`
  - `Class[] getServiceTypes(ServiceTemplate, String prefix)`
  - `Object lookup(ServiceTemplate tmpl)`
  - `ServiceMatches lookup(ServiceTemplate tmpl, int maxMatches)`

  Returns the service object from an item matching the template

- Information about Lookup service
  - `String[] getGroups()`

  Returns the groups that this lookup service is a member of.

- LookupLocator
  - `LookupLocator getLocator()`

  LookupLocator for unicast discovery of lookup service.

- ServiceID
  - `ServiceID getServiceID()`

- Notification, Registration
  - `EventRegistration notify(ServiceTemplate tmpl, int transitions, RemoteEventListener listener, MarshalledObject handback, long leaseDuration)`

  ServiceRegistration
  - `register(ServiceItem item, long leaseDuration)`

---

**HelloClient Continued**

```java
// Create ServiceTemplate used to match or find the server
// we want
Entry[] serverAttributes = new Entry[1];
serverAttributes[0] = new Name("HelloServer");
ServiceTemplate template = new ServiceTemplate(null, null, serverAttributes);
HelloInterface myServerInterface = (HelloInterface) registrar.lookup (template);
```

**Interface net.jini.core.entry.Entry**

Entry subclasses:

Are used in identifying and matching services
Contains public fields of non-primitive types
Must have no-argument constructor
All fields must be serializable

Each field is serialized separately, so references between two fields of an entry will not be
reconstituted to be shared references, but instead to separate copies of the original object
If a field of an entry being fetched is cannot be deserialized for any reason then
net.jini.core.entry.UnusableEntryException will be thrown when

Entry is a marker interface, it has no methods

Matching Entries

Let \( T \) (for template) be an object of a subclass \( \text{Entry} \)
Let \( E \) be an object of a subclass of \( \text{Entry} \)

A field of an Entry object is a wildcard it its value is null

The template \( T \) matches \( E \) if:

- The type of \( E \) must be that of \( T \), or be a subtype of the type of \( T \)

- Fields with values in \( T \) must be matched exactly by the value in the same field of \( E \).

- Wildcards in \( T \) match any value in the same field of \( E \).

MarshalledObject.equals is used to test equality of fields, not Object.equals

This means the bytes generated by their serialized form must match, ignoring differences of serialization
stream implementation (such as blocking factors for buffering). Class version differences that change the
bytes generated by serialization will cause objects not to match.

net.jini.core.lookup.ServiceTemplate

Constructor

ServiceTemplate(ServiceID serviceID, Class[] serviceTypes,
Entry[] attrSetTemplates)

Public Fields

Entry[] attributeSetTemplates
ServiceID serviceID
Class[] serviceTypes

ServiceTemplates are used to match for services

A service template (tmpl) matches service item (item) a if:

- item.serviceID equals tmpl.serviceID

If tmpl.serviceID is null then tmpl and item match if:

- item.service is an instance of every type in tmpl.serviceTypes

- and item.attributeSets contains at least one matching entry for each entry template in tmpl.attributeSetTemplates.

---

**Standard Entry Subclasses**

**net.jini.lookup.entry Classes**

Some basic entry classes with their fields. All fields are Strings

<table>
<thead>
<tr>
<th>Address</th>
<th>ServiceInfo</th>
<th>Location</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>country</td>
<td>manufacturer</td>
<td>building</td>
<td>name</td>
</tr>
<tr>
<td>locality</td>
<td>model</td>
<td>floor</td>
<td></td>
</tr>
<tr>
<td>organization</td>
<td>serialNumber</td>
<td>room</td>
<td></td>
</tr>
<tr>
<td>organizationalUnit</td>
<td>vendor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>postalCode</td>
<td>version</td>
<td></td>
<td>comment</td>
</tr>
<tr>
<td>stateOrProvince</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>street</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Status & StatusType**

**Status** is base class from which other status-related entry classes may be derived.

**StatusType** defines status types:

- ERROR
- NORMAL
- NOTICE
- WARNING
net.jini.entry.AbstractEntry

Useful base class for entry types
You don’t have to use this class, but will save you some effort

Entry & JavaBeans

Jini uses JavaBeans to display entry objects to humans

You should have a JavaBean class for an Entry class that a human may interact with

JavaBean Class Design Pattern

Let X be an entry class then:

- The JavaBean class for X is called XBean

- For each field F in X class XBean has methods:
  setF() and getF()

- XBean has an no-argument constructor

- XBean implements net.jini.lookup.entry.BeanEntry

- XBean has methods followLink(), makeLink()

  These methods set and get the bean’s field that contains the bean’s entry object

Location Entry Example

Location fields and LocationBean’s methods

<table>
<thead>
<tr>
<th>Location</th>
<th>LocationBean</th>
</tr>
</thead>
<tbody>
<tr>
<td>building</td>
<td>Entry followLink()</td>
</tr>
<tr>
<td>floor</td>
<td>getBuilding()</td>
</tr>
<tr>
<td>room</td>
<td>getFloor()</td>
</tr>
<tr>
<td></td>
<td>getRoom()</td>
</tr>
<tr>
<td></td>
<td>makeLink(Entry e)</td>
</tr>
<tr>
<td></td>
<td>setBuilding(String x)</td>
</tr>
<tr>
<td></td>
<td>setFloor(String x)</td>
</tr>
<tr>
<td></td>
<td>setRoom(String x)</td>
</tr>
</tbody>
</table>
**HelloClient Matching**

The following code creates a simple template to use to find a service. The service id is set to null, so it is not used. Using the service ID would be faster, but we need to know it. We also set the serviceTypes to null. This means the template will match any serviceTypes. If we were looking for printers, we might set serviceTypes to be an array containing a printer type. Jini does not define these serviceTypes. They must be supplied by Jini developers. Only one entry template is given. Hence this service template will match any service that contains an entry of type "Name" or subtype of "Name" with the field "name" equal to the value "HelloServer".

```java
Entry[] serverAttributes = new Entry[1];
serverAttributes[0] = new Name("HelloServer");
ServiceTemplate template =
   new ServiceTemplate(null, null, serverAttributes);
HelloInterface myServerInterface =
   (HelloInterface) registrar.lookup(template);
```

**HelloServer**

```java
public class HelloServer extends UnicastRemoteObject
   implements HelloInterface, ServiceIDListener
{
   private ServiceID myID;

   public void serviceIDNotify (ServiceID uniqueID)
   {
      myID = uniqueID;
      System.out.println("server: ID set: " + myID);
   }

   net.jini.core.lookup.ServiceID

   Each service has a unique service id

   The id is unique over time and space with respect to all other service ids generated by all lookup services. The id is a 128-bit value, to insure this uniqueness.

   It is to be set by the lookup service.

   To re-register an existing service, or to register the service in any other lookup service, item.serviceID should be set to the same service id that was returned by the initial registration.

   com.sun.jini.lookup.ServiceIDListener

   Contains one method, serviceIDNotify

   The lookup service calls this method to provide the service its service ID.
```
HelloServer Main

```java
HelloServer myServer = new HelloServer();
Entry[] identityingAttributes = new Entry[1];
identityingAttributes[0] = new Name("HelloServer");
JoinManager myManager = new JoinManager
{
    myServer,
    identityingAttributes,
    myServer,
    new LeaseRenewalManager()
};
```

**com.sun.jini.lookup.JoinManager**

This class manages the join protocol for a service. It discovers and keeps track of which lookup services to join, registers with them, keeps the registration leases renewed, and keeps the attributes up to date.

```java
JoinManager(Object service, Entry[] attrSets, ServiceIDLListener callback, LeaseRenewalManager leaseMgr)
```

**JoinManager Methods**

- `addAttributes(Entry[] attrSets)`
- `addAttributes(Entry[] attrSets, boolean checkSC)`
- `addGroups(String[] groups)`
- `addLocators(LookupLocator[] locators)`
- `Entry[] getAttributes()`
- `String[] getGroups()`
- `ServiceRegistrar[] getJoinSet()`
  - Get the list of lookup services that have currently been joined.
- `LookupLocator[] getLocators()`
  - Get the list of locators of specific lookup services to join.
- `modifyAttributes(Entry[] attrSetTemplates, Entry[] attrSets)`
- `modifyAttributes(Entry[] attrSetTemplates, Entry[] attrSets, boolean checkSC)`
- `removeGroups(String[] groups)`
- `removeLocators(LookupLocator[] locators)`
- `setAttributes(Entry[] attrSets)`
- `setGroups(String[] groups)`
- `setLocators(LookupLocator[] locators)`
- `terminate()`