



# Jini™ Discovery Utilities Specification

The Jini™ technology is a Java™ platform-centric distributed system designed around the goals of simplicity, flexibility, and federation. The Jini discovery protocols are used by entities that wish to participate in a system of Jini technology-enabled services and/or devices. This document specifies utility classes to simplify the task of using the discovery protocols.



Version 1.1Alpha  
November 1999

Copyright © 1999 Sun Microsystems, Inc.  
901 San Antonio Road, Palo Alto, CA 94303 USA.  
All rights reserved.

Sun Microsystems, Inc. has intellectual property rights (“Sun IPR”) relating to implementations of the technology described in this publication (“the Technology”). In particular, and without limitation, Sun IPR may include one or more patents or patent applications in the U.S. or other countries. Your limited right to use this publication does not grant you any right or license to Sun IPR nor any right or license to implement the Technology. Sun may, in its sole discretion, make available a limited license to Sun IPR and/or to the Technology under a separate license agreement. Please visit <http://www.sun.com/software/communitysource/>.

Sun, the Sun logo, Sun Microsystems, Jini, the Jini logo, JavaSpaces, Java, JavaBeans, Solaris, NFS, PC-NFS, EmbeddedJava, PersonalJava, and Solstice are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

THIS SPECIFICATION IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS SPECIFICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE SPECIFICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN ANY TECHNOLOGY, PRODUCT, OR PROGRAM DESCRIBED IN THIS SPECIFICATION AT ANY TIME.

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Dependencies	5
1.2	Comments	5
<b>2</b>	<b>Multicast Discovery Utility</b>	<b>7</b>
2.1	The LookupDiscovery Class	8
2.2	Useful Constants	9
2.3	Changing the Set of Groups to Discover	9
2.4	The DiscoveryEvent Class	10
2.5	The DiscoveryListener Interface	10
2.6	Security and Multicast Discovery	11
2.7	Serialized Forms	12
<b>3</b>	<b>Protocol Utilities</b>	<b>13</b>
3.1	Marshalling Multicast Requests	13
3.2	Unmarshalling Multicast Requests	14
3.3	Marshalling Multicast Announcements	15
3.4	Unmarshalling Multicast Announcements	16
3.5	Easy Access to Constants	16
3.6	Marshalling Unicast Discovery Requests	17
3.7	Unmarshalling Unicast Discovery Requests	17
3.8	Marshalling Unicast Discovery Responses	18
3.9	Unmarshalling Unicast Discovery Responses	18



# DU

---

# The Jini™ Discovery Utilities Specification

## DU.1 Introduction

**E**ACH individual party in a Java™ virtual machine (JVM) on a given host is independently responsible for obtaining references to lookup services. In this specification we first covers utility classes that such parties can use to simplify multicast discovery tasks. We then present lower-level utility classes that are useful in building these kinds of utilities.

### DU.1.1 Dependencies

This specification relies on the following other specifications:

- ◆ *Java™ Object Serialization Specification*
- ◆ *Jini™ Lookup Service Specification*
- ◆ *Jini™ Discovery and Join Specification*

### DU.1.2 Comments

Please direct comments to [jini-comments@java.sun.com](mailto:jini-comments@java.sun.com).



---

## DU.2 Multicast Discovery Utility

**P**ARTIES can obtain references to lookup services via the multicast discovery protocols by making use of the `LookupDiscovery` class.

```
package net.jini.discovery;

import net.jini.core.lookup.ServiceRegistrar;
import java.io.IOException;

public final class LookupDiscovery {
    public static final String[] ALL_GROUPS = null;
    public static final String[] NO_GROUPS = new String[0];

    public LookupDiscovery(String[] groups)
        throws IOException {...}
    public void addDiscoveryListener(DiscoveryListener l) {...}
    public void removeDiscoveryListener(DiscoveryListener l)
        {...}
    public void discard(ServiceRegistrar reg) {...}
    public String[] getGroups() {...}
    public void setGroups(String[] groups)
        throws IOException {...}
    public void addGroups(String[] groups)
        throws IOException {...}
    public void removeGroups(String[] groups) {...}
    public void terminate() {...}
}
```

The `LookupDiscovery` class relies upon the `DiscoveryEvent` class:

```
package net.jini.discovery;

import net.jini.core.lookup.ServiceRegistrar;
import java.util.EventListener;
```

```
import java.util.EventObject;

public class DiscoveryEvent extends EventObject {
    public DiscoveryEvent(Object source,
                          ServiceRegistrar[] regs) {...}
    public ServiceRegistrar[] getRegistrars() {...}
}
```

The LookupDiscovery class also relies upon the DiscoveryListener interface:

```
public interface DiscoveryListener extends EventListener {
    public void discovered(DiscoveryEvent e);
    public void discarded(DiscoveryEvent e);
}
```

These classes and interfaces hide the details of the underlying protocol implementation, but provide enough information to the programmer to be flexible and useful.

## DU.2.1 The LookupDiscovery Class

The `net.jini.discovery.LookupDiscovery` class encapsulates the operation of the multicast discovery protocols, including the automatic switch from use of the multicast request protocol to the multicast announcement protocol. Each instance of the `LookupDiscovery` class must behave as if it operated independently of all other instances. The semantics of the methods on this class are:

- ◆ The constructor takes a set of groups in which the caller is interested as parameter. This set is represented as an array, none of whose elements may be `null`. The empty set is represented by an empty array, and no set (indicating that all lookup services should be discovered) is indicated by a `null` reference. The constructor may throw a `java.io.IOException` if a problem occurs in starting discovery.
- ◆ The `addDiscoveryListener` method adds a listener to the set of objects listening for discovery events. Once a listener is registered, it is notified of all lookup services that have been discovered to date, and is then notified as new lookup services are discovered or existing lookup services are discarded.
- ◆ The `removeDiscoveryListener` method removes a listener from the set of objects that are listening for discovery events.



- ◆ The `discard` method removes a particular lookup service from the set that is considered to already have been discovered. This allows the lookup service to be discovered again; it is intended as a mechanism for programmers to remove stale entries from the set so that they do not have to keep trying to contact lookup services that no longer exist.
- ◆ The `getGroups` method returns the set of groups that this `LookupDiscovery` object is attempting to discover. If the set is empty, this method returns the empty array, and if there is no set, it returns the `null` reference.
- ◆ The `terminate` method ends discovery. After this method has been called, no new lookup services will be discovered.

Discovery usually starts as soon as an instance of this class is created and ends either when the instance is finalized prior to garbage collection, or when the `terminate` method is called. However, if the empty set is passed to the constructor, discovery will not be started until the `setGroups` method is called with either no set or a non-empty set.

## DU.2.2 Useful Constants

The `ALL_GROUPS` constant can be passed to the `LookupDiscovery` constructor and to the `setGroups` method to indicate that all lookup services within range should be discovered. The `NO_GROUPS` constant indicates that no groups should be discovered (implying that discovery should be postponed until another call to `setGroups`).

If the `getGroups` method returns the empty array, that array is guaranteed to be referentially equal to the `NO_GROUPS` constant (that is, it can be tested for equality using the `==` operator).

## DU.2.3 Changing the Set of Groups to Discover

Programmers may modify the set of groups to be discovered on the fly, using the methods described below. In each case, a set of groups is represented as an array of strings, none of whose elements may be `null`. The empty set is denoted by the empty array, and no set (indicating that all lookup services should be discovered) is indicated by `null`. Duplicated group names are ignored.

- ◆ The `setGroups` method changes the set of groups to be discovered to the given set (or to no set, if indicated).

- ◆ The `addGroups` method augments the set of groups to be discovered. This method throws a `java.lang.UnsupportedOperationException` if there is no set to be augmented.
- ◆ The `removeGroups` method removes members from the set of groups to be discovered. No exception is thrown if an attempt is made to remove a group that is not currently in the set to be discovered. This method throws a `java.lang.UnsupportedOperationException` if there is no set to remove members from.

When groups are removed from the set to be discovered, any already discovered lookup services that are no longer members of any of the groups to be discovered are removed from the set maintained by the particular `LookupDiscovery` object in use, and all listeners are notified that they have been discarded.

If groups are added to the set to be discovered, the multicast request protocol is used to discover lookup services for those groups. If there are no responses to multicast requests, the `LookupDiscovery` object switches over to listening for multicast announcements for those groups.

Since calling either the `setGroups` or `addGroups` method may result in the multicast request protocol being started afresh, either method may throw a `java.io.IOException` if a problem occurs in starting the protocol.

If any of the `setGroups`, `addGroups`, or `removeGroups` methods is called after the `terminate` method has been called, the invocation will throw a `java.lang.IllegalStateException`.

## DU.2.4 The *DiscoveryEvent* Class

The `net.jini.discovery.DiscoveryEvent` class encapsulates the information made available by the multicast discovery protocols. The sole new method of the `DiscoveryEvent` class is `getRegistrars`, which returns an array of lookup service registrars. The `getSource` method returns the `LookupDiscovery` object that originated the given event.

## DU.2.5 The *DiscoveryListener* Interface

Objects that wish to register for notifications of multicast discovery events must implement the `net.jini.discovery.DiscoveryListener` interface. Its `discovered` method is called whenever new lookup services are discovered, with an event containing a set of discovered lookup services represented as an array.

The `discard` method is called whenever previously discovered lookup services have been discarded by the originating `LookupDiscovery` object; the event contains a set of discarded lookup services represented as an array. An event is delivered to listeners whenever the `discard` method is called on a `LookupDiscovery` object, and also if a call to either its `removeGroups` or `setGroups` method results in lookup services being discarded.

## DU.2.6 Security and Multicast Discovery

When a `LookupDiscovery` object is created, the creator must have permission either to attempt discovery of each group specified in the set to discover, or to attempt discovery of all groups if the set is `null`. This is also true for the `addGroups` and `setGroups` methods on the `LookupDiscovery` class. If appropriate permissions have not been granted, the constructor and these methods will throw a `java.lang.SecurityException`.

Discovery permissions are controlled in security policy files using the `net.jini.discovery.DiscoveryPermission` permission.

```
package net.jini.discovery;

import java.security.Permission;
import java.io.Serializable;

public final class DiscoveryPermission extends Permission
    implements Serializable
{
    public DiscoveryPermission(String group) {...}
    public DiscoveryPermission(String group, String actions)
        {...}
}
```

The `actions` parameter is ignored. The following examples illustrate the use of this permission:

```
permission net.jini.discovery.DiscoveryPermission "*";
    All groups

permission net.jini.discovery.DiscoveryPermission "";
    Only the "public" group

permission net.jini.discovery.DiscoveryPermission "foo";
    The group "foo"
```

```
permission net.jini.discovery.DiscoveryPermission "*.sun.com";
    Groups ending in ".sun.com"
```

Each declaration grants permission to attempt discovery of one name. A name does not necessarily correspond to a single group:

- ◆ The name `*` grants permission to attempt discovery of *all* groups.
- ◆ A name beginning with `*.` grants permission to attempt discovery of all groups that match the *remainder* of that name; for example, the name `*.example.org` would match a group named `foonly.example.org` and also a group named `sf.ca.example.org`.
- ◆ The empty name `""` denotes the *public* group.
- ◆ All other names are treated as individual groups and must match exactly.

A restriction of the Java 2 platform security model requires that appropriate `net.jini.discovery.DiscoveryPermission` be granted to the Jini technology infrastructure software codebase itself, in addition to any codebases that may use Jini technology infrastructure software classes.

## DU.2.7 Serialized Forms

Class	serialVersionUID	Serialized Fields
DiscoveryEvent	5280303374696501479L	ServiceRegistrar[] regs
DiscoveryPermission	-3036978025008149170L	none

---

## DU.3 Protocol Utilities

**T**HE utilities we will now present are intended for use by implementors of multicast discovery utilities, and for others who might need to exercise more control over their usage of the Jini discovery protocols.

### DU.3.1 Marshalling Multicast Requests

The `OutgoingMulticastRequest` class provides facilities for marshalling multicast discovery requests into a form suitable for transmission over a network. This class is useful for programmers who are implementing the component of one of the discovery protocols that sits on a device that wishes to join a djinn.

```
package net.jini.discovery;

import net.jini.core.lookup.ServiceID;
import java.io.IOException;
import java.net.DatagramPacket;

public class OutgoingMulticastRequest {
    public static DatagramPacket[]
        marshal(int port, String[] groups, ServiceID[] heard)
            throws IOException {...}
}
```

This class cannot be instantiated, and its sole method, `marshal`, is static. This method takes as parameter the port of the multicast response service to advertise, along with a set of groups to look for and a set of service IDs from which this system has already heard. The latter two arguments are represented as arrays. No parameter may be `null`, and the arrays must have no members that are `null`, and none should be duplicated (implementations are not required to check for duplicated members).

This method returns an array of `DatagramPacket` objects; this array contains at least one member, and will contain more if the request is not small enough to fit

in a single packet. Each such object has been fully initialized; it contains a multicast request as payload and is ready to send over the network.

In the event of error, this method may throw a `java.io.IOException` if marshalling fails. In some instances the exception thrown may be a more specific subclass of this exception.

### DU.3.2 Unmarshalling Multicast Requests

The `IncomingMulticastRequest` class provides facilities for unmarshalling multicast discovery requests into a form in which the individual parameters of the request may be easily accessed. This class is useful for programmers who are implementing the component of one of the discovery protocols that works with a lookup service implementation within a djinn.

```
package net.jini.discovery;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import net.jini.core.lookup.ServiceID;

public class IncomingMulticastRequest {
    public IncomingMulticastRequest(DatagramPacket dgram)
        throws IOException {...}
    public InetAddress getAddress() {...}
    public int getPort() {...}
    public String[] getGroups() {...}
    public ServiceID[] getServiceIDs() {...}
}
```

This class may be instantiated using a `java.net.DatagramPacket`. The payload of the `DatagramPacket` is assumed to contain nothing but the marshalled discovery request. If the marshalled request is corrupt, a `java.io.IOException` or a `java.lang.ClassNotFoundException` will be thrown. In some such instances a more specific subclass of either exception may be thrown that will give more detailed information.

The methods of this class are mostly self-explanatory.

- ◆ The `getAddress` method returns the IP address of the host to which the caller should respond.

- ◆ The `getPort` method returns the TCP port number on that host to which the caller should connect.
- ◆ The `getGroups` method returns the groups in which the originator of this request is interested. The array returned by this method may be of zero length; none of its fields will be `null`; and items may or may not be duplicated.
- ◆ The `getServiceIDs` method returns the set of service IDs of lookup services from which the originator has already heard. The array returned by this method may have length equal to zero, but none of its fields will be `null`, and items may or may not be duplicated.
- ◆ The `equals` method returns `true` if both instances have the same address, port, groups, and service IDs.

### DU.3.3 Marshalling Multicast Announcements

The `OutgoingMulticastAnnouncement` class encapsulates details of announcing a lookup service.

```
package net.jini.discovery;

import java.io.IOException;
import java.net.DatagramPacket;
import net.jini.core.lookup.ServiceID;
import net.jini.core.discovery.LookupLocator;

public class OutgoingMulticastAnnouncement {
    public static DatagramPacket[]
        marshal(ServiceID id, LookupLocator loc,
               String[] groups)
        throws IOException {...}
}
```

The sole method of this class, `marshal`, is static. It takes as parameters the service ID of the lookup service being advertised, the locator via which unicast discovery of that lookup service may be performed, and the names of the groups of which that service is a member. If a problem occurs with marshalling the request, a `java.net.IOException` will be thrown.

This method returns an array of `DatagramPacket` objects, each of which has been initialized such that it is ready to be multicast.

### DU.3.4 Unmarshalling Multicast Announcements

The `IncomingMulticastAnnouncement` class permits access to the fields of a multicast announcement datagram that has been received.

```
package net.jini.discovery;

import java.io.IOException;
import java.net.DatagramPacket;
import net.jini.core.lookup.ServiceID;
import net.jini.core.discovery.LookupLocator;

public class IncomingMulticastAnnouncement {
    public IncomingMulticastAnnouncement(DatagramPacket p)
        throws IOException {...}
    public ServiceID getServiceID() {...}
    public LookupLocator getLocator() {...}
    public String[] getGroups() {...}
}
```

The constructor takes a datagram packet as argument. If it cannot decode the contents of the datagram packet, it throws a `java.lang.ClassNotFoundException` or a `java.io.IOException`. The `getServiceID` method returns the service ID of the originator. The `getLocator` method returns the locator via which unicast discovery of the originator may be performed. The `getGroups` method returns the groups represented by the originator; the array returned by this method may be null, will not be empty, and will contain no null elements. Elements may or may not be duplicated. The `equals` method returns true if both instances have the same service ID.

### DU.3.5 Easy Access to Constants

The `Constants` class provides easy access to some constants used during the lookup discovery process.

```
package net.jini.discovery;

import java.net.InetAddress;
import java.net.UnknownHostException;

public class Constants {
```



```

    public static final short discoveryPort = 4160;
    public static final InetAddress getRequestAddress()
        throws UnknownHostException {...}
    public static final InetAddress getAnnouncementAddress()
        throws UnknownHostException {...}
}

```

The value of the `discoveryPort` variable is the UDP port number over which the multicast request and announcement protocols operate, and also the TCP port number over which the unicast discovery protocol operates by default.

The `getRequestAddress` and `getAnnouncementAddress` methods return the addresses of the multicast groups over which multicast request and multicast announcement take place, respectively. These methods may throw a `java.net.UnknownHostException` if called in a circumstance under which multicast address resolution is not permitted.

### DU.3.6 Marshalling Unicast Discovery Requests

The `OutgoingUnicastRequest` class provides facilities for marshalling unicast discovery requests into a form suitable for transmission over a network.

```

package net.jini.discovery;

import java.io.IOException;
import java.io.OutputStream;

public class OutgoingUnicastRequest {
    public static void marshal(OutputStream str)
        throws IOException {...}
}

```

This class cannot be instantiated, and its only public method is static.

### DU.3.7 Unmarshalling Unicast Discovery Requests

The `IncomingUnicastRequest` class provides facilities for unmarshalling unicast discovery requests.

```

package net.jini.discovery;

import java.io.InputStream;

```

```
import java.io.IOException;

public class IncomingUnicastRequest {
    public IncomingUnicastRequest(InputStream str)
        throws IOException {...}
}
```

Since, under the current version of the unicast discovery protocol, no useful information is transmitted in a request, this class has no public methods.

### DU.3.8 Marshalling Unicast Discovery Responses

The `OutgoingUnicastResponse` class provides marshalling facilities for unicast discovery responses.

```
package net.jini.discovery;

import java.io.IOException;
import java.io.OutputStream;
import net.jini.core.lookup.ServiceRegistrar;

public class OutgoingUnicastResponse {
    public static void marshal(OutputStream s,
                               ServiceRegistrar reg,
                               String[] groups)
        throws IOException {...}
}
```

This class may not be instantiated. The sole static method, `marshal`, writes the given registrar proxy to the given output stream, and indicates that it is a member of the given set of groups (which is represented as an array which should have no null members, but may contain duplicates). If a problem occurs during marshalling or writing, it throws a `java.io.IOException`.

### DU.3.9 Unmarshalling Unicast Discovery Responses

The `IncomingUnicastResponse` class allows a caller to unmarshal a unicast discovery response.

```
package net.jini.discovery;

import java.io.IOException;
import java.io.InputStream;
import net.jini.core.lookup.ServiceRegistrar;

public class IncomingUnicastResponse {
    public IncomingUnicastResponse(InputStream s)
        throws IOException, ClassNotFoundException {...}
    public ServiceRegistrar getRegistrar() {...}
    public String[] getGroups() {...}
}
```

The constructor unmarshals a response from an input stream, and throws an exception if the reading or the unmarshalling fails. The `getRegistrar` method returns the unmarshalled registrar proxy. The `getGroups` method returns the set of groups of which the given lookup service is a member. This set is represented as an array of strings, with no null members (duplicate members may appear, however). The `equals` method returns true if both instances have the same registrar.

