SLP Manual

Glenn Hisdal

2004

Table of Contents

Introduction

The Service Location Protocol (SLP) specifies how services can be advertised and located over a network. The protocol is standardised by the Internet Engineering Task Force (IETF), and its specification can be found in RFC-2608. A service is identified by a Service URL. This URL is unique for each service. The Service URL specifies the type of service and the location of the service. Searching for a specific type of service, returns the service URLs for all services of that type. In addition to the URL, a service can have a set of attributes. These attributes can be used in searches in order to find only a subset of the services of a given type.

The Service Location Protocol defines three agents. The Service Agent (Advertiser), the User Agent (Locator) and the Directory Agent. The Service Agent is responsible for advertising one or more services. It replies to service requests sent by the User Agents. The User Agent works on behalf of a client, and discovers services in the network. The Directory Agent provides a cache for service advertisements. All Service Agents in the network registers their services with the Directory Agent. When possible, the User Agent will send service requests to a Directory Agent using unicast. When no Directory Agent can be used, the requests are multicast, and then received by any Service Agent reachable by multicast.

Each agent is assigned one or more scopes. A scope is given as a single String which is the name of the scope. An agent can only work with services that are in at least one of the scopes assigned to that agens. Thus, a User Agent can only discover services in the scopes it is configured to use. As a special case, the User Agent can be configured with an empty scope list. In which case it is able to discover services in any scope.

Service Location Protocol API

This section describes how to use the API methods of the SLP library for advertising or locating a service. These methods are used within SmartFrog components and other programs using the Service Location Protocol for advertising a service they provide, or locating a service they need. All classes needed to access the Service Location Protocol are found in the org.smartfrog.services.comm.slp package. The information given here should be enough to get started with SLP. More details can be found in the Javadoc files for the SLP library.

The Service Location Manager

The Service Location Manager manages the access to the SLP framework. It is implemented in the class **ServiceLocationManager**. The main use of this class is to obtain an Advertiser or Locator object that can be used for advertising or locating services. Below is the description of the most used methods belonging to this class.

```
static Advertiser getAdvertiser(Locale loc)
      Returns an Advertiser object for the given language. The returned object
      can be used for advertising a service.

static Locator getLocator(Locale loc)
      Returns a Locator object for the given language. The returned object can be
      used for discovering services, service types and service attributes
      advertised through SLP.

static Vector findScopes()
      Returns a Vector with the names of all scopes known by the Advertiser and
      Locator objects managed by this ServiceLocation-Manager.

static void setProperties(Properties p)
      Sets the properties to use for future Advertiser or Locator objects.
```

Advertising a Service

An Advertiser object is used to advertise services. The Advertiser is obtained by a call to `ServiceLocationManager.getAdvertiser(locale).` To register a service to be advertised through SLP, one must create a Service URL identifying the service. A Vector of ServiceLocationAttribute objects must also be provided with the registration in order to assign a set of attributes to the service. This may be an empty Vector if no attributes are to be given for the service. An overview of the methods used to register or de-register a service is given below:

```
void register(ServiceURL url, Vector attributes)
      Registers a service that is to be advertised using SLP.

Void deregister(ServiceURL url)
      De-registers a service. After this, the service is no longer advertised.
```

A typical program for advertising a service is shown below. More information on the classes and methods used can be found in the Javadoc files for the SLP library.

```
class MyAdvertisedService {
    void main(String[] args) {
        ...
        // create URL and attributes.
        ServiceURL url = new ServiceURL("service:my_service://myHost",
                                        ServiceURL.LIFETIME_PERMANENT);
        Vector attributes = new Vector();
        attributes.add(new ServiceLocationAttribute("anId", "aValue"));

        // advertise service.
        Advertiser adv = ServiceLocationManager.getAdvertiser(new Locale("en"));
        adv.register(url, attributes);
        ...
```

```
        // stop advertising service.
        adv.deregister(url);
    }
}
```

Locating a Service

In order to locate services, a Locator object is used. This is obtained by a call to `ServiceLocationManager.getLocator(locale)`. After obtaining the Locator object, one can use the following methods to discover services, service types and service attributes in the network.

`ServiceLocationEnumeration findServices(ServiceType t, Vector scopes, String filter)`

Searches for services of a given type. The returned ServiceLocationEnumeration contains zero or more ServiceURLs identifying the discovered services. The parameters are:

- t: The type of service to look for.
- Scopes: A Vector with the scopes to search in. (Only scopes supported by the Locator will be used)
- filter: An LDAPv3 search filter used to specify that some attributes must be present and have a certain value. This may be an empty String to find all services of the given type. The following operators are supported for use within the search filter in the current version of the SLP libbrary.
  - | (or): At least one of the attributes must be present with the correct value.
  - & (and): All given attributes must be present with the correct value.
  - = (equals): At least one of the values for the attribute must equal the value given in the search filter.
  - < (less than): At least one of the values for the attribute must be less than, or equal to the value given in the search filter.
  - > (greater than): At least one of the values for the attribute must be greater than, or equal to the value given in the search filter.

  A search filter may look like: (&(attr1=3)(attr4<8)).

`ServiceLocationEnumeration findServiceTypes(String namingAuthority, Vector scopes)`

Finds all the service types available in the given scopes. The namingAuthority parameter may be the name of a naming authority, an empty String or null. If a naming authority is given, only service types registered with that naming authority are returned. If an empty String is provided, only service types registered with the default naming authority (IANA) are returned. Setting this parameter to null will result in all available service types being returned. The ServiceLocationEnumeration returned by this method contains zero or more ServiceType objects. One for each type of service that is discovered.

`ServiceLocationEnumeration findAttributes(ServiceURL url, Vector scopes, Vector attributeIds)`

Finds all attributes for the service identified by the Service URL url (in the given scopes). The attributeIds Vector can be used to select only a subset of the attributes. If the Vector is not empty, only attributes listed in this Vector are returned. If empty, all attributes are returned. The ServiceLocationEnumeration returned by this method contains zero or more ServiceLocationAttribute objects. One for each attribute.

`ServiceLocationEnumeration findAttributes(ServiceType type, Vector scopes, Vector attributeIds)`

Identical to the above method, except that the attributes for all services with the Service Type serviceType are returned.

Configurable Properties

This section lists the properties that can be set to control how the SLP library works. If one does not want a Locator or Advertiser to use the default values for these properties, a Java Properties object can be created and passed on to the ServiceLocationManager by calling `ServiceLocationManager.setProperties(properties)` before the Locator or

Advertiser is created. The Directory Agent, as a standalone application, is currently not configurable. If the DA is started as a SmartFrog component, it will get its configuration from the SmartFrog description file in the same way as the Locator and Advertiser.

- **net.slp.multicastMaximumWait**

  Sets the time to wait for replies to multicast requests. During this time the original request may be re-sent a number of times. Default value is 15000 (15 seconds).

- **net.slp.randomWaitBound**

Maximum value for various random waits used in the library. Default is 1000 (1 second).

- **net.slp.initialTimeout**

  The time before a request is retransmitted for the first time. The time to wait is doubled each time a request is retransmitted. Default is 2000 (2 seconds).

- **net.slp.unicastMaximumWait**

  Maximum time to wait for a reply to a unicast request. During this time, the request can be retransmitted a number of times. Default value is 15000 (15 seconds).

- **net.slp.DAHeartBeat**

  The time between each multicast DAAdvert message sent by the Directory Agent to advertise its existence. Default is 10800000 (3 hours).

- **net.slp.DAActiveDiscoveryInterval**

  The time between each attempt by a User Agent or Service Agent to actively discover a Directory Agent. The discovery is done by sending a service request for the service "service:directory-agent". Default is 900000 (900 seconds).

- **net.slp.useScopes**

  The scopes to be used by the agent. Given as a comma-separated list of scope names. Default is "default".

- **net.slp.DAAddresses**

  Comma-separated list of IP-addresses or hostnames giving the location of one or more Directory Agents. This is useful if the DA can not be discovered by multicast. Defaultis "" (no predefined DA).

- **net.slp.passiveDADetection**

  A boolean enabling or disabling passive DA detection. If disabled, the UAs and SAs will not listen for DAAdvert messages on the multicast address. Default is true (enabled).

- **net.slp.MTU**

  MTU for SLP messages (maximum message size). Default is 1400.

- **net.slp.port**

  The default port for SLP messages. All requests are sent to this port. Default is 427.

- **net.slp.uaport**

Port used for sending messages from the UA. When set to 0, the system will select any free port. If, for some reason, a special port has to be used, set this attribute to that port number. Default is 0.

- **net.slp.saport**

  Port used for sending messages from the SA. This is also the port on which the SA will be able to receive TCP requests. Default is 0 (system selects).

- **net.slp.locale**

  The language supported by the agent. Each UA/SA supports one language. To register a service in multiple languages, one Advertiser for each language is required. This attribute is automatically set by the service location manager when the getAdvertiser or getLocator methods are called, as these take the language as a parameter. Default is "en" (English).

- **net.slp.multicastAddress**

  The multicast address used for SLP requests. Default is 239.255.255.253.

- **net.slp.interface**

  The IP-address of the network interface to use for SLP. Use this if a host has more than one network interface, and you do not want to use the default. Default is `InetAddress.getLocalHost()`.

- **net.slp.debug**

  Enable/disable debug output from the SLP library. Default is false.

- **net.slp.logErrors**

  Enable/disable logging of errors in the SLP library. Default is false.

- **net.slp.logMsg**

  Enable/disable logging of messages sent and received in the SLP library. Default is false.

- **net.slp.logfile**

  The name of the file to write debug, errors and message logs to. If an empty String is given, all output goes to stdout. Default is "" (stdout).

SmartFrog Components

This section will show how to use the SLP library directly in SmartFrog descriptions by using the provided SmartFrog components for SLP.

SFSlpAdvertiser

The SFSlpAdvertiser component is used to advertise things through SLP. The value of the attribute "toAdvertise" must be set to what you want to advertise. The value can be set directly, or it can be a link to another attribute. Some other attributes must also be set for the advertisement to work. These are listed below. In addition, a number of attributes can be set to change the configuration of the SLP system. These are given in a separate section: "SmartFrog SLP Configuration".

- **serviceType**

  The service type for the advertised service. The service type must be given in the format specified in the SLP standard (RFC-2608). An example is "service:sf-prim:printer". This attribute must always be given.

- **toAdvertise**

  The thing to advertise. This can be a simple value (String, Integer, ...) or a reference to another attribute. This attribute must always be given.

- **serviceAttributes**

  Used to assign a set of attributes to the advertised service. An attribute is given as a Vector where the first element is the name (id) of the attribute and the remaining elements are the values for that attribute. A Vector of such attributes should be given here.

  Example: [["a1Name","a1Value1","a1Value2"], ["a2Name","a2Value1"]]

- **serviceLifetime**

  The life time of the service. That is, how long the service is to be advertised. This is given as a positive Integer giving the number of seconds for which the service is advertised. Alternatively, a life time of -1 can be given. This makes the advertisement permantent (advertised as long as the advertiser component is running). The default is -1.

- **advertiseReference**

  When the "toAdvertise" attribute is a Reference, setting this to "true" will cause the reference to be advertised instead of the resolution of the reference. The default is "false".

Below is an example showing how the Printer component from the HelloWorld example provided with SmartFrog can be advertised using SLP.

```
#include "org/smartfrog/examples/helloworld/printer.sf"
#include "org/smartfrog/services/comm/slp/components.sf"

sfConfig extends Compound {
    p extends Printer;
    adv extends SFSlpAdvertiser {
        serviceType "service:sf-prim:printer";
        toAdvertise LAZY p;
    }
}
```

SFSlpLocator

The SFSlpLocator component is used to locate advertised services. Other components use a reference to the "result" attribute of the SFSlpLocator component in order to get the result of the discovery. Some attributes must be set for the SFSlpLocator component to be able to discover services. In addition to these, the configuration of the SLP system can be set as for the SFSlpAdvertiser component.

- **serviceType**

  The service type for the service to discover. This attribute must be given.

- **searchFilter**

A String giving the search filter to use for the discovery. The search filter must be given in the format shown in Section . Specifying the attribute is optional. The default is to use an empty String.

- **discoveryInterval**

  The search done by the component may be repeated periodically. This attribute sets the time (in milliseconds) between each search. A value of zero, means that the search is only performed once, and not repeated. This attribute is optional. Default is 0.

- **discoveryDelay**

  The delay (in milliseconds) before the first attempt at finding the service. This is to allow the User Agent to have time to discover any Directory Agents before it starts searching for services. This attribute is optional. Default value is 0.

- **returnEnumeration**

  This is a boolean controlling what is returned from the SFSlpLocator component when the result of the SLP discovery is requested. When set to true, the unmodified ServiceLocationEnumeration object is returned. The component requesting the service must then be able use this to find the services. The default value is false. In this case, the first service in the enumeration is returned. This is the service with the longest life time at the time of the search. The attribute is optional.

- **result**

  This is attribute is created dynamically each time a component tries to resolve it. Trying to resolve this will return the object found by the service discovery.

The example below shows how the Generator component in the HelloWorld example can use SLP to find an advertised Printer component.

```
#include "org/smartfrog/examples/helloworld/generator.sf"
#include "org/smartfrog/services/comm/slp/components.sf"

sfConfig extends Compound {
    g extends Generator {
        printer LAZY loc:result;
    }
    loc extends SFSlpLocator {
        serviceType "service:sf-prim:printer";
    }
}
```

SFSlpDA

The SFSlpDA component is used to start an SLP Directory Agent. This component has no attributes except for the configuration of the SLP system. Two scripts are provided to start a Directory Agent with the default configuration on a host. These scripts are bin/sfStartDA and bin/sfTerminateDA.

sfStartDA

This command can be used to start a Directory Agent on a host. The syntax is "sfStartDA <hostname>" where <hostname> is the name of the host to start the DA on.

sfTerminateDA

This command is used to terminate a Directory Agent. The syntax is "sfTerminateDA <hostname>" where <hostname> is the name of the host on which the DA is running.

SFSlpDeployerImpl

The SFSlpDeployerImpl class is a special SmartFrog deployer class that will use SLP to search for an advertised ProcessCompound in which to deploy a component. If no ProcessCompound is found in the SLP search, the standard deployer class is used to deploy the component. A ProcessCompound is advertised by setting the "toAdvertise" attribute in the SFSlpAdvertiser component to point to the "sfProcess" attribute. An example is given below:

```
sfConfig extends Compound {
    adv extends SFSlpAdvertiser {
        serviceType "service:sf-pc";
        toAdvertise LAZY sfProcess;
    }
}
```

In order to use SLP, the deployer class needs to know which service to look for. This is done by adding the attribute "slpConfig" to the description. This must be present, and have at least the "serviceType" attribute set. Other possible attributes are "searchFilter" and the various SLP configuration attributes. These are the same as for the SFSlpLocator component. Below is an example on how to use this class.

```
SfConfig extends Compound {
    sfDeployerClass "org.smartfrog.services.comm.slp.SFSlpDeployerImpl";
    slpConfig extends SFSlpConfiguration {
        serviceType "service:sf-pc";
    }
    ...
}
```

SmartFrog SLP Configuration

This section gives an overview of the possible SmartFrog attributes for configuring the SLP agents used by the SmartFrog components. The default values for these attributes are stored in the "org/smartfrog/services/comm/slp/sf/SFSlpConfiguration.sf" file. Each attribute corresponds to one of the properties listed in Section . The table below lists the possible attributes.

SmartFrog attribute	*Property*
slp_config_mc_max	net.slp.multicastMaximumWait
slp_config_rnd_wait	net.slp.randomWaitBound
slp_config_retry	net.slp.initialTimeout
slp_config_retry_max	net.slp.unicastMaximumWait
slp_config_da_beat	net.slp.DAHeartBeat
slp_config_da_find	net.slp.DAActiveDiscoveryInterval
slp_config_daAddresses	net.slp.DAAddresses
slp_config_scope_list	net.slp.useScopes
slp_config_mtu	net.slp.MTU
slp_config_port	net.slp.port
slp_config_locale	net.slp.locale
slp_config_mc_addr	net.slp.multicastAddress

SmartFrog attribute	*Property*
slp_config_interface	net.slp.interface
slp_config_debug	net.slp.debug
slp_config_log_errors	net.slp.logErrors
slp_config_log_msg	net.slp.logMsg
slp_config_logfile	net.slp.logfile