APS Groups

User Guide

Version: 1.0.0

Author: Tommy Svensson

Table of Contents

1 APSGroups

Provides network groups where named groups can be joined as members and then send and receive data messages to the group. This is based on multicast and provides a verified multicast delivery with acknowledgements of receive to the sender and resends if needed. The sender will get an exception if not all members receive all data. Member actuality is handled by members announcing themselves relatively often and will be removed when an announcement does not come in expected time. So if a member dies unexpectedly (network goes down, etc) its membership will resolve rather quickly. Members also tries to inform the group when they are doing a controlled exit.

Please note that this does not support streaming! That would require a far more complex protocol. APSGroups waits in all packets of a message before delivering the message.

1.1 OSGi service usage

The APSGroupsService can be used as an OSGi service and as a standalone library. This section describes the service.

1.1.1 Getting the service

```
APSServiceTracker<APSGroupService> apsGroupsServiceTracker =
    new APSServiceTracker<APSGroupsService>(bundleContext, APSConfigService.class,
    APSServiceTracker.LARGE_TIMEOUT);
APSGroupsService apsGroupsService = apsGroupsServiceTracker.getWrappedService();
```

1.1.2 Joining a group

```
GroupMember groupMember = apsGroupsService.joinGroup("mygroup");
```

1.1.3 Sending a message

To send a message you create a message, get its output stream and write whatever you want to send on that output stream, close it and then send it. *Note* that since the content of the message is any data you want, all members of the groups must know how the data sent looks like. In other words, you have to define your own message protocol for your messages. Note that you can wrap the OutputStream in an ObjectOutputStream and serialize any java object you want.

```
Message message = groupMember.createNewMessage();
OutputStream msgDataStream = message.getOutputStream();
try {
    ...
    msgDataStream.close();
    groupMember.sendMessage(message);
}
catch (IOException ioe) {
    ...
}
```

Note that the `groupMember.sendMessage(message)` does throw an IOException on failure to deliver the message to all members.

1.1.4 Receiving a message

To receive a message you have to register a message listener with the GroupMember object.

```
MessageListener msgListener = new MyMsgListener();
groupMember.addMessageListener(myMsgListener);
```

and then handle received messages:

```
public class MyMsgListener implements MessageListener {
    public void messageReceived(Message message) {
        InputStream msgDataStream = message.getInputStream();
        ...
    }
}
```

1.1.5 Leaving a group

```
apsGroupsService.leaveGroup(groupMember);
```

1.2 Library usage

The bundle jar file can also be used as a library outside of an OSGi server, with an API that has no other dependencies than what is in the jar. The API is then slightly different, and resides under the se.natusoft.apsgroups package.

1.2.1 Setting up

```
APSGroups apsGroups = new APSGroups(config, logger);
apsGroups.connect();
```

The config passed as argument to APSGroups will be explained further down under "Configuration".

The *logger* is an instance of an implementation of the APSGroupsLogger interface. Either you provide your own implementation of that or your use the APSGroupsSystemOutLogger implementation.

1.2.2 Joining a group

```
GroupMember groupMember = apsGroups.joinGroup("mygroup");
```

1.2.3 Sending and receiving messages

Sending and receiving works exactly like the OSGi examples above.

1.2.4 Leaving a group

```
apsGroups.leaveGroup(groupMember);
```

1.2.5 Shutting down

```
apsGroups.disconnect();
```

1.3 Net time

All APSGroups instances connected will try to sync their time. I call this synced time "net time".

It works like this: When an APSGroups instance comes up it waits a while for NET_TIME packets. If it gets such a packet then it enters receive mode and takes the time in the received NET_TIME packet and stores a diff to that time and local time. This diff can then be used to translate back and forth between local and net time. If no such packet arrives in expected time it enters send mode and starts sending NET_TIME packets itself using its current net time. If a NET_TIME packet is received when in send mode it directly goes over to listen mode. If in listen mode and no NET_TIME packet comes in reasonable time it goes over to send mode. So among all instances on the network only one is responsible for sending NET_TIME. If that instance leaves then there might be a short fight for succession, but it will resolve itself rather quickly.

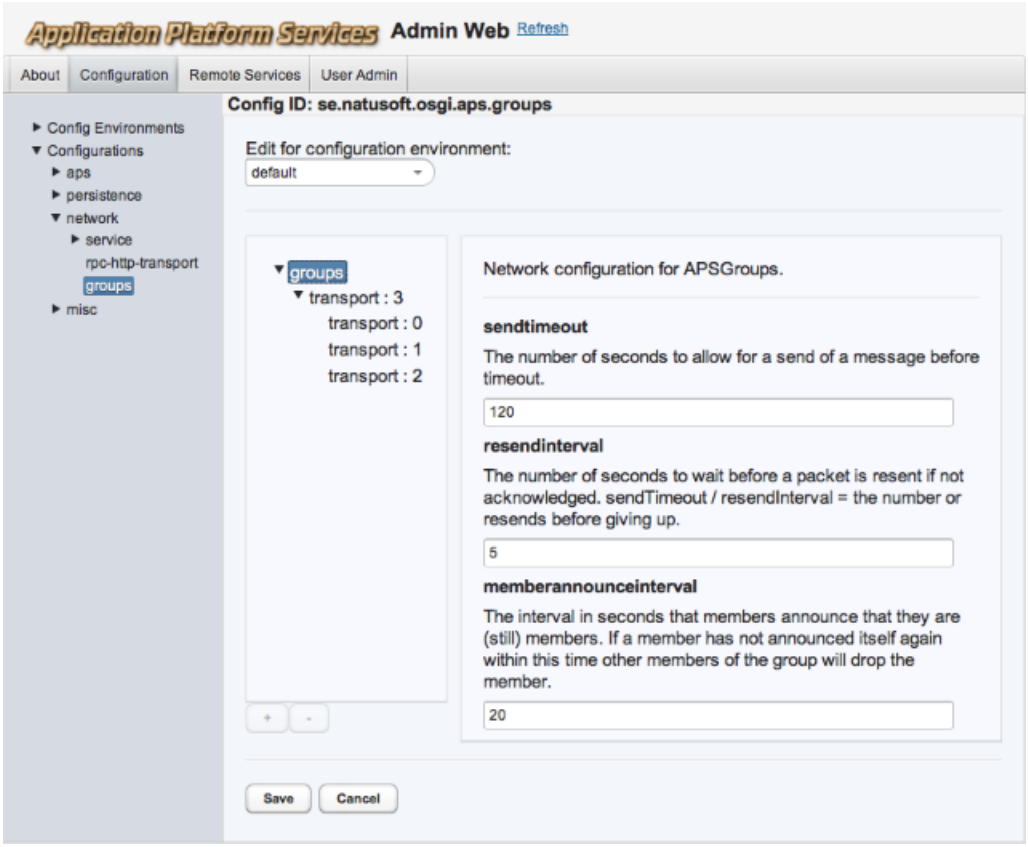
The GroupMember contains a few *create** methods to produce a *NetTime* object instance. See the API further down

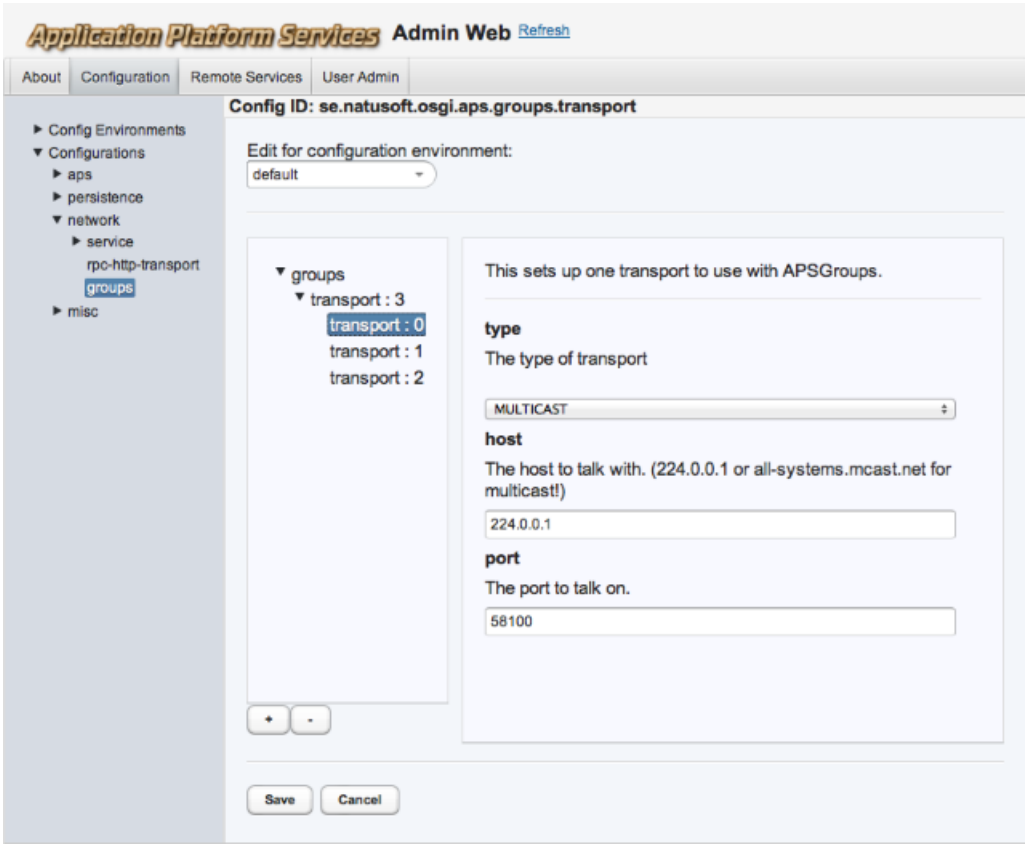
for more information on these.

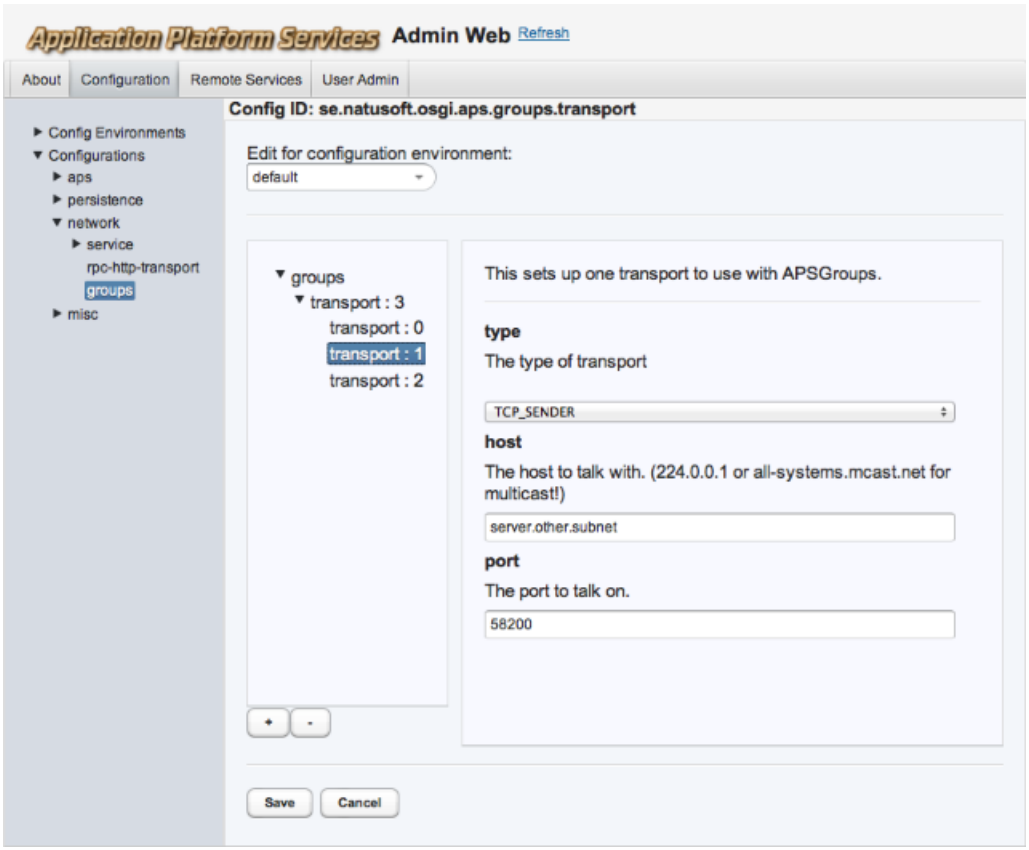
1.4 Configuration

1.4.1 OSGi service

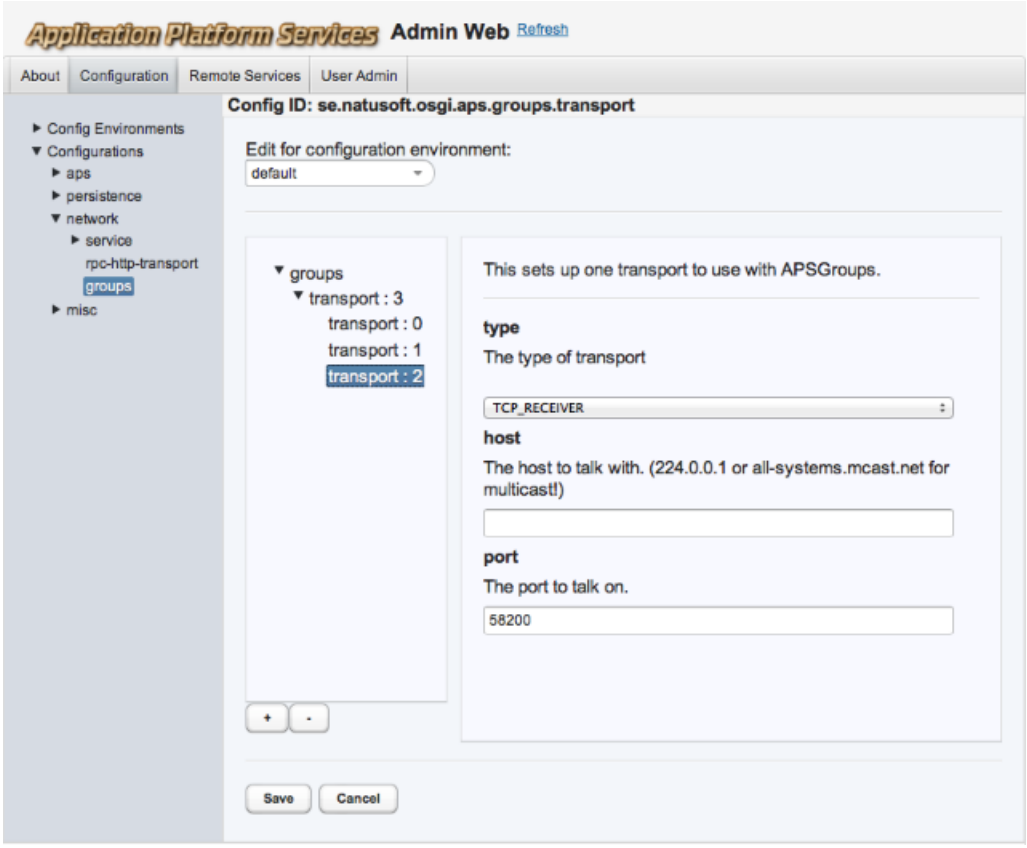
The OSGi service provides a configuration model that gets managed by the APSConfigService. It can be configured in the APS adminweb (http://host:port/apsadminweb/). Here are some screenshots of the config admin:







As can bee seen in the above screenshots transports need to be configured for communication to work. If you only need to talk to members on the same subnet the multicast transport is enough! The multicast transport makes sure that all transmitted data is received by all known group members. It will do resends if required, and throw an exception on failure of any member to acknowledge all sent packets.



If you need to talk to members on a different subnet then you need to use the TCP transports. Note that there are 2 of these: *TCP_SENDER*, and *TCP_RECEIVER*. One receiver must be configured and can receive messages from anyone. A sender is needed for each APSGroups installation you want to talk to, and should point to the receiver of that installation. Note that for a receiver you only need to specify a port. The host part is ignored by the receiver.

1.4.2 Library

The library wants an implementation of the APSGroupsConfig interface as its first argument to APSGroups(config, logger) constructor. Either you implement your own or use the APSGroupsConfigProvider implementation. This is a plain java bean with both setters and getters for the config values. It comes with quite reasonable default values. It contains exactly the same properties as shown in the screenshots above.

1.5 APIs

List<String> getGroupNames()

Returns the names of all available groups.

List<String> getGroupMembers(String groupName)

Returns a list of member ids for the specified group.

Parameters

groupName - *The name of the group to get member ids for.*

List<String> getGroupsAndMembers()

Returns a list of "groupName : groupMember" for all groups and members.

}

GroupMember joinGroup(String name) throws IOException

Joins a group.

Returns

A GroupMember that provides the API for sending and receiving data in the group.

Parameters

name - *The name of the group to join.*

Throws

java.io.IOException - *The unavoidable one!*

GroupMember joinGroup(String name, Properties memberUserData) throws IOException

Joins a group.

Returns

A GroupMember that provides the API for sending and receiving data in the group.

Parameters

name - *The name of the group to join.*

memberUserData - *Data provided by users of the service.*

Throws

java.io.IOException - *The unavoidable one!*

void leaveGroup(GroupMember groupMember) throws IOException

Leaves as member of group.

Parameters

groupMember - *The GroupMember returned when joined.*

Throws

java.io.IOException - *The unavoidable one!*

}

void addMessageListener(MessageListener listener)

Adds a listener for incoming messages.

Parameters

listener *- The listener to add.*

void removeMessageListener(MessageListener listener)

Removes a listener for incoming messages.

Parameters

listener *- The listener to remove.*

Message createNewMessage()

Creates a new Message to send. Use the sendMessage() method when ready to send it.

void sendMessage(Message message) throws IOException

Sends a previously created messaging to all current members of the group. If this returns without an exception then all members have received the messaging.

Parameters

message *- The messaging to send.*

Throws

java.io.IOException *- On failure to reach all members.*

UUID getMemberId()

Returns

The ID of the member.

List<String> getMemberInfo()

Returns information about members.

List<Properties> getMembersUserProperties()

Returns the user properties for the members.

NetTime getNow()

Returns

The current time as net time.

NetTime createFromNetTime(long netTimeMillis)

Creates from milliseconds in net time.

Parameters

netTimeMillis *- The net time milliseconds to create a NetTime for.*

NetTime createFromNetTime(Date netTimeDate)

Creates from a Date in net time.

Parameters

netTimeDate *- The Date in net time to create a NetTime for.*

NetTime createFromLocalTime(long localTimeMillis)

Creates from milliseconds in local time.

Parameters

localTimeMillis *- The local time milliseconds to create a NetTime for.*

NetTime createFromLocalTime(Date localTimeDate)

Creates from a Date in local time.

Parameters

localTimeDate *- The Date in local time to create a NetTime for.*

}

OutputStream getOutputStream()

Returns an *OutputStream* to write messaging on. Multiple calls to this will return the same *OutputStream*!

InputStream getInputStream()

Returns an *InputStream* for reading the messaging. Multiple calls to this will return new *InputStream*:s starting from the beginning!

UUID getId()

Returns the id of this messaging.

String getMemberId()

Returns

id of member as a string.

String getGroupName()

Returns

The name of the group this messaging belongs to.

}

---

public void messageReceived(Message message)

Notification of received messaging.

Parameters

message - *The received messaging.*

}

---

public long getNetTime()

Returns the number of milliseconds since January 1, 1970 in net time.

public Date getNetTimeDate()

Returns the net time as a Date.

public Calendar getNetTimeCalendar()

Returns the net time as a Calendar.

public Calendar getNetTimeCalendar(Locale locale)

Returns the net time as a Calendar.

Parameters

locale - *The locale to use.*

public Date getLocalTimeDate()

Converts the net time to local time and returns as a Date.

public Calendar getLocalTimeCalendar()

Converts the net time to local time and returns as a Calendar.

public Calendar getLocalTimeCalendar(Locale locale)

Converts the net time to local time and returns as a Calendar.

Parameters

locale - *The locale to use.*

}

```
}
```