University of Bremen
Department of
Mathematics and Computer Science
Project PoP

2004-2005
Peeranha42
Programmer's Guide

Project PoP
http://peeranha42.sf.net
mailto:grp-peeranha42@informatik.uni-bremen.de

May 2005

Contents

Introduction

Thid document is intended as a brief introduction to creating new plug-ins for the Peeranha42 framework.

Description of the Peeranha42

- Peeranha42 Client The P42 Client is the core of Peeranha42. The special plugin architecture of the client assures unlimited numbers of add-ons, so that the client can be used in all kinds of application areas. Moreover the client offeres client-server-like access to p2p technology. So that every developer can create p2p software easily.

- P42 Development Kit The P42 Development Kit gives support to developers of P42 Plugins. It contains a large collection of tools, documentations, tutorials and examples. Thus makes it possible to develope p2p applications without getting in touch with p2p technology itself.

Description of the P42 Plugins

There are already a few P42 Plugins for testing the P42 Client. They demonstrate the power of the Peeranha42 and can be seen as examples for developers of P42 Plugins as well.

- P42 Chat P42 Chat is an instant messanging plugin for the P42 Client that supports conferencing in JXTA(TM)-Peergroups.

- P42 Chess P42 Chess is a gaming plugin for the P42 Client. Besides 1-on-1 chess matches over the JXTA(TM) network the plugin furthermore supports group matches.

- P42 Filesharing P42 Filesharing is a filesharing plugin for the P42 Client. Its special feature is the use of so called overlaynetwork tech-

nologie that focus on reducing the signaling traffic in p2p networks, especially the traffic of search queries.

- P42 GameLobby P42 GameLobby is a game portal where people can meet each other and play games together.
- P42 Peerbay P42 Peerbay is a auction system that bases on the p2p networks of JXTA(TM).
- P42 Pirate Radio P42 Pirate Radio allows users to create their own radio station in the internet. Therefore P42 Pirate Radio streams mp3-files in the p2p network which can be received by other users of P42 Pirate Radio.

License

Chapter 1

Creating the PlugIn-SDK

The SDK is dedicated to developers of Peeranha42 plugIns. To create the SDK, some requirements are necessary:

- A Java-based build tool like Ant must properly run on your Operating System. If not you must first install it. The latest stable version of Ant is available from the Ant web page http://ant.apache.org.
- To build and use Ant, you must have a JAXP-compliant XML parser installed and available on your classpath.
- You will also need the JDK 1.4 or later installed on your system.

Now you can create the SDK by running `ant [option] [target]` in a shell window. In this case the needed option is `-Dsdk` and the target is `sdk`. EG. `ant -Dsdk sdk` When the build was successfully executed, a sdk directory will be created. In most cases, `ant sdk` is sufficient.

Chapter 2

Hello World Example

2.1 Creating a new Plugin

When the Plugin-SDK was successfully created, you will find a `sdk` directory inside the peeranha42 build directory. Inside, there is a `build.properties` file, which provides general information about the plugin you wish to create, as for example the name of the plugin, or the main class which is the entry point to the plugin. When you have finished editing the `build.properties`, you can use the SDK to create the basic plugin directories and advertisements: just run `ant advertisements` in the sdk directory.
Now you can start writing the plugin. Place your sourcecode in the `src` directory which the sdk has created.
The main class of a Peeranha42 plugin has to implement the interface: `de.uni\_bremen.informatik.p2p.peeranha42.core.plugin.Plugin`
As defined in the `Plugin` interface, a plugin main class has to provide the following functions:

```
public String name();
public Info getInfo();
public void run();
public void stop();
```

2.2 HelloWorld Sourcecode

The following is the complete source code of a basic Peeranha42 plugin:

```
import de.uni_bremen.informatik.p2p
        .peeranha42.plugin_core.*;
```

```
import net.jxta.endpoint.Message;
import net.jxta.peergroup.PeerGroup;
import net.jxta.protocol.PipeAdvertisement;
import java.math.BigInteger;

/**
 * This is a Hello world plugin for Peeranha42
 */
public class HelloWorld
        implements Plugin {

        /**
         * The run() method is the entry point
         * to this plugin
         * and is called once on startup
         */
        public void run() {
                System.out.println("Hello, world!");
                stop();
        }

        /**
         * The stop() method is invoked by Peeranha42
         * in order to stop this plugin
         */
        public void stop() {
        }

        /**
         * @return The name of this plugin
         */
        public String name() {
                return ("HelloWorld");
        }

        /**
         * @return The Info object for this plugin
         */
        public Info getInfo() {
                return new Info(new PluginID(
                        new BigInteger(
```

```
                "D20CE8B524B44C15BD77E243C433C08D05",
                16)),
                "HelloWorld",
                "Version 0.1",
                "I am the cool guy who wrote this."
                        );
        }

        /**
         * receive() is called by Peeranha42
         * to submit incoming messages to this plugin
         *
         * @param pg the peergroup
         * from which the message originates
         * @param pa the pipeadvertisement
         * of the message sender
         * @param msg the incoming message
         */
        public void receive(PeerGroup pg,
                PipeAdvertisement pa,
                Message msg) {
        }
}
```

2.3 Running the Plugin

When you wish to see the plugin in action, just run `ant` on the `build.xml` which was created by the SDK. This should compile the plugin. If done correctly, you can now copy the resulting `jar` file to the `peeranha42/plugins` directory and start Peeranha42.

Chapter 3

P42 GUI Interface

Peeranha42 offers a default GUI-Manager with the possibility to display one or more GUI-panels in a common tabbed window or each GUIpanel in an own frame.
Most public functions can be called with `P42_gui_default.[function()]`. Since there is no sense in instantiating more than one user interface, the public functions to be used by the Plugins are static. The constructor itself is private and invoked indirectly by a public static method which checks that there is no other instance running yet before actually invoking it.
To integrate an own GUI-panel in the common GUI, a developer just has to use a `P42_application_pane`. This is an extended class from `javax.swing.Jpanel` and can be either still extended or simply instantiated.
The swing components and layout inside the panel are completely up to the plugin programmer. The panel(s) shall then be added to a dynamic list and this list added to the GUI-manager.
After use, e.g. when the `stop()` method of the plugin is invoked, the panels that aren't needed shall be removed. Bellow there is a short list of the most important methods needed by plugin developers for using the common GUI. For more detailed information about the respective methods, constructors or classes, please refer to the online APIdocumentation.
The constructors of the `P42_application_pane`:

```
public P42_application_pane(String title,
        int state,
        PluginID pid)
```

The public field constants of the `P42_default_gui`:

```
public static final int NON_DETACHEABLE
```

```
public static final int ATTACHED
public static final int DETACHED
```

The public methods of the `P42_application_pane`:

```
public PluginID getPluginID()
public void setPluginID(PluginID pid)
public String getTitle()
public int getPreferredState()
public void setPreferredState(int state)
```

The public methods of the `P42_default_gui`:

```
public static void startGUI()
public static void main(String args[])
public static void addAppPanes(final Vector tempApps)
public static void removeAppPanes(final Vector tempApps)
public static void removeAppPanes(final PluginID pId)
public static boolean addPrefPanels(PluginID pid,
        JPanel p_panels[])
```

Chapter 4

P42 Preference Panel

You can find the preferences of Peeranha42 and the supplied Plugins as one tab of the GUI. As you can see here, the preference tab is organized by a `JTree`. Besides a main preference panel for the client, Peeranha42 offers two kinds of Preference panels for each plugin. The first one is generated automatically by the client.
Every plugin (jar file) placed in the plugin directory is listed in the Preference tab. The automatically generated panel displays information about the Plugin using the info class. Furthermore you can start and stop the plugin from this panel or mark it for auto load on start-up. If you as a developer want to use the preference tab to configure custom settings for your plugin, then you can supply these settings to the GUI by using the `addPrefPanels(PluginID pid,JPanel p_panels[])` method in the main GUI class: `P42_gui_default`. Your panels are inserted below the branch of your plugin by order of the supplied array. Adding new branches to build up an advanced hierarchic tree (with sub branches) is not implemented yet but will be added later versions.
Adding panels: This should happen while your plugin is starting. Call `addPrefPanels(..)` in your run method.
Removing the panels: If your plugin is exiting you have to call `unloadPlugin(PluginID)` in your `stop()` method. Your panels are automatically removed then.

Chapter 5

P42 Network

Requirements:

- The plugin-interface should be implemented.
- You have to create the advertisements of the plugin with the *latest* SDK. (See chapter 1.)

5.1 Basic setup and group functionality

5.1.1 Create your own service group

Before you can use the network functionality of the peeranha network you have to create a *unique* service group for your plugin. The following code example illustrates how you create your own service group:

```
PeerGroup svcPg = Network.createServiceGroup(PluginID);
```

Subsequently the service group has to be registered to our network. The following subsection shows you how to register a peer group to the network.

5.1.2 Registering of a peer group(also service group) to the network

when a peer group is registered you are able to use the methods `send()` and `receive()` of the network.

```
Network.register(PluginID, PeerGroup,
        [ Optional: UserName ]);
```

The network of peeranha42 supports the possibility for a client to have different names in several registered peer groups. Therefore you can specify any user name you want to. In the case of `null` as optional user name the network uses the given peer name in this peer group. This name is created when you start the configurator and specify a peer name.

5.1.3 Usage of the listeners in the registered peer groups

By implementing the listeners you can get events in the different registered peer groups. The listeners support different tasks like checking the state of the connection to the rendezvous and updating the arrays with the found peers, pipes or group advertisements. The advantage for the developer is that she/he does not worry about the discovery tasks. So you will be informed of changes from the different listener types.
The following listeners are provided when the peer group is registered:

- `P42RendezvousListener` (informs you about the actual state of the connection to the connected rendezvous)
- `P42PeerAdvListener` (gives you an array with the actually found Peer Advertisements)
- `P42PipeAdvListener` (gives you an array with the actually found Pipe Advertisements)
- `P42PeerGroupListener` (gives you an array with the actually found Peer Group Advertisements)

In order to use a listener you have to register it for the plugin in the right peer group. This works like it is shown in the following code example:

```
// register a P42RendezvousListeners
Network.addP42RendezvousListener(PluginID,
        PeerGroup,
        P42ConnectionListener);

// register a P42PeerAdvListeners
Network.addP42PeerAdvListener(PluginID,
        PeerGroup,
        P42PeerAdvListener);

// register a P42PipeAdvListeners
```

```
Network.addP42PipeAdvListener(PluginID,
        PeerGroup,
        P42PipeAdvListener);

// register a P42PeerGroupListeners
Network.addP42PeerGroupListener(PluginID,
        PeerGroup,
        P42PeerGroupListener);
```

As soon as a new listener is registered to the network you can handle the events which are given by these listeners.
If there are changes concerning the connection state you can get them like this:

```
public void rendezvousEvent(PeerGroup pg,
        boolean isConnectedToRendezvous) {
...
}
```

When the network finds new advertisements regarding peers, pipes or groups the corresponding listeners notice the plugin. Therefore different methods have to be implemented by Classes implementing Listeners.

```
// This method is invoked
// if the P42PeerAdvListener
// is registered
public void changedPeersEvent(PeerGroup pg,
        PeerAdvertisement peerAdv_arr[]) {
...
}

// This method is invoked
// if the P42PipeAdvListener
// is registered
public void changedPipesEvent(PeerGroup pg,
        PipeAdvertisement pipeAdv_arr[]) {
...
}

// This method is invoked
// if the P42PeerGroupListener
// is registered
```

```
public void changedGroupsEvent(PeerGroup pg,
        PeerGroupAdvertisement pgAdv_arr[]) {
...
}
```

5.1.4 Deregistering of Listeners

If a listener should be deregistered you have to do the following for the different registered listeners:

```
// deregister a P42RendezvousListeners
Network.removeP42RendezvousListener(PluginID,
        PeerGroup,
        P42ConnectionListener);

// deregister a P42PeerAdvListeners
Network.removeP42PeerAdvListener(PluginID,
        PeerGroup,
        P42PeerAdvListener);

// deregister a P42PipeAdvListeners
Network.removeP42PipeAdvListener(PluginID,
        PeerGroup,
        P42PipeAdvListener);

// deregister a P42PeerGroupListeners
Network.removeP42PeerGroupListener(PluginID,
        PeerGroup,
        P42PeerGroupListener);
```

5.1.5 Deregistering of peer groups

Before you can leave a registered peer group you have to invoke the deregister function in the network of peeranha42. Therefore you can use two different ways which are described in the following subsections.

Leave all registered peer groups of a plugin

With the help of this method you can leave all joined peer groups of a plugin. This method should be invoked at the latest in the stop method of the plugin. When the method is invoked you only have to give the pluginID as parameter.

```
Network.deregister(PluginID);
```

Leave a specified peer group of a plugin

To leave a specified peer group of a plugin you have to do the following:

```
Network.deregister(PluginID,
        PeerGroup);
```

5.1.6 Creating peer group(s)

Creation of a new peer group

You can create a new peer group with the following statement:

```
PeerGroup newpg =
        Network.createGroup(PeerGroup parentPeerGroup,
                String peerGroupName,
                String peerGroupDescription);
```

If `null` is inserted as ParentPeerGroup the new created peer group is a subgroup of the Peeranha42PeerGroup. You usually can use each arbitrary peer group. After the new peer group has been created it has to to be registered before you can use the network in this group.

Creating a peer group with a found peer group advertisement

If you have an existing PeerGroupAdvertisement you can use it to create a new peer group instance. This is especially used if you have found PeerGroupAdvertisements with the registered P42PeerGroupListener or manually with getGroupDiscoveryResults(described in the next section).

```
PeerGroup newpg =
        Network.createGroup(PeerGroupAdvertisement,
                ParentPeerGroup);
```

5.1.7 How do I get the Advertisements without using a listener of a specified peer group?

- It is possible to get the pipe advertisements in a specified peer group of plugin by the folowing statement:

```
PipeAdvertisement[] peers =
        Network.getPipeDiscoveryResults(PluginID,
                PeerGroup,
                searchTerm);
```

After you have done this you get an array with the discovered pipe advertisements in this peer group. By using the search term it is possible to specify the search. When you have found the pipe of a peer you are able to send this peer messages like it is described in the next section.

- If you want to find the peer group advertisements in a peer group of a plugin you can do it like this:

```
PeerGroupAdvertisement[] groups =
        Network.getGroupDiscoveryResults(PluginID,
                PeerGroup,
                searchTerm);
```

This function works analogue to getPipeDiscoveryResults above. The unique difference is that you get back an array with peer group advertisents.

- The way to get the peer advertisements works similar to the two methods above. If any peers are found in the specified group you get back an array with the found peer advertisements. If there are no peers found in the group `null` will be returned by the three functions.

```
PeerAdvertisement[] peeradvs =
        Network.getGroupDiscoveryResults(PluginID,
                PeerGroup,
                searchTerm);
```

This function is necessary if you want to acquire the IDs of peers in order to send a message to a specified set of peers(peerIDs).

5.2 Sending messages over the network

When a peer group has been registered with the help of the register method to the network the plugin is able to send messages in the registered peer group. Therefore you have three possiblities:

5.2.1 Sending unicast messages

If you only want to send a message to one peer you can do this by executing the following statement:

```
Network.sendMsg(PluginID,
        PeerGroup,
        PipeAdvertisement,
        Message);
```

The parameter peer group indicates the peer group in which the message should be sent. With the specification of the pipe advertisement you define the peer who should be receive the message.

5.2.2 Sending multicast messages

If you want to send a message to all peers in peer group except yourself you can do this by leaving out the pipe advertisement of the unicast sending method. The other parameters remain the same.

```
Network.sendMsg(PluginID,
        PeerGroup,
        Message);
```

5.2.3 Sending messages to a Set of peers(peerIDs)

If you have the desire to send messages to a specified set of peers you get a Set or Collection(for more information look at the java API at http://java.sun.com) with the corresponding peerIDs of the specified peers.

```
Network.sendMsgToSet(PeerGroup,
        Set,
        Message);
```

5.3 Receiving messages

If you want to use the following receive function there has to be an implemented Receiver-Interface. By implementing this Receiver you also have to implement the method `public Plugin getPlugin()`. This should return the reference to your plugin. Unless you have not implemented this method you are not able to receive messages in the following receive method:

```
public void receive(PeerGroup pg,
        PipeAdvertisement pa,
        Message msg)
```

The parameter peer group indicates in which peer group the received message was sent. Furthermore the method gives returns the pipe advertismement of the peer who sent this message. So you are able to reply to this message.

Chapter 6

P42 PeerListManager

The `PeerListManager` is started when `Network.startServices()` is called. After this intialization all services provided by the `PeerListManager` are available. It is neither neccessary, nor intended to have the `PeerListManager` started by a plugin thereafter. All needed methods for the usage of the `PeerListManager` are located in the `PeerListManager`-class.

6.1 Basic setup and usage

Most of the `PeerListManager`-actions that are performed are concerned with the interchange of messages between peers. This can either work properly, take a long time to perform or not work at all. The interface between `PeerListManager` and a plugin is realized by an event listener system. A plugin which shall benefit from using the `PeerListManager` has to implement the following interface: `PeerListEventListener`. The initialization and setup of such a listener-object goes as follows:

```
//initialize an own listener
MyPeerListEventListener listener =
        new MyPeerListEventListener();

//setup listener
PeerListEventManager.
        addPeerListEventListener(listener);
```

Which methods should be implemented, and how, is not prescribed and depends on which information seem important to the plugin designer.

6.2 Add a peer

```
PeerListManager.addPeer(Peer p);
```

Description of parameters:

- Peer p: Peer Object

6.3 Remove a peer

```
PeerListManager.removePeer(PeerID peerID) ;
```

Description of parameters:

- PeerID peerID: The unique `PeerID` of the peer which is to be removed.

6.4 Information about peers in the peerlist

The `PeerListManager` contains various information profiles about each peer. There is volatile information about a peer on one hand, and static information about a peer on the other hand. This distinction enables plugins to gather information about a peer which may be relevant at one moment, but becomes irrelevant the next time the plugin is started, and to seperate those information from the static information about a peer, which never changes. Such information is transferred to the peerlist the first time when a peer is added. Static information transferred in this way is locally stored on the hard disc.
To access volatile information about a peer at a later time as well as to update it at runtime, one can use the following method:

```
PeerListManager.updatePeerProfile(PeerID peerID);
```

To access information about a peer one needs the `Peer`-object of that peer:

```
Peer p=PeerListManager.getPeer(PeerID peerID);
```

Description of parameters:

- PeerID peerID: The unique ID of a peer.

Now volatile and static information about a peer are accessible in the following ways:

```
// volatile information
p.getVolatilePeerProfileEntry(String key)
// or static information
p.getStaticPeerProfileEntry(String key)
```

Description of parameters:

- String key: A key string for the information in a profile. Key strings have the following format:
  - key='foo'
  - key= 'foo.bar' , where *bar* is hierarchically ordered beneath *foo*.
  - key= 'foo.bar.something' accordingly with *something* one level deeper.

Under certain circumstances it might be useful to store information in a plugin-specific way, to make information accessible to other applications. For this purpose, there are further methods which also work according to the static/volatile scheme. Those methods need a `PluginID` as an additional parameter. Information stored in that way have keys of the form:

`key=pluginID.key`

```
// volatile plugin information
p.getVolatilePluginProfileEntry(String key,
        String pluginID)
// or static plugin information
p.getStaticPluginProfileEntry(String key,
        String pluginID)
```

Chapter 7

Appendix

Homepage

http://peeranha42.sf.net

Contact

mailto:grp-peeranha42@informatik.uni-bremen.de