HTML Parser API

by David McNicol

Package
nippu.printingmanagement

nippu.printingmanagement

Class NippuPrinterSimulator

```
java.lang.Object
   ┗-nippu.NippuPrinter
      ┗-nippu.printingmanagement.NippuPrinterSimulator
```

public class **NippuPrinterSimulator**
extends NippuPrinter

The class representing a simulation of a printer in the nippu printing management system.

Author:
Michael Forsstrom

Fields

logger

`protected static org.apache.log4j.Logger` **`logger`**

The logger

parentPrintingManagement

`private nippu.printingmanagement.PrintingManagement` **`parentPrintingManagement`**

The PrintingManagement object controlling this printer instance

printerQueue

`private java.util.List` **`printerQueue`**

The printer queue

printerTimer

`private java.util.Timer` **`printerTimer`**

The timer for the printer

timerJob

`private nippu.PrintJob` **`timerJob`**

The job in the timer

timerHasJobs

`private boolean` **`timerHasJobs`**

A boolean representing the state of the timer

Constructors

(continued on next page)

(continued from last page)

NippuPrinterSimulator

```
public NippuPrinterSimulator(java.lang.String ID,
                             int X,
                             int Y,
                             int Z,
                             int speed,
                             PrintingManagement callerPrintingManagement)
```

Constructor

Parameters:

`ID` -
The printer identification number
`X` -
The x-coordinate for the printer
`Y` -
The y-coordinate for the printer
`Z` -
The z-coordinate for the printer
`speed` -
The speed of the printer
`callerPrintingManagement` -
The printing management instance that owns this Simulator

Methods

getPrinterQueue

```
public java.util.List getPrinterQueue()
```

Getter for the printer queue

Returns:

The printer queue List

---

getPrinterTimer

```
public java.util.Timer getPrinterTimer()
```

Getter for the printer timer

Returns:

The printerTimer Timer

---

getTimerJob

```
public PrintJob getTimerJob()
```

Getter for the job in the timer

Returns:

The timerJob PrintJob

---

timerHasJobs

```
public boolean timerHasJobs()
```

Getter for the timerHasJobs field. Tells wheter there are jobs in the timer or not

(continued on next page)

(continued from last page)

Returns:

The value of timerHasJobs

setTimerHasJobs

`public void setTimerHasJobs(boolean newValue)`

Setter public methods

setPrinterTimer

`public void setPrinterTimer(java.util.Timer newTimer)`

setTimerJob

`public void setTimerJob(PrintJob job)`

Setter for the timerJob

Parameters:

`job` -
The job to set as the timerJob

toString

`public java.lang.String toString()`

Implementation of the toString method.

Returns:

A string representation of the NippuPrinterSimulator.

addJob

`public void addJob(PrintJob newJob)`

Adds a new job to the timer or printer queue

Parameters:

`newJob` -
The PrintJob to add

getNumberOfJobs

`public int getNumberOfJobs()`

Returns the number of jobs in the timer and printer queue

Returns:

The number of jobs in the system as an integer

main

`public static void main(java.lang.String args)`

A main method for testing purposes

nippu.printingmanagement

Class PMTimerTask

```
java.lang.Object
    |
    +-java.util.TimerTask
        |
        +-nippu.printingmanagement.PMTimerTask
```

public class **PMTimerTask**
extends java.util.TimerTask

A class for the TimerTask used for the printers in the printing management

Author:
Michael Forsstrom

Fields

job

```
private nippu.PrintJob job
```

The job which is to be printed

parentNPS

```
private nippu.printingmanagement.NippuPrinterSimulator parentNPS
```

The parent NippuPrinterSimulator instance associated with this task

parentPM

```
private nippu.printingmanagement.PrintingManagement parentPM
```

Then parent PrintingManagement instance associated with this task

logger

```
protected static org.apache.log4j.Logger logger
```

The logger

Constructors

PMTimerTask

```
public PMTimerTask(PrintJob assignedJob,
                   NippuPrinterSimulator parentNippuPrinterSimulator,
                   PrintingManagement parentPrintingManagement)
```

The constructor.

Parameters:

`assignedJob` -
The job to "print"
`parentNippuPrinterSimulator` -
The parent NippuPrinterSimulator

(continued from last page)

`parentPrintingManagement` -
Then parent PrintingManagement

Methods

run

```
public void run()
```

The run method

---

main

```
public static void main(java.lang.String args)
```

A main method for testing purposes.

nippu.printingmanagement

Class PrintingManagement

```
java.lang.Object
   |
   +-nippu.printingmanagement.PrintingManagement
```

All Implemented interfaces:
terttu.server.scheduler.Executor

---

public class **PrintingManagement**
extends java.lang.Object
implements terttu.server.scheduler.Executor

Fields

logger

`protected static org.apache.log4j.Logger` **`logger`**

The logger

---

configuration

`private java.lang.String` **`configuration`**

The contents of the main configuration file.

---

registered

`private boolean` **`registered`**

True iff we have been registered to the scheduler.

---

prScProtocol

`private nippu.communications.PrintingManagementSchedulerProtocol` **`prScProtocol`**

Our connection to the scheduler.

---

uIVector

`private java.util.Vector` **`uIVector`**

A vector containing all the user interface connections

---

printerVector

`private java.util.Vector` **`printerVector`**

A vector containing all the printers

---

iDCount

`private int` **`iDCount`**

Counter for the user interface identification numbers

superuserID

`private int` **`superuserID`**

The identification number of the first superuser instance

schedulerStopped

`private boolean` **`schedulerStopped`**

A boolean telling whether the scheduler is stopped or not

pendingJobs

`private java.util.List` **`pendingJobs`**

A linked list containing pending jobs

schedulerLocation

`private java.lang.String` **`schedulerLocation`**

The location of the scheduler

schedulerPort

`private int` **`schedulerPort`**

The port used by the scheduler

TCPPORT

`final java.lang.String` **`TCPPORT`**

The TCP port to use when functioning as a server

PMCONFIG

`final java.lang.String` **`PMCONFIG`**

The location of the printing management configuration file.

NIPPUCONFIG

`final java.lang.String` **`NIPPUCONFIG`**

The location of the Nippu configuration file.

schedulerStateLock

`private java.lang.Object` **`schedulerStateLock`**

A synchronization lock for changing the schduler state.

executeLock

`private java.lang.Object` **`executeLock`**

A synchronization lock for executing a task.

(continued from last page)

registrationLock

`private java.lang.Object registrationLock`

A synchronization lock for registering to scheduler.

Constructors

PrintingManagement

`public PrintingManagement()`

Creates a new printing management. To initialize communications, the methods initializeCommunicationsServer and initializeCommunicationsClient should be called after this constructor returns. The call is done f.ex. by the class PrintingManagementWrapper.

Methods

getPendingJobs

`public java.util.List getPendingJobs()`

Getter for pendingJobs

Returns:

Then pendingJobs List

---

getSchedulerStopped

`public boolean getSchedulerStopped()`

Getter for the schedulerStopped

Returns:

A boolean telling wheter the scheduler is stopped or not

---

getUIConnections

`public java.util.Vector getUIConnections()`

Getter for the user interface connections

Returns:

The uIVector Vector object containing all the user interface connections

---

execute

`public java.lang.Object execute(java.lang.Object request_data)`

The executor for the printing jobs.

Parameters:

`request_data` -
The request data. This should be a Bundle in the Nippu system.

Returns:

Returns always null.

(continued from last page)

Exceptions:

`IllegalArgumentException` -
if the argument is invalid

---

register

```
public int register(PrintingManagementUserInterfaceProtocol connection,
                    int listenerType)
```

Registers a new UI to the Printing Management. The UI can be of the following types: 1 = a super user interface, 0 = other.

Parameters:

`connection` -
The connection to the UI
`listenerType` -
The type of the UI

Returns:

The ID for the UI

---

unregister

```
public void unregister(int listenerID)
```

Unregisters a component from the Printing Management

Parameters:

`listenerID` -
The ID of the component

---

changeSchedulerState

```
public void changeSchedulerState(int newState)
```

Changes the scheduler state, if possible.

Parameters:

`newState` -
the new scheduler state: 3 = pause, 4 = resume

---

schedulerStateChanged

```
public void schedulerStateChanged(int newState)
```

Called by communications when the scheduler has sent a state change notification.

Parameters:

`newState` -
the new scheduler state; 1 = started, 2 = stopped, 3 = paused, 4 = resumed

---

registrationAccepted

```
public void registrationAccepted()
```

Called by communications when the scheduler has accepted our registration.

---

initializeCommunicationsServer

```
public void initializeCommunicationsServer()
```

Sets up everything related to communications of the printing management as a server.

(continued from last page)

initializeCommunicationsClient

```
public void initializeCommunicationsClient()
```

Sets up everything related to communications of the printing management as a client.

Exceptions:

`java.lang.RuntimeException -`
if initialization failed

---

chooseOptimal

```
private java.lang.String chooseOptimal(Bundle bundle)
```

Chooses a free optimal printer (if any) from the optimalPrinters in the Bundle.

Parameters:

`bundle -`
The Bundle to choose the printer for.

Returns:

The printer ID or null if no printer was found.

---

chooseAny

```
private java.lang.String chooseAny(Bundle bundle)
```

Chooses a free printer (if any) for the Bundle.

Parameters:

`bundle -`
The Bundle to choose the printer for.

Returns:

The printer ID or null if no printer was found.

---

countDistance

```
private float countDistance(java.lang.String printerID,
                           Bundle bundle)
```

Counts the distance from a bundle to a printer.

Parameters:

`printerID -`
The printed identification number to count the distance to.
`bundle -`
The Bundle instance to count the distance from.

Returns:

The distance as a float.

---

main

```
public static void main(java.lang.String args)
```

A main method for testing purposes.

(continued from last page)

checkUserPrinter

```
private static boolean checkUserPrinter(java.lang.String id,
                                        int minLength,
                                        int maxLength,
                                        java.lang.String content)
```

nippu.printingmanagement

Class PrintingManagementWrapper

```
java.lang.Object
   |
   +-nippu.printingmanagement.PrintingManagementWrapper
```

All Implemented interfaces:
terttu.server.scheduler.Executor

---

public class **PrintingManagementWrapper**
extends java.lang.Object
implements terttu.server.scheduler.Executor

A wrapper for the actual PrintingManagement class. The wrapper was initially made in the belief that it was essential in order to initialize printing management. Later it was found out that this wasn't necessarily the case. The wrapper is still here because it works. (And you know; if it ain't broken, don't fix it!) This is the original explanation of the wrapper: The wrapper is needed for proper initialization of PrintingManagement. When Terttu creates an instance of its printing management, it only calls the constructor having no parameters. All initialization of the printing management must be done in the constructor. However, we need to initialize connections to other components using the communications classes. This requires storing printing management's reference (i.e. the value of 'this') into other objects. But 'this' isn't available in the constructor. We work around this problem with PrintingManagementWrapper. PrintingManagementWrapper creates the actual PrintingManagement, and then calls another initializing method of the PrintingManagement. There we can use 'this' and the communications initialization can be done. Basically PrintingManagementWrapper forwards method calls directly to PrintingManagement. PrintingManagementWrapper does not use the communications package.

Author:
Ville Nurmi

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

---

printingManagement

```
private nippu.printingmanagement.PrintingManagement printingManagement
```

The actual printing management instance.

Constructors

PrintingManagementWrapper

```
public PrintingManagementWrapper()
```

Creates a printing management that is able to establish a server.

Methods

(continued from last page)

execute

```
public java.lang.Object execute(java.lang.Object request_data)
```

Forwards an execution request given by Terttu to the actual printing management.

Parameters:

`request_data` -
the request data

nippu.printingmanagement

Class PrintingManagementWrapper.PrintingManagementInitializationTask

```
java.lang.Object
    |
    +-java.util.TimerTask
          |
          +-
nippu.printingmanagement.PrintingManagementWrapper.PrintingManagementInitializationTask
```

---

public class **PrintingManagementWrapper.PrintingManagementInitializationTask**
extends java.util.TimerTask

A helper class used in initializing printing management.

Constructors

PrintingManagementWrapper.PrintingManagementInitializationTask

```
public PrintingManagementWrapper.PrintingManagementInitializationTask()
```

Methods

run

```
public void run()
```

See Also:

java.util.TimerTask

nippu.printingmanagement

Class UIConnection

```
java.lang.Object
   |
   +-nippu.printingmanagement.UIConnection
```

public class **UIConnection**
extends java.lang.Object

A datastructure for the UI connections

Fields

iD

```
private int iD
```

The ID of the UI Connection

uIProtocol

```
private nippu.communications.PrintingManagementUserInterfaceProtocol uIProtocol
```

The protocol of the connection

Constructors

UIConnection

```
public UIConnection(int newID,
                    PrintingManagementUserInterfaceProtocol newUIProtocol)
```

The construtor

Methods

equals

```
public boolean equals(java.lang.Object test)
```

Checks if the given object is the same as this

getID

```
public int getID()
```

Returns the connection id

getProtocol

```
public PrintingManagementUserInterfaceProtocol getProtocol()
```

Returns the protocol object of the connection

Package
nippu.analyzer

nippu.analyzer

Class Analyzer

```
java.lang.Object
    |
    +- terttu.server.scheduler.Analyzer
        |
        +-nippu.analyzer.Analyzer
```

public class **Analyzer**
extends terttu.server.scheduler.Analyzer

Analyzer of bundles of print jobs. Provides five bundle services: get coordinates, cost, combined cost, fusion, and toString. This class extends the abstract class of the same name in Terttu. There are three sets of five methods providing the five bundle services. The first set is the method interface required by Terttu. The second set is for the communications component. Those methods take as a parameter a protocol instance that can be used to send answer messages. The third set is similar to the Terttu set but takes parameters in another form (natural Bundles and not byte[] objects). The third interface is mainly for internal use although it can be used by outsiders too. Analyzer can be configured. The configuration is received from printing management via communications. The configuration holds information about printers, users and analyzer's constants.

Author:
Ville Nurmi

Fields

anPrProtocol

`private nippu.communications.AnalyzerPrintingManagementProtocol` **`anPrProtocol`**

Our connection to printing management.

printingManagementLocation

`private java.lang.String` **`printingManagementLocation`**

The location of the printing management.

printingManagementPort

`private int` **`printingManagementPort`**

The printing management's port for clients.

communicationsLock

`private java.lang.Object` **`communicationsLock`**

A lock for variables that are tampered by communications threads. This is just *any* object. Fields to synchronize are listenerID, users, printers, listenerIDInitialized, configurationInitialized, sameUserMultiplier, maxNumberOfPages, bigBundleMultiplier, distanceLimit, minDistanceMultiplier, sameForcedPrinterMultiplier, coordinateStyle, fastPrinterDistance.

users

`private java.util.Vector` **`users`**

All users in the system. Got from the configuration. This vector contains NippuUsers.

(continued from last page)

printers

`private java.util.Vector printers`

All printers in the system. Got from the configuration. This vector contains NippuPrinters.

listenerIDInitialized

`private boolean listenerIDInitialized`

True iff listenerID has been initialized.

configurationInitialized

`private boolean configurationInitialized`

True iff configuration has been initialized.

listenerID

`private int listenerID`

Our ID given by printing management.

numberOfCreatedBundles

`private int numberOfCreatedBundles`

The number of bundles created by this analyzer instance.

sameUserMultiplier

`private float sameUserMultiplier`

Combined cost multiplier for bundles with the same submitters.

maxNumberOfPages

`private int maxNumberOfPages`

The page limit of a big bundle .

bigBundleMultiplier

`private float bigBundleMultiplier`

Combined cost multiplier for big bundles.

distanceLimit

`private float distanceLimit`

Maximum distance for bundles to not avoid fusion.

minDistanceMultiplier

`private float minDistanceMultiplier`

Combined cost multiplier for bundles with the same coordinates.

sameForcedPrinterMultiplier

`private float sameForcedPrinterMultiplier`

Combined cost multiplier for bundles with the same forced printer.

coordinateStyle

`private java.lang.String` **`coordinateStyle`**

Style of bundle coordinates; "submitter", "printer", or "submitter and printer".

fastPrinterDistance

`private float` **`fastPrinterDistance`**

The maximum such distance for a printer with speed 50 from a 50-page bundle that the printer is an optimal printer for the bundle.

logger

`protected static org.apache.log4j.Logger` **`logger`**

Constructors

Analyzer

`public` **`Analyzer`**`()`

Creates an analyzer. Communications initialization must be done with calls to Analyzer.initializeCommunicationsClient() and Analyzer.initializeCommunicationsServer()

Methods

initializeCommunicationsClient

`public void` **`initializeCommunicationsClient`**`()`

Sets up everything related to communications of the analyzer as a client. More specifically, establishes a connection to printing management. The connection gives us our listenerID which we need later for creating new Bundles etc.

Exceptions:

`java.lang.RuntimeException` -
if initialization failed

initializeCommunicationsServer

`public void` **`initializeCommunicationsServer`**`()`

Sets up everything related to communications of the analyzer as a server. More specifically, creates a server to which reception can connect to.

setListenerID

`public void` **`setListenerID`**`(int listenerID)`

Set listener ID. This method is to be called by the communications component only.

setConfiguration

`public void` **`setConfiguration`**`(java.lang.String configuration)`

Set configuration. This method is to be called by the communications component only.

(continued from last page)

getCoordinates

```
public float[] getCoordinates(java.lang.Object data)
                       throws terttu.server.scheduler.AnalyzerException
```

Returns the coordinates of the given bundle.

Parameters:

`data` -
the bundle; a serialized Bundle in a byte array

Returns:

the coordinates of the given bundle

Exceptions:

`terttu.server.scheduler.AnalyzerException` -
if data is not an instance of Bundle

---

cost

```
public float cost(java.lang.Object data)
           throws terttu.server.scheduler.AnalyzerException
```

Returns the cost of the given bundle. The cost is always non-negative.

Parameters:

`data` -
the bundle; a serialized Bundle in a byte array

Returns:

the cost of the given bundle

Exceptions:

`terttu.server.scheduler.AnalyzerException` -
if data is not an instance of Bundle

---

combinedCost

```
public float combinedCost(java.lang.Object data1,
                          java.lang.Object data2)
                   throws terttu.server.scheduler.AnalyzerException
```

Returns the cost of the union of the two given bundles. The cost is always non-negative.

Parameters:

`data1` -
the first bundle; a serialized Bundle in a byte array
`data2` -
the second bundle; a serialized Bundle in a byte array

Returns:

the cost of the union of the given bundles

Exceptions:

`terttu.server.scheduler.AnalyzerException` -
if data1 or data2 is not an instance of Bundle

(continued from last page)

fusion

```
public java.lang.Object fusion(java.lang.Object data1,
                               java.lang.Object data2)
                        throws terttu.server.scheduler.AnalyzerException
```

Returns the union of the two given bundles.

Parameters:

`data1` -
the first bundle; a serialized Bundle in a byte array
`data2` -
the second bundle; a serialized Bundle in a byte array

Returns:

the bundle formed by the two bundles fused together; a serialized Bundle in a byte array

Exceptions:

`terttu.server.scheduler.AnalyzerException` -
if data1 or data2 is not an instance of Bundle

---

toString

```
public java.lang.String toString(java.lang.Object data)
                          throws terttu.server.scheduler.AnalyzerException
```

Returns a human-readable string describing the given bundle.

Parameters:

`data` -
the bundle; a serialized Bundle in a byte array

Returns:

the string describing the bundle

Exceptions:

`terttu.server.scheduler.AnalyzerException` -
if data is not an instance of Bundle

---

getCoordinates

```
public void getCoordinates(AnalyzerReceptionProtocol protocol,
                           Bundle bundle)
```

See Also:

getCoordinates(Object)

---

cost

```
public void cost(AnalyzerReceptionProtocol protocol,
                 Bundle bundle)
```

See Also:

cost(Object)

(continued from last page)

combinedCost

```
public void combinedCost(AnalyzerReceptionProtocol protocol,
                         Bundle bundle1,
                         Bundle bundle2)
```

See Also:

combinedCost(Object, Object)

---

fusion

```
public void fusion(AnalyzerReceptionProtocol protocol,
                   Bundle bundle1,
                   Bundle bundle2)
```

See Also:

fusion(Object, Object)

---

toString

```
public void toString(AnalyzerReceptionProtocol protocol,
                     Bundle bundle)
```

See Also:

toString(Object)

---

getCoordinates

```
public float[] getCoordinates(Bundle bundle)
                       throws terttu.server.scheduler.AnalyzerException
```

See Also:

getCoordinates(Object)

---

cost

```
public float cost(Bundle bundle)
           throws terttu.server.scheduler.AnalyzerException
```

See Also:

cost(Object)

---

combinedCost

```
public float combinedCost(Bundle bundle1,
                          Bundle bundle2)
                   throws terttu.server.scheduler.AnalyzerException
```

(continued from last page)

See Also:

combinedCost(Object, Object)

fusion

```
public Bundle fusion(Bundle bundle1,
                     Bundle bundle2)
              throws terttu.server.scheduler.AnalyzerException
```

See Also:

fusion(Object, Object)

toString

```
public java.lang.String toString(Bundle bundle)
                          throws terttu.server.scheduler.AnalyzerException
```

See Also:

toString(Object)

distance

```
private float distance(float p1,
                      float p2)
```

Returns the Euclidean distance between the two given points in an N-dimensional space.

Exceptions:

`IllegalArgumentException` -
if the points are of different dimensionality or are null

getDistanceMultiplier

```
private float getDistanceMultiplier(float distance)
```

Returns the bundle combined cost multiplier for the distance between the bundles to fuse.

maxPrinterDistance

```
private float maxPrinterDistance(float numberOfPages,
                                 float printerSpeed)
```

Returns the maximum such distance between a printer with the given speed from a bundle with the given number of pages that the printer will become an optimal printer for the bundle. If the printer is further away, it will not become an optimal printer for the bundle.

Parameters:

`numberOfPages` -
the number of pages in the bundle
`printerSpeed` -
the speed of the printer

Returns:

the maximum distance for the printer to be optimal for the bundle

getSubmitterCoordinates

```
private float[] getSubmitterCoordinates(Bundle bundle)
```

Returns the average coordinates of the submitters of the bundle's print jobs.

getPrinterCoordinates

```
private float[] getPrinterCoordinates(Bundle bundle)
```

Returns the coordinates of the printer nearest to the average coordinates of the submitters of the bundle's print jobs.

getNearestPrinter

```
private NippuPrinter getNearestPrinter(int x,
                                       int y,
                                       int z)
```

Returns the printer nearest to the given point.

schedulingActionPerformed

```
public void schedulingActionPerformed(terttu.server.scheduler.event.SchedulingEvent e)
```

Called when there's a scheduling event in the scheduler.

Parameters:

`e` -
the scheduling event

schedulerStateChanged

```
public void
schedulerStateChanged(terttu.server.scheduler.event.SchedulerStateChangedEvent e)
```

Called when the scheduler's state has changed.

Parameters:

`e` -
the state changed event

testCommunicationsClient

```
public void testCommunicationsClient()
```

testGetCoordinatesInner

```
public void testGetCoordinatesInner()
                             throws terttu.server.scheduler.AnalyzerException
```

testGetCoordinates

```
public void testGetCoordinates()
```

testCombinedCost

```
public void testCombinedCost()
```

(continued from last page)

testFusion

```
public void testFusion()
```

toString

```
public static java.lang.String toString(float coords)
```

Returns a human-readable string representation of the given coordinates.

main

```
public static void main(java.lang.String args)
```

nippu.analyzer

Class AnalyzerWrapper

```
java.lang.Object
    |
    +- terttu.server.scheduler.Analyzer
        |
        +-nippu.analyzer.AnalyzerWrapper
```

public class **AnalyzerWrapper**
extends terttu.server.scheduler.Analyzer

A wrapper for the actual Analyzer class. The wrapper was initially made in the belief that it was essential in order to initialize the analyzer. Later it was found out that this wasn't necessarily the case. The wrapper is still here because it works. (And you know; if it ain't broken, don't fix it!) This is the original explanation of the wrapper: The wrapper is needed for proper initialization of Analyzer. When Terttu creates an instance of its analyzer, it only calls the constructor having no parameters. All initialization of the analyzer must be done in the constructor. However, we need to initialize connections to other components using the communications classes. This requires storing analyzer's reference (i.e. the value of 'this') into other objects. But 'this' isn't available in the constructor. We work around this problem with AnalyzerWrapper. AnalyzerWrapper creates the actual Analyzer, and then calls another initializing method of the Analyzer. There we can use 'this' and the communications initialization can be done. Basically AnalyzerWrapper forwards method calls directly to Analyzer. AnalyzerWrapper does not use the communications package.

Author:
Ville Nurmi

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

analyzer

```
private nippu.analyzer.Analyzer analyzer
```

The actual analyzer instance.

Constructors

AnalyzerWrapper

```
public AnalyzerWrapper()
```

Creates an analyzer that is able to connect to a printing management.

Methods

getCoordinates

```
public float[] getCoordinates(java.lang.Object data)
                       throws terttu.server.scheduler.AnalyzerException
```

Returns the coordinates of the given bundle.

(continued from last page)

Parameters:

`data` -
the bundle; an instance of class Bundle

Returns:

the coordinates of the given bundle

Exceptions:

`terttu.server.scheduler.AnalyzerException` -
if data is not an instance of Bundle

---

cost

```
public float cost(java.lang.Object data)
           throws terttu.server.scheduler.AnalyzerException
```

Returns the cost of the given bundle. The cost is always non-negative.

Parameters:

`data` -
the bundle; an instance of class Bundle

Returns:

the cost of the given bundle

Exceptions:

`terttu.server.scheduler.AnalyzerException` -
if data is not an instance of Bundle

---

combinedCost

```
public float combinedCost(java.lang.Object data1,
                          java.lang.Object data2)
                   throws terttu.server.scheduler.AnalyzerException
```

Returns the cost of the union of the two given bundles. The cost is always non-negative.

Parameters:

`data1` -
the first bundle; an instance of class Bundle
`data2` -
the second bundle; an instance of class Bundle

Returns:

the cost of the union of the given bundles

Exceptions:

`terttu.server.scheduler.AnalyzerException` -
if data1 or data2 is not an instance of Bundle

---

fusion

```
public java.lang.Object fusion(java.lang.Object data1,
                               java.lang.Object data2)
                        throws terttu.server.scheduler.AnalyzerException
```

Returns the union of the two given bundles.

Parameters:

`data1` -
the first bundle; an instance of class Bundle

(continued from last page)

`data2 -`
the second bundle; an instance of class Bundle

Returns:

the bundle formed by the two bundles fused together

Exceptions:

`terttu.server.scheduler.AnalyzerException -`
if data1 or data2 is not an instance of Bundle

---

toString

```
public java.lang.String toString(java.lang.Object data)
                          throws terttu.server.scheduler.AnalyzerException
```

Returns a human-readable string describing the given bundle.

Parameters:

`data -`
the bundle; an instance of class Bundle

Returns:

the string describing the bundle

Exceptions:

`terttu.server.scheduler.AnalyzerException -`
if data is not an instance of Bundle

---

schedulingActionPerformed

```
public void schedulingActionPerformed(terttu.server.scheduler.event.SchedulingEvent e)
```

Called when there's a scheduling event in the scheduler.

Parameters:

`e -`
the scheduling event

---

schedulerStateChanged

```
public void
schedulerStateChanged(terttu.server.scheduler.event.SchedulerStateChangedEvent e)
```

Called when the scheduler's state has changed.

Parameters:

`e -`
the state changed event

nippu.analyzer

Class AnalyzerWrapper.AnalyzerInitializationTask

```
java.lang.Object
    |
    +-java.util.TimerTask
        |
        +-nippu.analyzer.AnalyzerWrapper.AnalyzerInitializationTask
```

---

public class **AnalyzerWrapper.AnalyzerInitializationTask**
extends java.util.TimerTask

A helper class used in initializing the analyzer.

Fields

communicationsClientInitialized

`private boolean` **`communicationsClientInitialized`**

True iff communications client initialization has been done.

---

communicationsServerInitialized

`private boolean` **`communicationsServerInitialized`**

True iff communications server initialization has been done.

Constructors

AnalyzerWrapper.AnalyzerInitializationTask

`public` **`AnalyzerWrapper.AnalyzerInitializationTask`**`()`

Methods

run

`public void` **`run`**`()`

See Also:

java.util.TimerTask

Package
nippu.communications

nippu.communications

Class AnalyzerPrintingManagementProtocol

```
java.lang.Object
    ┗-terttu.server.network.Protocol
        ┗-terttu.server.network.TerttuProtocol
            ┗-nippu.communications.NippuProtocol
                ┗-nippu.communications.AnalyzerPrintingManagementProtocol
```

public class **AnalyzerPrintingManagementProtocol**
extends `NippuProtocol`

A protocol class for analyzer towards printing management. A protocol class handles the conversion of communications messages of one connection type on the end of the protocol's component. The conversion is made in the send methods from method calls to byte stream, and in the handleMessage method from byte stream to method calls.

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

services

```
private nippu.communications.AnalyzerPrintingManagementServices services
```

The instance who knows what to do with incoming messages.

Constructors

AnalyzerPrintingManagementProtocol

```
public AnalyzerPrintingManagementProtocol()
```

Methods

setServices

```
public void setServices(AnalyzerPrintingManagementServices services)
```

handleMessage

```
protected void handleMessage(byte header,
                             int dataLength)
                      throws java.io.IOException
```

(continued from last page)

See Also:

NippuProtocol

sendListenerRegistration

```
public void sendListenerRegistration()
                              throws java.io.IOException
```

Sends a request to be registered as a listener.

sendListenerRegistration

```
public void sendListenerRegistration(int optionalData)
                              throws java.io.IOException
```

Sends a request to be registered as a listener.

Parameters:

`optionalData` -
optional data, meaning undefined

sendListenerUnregistration

```
public void sendListenerUnregistration(int listenerID)
                                throws java.io.IOException
```

Sends a request to be unregistered from being a listener.

Parameters:

`listenerID` -
the ID of the listener

nippu.communications

Class AnalyzerPrintingManagementServices

```
java.lang.Object
   |
   +-nippu.communications.AnalyzerPrintingManagementServices
```

public class **AnalyzerPrintingManagementServices**
extends java.lang.Object

A services class for connections of analyzer towards printing management. A services class forwards messages interpreted by a protocol class to the classes who are concerned. Thus, the services class can be seen as a higher level of communications handling whereas the protocol class is a lower level. A services class consists mainly of service methods. They are called by the protocol class only.

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

protocol

```
private nippu.communications.AnalyzerPrintingManagementProtocol protocol
```

The instance who can send actual messages to the other end of our connection.

analyzer

```
private nippu.analyzer.Analyzer analyzer
```

The analyzer whose communications we are handling.

Constructors

AnalyzerPrintingManagementServices

```
public AnalyzerPrintingManagementServices()
```

Methods

setProtocol

```
public void setProtocol(AnalyzerPrintingManagementProtocol protocol)
```

setAnalyzer

```
public void setAnalyzer(Analyzer analyzer)
```

Sets the analyzer instance to forward messages to.

registrationAccepted

```
void registrationAccepted(int listenerID)
```

unregistrationAccepted

```
void unregistrationAccepted(int returnValue)
```

printJobQueued

```
void printJobQueued(PrintJob printJob,
                    java.lang.String printer)
```

printJobPrinted

```
void printJobPrinted(PrintJob printJob)
```

schedulerStateChanged

```
void schedulerStateChanged(int newState)
```

configuration

```
void configuration(java.lang.String configuration)
```

nippu.communications

Class AnalyzerReceptionProtocol

```
java.lang.Object
    |-terttu.server.network.Protocol
        |-terttu.server.network.TerttuProtocol
            |-nippu.communications.NippuProtocol
                |-nippu.communications.AnalyzerReceptionProtocol
```

public class **AnalyzerReceptionProtocol**
extends NippuProtocol

A protocol class for connections of analyzer towards reception. A protocol class handles the conversion of communications messages of one connection type on the end of the protocol's component. The conversion is made in the send methods from method calls to byte stream, and in the handleMessage method from byte stream to method calls.

Author:
Ville Nurmi

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

services

```
private nippu.communications.AnalyzerReceptionServices services
```

The instance who knows what to do with incoming messages.

Constructors

AnalyzerReceptionProtocol

```
public AnalyzerReceptionProtocol()
```

Methods

setServices

```
public void setServices(AnalyzerReceptionServices services)
```

(continued from last page)

handleMessage

```
protected void handleMessage(byte header,
                             int dataLength)
                      throws java.io.IOException
```

See Also:

NippuProtocol

---

sendGetCoordinatesAnswer

```
public void sendGetCoordinatesAnswer(int bundleID,
                                     float coordinates)
                              throws java.io.IOException
```

---

sendCostAnswer

```
public void sendCostAnswer(int bundleID,
                           float cost)
                    throws java.io.IOException
```

---

sendCombinedCostAnswer

```
public void sendCombinedCostAnswer(int bundle1ID,
                                   int bundle2ID,
                                   float cost)
                            throws java.io.IOException
```

---

sendFusionAnswer

```
public void sendFusionAnswer(int bundle1ID,
                             int bundle2ID,
                             Bundle fusion)
                      throws java.io.IOException
```

---

sendToStringAnswer

```
public void sendToStringAnswer(int bundleID,
                               java.lang.String bundleString)
                        throws java.io.IOException
```

nippu.communications

Class AnalyzerReceptionServices

```
java.lang.Object
   |
   +-nippu.communications.AnalyzerReceptionServices
```

public class **AnalyzerReceptionServices**
extends java.lang.Object

A services class for connections of analyzer towards reception. A services class forwards messages interpreted by a protocol class to the classes who are concerned. Thus, the services class can be seen as a higher level of communications handling whereas the protocol class is a lower level. A services class consists mainly of service methods. They are called by the protocol class only.

Author:
Ville Nurmi

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

protocol

```
private nippu.communications.AnalyzerReceptionProtocol protocol
```

The instance who can send actual messages to the other end of our connection.

analyzer

```
private nippu.analyzer.Analyzer analyzer
```

The instance to forward messages to.

Constructors

AnalyzerReceptionServices

```
public AnalyzerReceptionServices(Analyzer analyzer)
```

Creates a services instance.

Parameters:
`analyzer` -
the instance to forward messages to.

Methods

setProtocol

```
public void setProtocol(AnalyzerReceptionProtocol protocol)
```

setAnalyzer

```
public void setAnalyzer(Analyzer analyzer)
```

Sets the instance to forward messages to.

getCoordinates

```
void getCoordinates(Bundle bundle)
```

cost

```
void cost(Bundle bundle)
```

combinedCost

```
void combinedCost(Bundle bundle1,
                  Bundle bundle2)
```

fusion

```
void fusion(Bundle bundle1,
            Bundle bundle2)
```

toString

```
void toString(Bundle bundle)
```

nippu.communications

Class Communications

```
java.lang.Object
   |
   +-java.lang.Thread
        |
        +-nippu.communications.Communications
```

public class **Communications**
extends java.lang.Thread

The main class of communications in Nippu. There is one instance of Communications for each communication type. Class Communications holds a server socket and creates new sockets for accepted connection requests. A new socket is given a thread of its own. The specific behavious of the connection type is specified by the given references to a protocol class and a services class. Thus class Communications can be used for all kinds of connection types. Class Communications is based on the similar class terttu.server.network.TerttuServerTCP in Terttu. Terminology: connection type = the type of connections that will be established between a certain pair of Nippu components, includes f.ex. the protocol that the components use with each other connection = a communications connection between two instances of Nippu components

Author:
Ville Nurmi

Fields

OPTNAME_PORTLISTENER_TIMEOUT

`public static final java.lang.String` **`OPTNAME_PORTLISTENER_TIMEOUT`**

Parametrin nimi.

DEFAULT_PORTLISTENER_TIMEOUT

`private static final int` **`DEFAULT_PORTLISTENER_TIMEOUT`**

Parametrin oletusarvo.

OPTNAME_SERVICETHREAD_TIMEOUT

`public static final java.lang.String` **`OPTNAME_SERVICETHREAD_TIMEOUT`**

Parametrin nimi.

DEFAULT_SERVICETHREAD_TIMEOUT

`private static final int` **`DEFAULT_SERVICETHREAD_TIMEOUT`**

Parametrin oletusarvo.

OPTNAME_MAX_MESSAGE_SIZE

`public static final java.lang.String` **`OPTNAME_MAX_MESSAGE_SIZE`**

Parametrin nimi.

DEFAULT_MAX_MESSAGE_SIZE

`private static final int` **`DEFAULT_MAX_MESSAGE_SIZE`**

(continued from last page)

Parametrin oletusarvo.

OPTNAME_MESSAGE_TIMEOUT

```
public static final java.lang.String OPTNAME_MESSAGE_TIMEOUT
```

Parametrin nimi.

DEFAULT_MESSAGE_TIMEOUT

```
private static final int DEFAULT_MESSAGE_TIMEOUT
```

Parametrin oletusarvo.

OPTNAME_MAX_CLIENTS

```
public static final java.lang.String OPTNAME_MAX_CLIENTS
```

Parametrin nimi.

DEFAULT_MAX_CLIENTS

```
private static final int DEFAULT_MAX_CLIENTS
```

Parametrin oletusarvo.

OPTNAME_CLIENT_PORT

```
public static final java.lang.String OPTNAME_CLIENT_PORT
```

Parametrin nimi.

logger

```
protected static org.apache.log4j.Logger logger
```

optPortListenerTimeOut

```
private int optPortListenerTimeOut
```

Parametrin arvo.

optServiceThreadTimeOut

```
private int optServiceThreadTimeOut
```

Parametrin arvo.

optMaxMessageSize

```
private int optMaxMessageSize
```

Parametrin arvo.

optMessageTimeOut

```
private int optMessageTimeOut
```

Parametrin arvo.

(continued from last page)

optMaxClients

```
private int optMaxClients
```

Parametrin arvo.

optClientPort

```
private int optClientPort
```

Parametrin arvo.

portListener

```
private nippu.communications.Communications.PortListener portListener
```

A listener of incoming connections.

serverRunning

```
private boolean serverRunning
```

True iff incoming connections are accepted.

Constructors

Communications

```
public Communications(java.util.Properties configuration,
                      java.lang.reflect.Constructor protocolConstructor,
                      java.lang.Object protocolConstructorArgs,
                      java.lang.reflect.Method protocolSetServicesMethod,
                      java.lang.reflect.Constructor servicesConstructor,
                      java.lang.Object servicesConstructorArgs,
                      java.lang.reflect.Method servicesSetProtocolMethod)
```

Creates a new instance of Communications using the given settings and information about the protocol and services classes that implement the functionality specific to this connection type.

Parameters:

`configuration` -
the settings to use
`protocolConstructor` -
the constructor of the correct protocol
`protocolConstructorArgs` -
the arguments for the protocol constructor
`protocolSetServicesMethod` -
the method of the protocol class to make an instance use a certain services instance. The method must take exactly one argument which is the services instance.
`servicesConstructor` -
the constructor of the correct services
`servicesConstructorArgs` -
the arguments for the services constructor
`servicesSetProtocolMethod` -
the method of the services class to make an instance use a certain protocol instance. The method must take exactly one argument which is the protocol instance.

Methods

parseConfiguration

```
private void parseConfiguration(java.util.Properties config)
                         throws terttu.server.network.ParameterException
```

Lukee asetukset.

parseIntegerProperty

```
private int parseIntegerProperty(java.util.Properties config,
                                 java.lang.String key,
                                 int defaultValue,
                                 int minValue,
                                 int maxValue)
```

Käsittelee kokonaislukuparametrin.

parseIntegerProperty

```
private int parseIntegerProperty(java.util.Properties config,
                                 java.lang.String key,
                                 int minValue,
                                 int maxValue)
                          throws terttu.server.network.ParameterException
```

Käsittelee välttämättömän kokonaislukuparametrin. Heittää poikkeuksen, jos (laillista) arvoa ei ole määritelty.

startService

```
public void startService()
                  throws java.io.IOException
```

Starts listening to the server socket.

stopService

```
public void stopService()
```

Stops the server socket listening for good.

nippu.communications

Class Communications.PortListener

```
java.lang.Object
    |
    +-java.lang.Thread
        |
        +-nippu.communications.Communications.PortListener
```

private class **Communications.PortListener**
extends java.lang.Thread

Takes care of incoming connections. Contains a server socket for a specified port. Starts a new instance of CommunicationsThread for every new connection.

Fields

port

`private int` **`port`**

protocolConstructor

`private java.lang.reflect.Constructor` **`protocolConstructor`**

The correct protocol's correct constructor to use.

protocolConstructorArgs

`private java.lang.Object` **`protocolConstructorArgs`**

The arguments for the protocol constructor.

protocolSetServicesMethod

`private java.lang.reflect.Method` **`protocolSetServicesMethod`**

The method of the given protocol class to make it use an instance of given type of services.

servicesConstructor

`private java.lang.reflect.Constructor` **`servicesConstructor`**

The correct services's correct constructor to use.

servicesConstructorArgs

`private java.lang.Object` **`servicesConstructorArgs`**

The arguments for the services constructor.

servicesSetProtocolMethod

`private java.lang.reflect.Method` **`servicesSetProtocolMethod`**

The method of the given services class to make it use an instance of given type of protocol.

listenerRunning

```
private boolean listenerRunning
```

When this changes from true to false the first time, the port listener stops listening.

childThreads

```
private java.lang.ThreadGroup childThreads
```

maxChildren

```
private int maxChildren
```

serverSocket

```
private java.net.ServerSocket serverSocket
```

The socket to which incoming connections come.

numberOfCreatedThreads

```
private int numberOfCreatedThreads
```

The number of created connection threads.

Constructors

Communications.PortListener

```
public Communications.PortListener(int port,
                                   int maxChildren,
                                   java.lang.reflect.Constructor protocolConstructor,
                                   java.lang.Object protocolConstructorArgs,
                                   java.lang.reflect.Method protocolSetServicesMethod,
                                   java.lang.reflect.Constructor servicesConstructor,
                                   java.lang.Object servicesConstructorArgs,
                                   java.lang.reflect.Method servicesSetProtocolMethod)
```

Creates a new server socket handler that listens to a given port for connection requests.

Parameters:

`port` -
the port to listen to
`maxChildren` -
the maximum number of connections to create
`protocolConstructor` -
the constructor of the correct protocol
`protocolConstructorArgs` -
the arguments for the protocol constructor
`protocolSetServicesMethod` -
the method of the protocol class to make an instance use a certain services instance. The method must take exactly one argument which is the services instance.
`servicesConstructor` -
the constructor of the correct services
`servicesConstructorArgs` -
the arguments for the services constructor
`servicesSetProtocolMethod` -
the method of the services class to make an instance use a certain protocol instance. The method must take exactly one argument which is the protocol instance.

Methods

getPort

```
public int getPort()
```

stopListener

```
public void stopListener()
```

Stops the listener for good.

startListener

```
public void startListener()
                    throws java.io.IOException
```

Starts the listener.

run

```
public void run()
```

See Also:

java.lang.Thread

nippu.communications

Class CommunicationsThread

```
java.lang.Object
    |
    +-java.lang.Thread
        |
        +-nippu.communications.CommunicationsThread
```

All Implemented interfaces:

terttu.server.network.ProtocolOutput, java.lang.Runnable

---

public class **CommunicationsThread**
extends java.lang.Thread
implements java.lang.Runnable, terttu.server.network.ProtocolOutput

A thread taking care of one connection. There is one instance of CommunicationsThread for each end of each connection. Class CommunicationsThread listens to incoming data forwarding it to a given instance of class Protocol. CommunicationsThread is also able to send data to the other end of its connection. The type of the thread can be determined, so class CommunicationsThread can be used for all kinds of connections. Class CommunicationsThread is based on the similar class terttu.server.network.TerttuServerThread in Terttu. Terminology: connection = a communications connection between two instances of Nippu components

Author:

Ville Nurmi

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

---

socket

```
protected java.net.Socket socket
```

Pistokeyhteys

---

protocol

```
protected terttu.server.network.Protocol protocol
```

Käytettävä protokolla

---

inStream

```
protected java.io.InputStream inStream
```

Pistokeyhteyden syötevirta.

---

outStream

```
protected java.io.OutputStream outStream
```

Pistokeyhteyden tulostevirta.

threadInitNumber

```
private static int threadInitNumber
```

Säikeen numero.

isOpen

```
private boolean isOpen
```

Yhteyden tila.

Constructors

CommunicationsThread

```
public CommunicationsThread(java.lang.String host,
                            int port,
                            int timeOut,
                            terttu.server.network.Protocol protocol)
```

Creates a new communications thread by requesting a connection to the given host through the given port. The thread must then be started with the method start(). This constructor is for clients who want to connect to a server.

Parameters:

`host` -
the host to connect to, f.ex. "localhost"
`port` -
the port to use
`timeOut` -
socket's timeout in blocking read
`protocol` -
the protocol to use in the connection

See Also:

java.lang.Thread.start()

CommunicationsThread

```
public CommunicationsThread(java.net.Socket socket,
                            int timeOut,
                            terttu.server.network.Protocol protocol,
                            java.lang.ThreadGroup threadGroup)
```

Creates a new communications thread. It must be started with the method start(). This constructor is called by servers who have received a connection request from a client and are now establishing the connection. @see java.lang.Thread.start()

Methods

nextThreadNum

```
private static int nextThreadNum()
```

Palauttaa seuraavan säikeen numeron.

close

```
public void close()
```

Sulkee yhteyden ja lopettaa säikeen toiminnan.

isOpen

```
private boolean isOpen()
```

Kertoo, onko säikeen yhteys päällä.

write

```
public void write(byte data,
                  int offset,
                  int length)
           throws java.io.IOException
```

ProtocolOutput rajapinnan toteuttava metodi

getProtocol

```
public terttu.server.network.Protocol getProtocol()
```

Returns the protocol used by this thread.

getSocket

```
public java.net.Socket getSocket()
```

Returns the socket listened by this thread

run

```
public void run()
```

Säikeen ajometodi.

nippu.communications

Class NippuProtocol

```
java.lang.Object
    |
    +- terttu.server.network.Protocol
        |
        +- terttu.server.network.TerttuProtocol
            |
            +- nippu.communications.NippuProtocol
```

Direct Known Subclasses:
AnalyzerPrintingManagementProtocol, AnalyzerReceptionProtocol, PrintingManagementSchedulerProtocol, PrintingManagementUserInterfaceProtocol, ReceptionAnalyzerProtocol, ReceptionSchedulerProtocol, ReceptionUserInterfaceProtocol, UserInterfacePrintingManagementProtocol, UserInterfaceReceptionProtocol

---

public abstract class **NippuProtocol**
extends terttu.server.network.TerttuProtocol

A superclass for protocol classes of Nippu communications. Provides an easy-to-use, fool-proof method interface for sending messages. A message is begun with a call to beginMessage(byte). There you can specify the message header. Then you call any number of the addXxx methods, feeding them parameters you want to store inside the message, in order. Finally, the message can be finished and sent with a call to endMessage(). Other things required by the Terttu communications protocol is done automatically. This includes writing down data length and padding the message. NippuProtocol also provides a fool-proof method interface for parsing an incoming message. The method interface consists of the abstract method handleMessage(byte, int) which is called on an incoming message. In the implementation of that method, the actual message data can be read with calls to the various readXxx methods. The read methods remember the position of the message where the previous call stopped parsing. Thus, the subclass needs not handle message data indices explicitly. NippuProtocol is *the*place where the codings used for primitive types in communications messages are decided.

Author:
Ville Nurmi

Fields

logger

```
private static org.apache.log4j.Logger logger
```

---

sendLock

```
private java.lang.Object sendLock
```

Synchronization lock for sending.

---

receiveLock

```
private java.lang.Object receiveLock
```

Synchronization lock for receiving.

---

charsetName

```
private java.lang.String charsetName
```

(continued from last page)

The name of the charset to use when coding and decoding strings.

INITIALBUFFERSIZE

`private int INITIALBUFFERSIZE`

Initial size of the message buffer.

LARGEBUFFERSIZE

`private int LARGEBUFFERSIZE`

The least such size of a buffer that is considered large. This affects memory saving mechanisms.

message

`private byte message`

The data of the message in construction, including the 8 bytes of the header, reserved and datalen fields.

dataLength

`private int dataLength`

The number of data bytes already in the message. This excludes the first 8 bytes of the message.

messageInConstruction

`private boolean messageInConstruction`

True iff a message is currently in construction.

parseMessage

`private byte parseMessage`

The message in parsing, including its header, reserved and datalen fields, and possibly some additional trailing bytes.

parseMessageLength

`private int parseMessageLength`

The number of bytes in the message in parsing. This includes the header, reserved and datalen fields.

nextToParse

`private int nextToParse`

The index of the next unread data byte in the message in parsing.

Constructors

NippuProtocol

`public NippuProtocol()`

Creates a new protocol instance. By default, the maximum length (in bytes) of a received message is 10 kB.

See Also:

NippuProtocol(int maxMessageLength)

(continued from last page)

NippuProtocol

```
public NippuProtocol(int maxMessageLength)
```

Creates a new protocol instance. The maximum length (in bytes) of a received message can be set with the parameter.

Parameters:

`maxMessageLength` -
the maximum message length in bytes

See Also:

NippuProtocol()

Methods

sendErrorMessage

```
public void sendErrorMessage()
                    throws java.io.IOException
```

Sends a message signaling an error. This method will call beginMessage and endMessage.

See Also:

sendErrorMessage(int)

---

sendErrorMessage

```
public void sendErrorMessage(int errorType)
                    throws java.io.IOException
```

Sends a message signaling an error. This method will call beginMessage and endMessage.

Parameters:

`errorType` -
the type of the error

See Also:

sendErrorMessage()

---

beginMessage

```
void beginMessage(byte header)
```

Begins constructing a new message. The message is sent by a call to endMessage(). Another call to this method is not allowed until endMessage() has been called.

Parameters:

`header` -
the message header

Exceptions:

`IllegalStateException` -
if a message is already in construction

See Also:

endMessage()

(continued from last page)

addInt

`void` **`addInt`**`(int data)`

Adds an int to the end of the message in construction. The method beginMessage must have been called earlier to start the construction of a message.

Parameters:

`data` -
the data to add

Exceptions:

`IllegalStateException` -
if no message is in construction

---

addFloat

`void` **`addFloat`**`(float data)`

Adds a float to the end of the message in construction. The method beginMessage must have been called earlier to start the construction of a message.

Parameters:

`data` -
the data to add

Exceptions:

`IllegalStateException` -
if no message is in construction

---

addString

`void` **`addString`**`(java.lang.String data)`

Adds a string to the end of the message in construction. The string is coded into bytes using the ISO-8859-15 charset (or ISO-8859-1 if the other is not available). The actual bytes are preceded by an int field describing the number of bytes in the coding (which may or may not equal the length of the string!). The method beginMessage must have been called earlier to start the construction of a message.

Parameters:

`data` -
the data to add

Exceptions:

`IllegalStateException` -
if no message is in construction

---

addBytes

`void` **`addBytes`**`(byte data)`

Adds a byte array to the end of the message in construction. The method beginMessage must have been called earlier to start the construction of a message.

Parameters:

`data` -
the data to add

Exceptions:

`IllegalStateException` -
if no message is in construction

See Also:

addBytes(byte[], int offset, int length)

addBytes

```
void addBytes(byte data,
              int offset,
              int length)
```

Adds the specified interval of the given byte array to the end of the message in construction. The method beginMessage must have been called earlier to start the construction of a message.

Parameters:

`data` -
the data to add
`offset` -
the index of the first data byte to add
`length` -
the number of consequent data bytes to add

Exceptions:

`IllegalStateException` -
if no message is in construction
`IllegalArgumentException` -
if data offset or length are out of bounds, or data == null

See Also:

addBytes(byte[])

endMessage

```
void endMessage()
         throws java.io.IOException
```

Ends and sends a message previously began with a call to beginMessage. Data length and message padding are done automatically.

Exceptions:

`IllegalStateException` -
if no message is in construction
`java.io.IOException` -
if the message couldn't be sent

See Also:

beginMessage(byte)

readInt

```
int readInt()
     throws java.io.IOException
```

Reads an int from the message in parsing.

Exceptions:

`java.io.IOException` -
if there is not enough data left for the data

readFloat

```
float readFloat()
         throws java.io.IOException
```

Reads a float from the message in parsing.

Exceptions:

(continued from last page)

`java.io.IOException` -
if there is not enough data left for the data

readString

```
java.lang.String readString()
                 throws java.io.IOException
```

Reads a string from the message in parsing.

Exceptions:

`java.io.IOException` -
if there is not enough data left for the data

readBytes

```
byte[] readBytes(int numberOfBytes)
       throws java.io.IOException
```

Reads a byte array from the message in parsing.

Parameters:

`numberOfBytes` -
the number of bytes to read

Exceptions:

`java.io.IOException` -
if there is not enough data left for the data

bytesToString

```
public static java.lang.String bytesToString(byte data)
                                       throws java.lang.IllegalArgumentException
```

Interprets and returns the string contained in the given byte array. The byte array is assumed to have been created along the communications conventions (f.ex. by NippuProtocol.stringToBytes or NippuProtocol.addString).

Exceptions:

`IllegalArgumentException` -
if the byte array doesn't contain a proper string

stringToBytes

```
public static byte[] stringToBytes(java.lang.String data)
                           throws java.lang.IllegalArgumentException
```

Converts and returns the string into a byte array using the correct charset and communications conventions.

Exceptions:

`IllegalArgumentException` -
if the conversion fails.

getCharsetName

```
private static java.lang.String getCharsetName()
```

Returns the name of the default charset to use.

getNextFreeByte

```
private int getNextFreeByte()
```

Returns the index in the message buffer that contains the next free byte to write onto.

(continued from last page)

ensureSpace

```
private void ensureSpace(int numberOfBytes)
```

Makes sure that the data buffer can store the given number of bytes.

---

ensureParseSpace

```
private void ensureParseSpace(int numberOfBytes)
                       throws java.io.IOException
```

Makes sure that the message in parsing contains at least the given number of unread bytes.

Exceptions:

`java.io.IOException` -
if there are not enough unread bytes

---

debugDataPrint

```
private void debugDataPrint(byte data,
                            int offset,
                            int length)
```

Prints the given data to the logger. This method is made for debugging the message buffer.

Parameters:

`data` -
the data to print
`offset` -
the first index in data to print
`length` -
the number of consequent bytes to print

---

handleMessage

```
protected abstract void handleMessage(byte header,
                                      int dataLength)
                               throws java.io.IOException
```

Called when an incoming message has been received. The message's header and number of data bytes are given as parameters. The message data itself can be accessed with calls to various readXxx methods. If there is not enough data for a readXxx call, java.io.IOException is thrown. A subclass can decide whether to implement NippuProtocol.handleMessage(byte, int) or the lower level TerttuProtocol.handleMessage(int, byte[], int). The lower level method gives direct access to the data bytes. If only the TerttuProtocol's handleMessage is implemented, then the NippuProtocol's handleMessage may be implemented as a method doing nothing.

Parameters:

`header` -
the header of the message
`dataLength` -
the number of data bytes in the message

Exceptions:

`java.io.IOException` -
if there is an error while parsing data, or if there is not enough data

See Also:

TerttuProtocol.handleMessage(int, byte[], int)

(continued from last page)

handleMessage

```
protected void handleMessage(int header,
                             byte data,
                             int length)
                      throws java.io.IOException
```

Called by our superclass when a new message is received. Message parsing is then delegated to handleMessage(byte, int).

See Also:

terttu.server.network.TerttuProtocol

handleMessage(byte, int)

nippu.communications

Class PrintingManagementSchedulerProtocol

```
java.lang.Object
    |-terttu.server.network.Protocol
        |-terttu.server.network.TerttuProtocol
            |-nippu.communications.NippuProtocol
                |-nippu.communications.PrintingManagementSchedulerProtocol
```

public class **PrintingManagementSchedulerProtocol**
extends `NippuProtocol`

A protocol class for connections of printing management towards scheduler. A protocol class handles the conversion of communications messages of one connection type on the end of the protocol's component. The conversion is made in the send methods from method calls to byte stream, and in the handleMessage method from byte stream to method calls.

Author:
Ville Nurmi

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

services

```
private nippu.communications.PrintingManagementSchedulerServices services
```

The instance who knows what to do with incoming messages.

Constructors

PrintingManagementSchedulerProtocol

```
public PrintingManagementSchedulerProtocol()
```

Methods

setServices

```
public void setServices(PrintingManagementSchedulerServices services)
```

(continued from last page)

handleMessage

```
protected void handleMessage(byte header,
                             int dataLength)
                      throws java.io.IOException
```

See Also:

NippuProtocol

---

sendSchedulerStateChangeQuery

```
public void sendSchedulerStateChangeQuery(int newState)
                                   throws java.io.IOException
```

Sends a request to change the state of the scheduler, 3 = pause, 4 = resume.

Parameters:

`newState` -
the requested new state

---

sendListenerRegistration

```
public void sendListenerRegistration(int listenerType)
                              throws java.io.IOException
```

Sends a request to be registered as a listener. Listener type has these possible values, and they can be : 1 = listening listener, bit 2 = commanding listener.

Parameters:

`listenerType` -
type of the listener

nippu.communications

Class PrintingManagementSchedulerServices

```
java.lang.Object
   |
   +-nippu.communications.PrintingManagementSchedulerServices
```

public class **PrintingManagementSchedulerServices**
extends java.lang.Object

A services class for connections of printing management towards scheduler. A services class forwards messages interpreted by a protocol class to the classes who are concerned. Thus, the services class can be seen as a higher level of communications handling whereas the protocol class is a lower level. A services class consists mainly of service methods. They are called by the protocol class only.

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

protocol

```
private nippu.communications.PrintingManagementSchedulerProtocol protocol
```

The instance who can send actual messages to the other end of our connection.

printingManagement

```
private nippu.printingmanagement.PrintingManagement printingManagement
```

The printing management to forward messages to.

Constructors

PrintingManagementSchedulerServices

```
public PrintingManagementSchedulerServices()
```

Methods

setProtocol

```
public void setProtocol(PrintingManagementSchedulerProtocol protocol)
```

setPrintingManagement

```
public void setPrintingManagement(PrintingManagement printingManagement)
```

Sets the printing management to forward messages to.

registrationAccepted

`void` **`registrationAccepted`**`()`

Called when the scheduler has accepted our registration.

schedulerStateChanged

`void` **`schedulerStateChanged`**`(int newState)`

Called when the scheduler informs of its state change.

Parameters:

`newState` -
the scheduler's new state: 1 = running, 2 = stopped

nippu.communications

Class PrintingManagementUserInterfaceProtocol

```
java.lang.Object
    |-terttu.server.network.Protocol
        |-terttu.server.network.TerttuProtocol
            |-nippu.communications.NippuProtocol
                |-nippu.communications.PrintingManagementUserInterfaceProtocol
```

public class **PrintingManagementUserInterfaceProtocol**
extends `NippuProtocol`

A protocol class for connections of printing management towards user interface. A protocol class handles the conversion of communications messages of one connection type on the end of the protocol's component. The conversion is made in the send methods from method calls to byte stream, and in the handleMessage method from byte stream to method calls. In addition to user interfaces, this class can also be used towards analyzers.

Author:
Ville Nurmi

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

services

```
private nippu.communications.PrintingManagementUserInterfaceServices services
```

The instance who knows what to do with incoming messages.

Constructors

PrintingManagementUserInterfaceProtocol

```
public PrintingManagementUserInterfaceProtocol()
```

Methods

setServices

```
public void setServices(PrintingManagementUserInterfaceServices services)
```

(continued from last page)

handleMessage

```
protected void handleMessage(byte header,
                             int dataLength)
                      throws java.io.IOException
```

See Also:

NippuProtocol

---

sendRegistrationAccepted

```
public void sendRegistrationAccepted(int listenerID,
                                     int listenerType)
                              throws java.io.IOException
```

Sends acception of earlier registration and the listener ID assigned to the registree. The listener can be of one of the following types: 1 = the first super user interface, 0 = other.

Parameters:

`listenerID` -
the ID of the listener
`listenerType` -
the type of the listener

---

sendUnregistrationAccepted

```
public void sendUnregistrationAccepted(int returnValue)
                                throws java.io.IOException
```

Sends acception of earlier unregistration.

Parameters:

`returnValue` -
meaning undefined

---

sendPrintJobQueued

```
public void sendPrintJobQueued(java.lang.String printer,
                               PrintJob printJob)
                        throws java.io.IOException
```

Sends a notification that a print job has been queued for some printer.

Parameters:

`printer` -
name of the printer
`printJob` -
the queued print job

---

sendPrintJobPrinted

```
public void sendPrintJobPrinted(PrintJob printJob)
                         throws java.io.IOException
```

Sends a notification that a print job has been printed.

Parameters:

`printJob` -
the printed print job

(continued from last page)

sendSchedulerStateChanged

```
public void sendSchedulerStateChanged(int newState)
                               throws java.io.IOException
```

Sends a notification that the scheduler's run state has changed.

Parameters:

`newState` -
the new run state, 1 = running, 2 = stopped.

---

sendConfiguration

```
public void sendConfiguration(java.lang.String configuration)
                       throws java.io.IOException
```

Sends the given configuration.

nippu.communications

Class PrintingManagementUserInterfaceServices

```
java.lang.Object
   |
   +-nippu.communications.PrintingManagementUserInterfaceServices
```

public class **PrintingManagementUserInterfaceServices**
extends java.lang.Object

A services class for connections of printing management towards user interface. A services class forwards messages interpreted by a protocol class to the classes who are concerned. Thus, the services class can be seen as a higher level of communications handling whereas the protocol class is a lower level. A services class consists mainly of service methods. They are called by the protocol class only. In addition to user interfaces, this class can also be used towards analyzers.

Author:
Ville Nurmi

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

protocol

```
private nippu.communications.PrintingManagementUserInterfaceProtocol protocol
```

The instance who can send actual messages to the other end of our connection.

printingManagement

```
private nippu.printingmanagement.PrintingManagement printingManagement
```

The printing management to forward messages to.

Constructors

PrintingManagementUserInterfaceServices

```
public PrintingManagementUserInterfaceServices(PrintingManagement printingManagement)
```

Creates a new services object.

Parameters:
`printingManagement` -
the printing management to forward messages to.

Methods

setProtocol

```
public void setProtocol(PrintingManagementUserInterfaceProtocol protocol)
```

setPrintingManagement

`public void **setPrintingManagement**(PrintingManagement printingManagement)`

listenerRegistration

`void **listenerRegistration**(int listenerType)`

Called when a new listener asks to be registered. The listener can be of one of the following types: 1 = a super user interface, 0 = other.

Parameters:

`listenerType -`
the type of the listener

listenerUnregistration

`void **listenerUnregistration**(int listenerID)`

Called when an existing listener asks to be unregistered.

Parameters:

`listenerID -`
the ID of the listener.

schedulerStateChangeQuery

`void **schedulerStateChangeQuery**(int newState)`

Called when an existing listener asks to change the run state of the scheduler.

Parameters:

`newState -`
the asked new run state.

nippu.communications

Class ReceptionAnalyzerProtocol

```
java.lang.Object
    |-terttu.server.network.Protocol
        |-terttu.server.network.TerttuProtocol
            |-nippu.communications.NippuProtocol
                |-nippu.communications.ReceptionAnalyzerProtocol
```

---

public class **ReceptionAnalyzerProtocol**
extends NippuProtocol

A protocol class for connections of reception towards analyzer. A protocol class handles the conversion of communications messages of one connection type on the end of the protocol's component. The conversion is made in the send methods from method calls to byte stream, and in the handleMessage method from byte stream to method calls.

Author:
Ville Nurmi

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

---

services

```
private nippu.communications.ReceptionAnalyzerServices services
```

The instance who knows what to do with incoming messages.

Constructors

ReceptionAnalyzerProtocol

```
public ReceptionAnalyzerProtocol()
```

Methods

setServices

```
public void setServices(ReceptionAnalyzerServices services)
```

(continued from last page)

handleMessage

```
protected void handleMessage(byte header,
                             int dataLength)
                      throws java.io.IOException
```

See Also:

NippuProtocol

---

sendGetCoordinatesQuery

```
public void sendGetCoordinatesQuery(Bundle bundle)
                             throws java.io.IOException
```

---

sendCostQuery

```
public void sendCostQuery(Bundle bundle)
                   throws java.io.IOException
```

---

sendCombinedCostQuery

```
public void sendCombinedCostQuery(Bundle bundle1,
                                  Bundle bundle2)
                           throws java.io.IOException
```

---

sendFusionQuery

```
public void sendFusionQuery(Bundle bundle1,
                            Bundle bundle2)
                     throws java.io.IOException
```

---

sendToStringQuery

```
public void sendToStringQuery(Bundle bundle)
                       throws java.io.IOException
```

nippu.communications

Class ReceptionAnalyzerServices

```
java.lang.Object
   |
   +-nippu.communications.ReceptionAnalyzerServices
```

public class **ReceptionAnalyzerServices**
extends java.lang.Object

A services class for connections of reception towards analyzer. A services class forwards messages interpreted by a protocol class to the classes who are concerned. Thus, the services class can be seen as a higher level of communications handling whereas the protocol class is a lower level. A services class consists mainly of service methods. They are called by the protocol class only.

Author:
Ville Nurmi

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

protocol

```
private nippu.communications.ReceptionAnalyzerProtocol protocol
```

The instance who can send actual messages to the other end of our connection.

reception

```
private nippu.reception.Reception reception
```

The instance to forward messages to.

Constructors

ReceptionAnalyzerServices

```
public ReceptionAnalyzerServices()
```

Methods

setProtocol

```
public void setProtocol(ReceptionAnalyzerProtocol protocol)
```

setReception

```
public void setReception(Reception reception)
```

(continued from last page)

Sets the instance to forward messages to.

getCoordinatesAnswer

```
void getCoordinatesAnswer(int bundleID,
                          float coordinates)
```

costAnswer

```
void costAnswer(int bundleID,
                float cost)
```

combinedCostAnswer

```
void combinedCostAnswer(int bundle1ID,
                        int bundle2ID,
                        float cost)
```

fusionAnswer

```
void fusionAnswer(int bundle1ID,
                  int bundle2ID,
                  Bundle fusion)
```

toStringAnswer

```
void toStringAnswer(int bundleID,
                    java.lang.String bundleString)
```

error

```
void error()
```

nippu.communications

Class ReceptionSchedulerProtocol

```
java.lang.Object
    |-terttu.server.network.Protocol
        |-terttu.server.network.TerttuProtocol
            |-nippu.communications.NippuProtocol
                |-nippu.communications.ReceptionSchedulerProtocol
```

public class **ReceptionSchedulerProtocol**
extends `NippuProtocol`

A protocol class for reception towards scheduler. A protocol class handles the conversion of communications messages of one connection type on the end of the protocol's component. The conversion is made in the send methods from method calls to byte stream, and in the handleMessage method from byte stream to method calls.

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

services

```
private nippu.communications.ReceptionSchedulerServices services
```

The instance who knows what to do with incoming messages.

Constructors

ReceptionSchedulerProtocol

```
public ReceptionSchedulerProtocol()
```

Methods

setServices

```
public void setServices(ReceptionSchedulerServices services)
```

handleMessage

```
protected void handleMessage(byte header,
                             int dataLength)
                      throws java.io.IOException
```

(continued from last page)

See Also:

NippuProtocol

sendNewBundle

```
public void sendNewBundle(Bundle bundle)
                   throws java.io.IOException
```

Sends a new bundle.

nippu.communications

Class ReceptionSchedulerServices

```
java.lang.Object
   |
   +-nippu.communications.ReceptionSchedulerServices
```

public class **ReceptionSchedulerServices**
extends java.lang.Object

A services class for connections of reception towards scheduler. A services class forwards messages interpreted by a protocol class to the classes who are concerned. Thus, the services class can be seen as a higher level of communications handling whereas the protocol class is a lower level. A services class consists mainly of service methods. They are called by the protocol class only.

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

protocol

```
private nippu.communications.ReceptionSchedulerProtocol protocol
```

The instance who can send actual messages to the other end of our connection.

reception

```
private nippu.reception.Reception reception
```

The reception to forward messages to.

Constructors

ReceptionSchedulerServices

```
public ReceptionSchedulerServices()
```

Methods

setProtocol

```
public void setProtocol(ReceptionSchedulerProtocol protocol)
```

setReception

```
public void setReception(Reception reception)
```

Set the reception to forward messages to.

(continued from last page)

newBundleReceived

`void` **`newBundleReceived`**`(int returnValue)`

Called when an acknowledgement of an added bundle is received.

Parameters:

`returnValue` -
a return value from the add functionality, meaning undefined

---

newBundleRejected

`void` **`newBundleRejected`**`(int failReason)`

Called when a negative acknowledgement of an added bundle is received. This means that the bundle was rejected. The reason of failure may be one of the following: 2 = service unavailable, 3 = too many jobs, 5 = bad job, 15 = unknown failure.

Parameters:

`failReason` -
the reason of failure

nippu.communications

Class ReceptionUserInterfaceProtocol

```
java.lang.Object
    |-terttu.server.network.Protocol
        |-terttu.server.network.TerttuProtocol
            |-nippu.communications.NippuProtocol
                |-nippu.communications.ReceptionUserInterfaceProtocol
```

public class **ReceptionUserInterfaceProtocol**
extends `NippuProtocol`

A protocol class for reception towards user interface. A protocol class handles the conversion of communications messages of one connection type on the end of the protocol's component. The conversion is made in the send methods from method calls to byte stream, and in the handleMessage method from byte stream to method calls.

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

services

```
private nippu.communications.ReceptionUserInterfaceServices services
```

The instance who knows what to do with incoming messages.

Constructors

ReceptionUserInterfaceProtocol

```
public ReceptionUserInterfaceProtocol()
```

Methods

setServices

```
public void setServices(ReceptionUserInterfaceServices services)
```

handleMessage

```
protected void handleMessage(byte header,
                             int dataLength)
                      throws java.io.IOException
```

(continued from last page)

See Also:

NippuProtocol

sendRegistrationAccepted

```
public void sendRegistrationAccepted(int listenerID)
                              throws java.io.IOException
```

Sends acception of earlier registration and the listener ID assigned to the registree. The listener can be of one of the following types: 1 = the first super user interface, 0 = other.

Parameters:

`listenerID` -
the ID of the listener

sendUnregistrationAccepted

```
public void sendUnregistrationAccepted(int returnValue)
                                throws java.io.IOException
```

Sends acception of earlier unregistration.

Parameters:

`returnValue` -
meaning undefined

sendNewBundleReceived

```
public void sendNewBundleReceived(int returnValue)
                           throws java.io.IOException
```

sendMaxServiceDelayChange

```
public void sendMaxServiceDelayChange(int newMaxServiceDelay)
                               throws java.io.IOException
```

Signals a change of the maximum service delay.

Parameters:

`newMaxServiceDelay` -
the new delay in seconds.

nippu.communications

Class ReceptionUserInterfaceServices

```
java.lang.Object
   |
   +-nippu.communications.ReceptionUserInterfaceServices
```

public class **ReceptionUserInterfaceServices**
extends java.lang.Object

A services class for connections of reception towards user interface. A services class forwards messages interpreted by a protocol class to the classes who are concerned. Thus, the services class can be seen as a higher level of communications handling whereas the protocol class is a lower level. A services class consists mainly of service methods. They are called by the protocol class only.

Author:
Ville Nuutinen, Ville Nurmi

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

protocol

```
private nippu.communications.ReceptionUserInterfaceProtocol protocol
```

The instance who can send actual messages to the other end of our connection.

reception

```
private nippu.reception.Reception reception
```

The reception instance to forward messages to.

Constructors

ReceptionUserInterfaceServices

```
public ReceptionUserInterfaceServices(Reception reception)
```

Methods

setProtocol

```
public void setProtocol(ReceptionUserInterfaceProtocol protocol)
```

listenerRegistration

```
void listenerRegistration(int listenerType)
```

(continued from last page)

Called when a new listener asks to be registered.

Parameters:

`listenerType` -
meaningless

---

listenerUnregistration

`void` **`listenerUnregistration`**`(int listenerID)`

Called when an existing listener asks to be unregistered.

Parameters:

`listenerID` -
the ID of the listener.

---

newBundle

`void` **`newBundle`**`(Bundle bundle)`

A message to add a new bundle to reception.

---

maxServiceDelayChangeQuery

`void` **`maxServiceDelayChangeQuery`**`(int newMaxServiceDelay)`

A message to change the maximum service delay.

Parameters:

`newMaxServiceDelay` -
the new delay in seconds

nippu.communications

Class UserInterfacePrintingManagementProtocol

```
java.lang.Object
    |
    +-terttu.server.network.Protocol
        |
        +-terttu.server.network.TerttuProtocol
            |
            +-nippu.communications.NippuProtocol
                |
                +-nippu.communications.UserInterfacePrintingManagementProtocol
```

public class **UserInterfacePrintingManagementProtocol**
extends `NippuProtocol`

A protocol class for user interface towards printing management. A protocol class handles the conversion of communications messages of one connection type on the end of the protocol's component. The conversion is made in the send methods from method calls to byte stream, and in the handleMessage method from byte stream to method calls.

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

services

```
private nippu.communications.UserInterfacePrintingManagementServices services
```

The instance who knows what to do with incoming messages.

Constructors

UserInterfacePrintingManagementProtocol

```
public UserInterfacePrintingManagementProtocol()
```

Methods

setServices

```
public void setServices(UserInterfacePrintingManagementServices services)
```

handleMessage

```
protected void handleMessage(byte header,
                             int dataLength)
                      throws java.io.IOException
```

(continued from last page)

See Also:

NippuProtocol

sendListenerRegistration

```
public void sendListenerRegistration()
                              throws java.io.IOException
```

Sends a request to be registered as a listener.

sendListenerRegistration

```
public void sendListenerRegistration(int listenerType)
                              throws java.io.IOException
```

Sends a request to be registered as a listener. The listener type can be specified and it must be one of the following: 1 = a super user interface, 0 = other.

Parameters:

`listenerType` -
the type of the listener

sendListenerUnregistration

```
public void sendListenerUnregistration(int listenerID)
                                throws java.io.IOException
```

Sends a request to be unregistered from being a listener.

Parameters:

`listenerID` -
the ID of the listener

sendSchedulerStateChangeQuery

```
public void sendSchedulerStateChangeQuery(int newState)
                                   throws java.io.IOException
```

Sends a request to change the state of the scheduler, 3 = pause, 4 = resume.

Parameters:

`newState` -
the requested new state

nippu.communications

Class UserInterfacePrintingManagementServices

```
java.lang.Object
   |
   +-nippu.communications.UserInterfacePrintingManagementServices
```

public class **UserInterfacePrintingManagementServices**
extends java.lang.Object

A services class for connections of user interface towards printing management. A services class forwards messages interpreted by a protocol class to the classes who are concerned. Thus, the services class can be seen as a higher level of communications handling whereas the protocol class is a lower level. A services class consists mainly of service methods. They are called by the protocol class only.

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

protocol

```
private nippu.communications.UserInterfacePrintingManagementProtocol protocol
```

The instance who can send actual messages to the other end of our connection.

frame

```
nippu.userinterface.Frame1 frame
```

The frame to forward messages to.

Constructors

UserInterfacePrintingManagementServices

```
public UserInterfacePrintingManagementServices()
```

Methods

setProtocol

```
public void setProtocol(UserInterfacePrintingManagementProtocol protocol)
```

setFrame

```
public void setFrame(Frame1 frame)
```

Set the user interface frame to forward messages to.

registrationAccepted

```
void registrationAccepted(int listenerID,
                          int listenerType)
```

Called when printing management informs that our earlier registration has been accepted.

Parameters:

`listenerID` -
the ID assigned for us as a listener
`listenerType` -
our type: 1 = the first super user interface, 0 = other (a normal user interface, or not-first super user interface).

unregistrationAccepted

```
void unregistrationAccepted(int returnValue)
```

Called when printing management informs that our earlier unregistration has been accepted.

Parameters:

`returnValue` -
undefined

printJobQueued

```
void printJobQueued(PrintJob printJob,
                    java.lang.String printer)
```

Called when printing management informs that the given print job has been queued to the specified printer.

printJobPrinted

```
void printJobPrinted(PrintJob printJob)
```

Called when printing management informs that the given print job has been printed.

schedulerStateChanged

```
void schedulerStateChanged(int newState)
```

Called when printing management informs of scheduler's state change.

Parameters:

`newState` -
the scheduler's new state: 1 = running, 2 = stopped

configuration

```
void configuration(java.lang.String configuration)
```

Called when printing management sends the configuration.

nippu.communications

Class UserInterfaceReceptionProtocol

```
java.lang.Object
    |-terttu.server.network.Protocol
        |-terttu.server.network.TerttuProtocol
            |-nippu.communications.NippuProtocol
                |-nippu.communications.UserInterfaceReceptionProtocol
```

public class **UserInterfaceReceptionProtocol**
extends `NippuProtocol`

A protocol class for user interface towards reception. A protocol class handles the conversion of communications messages of one connection type on the end of the protocol's component. The conversion is made in the send methods from method calls to byte stream, and in the handleMessage method from byte stream to method calls.

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

services

```
private nippu.communications.UserInterfaceReceptionServices services
```

The instance who knows what to do with incoming messages.

Constructors

UserInterfaceReceptionProtocol

```
public UserInterfaceReceptionProtocol()
```

Methods

setServices

```
public void setServices(UserInterfaceReceptionServices services)
```

handleMessage

```
protected void handleMessage(byte header,
                             int dataLength)
                      throws java.io.IOException
```

(continued from last page)

See Also:

NippuProtocol

sendNewBundle

```
public void sendNewBundle(Bundle bundle)
                  throws java.io.IOException
```

sendMaxServiceDelayChangeQuery

```
public void sendMaxServiceDelayChangeQuery(int newMaxServiceDelay)
                                    throws java.io.IOException
```

sendListenerRegistration

```
public void sendListenerRegistration()
                             throws java.io.IOException
```

Sends a request to be registered as a listener.

sendListenerUnregistration

```
public void sendListenerUnregistration(int listenerID)
                               throws java.io.IOException
```

Sends a request to be unregistered from being a listener.

Parameters:

`listenerID` -
the ID of the listener

nippu.communications

Class UserInterfaceReceptionServices

```
java.lang.Object
   |
   +-nippu.communications.UserInterfaceReceptionServices
```

public class **UserInterfaceReceptionServices**
extends java.lang.Object

A services class for connections of user interface towards reception. A services class forwards messages interpreted by a protocol class to the classes who are concerned. Thus, the services class can be seen as a higher level of communications handling whereas the protocol class is a lower level. A services class consists mainly of service methods. They are called by the protocol class only.

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

protocol

```
private nippu.communications.UserInterfaceReceptionProtocol protocol
```

The instance who can send actual messages to the other end of our connection.

frame

```
nippu.userinterface.Frame1 frame
```

The frame to forward messages to.

Constructors

UserInterfaceReceptionServices

```
public UserInterfaceReceptionServices()
```

Methods

setProtocol

```
public void setProtocol(UserInterfaceReceptionProtocol protocol)
```

setFrame

```
public void setFrame(Frame1 frame)
```

Set the user interface frame to forward messages to. If the frame is null, then all received messages will be quietly scrapped.

registrationAccepted

```
void registrationAccepted(int listenerID,
                          int listenerType)
```

unregistrationAccepted

```
void unregistrationAccepted(int returnValue)
```

newBundleReceived

```
void newBundleReceived(int returnValue)
```

newMaxServiceDelay

```
void newMaxServiceDelay(int newMaxServiceDelay)
```

Package
nippu

nippu
Class Bundle

```
java.lang.Object
   |
   +-nippu.Bundle
```

public class **Bundle**
extends java.lang.Object

This class is implementation of print job -bundle. One Bundle includes >=1 PrintJob-objects

Version:
1.24.

Author:
Ville Nuutinen

Fields

bundleID
```
private int bundleID
```

x
```
private int x
```

y
```
private int y
```

z
```
private int z
```

optimalPrinters
```
private java.lang.String optimalPrinters
```

jobs
```
private java.util.Vector jobs
```

(continued from last page)

NUMBEROFOPTIMALPRINTERS

```
static final int NUMBEROFOPTIMALPRINTERS
```

Constructors

Bundle

```
public Bundle(int bundleID)
```

CONSTRUCTORS

Bundle

```
public Bundle(PrintJob job,
              int bundleID)
```

Used when made Bundle which includes only one PrintJob. Here method add(job) also calculates and sets automatically values of x,y and z of bundle. Please use Bundle.createID() for creating a unique ID for the bundle.

Parameters:

`bundleID` -
a unique ID for the bundle

Bundle

```
public Bundle(int bundleID,
              java.lang.String optimalPrinters,
              java.util.Vector jobs)
```

Here method setJobs also calculates and sets automatically values of x,y and z of bundle. Please use Bundle.createID() for creating a unique ID for the bundle and give it as a parameter.

Parameters:

`bundleID` -
a unique ID for the bundle

Methods

getBundleID

```
public int getBundleID()
```

Get-methods********** These methods returns the value of Objects field.

---

getX

```
public int getX()
```

---

getY

```
public int getY()
```

---

getZ

```
public int getZ()
```

(continued from last page)

getOptimalPrinters

```
public java.lang.String[] getOptimalPrinters()
```

getJobs

```
public java.util.Vector getJobs()
```

getNumberOfPages

```
public int getNumberOfPages()
```

Returns the sum of the number of pages of all print jobs in the bundle.

getForcedPrinter

```
public java.lang.String getForcedPrinter()
```

Returns the forced printer of the bundle, if there is such. Otherwise, returns a string indicating the absence of a forced printer. Note: If there are several print jobs in the bundle with different forced printers, this method will return only one of them.

See Also:

hasForcedPrinter()

setX

```
private int setX(int x)
```

Set-methods********** The values of fields can be set by using these methods.

setY

```
private int setY(int y)
```

setZ

```
private int setZ(int z)
```

setOptimalPrinters

```
public boolean setOptimalPrinters(java.lang.String printers)
```

setJobs

```
public boolean setJobs(java.util.Vector printjobs)
```

(continued from last page)

sort

```
public void sort()
```

Sorts the jobs inside a bundle.

---

union

```
public Bundle union(Bundle otherBundle,
                    int componentID,
                    int componentBundleCounter)
```

Makes union with parameter otherBundle. The parameters componentID and componentBundleCounter are as in the method createID(). They help create a unique ID for the new bundle.

Parameters:

`Bundle` -
otherBundle is Bundle to union with.
`componentID` -
is idnumber of component, which called this method.
`componentBundleCounter` -
is value of counter of Bundles. Every component has its own counter for created Bundles.

Returns:

Bundle new bundle, which is union of two bundles

---

createID

```
public static int createID(int componentID,
                           int componentBundleCounter)
```

Creates bundleID, which can be given for Bundle-constructor.

Parameters:

`componentID` -
is idnumber of component, which called this method.
`componentBundleCounter` -
is value of counter of Bundles. Every component has its own counter for created Bundles.

---

serialize

```
public java.lang.String serialize()
```

Makes String representation of bundle, which is ment for example for other methods - not for human!

Returns:

String String representation of bundle

---

deserialize

```
public static Bundle deserialize(java.lang.String serializedBundle)
```

Makes new bundle out of String representation of Bundle

Parameters:

`String` -
serializedBundle: String representation of Bundle

Returns:

Bundle

(continued from last page)

breakBundle

```
public Bundle[] breakBundle(int componentID,
                            int componentBundleCounter)
```

Method breaks bundle.

Parameters:

`componentID` -
the ID number of the component that called this method
`componentBundleCounter` -
the value of counter of Bundles. Every component has its own counter for created Bundles.

Returns:

Bundle[] array of bundles. Every new bundle gets same optimal printers as Mother-bundle! The parameters componentID and componentBundleCounter are as in the method createID(). They help create a unique ID for all new bundles. Remember to increase the original bundle counter by the number of elements in the array this method returns.

---

addJob

```
public boolean addJob(PrintJob job)
```

Adds one PrintJob in the jobs-Vector and calculate new values of x, y and z after addition (averages).

Parameters:

`PrintJob` -
job, PrintJob to be added

---

addOptimalPrinter

```
public boolean addOptimalPrinter(java.lang.String printer)
```

Adds one more optimal printer.

Parameters:

`String` -
printer, printer to be added.Has to be <= 10!!

Returns:

true, if succeeded, otherwise false.

---

removeOptimalPrinter

```
public boolean removeOptimalPrinter(java.lang.String printer)
```

Removes optimal printer.

Parameters:

`String` -
printer, printer to be removed Method sets value null in that index where removed printer did exists.

Returns:

true, if succeeded, otherwise false.

---

hasForcedPrinter

```
public boolean hasForcedPrinter()
```

This method checks if some of the printjobs of this bundle has forcedPrinter.

Returns:

(continued from last page)

boolean. True if there is >=1 forcedPrinters, else false.

toString

`public java.lang.String **toString**()`

String representation of Bundle for human use, example for testing.

main

`public static void **main**(java.lang.String args)`

Main program for testing purposes

nippu

Class ConfigurationParser

```
java.lang.Object
   |
   +-nippu.ConfigurationParser
```

public class **ConfigurationParser**
extends java.lang.Object

This class parses configuration file's String representation. With methods it is able to get values out from configuration file.

Version:
1.7.

Author:
Ville Nuutinen

Fields

confFile

```
private java.lang.String confFile
```

userList

```
private java.util.Vector userList
```

printerList

```
private java.util.Vector printerList
```

Constructors

ConfigurationParser

```
public ConfigurationParser(java.lang.String configuration)
```

Constructor.

Parameters:
`configuration` -
is what configuration file includes - not a filename!

Methods

getUsers

```
public java.util.Vector getUsers()
```

Method returns the userList Vector.

getPrinters

```
public java.util.Vector getPrinters()
```

Method returns the printerList Vector.

getString

```
public java.lang.String getString(java.lang.String constant)
```

Returns the value of the specified constant interpreted as a string. If the value cannot be received, null is returned.

getInt

```
public int getInt(java.lang.String constant)
```

Returns the value of the specified constant interpreted as an int. If the value cannot be received, 0 is returned.

getFloat

```
public float getFloat(java.lang.String constant)
```

Returns the value of the specified constant interpreted as a float. If the value cannot be received, 0.0f is returned.

parseFile

```
private boolean parseFile(java.lang.String file)
```

This method finds all users and printers and puts them into userList and printerList Vectors.

createNippuUser

```
private NippuUser createNippuUser(java.lang.String userID,
                                  java.lang.String userPos)
```

Creates one NippuUser-object.

createNippuPrinter

```
private NippuPrinter createNippuPrinter(java.lang.String printerID,
                                        java.lang.String printerPos)
```

Creates on NippuPrinter-object.

toString

```
public java.lang.String toString()
```

toString-method for testing.

main

```
public static void main(java.lang.String args)
```

Main-program for testing purposes.

nippu

Class NippuPrinter

```
java.lang.Object
   |
   +-nippu.NippuPrinter
```

Direct Known Subclasses:
NippuPrinterSimulator

---

public class **NippuPrinter**
extends java.lang.Object

The class representing a printer in the Nippu system.

Version:
1.8.

Author:
Michael Forström, Ville Nuutinen, Chen Zhao

Fields

printerID

`private java.lang.String printerID`

Printer identification number

---

printerX

`private int printerX`

x-coordinate for the printer

---

printerY

`private int printerY`

y-coordinate for the printer

---

printerZ

`private int printerZ`

z-coordinate for the printer

---

printerSpeed

`private int printerSpeed`

The speed of the printer

Constructors

(continued from last page)

NippuPrinter

```
public NippuPrinter(java.lang.String ID,
                    int X,
                    int Y,
                    int Z,
                    int speed)
```

Constructor

Parameters:

`ID` -
The printer identification number
`X` -
The x-coordinate for the printer
`Y` -
The y-coordinate for the printer
`Z` -
The z-coordinate for the printer
`speed` -
The speed of the printer

Methods

getPrinterID

```
public java.lang.String getPrinterID()
```

Getter for the printer identification number

Returns:

The printer identification number

---

getPrinterX

```
public int getPrinterX()
```

Getter for the x-coordinate

Returns:

The x-coordinate

---

getPrinterY

```
public int getPrinterY()
```

Getter for the y-coordinate

Returns:

The y-coordinate

---

getPrinterZ

```
public int getPrinterZ()
```

Getter for the z-coordinate

Returns:

The z-coordinate for the printer

getPrinterSpeed

`public int` **`getPrinterSpeed`**`()`

Getter for the printer speed

Returns:

The printer speed

toString

`public java.lang.String` **`toString`**`()`

toString-method for testing.

main

`public static void` **`main`**`(java.lang.String args)`

Main program for testing purposes.

nippu

Class NippuUser

```
java.lang.Object
   |
   +-nippu.NippuUser
```

public class **NippuUser**
extends java.lang.Object

This class is implementation of user of this Nippu-system.

Version:
1.4.

Author:
Ville Nuutinen

Fields

userID

```
private java.lang.String userID
```

posX

```
private int posX
```

posY

```
private int posY
```

posZ

```
private int posZ
```

Constructors

NippuUser

```
public NippuUser(java.lang.String ID,
                 int X,
                 int Y,
                 int Z)
```

Methods

(continued from last page)

getPosX

```
public int getPosX()
```

Get-methods

getPosY

```
public int getPosY()
```

getPosZ

```
public int getPosZ()
```

getUserID

```
public java.lang.String getUserID()
```

toString

```
public java.lang.String toString()
```

main

```
public static void main(java.lang.String args)
```

Test main program.

nippu

Class PrintJob

```
java.lang.Object
   |
   +-nippu.PrintJob
```

public class **PrintJob**
extends java.lang.Object

This class is implementation of print job.

Version:
1.13.

Author:
Ville Nuutinen

Fields

jobName

```
private java.lang.String jobName
```

jobID

```
private int jobID
```

user

```
private java.lang.String user
```

x

```
private int x
```

y

```
private int y
```

z

```
private int z
```

(continued from last page)

numberOfPages

```
private int numberOfPages
```

forcedPrinter

```
private java.lang.String forcedPrinter
```

senderUI

```
private int senderUI
```

Constructors

PrintJob

```
public PrintJob()
```

Constructors

PrintJob

```
public PrintJob(java.lang.String jobName,
                int jobID,
                java.lang.String user,
                int x,
                int y,
                int z,
                int numberOfPages,
                java.lang.String forcedPrinter,
                int senderUI)
```

Methods

getJobName

```
public java.lang.String getJobName()
```

get-methods********** These methods

Returns:

the value of Objects field.

getJobID

```
public int getJobID()
```

getUser

```
public java.lang.String getUser()
```

getX

```
public int getX()
```

getY

```
public int getY()
```

getZ

```
public int getZ()
```

getNumberOfPages

```
public int getNumberOfPages()
```

getForcedPrinter

```
public java.lang.String getForcedPrinter()
```

getSenderUI

```
public int getSenderUI()
```

setJobName

```
public boolean setJobName(java.lang.String name)
```

set-methods********** The values of fields can be set by using these methods. WARNING:User has to know, what is doing, when uses these PUBLIC set-methods!!

setJobID

```
public int setJobID(int id)
```

setUser

```
public boolean setUser(java.lang.String user)
```

setX

```
public int setX(int x)
```

(continued from last page)

setY

```
public int setY(int y)
```

setZ

```
public int setZ(int z)
```

setNumberOfPages

```
public int setNumberOfPages(int pages)
```

setForcedPrinter

```
public boolean setForcedPrinter(java.lang.String printer)
```

setSenderUI

```
public int setSenderUI(int ui)
```

equals

```
public boolean equals(java.lang.Object other)
```

Method compare PrintJob to other PrintJob. If every field is equal, true is returned otherwise false.

Parameters:

`other` -
Object, which has to be instance of PrintJob.

Returns:

true, if PrintJobs are equal, else false.

serialize

```
public java.lang.String serialize()
```

Return serializd String of PrintJob.

Returns:

String serializd String of PrintJob.

deserialize

```
public static PrintJob deserialize(java.lang.String serializedPrintJob)
```

Makes new PrintJob-object out of serialized PrintJob-String.

Parameters:

`String` -
serializedPrintJob, which is String-representation of PrintJob

(continued from last page)

Returns:

new PrintJob-object

toString

```
public java.lang.String toString()
```

Returns:

String representation of PrintJob. String is ment for human, for example for testing purposes.

hasForcedPrinter

```
public boolean hasForcedPrinter()
```

Method checks if there is forcedPrinter.

Returns:

boolean. True if there is forcedPrinter, else false.

main

```
public static void main(java.lang.String args)
```

Test main-program.

Package
nippu.userinterface

nippu.userinterface

Class Frame1

```
java.lang.Object
    |-java.awt.Component
        |-java.awt.Container
            |-java.awt.Window
                |-java.awt.Frame
                    |-javax.swing.JFrame
                        |-nippu.userinterface.Frame1
```

public class **Frame1**
extends javax.swing.JFrame

Title: Nippu

Description: Nippu käyttöliittymä

Copyright: Copyright (c) 2003

Company:

Version:
1.0

Author:
Chen Zhao

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

usReProtocol

```
private nippu.communications.UserInterfaceReceptionProtocol usReProtocol
```

The protocol to use with communication to reception.

usPrProtocol

```
private nippu.communications.UserInterfacePrintingManagementProtocol usPrProtocol
```

The protocol to use with communication to printing management.

listenerID

```
private int listenerID
```

The listener ID of the user interface given by printing management.

receptionListenerID

`private int receptionListenerID`

The listener ID of the user interface given by reception.

initializationLock

`private java.lang.String initializationLock`

A lock for initialization. It is just *some* object!

printerQueueLock

`private java.lang.String printerQueueLock`

A lock for the printer queues. It is just *some* object!

mainLogLock

`private java.lang.String mainLogLock`

A lock for the main log. It is just *some* object!

ownJobsLogLock

`private java.lang.String ownJobsLogLock`

A lock for the own jobs log. It is just *some* object!

isInitialized

`private int isInitialized`

Counter of items that have been initialized. Such items are listenerID, receptionListenerID, isFirstSuperUser and configuration. Thus, when isInitialized == 4, then we are fully initialized.

isFirstSuperUser

`private boolean isFirstSuperUser`

NUMBEROFPRINTERS

`private final int NUMBEROFPRINTERS`

The maximum number of printers.

NUMBEROFJOBS

`private final int NUMBEROFJOBS`

The number of print jobs visible per queue.

OWNLOG_RECEIVED

`private java.lang.String OWNLOG_RECEIVED`

(continued from last page)

OWNLOG_QUEUED

private java.lang.String **OWNLOG_QUEUED**

OWNLOG_PRINTING

private java.lang.String **OWNLOG_PRINTING**

OWNLOG_PRINTED

private java.lang.String **OWNLOG_PRINTED**

howManyChars

int **howManyChars**

exitWindows

javax.swing.JButton **exitWindows**

errWarningLabel

javax.swing.JLabel **errWarningLabel**

sUserErrLabel

javax.swing.JLabel **sUserErrLabel**

sUserErrLabel2

javax.swing.JLabel **sUserErrLabel2**

currentMaxDelayLabel

javax.swing.JLabel **currentMaxDelayLabel**

sendMaxDelayButton

javax.swing.JButton **sendMaxDelayButton**

nippuPrinters

nippu.NippuPrinter **nippuPrinters**

psCurrentJob

nippu.PrintJob **psCurrentJob**

psCurrentTextArea

javax.swing.JTextArea **psCurrentTextArea**

psNoTextArea

javax.swing.JTextArea **psNoTextArea**

contentPane

javax.swing.JPanel **contentPane**

printerLogPanel

javax.swing.JPanel **printerLogPanel**

titledBorder1

javax.swing.border.TitledBorder **titledBorder1**

xYLayout1

com.borland.jbcl.layout.XYLayout **xYLayout1**

normalUserPanel

javax.swing.JPanel **normalUserPanel**

titledBorder2

javax.swing.border.TitledBorder **titledBorder2**

userLogPanel

javax.swing.JPanel **userLogPanel**

(continued from last page)

border1

javax.swing.border.Border **border1**

titledBorder3

javax.swing.border.TitledBorder **titledBorder3**

superUserPanel

javax.swing.JPanel **superUserPanel**

border2

javax.swing.border.Border **border2**

titledBorder4

javax.swing.border.TitledBorder **titledBorder4**

printingLogPanel

javax.swing.JPanel **printingLogPanel**

titledBorder5

javax.swing.border.TitledBorder **titledBorder5**

printJobNameLabel

javax.swing.JLabel **printJobNameLabel**

xYLayout2

com.borland.jbcl.layout.XYLayout **xYLayout2**

printJobNameTextField

javax.swing.JTextField **printJobNameTextField**

pageNumberLabel

javax.swing.JLabel **pageNumberLabel**

pageNumberTextField

javax.swing.JTextField **pageNumberTextField**

userIDLabel

javax.swing.JLabel **userIDLabel**

userIDBox

javax.swing.JComboBox **userIDBox**

printerIDLabel

javax.swing.JLabel **printerIDLabel**

printerIDBox

javax.swing.JComboBox **printerIDBox**

printButton

javax.swing.JButton **printButton**

border3

javax.swing.border.Border **border3**

border4

javax.swing.border.Border **border4**

xYLayout3

com.borland.jbcl.layout.XYLayout **xYLayout3**

border5

javax.swing.border.Border **border5**

(continued from last page)

titledBorder6

javax.swing.border.TitledBorder **titledBorder6**

xYLayout4

com.borland.jbcl.layout.XYLayout **xYLayout4**

border6

javax.swing.border.Border **border6**

showSuperUser

boolean **showSuperUser**

userList

java.util.Vector **userList**

printerList

java.util.Vector **printerList**

userID

java.lang.String **userID**

printerName

java.lang.String **printerName**

psQ

java.util.Vector **psQ**

Printer queues. Contain nippu.PrintJob instances.

superuserRandomJob

nippu.userinterface.RunRandomJob **superuserRandomJob**

randomJobThread

private volatile java.lang.Thread **randomJobThread**

jScrollPane1

javax.swing.JScrollPane **jScrollPane1**

printingTextArea

javax.swing.JTextArea **printingTextArea**

jScrollPane2

javax.swing.JScrollPane **jScrollPane2**

userLogTextArea

javax.swing.JTextArea **userLogTextArea**

userLogTitleLabel

javax.swing.JLabel **userLogTitleLabel**

userLogAckButton

javax.swing.JButton **userLogAckButton**

printingLogTitleLabel

javax.swing.JLabel **printingLogTitleLabel**

xYLayout5

com.borland.jbcl.layout.XYLayout **xYLayout5**

sPageNumberLabel

javax.swing.JLabel **sPageNumberLabel**

sConstantSizeCheckBox

javax.swing.JCheckBox **sConstantSizeCheckBox**

(continued from last page)

sPageNumberLabel1

javax.swing.JLabel **sPageNumberLabel1**

sConstantFrequencyCheckBox

javax.swing.JCheckBox **sConstantFrequencyCheckBox**

sPageNumberLabel2

javax.swing.JLabel **sPageNumberLabel2**

sUserNameBox

javax.swing.JComboBox **sUserNameBox**

sPageNumberLabel3

javax.swing.JLabel **sPageNumberLabel3**

sForcedPrinterBox

javax.swing.JComboBox **sForcedPrinterBox**

sRandomJobButton

javax.swing.JButton **sRandomJobButton**

sActionPanel

javax.swing.JPanel **sActionPanel**

titledBorder7

javax.swing.border.TitledBorder **titledBorder7**

xYLayout6

com.borland.jbcl.layout.XYLayout **xYLayout6**

jLabel3

javax.swing.JLabel **jLabel3**

schedulerLabel

javax.swing.JLabel **schedulerLabel**

sStopSchedulerButton

javax.swing.JButton **sStopSchedulerButton**

currentSchedulerStateLabel

javax.swing.JLabel **currentSchedulerStateLabel**

xYLayout7

com.borland.jbcl.layout.XYLayout **xYLayout7**

titledBorder8

javax.swing.border.TitledBorder **titledBorder8**

sPageMinTextField

javax.swing.JTextField **sPageMinTextField**

sPageMaxTextField

javax.swing.JTextField **sPageMaxTextField**

sFrequencyMinTextField

javax.swing.JTextField **sFrequencyMinTextField**

sFrequencyMaxTextField

javax.swing.JTextField **sFrequencyMaxTextField**

sMaxdelayTextField

javax.swing.JTextField **sMaxdelayTextField**

(continued from last page)

titledBorder9

```
javax.swing.border.TitledBorder titledBorder9
```

printerPanel

```
javax.swing.JPanel printerPanel
```

printerLayout

```
com.borland.jbcl.layout.XYLayout printerLayout
```

printerLabel

```
javax.swing.JLabel printerLabel
```

sRandomJobButtonLabel1

```
javax.swing.JLabel sRandomJobButtonLabel1
```

Constructors

Frame1

```
public Frame1(java.lang.String currentUser,
              boolean superUser)
```

Constructor

Methods

jbInit

```
private void jbInit()
             throws java.lang.Exception
```

Component initialization

updatePrinterTexts

```
private void updatePrinterTexts(int i)
```

Updates printer name and tool tip.

Parameters:

`i` -
the index of the printer

processWindowEvent

```
protected void processWindowEvent(java.awt.event.WindowEvent e)
```

printJobNameTextField_keyTyped

```
void printJobNameTextField_keyTyped(java.awt.event.KeyEvent e)
```

verifyNameLength

```
void verifyNameLength(java.awt.event.KeyEvent e)
```

printButton_actionPerformed

```
void printButton_actionPerformed(java.awt.event.ActionEvent e)
```

when normal user print buttone is clicked, create PrintJob and Bundle - instanssit, send Bundle- instanssi to Reception using methods in networking component.

userLogAckButton_actionPerformed

```
void userLogAckButton_actionPerformed(java.awt.event.ActionEvent e)
```

Called when the user's own jobs log's acknowledgement button is clicked. Removes printed jobs from the user's own jobs log.

sendNewBundle

```
public void sendNewBundle(Bundle newBundle)
```

Sends a bundle to reception. This method is thread-safe.

setProtocol

```
public void setProtocol(UserInterfaceReceptionProtocol usReProtocol)
```

setProtocol

```
public void setProtocol(UserInterfacePrintingManagementProtocol usPrProtocol)
```

initialize

```
public void initialize()
```

Initializes the frame. Sends registration to printing management and receives configuration from it. Also sends a registration to reception. Initializes Swing components.

setListenerID

```
public void setListenerID(int component,
                          int listenerID)
```

Sets the listener ID given by a component. The component is one of the following: 0 = printing managment, 1 = reception. This method is called by communications.

Parameters:

`component` -
the component accepting the registration

setListenerType

```
public void setListenerType(int listenerType)
```

Sets the listener type given by printing managment. This method is called by communications. The listener type can be one of the following: 1 = the first super user interface, 0 = other.

receptionRegistrationAnwer

```
public void receptionRegistrationAnwer()
```

Signals that our registration to reception has been accepted. This method is called by communications.

unregisterAnswer

```
public void unregisterAnswer(int component)
```

Signals that the user interface has been unregistered from a component. The component may be one of the following: 0 = printing management, 1 = reception.

Parameters:

`type -`
type of unregistration

setConfiguration

```
public void setConfiguration(java.lang.String configuration)
```

Sets the configuration given by printing managment. This method is called by communications.

schedulerStateChanged

```
public void schedulerStateChanged(int newState)
```

Sets the scheduling state label. This method is called by communications.

Parameters:

`newState -`
the scheduler's new state: 1 = running, 2 = stopped

appendToMainLog

```
public void appendToMainLog(java.lang.String text)
```

Appends the given text to the main log. This method is thread-safe.

updateToOwnJobsLog

```
public void updateToOwnJobsLog(int senderUI,
                               int jobID,
                               java.lang.String newText)
```

Updates the given string to the specified line of the own jobs log. However, the log is not modified unless the specified user interface ID refers to this user interface. This method is thread-safe.

Parameters:

`senderUI -`
the ID of the user interface that sent this job
`jobID -`
the ID of the print job the string is about
`newText -`
the text string

(continued from last page)

printJobReceived

```
public void printJobReceived(PrintJob printJob)
```

Updates the printing log for another job being created. This method is called internally by the user interface (classes Frame1 and RunRandomJob).

---

printJobQueued

```
public void printJobQueued(PrintJob printJob,
                           java.lang.String printer)
```

Updates the printing log for another job being queued. This method is called by communications.

---

printJobToPrint

```
public void printJobToPrint(PrintJob printJob,
                            java.lang.String printer)
```

Updates the printing log for another job going to print. This method is called internally by user interface.

---

printJobPrinted

```
public void printJobPrinted(PrintJob printJob)
```

Updates the printing log for another job being printed. This method is called by communications.

---

updatePSLogs

```
void updatePSLogs(int queue,
                  java.util.Vector psQ)
```

Updates the visual representation of a printer queue.

Parameters:

`queue` -
the queue to update
`psQ` -
the contents of the queue

---

sRandomJobButton_actionPerformed

```
void sRandomJobButton_actionPerformed(java.awt.event.ActionEvent e)
```

---

sStopSchedulerButton_actionPerformed

```
void sStopSchedulerButton_actionPerformed(java.awt.event.ActionEvent e)
```

---

setNewMaxServiceDelay

```
public void setNewMaxServiceDelay(int newDelay)
```

---

sConstantFrequencyCheckBox_actionPerformed

```
void sConstantFrequencyCheckBox_actionPerformed(java.awt.event.ActionEvent e)
```

(continued from last page)

sConstantSizeCheckBox_actionPerformed

void **sConstantSizeCheckBox_actionPerformed**(java.awt.event.ActionEvent e)

sPageMinTextField_keyPressed

void **sPageMinTextField_keyPressed**(java.awt.event.KeyEvent e)

sPageMaxTextField_keyPressed

void **sPageMaxTextField_keyPressed**(java.awt.event.KeyEvent e)

sFrequencyMinTextField_keyPressed

void **sFrequencyMinTextField_keyPressed**(java.awt.event.KeyEvent e)

sFrequencyMaxTextField_keyPressed

void **sFrequencyMaxTextField_keyPressed**(java.awt.event.KeyEvent e)

refreshRandomJobButton

private void **refreshRandomJobButton**(java.lang.String startStop)

sConstantSizeCheckBox_mouseClicked

void **sConstantSizeCheckBox_mouseClicked**(java.awt.event.MouseEvent e)

verifyNumericInput

void **verifyNumericInput**(java.awt.event.KeyEvent e)

userExists

public boolean **userExists**(java.lang.String userName)

Returns true if the given user exists in the configuration. Before calling this method, initialize() must have been called.

sendMaxDelayButton_actionPerformed

void **sendMaxDelayButton_actionPerformed**(java.awt.event.ActionEvent e)

sUserNameBox_actionPerformed

void **sUserNameBox_actionPerformed**(java.awt.event.ActionEvent e)

sForcedPrinterBox_actionPerformed

```
void sForcedPrinterBox_actionPerformed(java.awt.event.ActionEvent e)
```

exitWindows_actionPerformed

```
void exitWindows_actionPerformed(java.awt.event.ActionEvent e)
```

nippu.userinterface

Class GetJobID

```
java.lang.Object
   |
   +-nippu.userinterface.GetJobID
```

public class **GetJobID**
extends java.lang.Object

Fields

jobID

```
private static volatile int jobID
```

Constructors

GetJobID

```
public GetJobID()
```

Methods

grantJobID

```
public static int grantJobID()
```

nippu.userinterface

Class Nippu

```
java.lang.Object
   |
   +-nippu.userinterface.Nippu
```

public class **Nippu**
extends java.lang.Object

Title: Nippu

Description: Nippu käyttöliittymä

Copyright: Copyright (c) 2003

Company:

Version:
1.0

Author:
Chen Zhao

Fields

logger

```
protected static org.apache.log4j.Logger logger
```

packFrame

```
boolean packFrame
```

superUser

```
static boolean superUser
```

currentUser

```
static java.lang.String currentUser
```

reception_ip

```
static java.lang.String reception_ip
```

(continued from last page)

reception_port

```
static java.lang.String reception_port
```

printingmanagement_ip

```
static java.lang.String printingmanagement_ip
```

printingmanagement_port

```
static java.lang.String printingmanagement_port
```

frame

```
private nippu.userinterface.Frame1 frame
```

usReProtocol

```
private nippu.communications.UserInterfaceReceptionProtocol usReProtocol
```

usPrProtocol

```
private nippu.communications.UserInterfacePrintingManagementProtocol usPrProtocol
```

Constructors

Nippu

```
public Nippu()
```

Constructs the application.

Exceptions:

`IllegalArgumentException` -
if currentUser doesn't exist

Methods

initializeCommunications

```
private void initializeCommunications()
```

Initializes communications for the user interface.

connectToPrintingManagement

```
private boolean connectToPrintingManagement()
```

Tries to connect to printing management. @return true iff the connection failed

(continued from last page)

connectToReception

```
private boolean connectToReception()
```

Tries to connect to reception. @return true iff the connection failed

main

```
public static void main(java.lang.String args)
```

getEnvUser

```
private static java.lang.String getEnvUser()
```

getUserErrMsg

```
private static void getUserErrMsg(int msgNo)
```

nippu.userinterface

Class RunRandomJob

```
java.lang.Object
   |
   +-nippu.userinterface.RunRandomJob
```

All Implemented interfaces:
java.lang.Runnable

---

public class **RunRandomJob**
extends java.lang.Object
implements java.lang.Runnable

Fields

randomGenerator

`java.util.Random` **`randomGenerator`**

---

t

`java.lang.Thread` **`t`**

---

runme

`public boolean` **`runme`**

---

i

`int` **`i`**

---

randomUserID

`java.lang.String` **`randomUserID`**

---

randomPrinterID

`java.lang.String` **`randomPrinterID`**

---

randomSizeMin

`int` **`randomSizeMin`**

randomSizeMax

int **randomSizeMax**

randomFrequencyMin

int **randomFrequencyMin**

randomFrequencyMax

int **randomFrequencyMax**

randomSize

int **randomSize**

randomFrequency

int **randomFrequency**

userList

java.util.Vector **userList**

listenerID

int **listenerID**

printerName

java.lang.String **printerName**

frame

private nippu.userinterface.Frame1 **frame**

The frame containing the user interface.

Constructors

RunRandomJob

public **RunRandomJob**()

RunRandomJob

```
public RunRandomJob(java.util.Vector allUsers,
                    java.lang.String printerName,
                    int sizeMin,
                    int sizeMax,
                    int frequencyMin,
                    int frequencyMax,
                    java.lang.String userID,
                    java.lang.String printerID,
                    int listenerID,
                    Frame1 frame)
```

Methods

run

```
public void run()
```

stopJee

```
public void stopJee()
```

checkPageSize

```
private int checkPageSize(int minPageSize,
                          int maxPageSize)
```

checkFrequency

```
private int checkFrequency(int minFrequency,
                           int maxFrequency)
```

checkUser

```
private NippuUser checkUser(java.lang.String userID)
```

Package
nippu.reception

nippu.reception

Class RBundle

```
java.lang.Object
   |
   +-nippu.reception.RBundle
```

public class **RBundle**
extends java.lang.Object

RBundle Class that holds the bundle and its timer. RBundle is a waiting bundle inside a reception. The bundle is waiting for the reception to forward it to the scheduler.

Fields

reception

`private nippu.reception.Reception` **`reception`**

The reception this waiting bundle belongs to.

rBundleTimer

`private java.util.Timer` **`rBundleTimer`**

The timer for the bundle.

rBundleTimerTask

`private nippu.reception.RTimerTask` **`rBundleTimerTask`**

The task of forwarding the bundle to the scheduler.

rBundle

`private nippu.Bundle` **`rBundle`**

The bundle.

rTotalWaitTime

`private long` **`rTotalWaitTime`**

The total amount of current wait time.

Constructors

RBundle

```
public RBundle(Bundle bundle,
               long waitTime,
               Reception reception)
```

Creates a new waiting bundle from the given bundle. The wait time will be the given amount of milliseconds.

Parameters:

`bundle` -
the bundle to put waiting

(continued from last page)

`waitTime` -
the number of milliseconds to wait

Methods

getBundle

```
public Bundle getBundle()
```

getTimer

```
public java.util.Timer getTimer()
```

includeBundle

```
public void includeBundle(Bundle bundle,
                          int componentID,
                          int componentBundleCounter)
```

Includes the given bundle into the waiting bundle. This means resetting the waiting bundle's timer. The new timer interval will be half of what it was the last time. This method creates a new bundle and needs certain parameters in order to do that; the parameters componentID and componentBundleCounter are as in the method Bundle.createID().

Parameters:

`Bundle` -
otherBundle is Bundle to union with.
`componentID` -
is idnumber of component, which called this method.
`componentBundleCounter` -
is value of counter of Bundles. Every component has its own counter for created Bundles.

nippu.reception

Class Reception

```
java.lang.Object
   |
   +-nippu.reception.Reception
```

public class **Reception**
extends java.lang.Object

Fields

logger

`protected static org.apache.log4j.Logger` **`logger`**

waitTime

`long` **`waitTime`**

The wait time for a new waiting bundle in milliseconds. The value is half of the maximum service delay given by a user interface. The initial maximum service delay is 10 seconds.

reScProtocol

`private nippu.communications.ReceptionSchedulerProtocol` **`reScProtocol`**

Our connection to scheduler.

reAnProtocol

`private nippu.communications.ReceptionAnalyzerProtocol` **`reAnProtocol`**

Our connection to analyzer.

reReProtocol

`private nippu.communications.UserInterfaceReceptionProtocol` **`reReProtocol`**

Our connection to ourself.

nextListenerID

`private int` **`nextListenerID`**

The next reception listener ID to give.

listenerMap

`private java.util.HashMap` **`listenerMap`**

A map from reception listener IDs to connections to user interfaces.

(continued from last page)

RECEPTIONSERVERPORT

`private int RECEPTIONSERVERPORT`

Our port for incoming connections.

analyzerLocation

`private java.lang.String analyzerLocation`

The location of the analyzer.

analyzerPort

`private int analyzerPort`

The analyzer's port for clients.

schedulerLocation

`private java.lang.String schedulerLocation`

The location of the scheduler.

schedulerPort

`private int schedulerPort`

The scheduler's port for external clients like us.

sendSingle

`private boolean sendSingle`

If true, then print jobs are sent one in a bundle. Otherwise, print jobs are sent as they are bundled in reception.

waitingBundles

`private java.util.Vector waitingBundles`

The waiting bundles. Contains RBundle instances.

answersToCome

`private int answersToCome`

The number of answers to our cost queries yet to receive from analyzer. One answer is either the cost of a waiting bundle, or the combined cost of a waiting bundle and the incoming bundle.

analyzerError

`private boolean analyzerError`

True iff an error has occured in analyzer communications.

incomingBundleCost

`private float incomingBundleCost`

The cost of the incoming bundle.

waitingBundleCost

`private float waitingBundleCost`

(continued from last page)

The cost of a waiting bundle.

incomingBundleID

`private int` **`incomingBundleID`**

The ID of the incoming bundle.

waitingBundle

`private nippu.reception.RBundle` **`waitingBundle`**

The waiting bundle currently in inspection for union with the incoming bundle.

bestWaitingBundle

`private nippu.reception.RBundle` **`bestWaitingBundle`**

The best candidate from our waiting bundles for a union with the incoming bundle.

bestCostChange

`private float` **`bestCostChange`**

The cost change with the best bundle union candidate.

sentBundle

`private nippu.Bundle` **`sentBundle`**

The sent bundle that will shortly receive an ACK or a NACK.

numberOfNewBundles

`private int` **`numberOfNewBundles`**

The number of bundles created in the reception.

analyzerAnswerLock

`private java.lang.Object` **`analyzerAnswerLock`**

Synchronizing lock for receiving answers from analyzer.

waitingBundlesLock

`private java.lang.Object` **`waitingBundlesLock`**

Synchronizing lock for our waiting bundles.

waitTimeLock

`private java.lang.Object` **`waitTimeLock`**

Synchronizing lock for our waitTime field.

sentBundleLock

`private java.lang.Object` **`sentBundleLock`**

Synchronizing lock for our sentBundle field.

Constructors

(continued from last page)

Reception

```
public Reception()
```

The default constructor for the Reception

Methods

costAnswer

```
public void costAnswer(int bundleID,
                       float cost)
```

Writes down an answer to an earlier cost query. This method is called by communications.

---

combinedCostAnswer

```
public void combinedCostAnswer(int bundle1ID,
                               int bundle2ID,
                               float cost)
```

Writes down an answer to an earlier cost query. This method is called by communications.

---

analyzerError

```
public void analyzerError()
```

Reports an error in communications with analyzer. This method is called by communications.

---

addJob

```
public boolean addJob(Bundle newJob)
```

Parameters:

`newJob` -
The new job in a Bundle that is to be added to the reception The method that is used in adding new jobs to the reception. This method is called by communications.

---

newBundleReceived

```
public void newBundleReceived(int returnValue)
```

Called when an acknowledgement of an added bundle is received. This method is called by communications.

Parameters:

`returnValue` -
a return value from the add functionality, meaning undefined

---

newBundleRejected

```
public void newBundleRejected(int failReason)
```

Called when a negative acknowledgement of an added bundle is received. This means that the bundle was rejected. The reason of failure may be one of the following: 2 = service unavailable, 3 = too many jobs, 5 = bad job, 15 = unknown failure. This method is called by communications.

Parameters:

`failReason` -
the reason of failure

(continued from last page)

setMaxServiceDelay

`public void **setMaxServiceDelay**(long newMaxServiceDelay)`

Sets the maximum service delay. This method is called by communications.

Parameters:

`newMaxServiceDelay -`
the maximum service delay in milliseconds

---

sendWaitingBundle

`public void **sendWaitingBundle**(RBundle waitingBundle)`

Sends the given waiting bundle to the scheduler. The bundle is removed from the internal container of waiting bundles. The given bundle should already be contained in the reception.

---

getNumberOfJobs

`public int **getNumberOfJobs**()`

Returns the number of jobs in the reception. Not thread-safe.

---

getNumberOfBundles

`public int **getNumberOfBundles**()`

Returns the number of bundles in the reception. Not thread-safe.

---

toString

`public java.lang.String **toString**()`

Not thread-safe.

See Also:

java.lang.Object.toString()

---

register

`public void **register**(ReceptionUserInterfaceProtocol protocol)`

Registers a new user interface to the reception.

Parameters:

`protocol -`
the connection to the user interface

---

unregister

`public void **unregister**(int listenerID)`

Unregisters a user interface from the reception.

Parameters:

`listenerID -`
the ID of the user interface

---

initializeCommunicationsServer

`public void **initializeCommunicationsServer**()`

Sets up everything related to communications of the reception as a server.

initializeCommunicationsClient

`public void` **`initializeCommunicationsClient`**`()`

Sets up everything related to communications of the reception as a client.

connectToScheduler

`public boolean` **`connectToScheduler`**`()`

Makes a connection to scheduler. Returns true iff the connection failed.

connectToAnalyzer

`public void` **`connectToAnalyzer`**`()`

Makes a connection to analyzer.

connectToSelf

`public void` **`connectToSelf`**`()`

Makes a connection to ourself.

getWaitingBundle

`public` RBundle **`getWaitingBundle`**`(int bundleID)`

Returns the waiting bundle consisting of the bundle with the given ID, or null if such a waiting bundle cannot be found. Not thread-safe.

getFailReasonString

`private java.lang.String` **`getFailReasonString`**`(int failReason)`

Returns a short, textual, human-understandable of a job addition rejection reason returned by the scheduler.

main

`public static void` **`main`**`(java.lang.String args)`

Main method for running a reception instance.

nippu.reception

Class RTimerTask

```
java.lang.Object
   |
   +-java.util.TimerTask
         |
         +-nippu.reception.RTimerTask
```

public class **RTimerTask**
extends java.util.TimerTask

Reception's timed task of sending a waiting bundle forward to the scheduler.

Fields

parentRBundle

```
private nippu.reception.RBundle parentRBundle
```

The bundle that is waiting.

reception

```
private nippu.reception.Reception reception
```

The reception sending the bundle forward.

Constructors

RTimerTask

```
public RTimerTask(RBundle parent,
                  Reception reception)
```

Creates a waiting bundle sending timer task.

Parameters:

`parent` -
the waiting bundle
`reception` -
the reception that will send the waiting bundle forward

Methods

run

```
public void run()
```

The run method

Index

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z