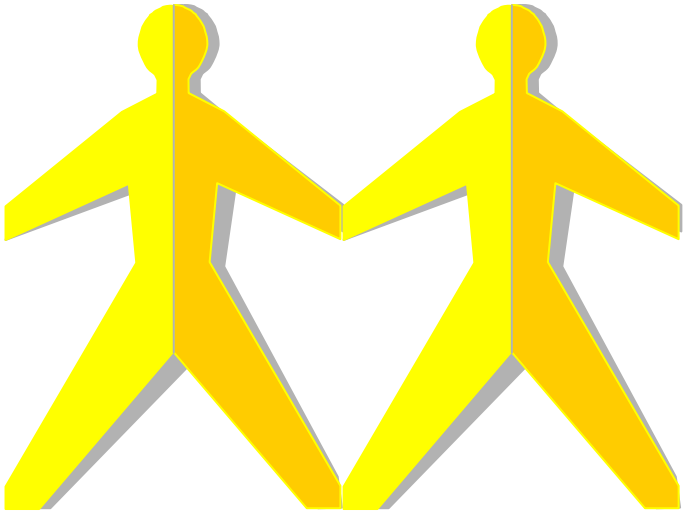


rvs® Client/Server

rvs® Client API 2

Benutzerhandbuch

Version 1.0

rvs® Client Server - rvs® Client API 2

Benutzerhandbuch

Version 1.0

Pascalstraße 11

10587 Berlin

Inhaltsverzeichnis

1 Einleitung

Dieses Kapitel gibt Ihnen einen Überblick über das rvs® Client API-Benutzerhandbuch. Es erklärt, welche Darstellungskonventionen verwendet werden und für welche Zielgruppe das Handbuch gedacht ist.

1.1 Aufbau des rvs® Client API-Benutzerhandbuchs

Das rvs® Client API-Benutzerhandbuch enthält die Beschreibung der Client Java-Programmierschnittstelle, Version 2. (rvs® Client API 2).

Im Unterschied zur Version Version 1 werden in der rvs® Client API Version 2 keine generischen Strukturen mehr verwendet (siehe mitgelieferte JavaDoc-Dateien für die Übersicht der neuen Strukturen).

Hinweis: Der Einfachheit halber wird in diesem Dokument die rvs® Client API 2 verkürzt als rvs® Client API bezeichnet.

Das vorliegende Handbuch enthält zuerst eine Installationsbeschreibung der rvs® Client API. Danach folgt eine ausführliche Beschreibung der Schnittstelle mit abschliessend vorgestellten Programmierbeispielen.

In diesem Benutzerhandbuch werden nicht die Grundlagen von rvs® oder rvs® Client/Server beschrieben. Hier wird auf die rvs® Handbücher und vor allem das "rvs® Client/Server Benutzerhandbuch" verwiesen.

1.2 Repräsentationsmittel

Dieser Abschnitt enthält die Beschreibung welche Darstellungskonventionen in diesem Handbuch verwendet werden und welche Bedeutung besonders gekennzeichnete Ausdrücke haben.

Darstellungskonventionen

"Hochkommata"	Verweise auf andere Handbücher, Kapitel und Abschnitte, Literatur
`Courier`	Sourcecode, Klassen- und Methodenbezeichner

fett wichtige Begriffe

1.3 Zielgruppe

Dieses Handbuch ist für Menschen gedacht, die mit der rvs® Client API arbeiten und hiermit Anwendungsprogramme schreiben werden, welche auf die rvs® Client API zugreifen und benutzen.

Folgende Fähigkeiten sind erforderlich, um die rvs® Client API nutzen zu können:

- gute Java-Kenntnisse
- Kenntnis der Grundfunktionen von rvs®
- Kenntnis der Grundfunktionen von rvs® Client/Server

Neben diesem Handbuch ist es vorteilhaft, die mitgelieferte HTML-Dokumentation (JavaDoc) zur Verfügung zu haben, sowie die im Source-Code gelieferten Beispiele.

Wir empfehlen dieses Handbuch zu lesen, bevor Sie anfangen mit der rvs® Client API zu arbeiten.

2 Systemüberblick

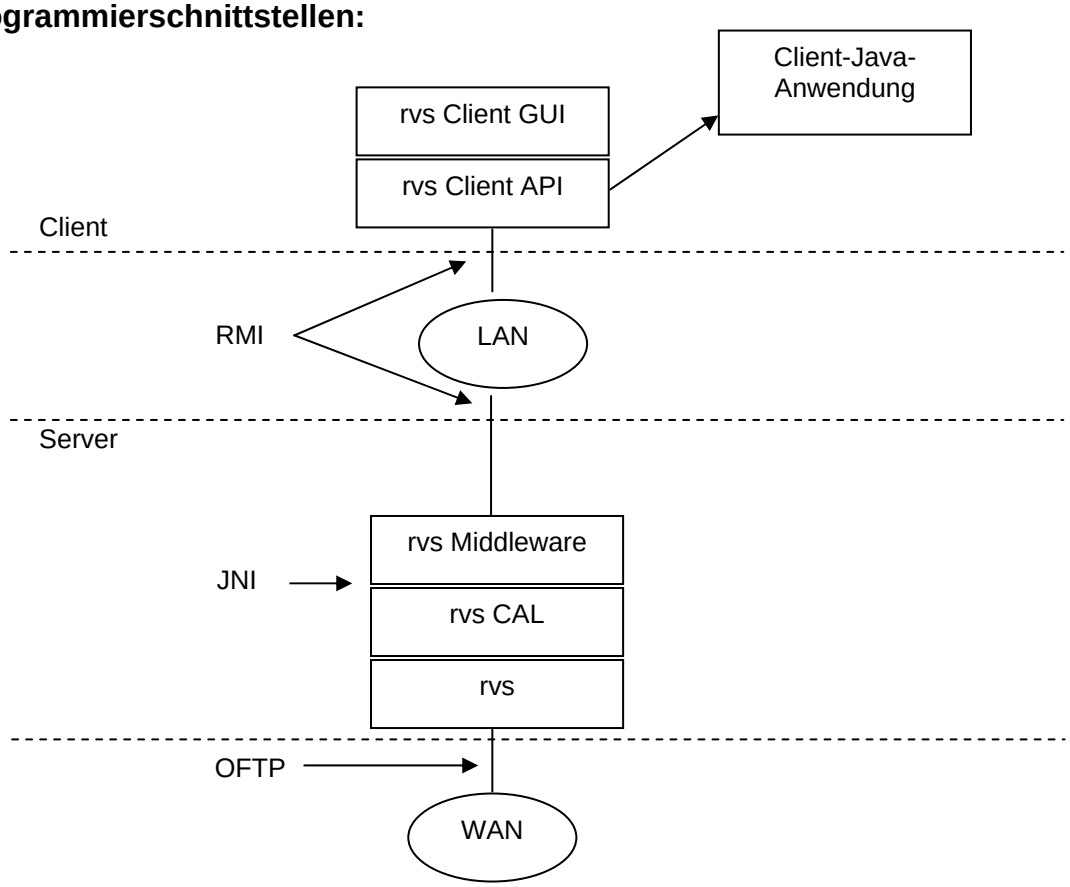
Die Abkürzung rvs® steht für die Bezeichnung Rechner-Verbund-System. Das rvs® Rechner-Kommunikations-System ist ein Basisdienst für den elektronischen Datenaustausch und die Integration von Business Applikationen.

Dateien werden von rvs® mit dem Odette File Transfer Protocol (OFTP) übertragen. rvs® auf den UNIX- und Windows-Plattformen bietet eine C-Programmierschnittstelle an (C-API). Auf dieser C-API setzt rvs® Client/Server auf.

Das in Java realisierte rvs® Client/Server-Paket erweitert die Stand-Alone-Applikation rvs® um eine Client/Server-Lösung. Die Kommunikation zwischen Client und Server findet über RMI statt. Clientseitig wird neben einer Swing-basierten GUI auch eine Java-Programmierschnittstelle angeboten, die rvs® Client API.

Die serverseitige Java-Programmierschnittstelle wird als rvs® J-CAL bezeichnet. Sie ist aber nicht Thema des vorliegenden Handbuchs.

Diagramm: Aufbau rvs® Client/Server und Programmierschnittstellen:



3 Installation

In diesem Kapitel werden die Systemvoraussetzungen für die Installation und den Betrieb der rvs® Client API von rvs® Client/Server und der Installationsvorgang selbst beschrieben.

3.1 Systemvoraussetzungen Installation

Für die Installation selbst existieren, außer dem benötigten Plattenplatz, keine Systemvoraussetzungen. Die rvs® Client API ist Bestandteil der rvs® Client/Server-Distribution.

3.2 Systemvoraussetzungen: Betrieb

Für einen erfolgreichen Betrieb der rvs® Client API benötigen Sie folgende Software:

- Unix: JRE Version 1.4.1 oder höher, installiert auf jedem rvs® Client.
- Unix: JRE (Java Runtime Enviroment) Version 1.4.1 oder höher, für die Middleware, installiert auch auf dem rvs® Server.
- Windows: Installation von JRE ist nicht notwendig, da sie vom Setup-Programm mitgebracht wird.
- Zum rvs® Client/Server passende rvs® Version, installiert auf dem rvs® Server. Lesen Sie bitte die Datei `readme.txt` für mehr Informationen.
- rvs® Middleware (Serverteil von rvs® Client/Server), installiert auf dem rvs® Server.

Die rvs® Middleware (Server) von rvs® Client/Server steht auf den Plattformen Windows NT, XP und 2000 und etlichen UNIX-Derivaten zur Verfügung. Eine vollständige Liste der unterstützten Plattformen finden Sie im "rvs® Client/Server-Benutzerhandbuch" oder auf der Internet-Seite http://rvs.info.gedas.de.

Die rvs® Client API ist auf allen Plattformen lauffähig, auf denen JRE Version 1.4.1 installiert ist.

Die Installation von rvs® ist im Benutzerhandbuch von rvs® beschreiben, die Installation der rvs® Middleware im rvs® Client Server-Benutzerhandbuch.

3.3 Installationsvorgang

Die rvs® Client API in der rvs® Client/Server-Distribution besteht aus den folgenden Dateien:

`rvsClientAPI2.zip`	Zip-Datei, in der die Verzeichnisse `com` (Source-Code der Beispielprogramme) und `doc` (HTML-Dokumentation der rvs® Client API (JavaDoc)) enthalten sind.
`rvsClient.jar`	rvs® Client API-Software (Klassen und Schnittstellen)
`rvsClientServer_ClientAPI2_Doc10de.doc`	rvs® Client API-Benutzerhandbuch (das vorliegende Dokument)

Entpacken sie in einem Verzeichnis die ausgelieferte Zip-Datei.

Anwendungen, welche die rvs® Client API benutzen wollen, müssen die Datei `rvsClient.jar` in den CLASSPATH mit aufnehmen. Die Benutzung von `rvsClient.jar` setzt die Verwendung von JDK ab Version 1.4.1 voraus.

Die mitgelieferten Beispiele demonstrieren die Verwendung der rvs® Client API und sind aufgrund der implementierten main-Methode als eigenständige Programme auf der Konsole lauffähig.

4 Clientseitige Programmierschnittstelle

In diesem Kapitel wird die clientseitige Programmierschnittstelle (rvs® Client API 2) von rvs® Client/Server vorgestellt. rvs® Client API 2 enthält keine generischen Strukturen (wie rvs® Client API 1), sondern nur konkrete Datenstrukturen. Für den Anwender bedeutet dies, dass die Transformation der generischen Strukturen in konkrete entfällt und dadurch die clientseitige Anwendungsprogrammierung erleichtert wird. Der weitere Vorteil ist, dass Programmierfehler nicht erst zur Laufzeit, sondern bereits zur Entwicklungszeit (Kompilierung) erkannt werden.

Die aktuelle Version der rvs® Client API unterstützt:

- Stations-, Benutzer- Job-, Jobstart und Parameterverwaltung
- Anzeigen von Monitor- und Statistic-Log-Datei und
- Starten/Stoppen von rvs® .

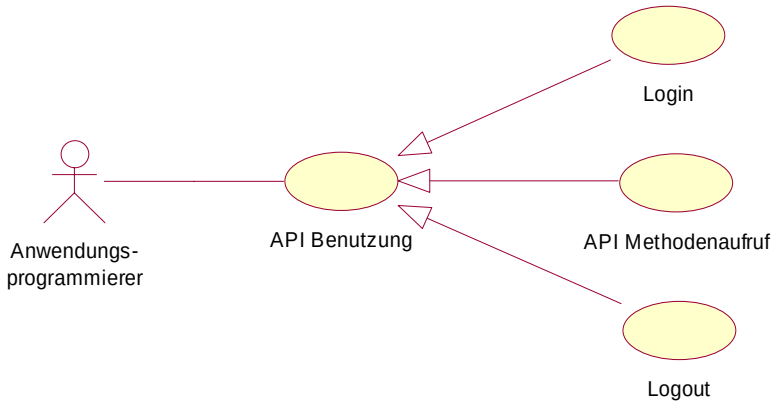
Vor der Beschreibung der Programmierschnittstelle wird auf den Aufbau und die verwendete Datenstruktur eingegangen.

4.1 Aufbau der Programmierschnittstelle

In diesem Kapitel finden Sie einen Überblick über die Strukturen von rvs® Client API. Zuerst wird gezeigt, wie ein praktischer Anwendungsfall der rvs® Client API aussieht. Danach wird die grundsätzliche Unterteilung in Daten und Serverfunktionalität erläutert. Anschließend werden die konkreten Pakete aus der ausgelieferte Datei `rvsClient.jar` aufgezählt.

4.1.1 Anwendungsfälle der rvs® Client API

In diesem Kapitel wird ein kurzer Überblick über die Anwendungsfälle (Benutzung) der rvs® Client API geboten. Folgende Abbildung soll das Muster der Benutzung veranschaulichen:



Beschreibung der Anwendungsfälle:

API-Benutzung

Übergeordneter Anwendungsfall für die untergeordneten Anwendungsfälle Login, API-Methodenaufruf und Logout.

Login

Das Login authentifiziert den Benutzer gegenüber der rvs® Middleware und liefert bei Erfolg ein Objekt der Klasse `UserProfile`. Ein erfolgreiches Login ist Voraussetzung für den Aufruf der Remote-Methoden der rvs® Middleware, da alle Remote-Methoden die Übergabe eines `UserProfile`-Objektes verlangen.

API Methodenaufruf

Mögliche Anwendungsfälle: Stations-, Benutzer-, Job-, Jobstarts- und Parameterverwaltung; Starten und Stopen von rvs®, Anzeigen von Monitor- und Statistic-Log-Datei. Nach erfolgreichem Login können die API-Methoden benutzt werden.

Auf den Umfang und die Verwendung der API-Methoden für die Stations-, Benutzer-, Job-, Jobstart und Parameterverwaltung wird im Einzelnen in den Kapiteln 4.3, 4.4, 4.5 und 4.6 eingegangen.

Logout

Das Logout beendet eine Client-Session in der rvs® Middleware. Dabei werden Client-bezogene Ressourcen in der rvs® Middleware freigegeben. Zur Entlastung der rvs® Middleware und des Servers sollte daher immer ein Logout erfolgen.

4.1.2 Datei `rvsClient.jar`

Die ausgelieferte Datei `rvsClient.jar` (im rvsClient-Verzeichnis `classes`) enthält die benötigten Java-Pakete der Client API. Diese umfassen Pakete der rvs® Client API, die dem Anwendungsprogrammierer als Schnittstelle angeboten werden und Pakete mit Klassen, die für die Funktionstüchtigkeit der Schnittstelle selbst benötigt werden (z.B. Ausnahme-Behandlungen).

Folgende Pakete aus der `rvsClient.jar`-Datei gehören zur rvs® Client API:

`com.gedas.rvs`	Ausnahmen (Exceptions) und Benachrichtigungsklassen
`com.gedas.rvs.client.api`	Klassen für den Zugriff auf die Funktionalitäten (Business-Logik) der rvs® Middleware
`com.gedas.rvs.client.api.tools`	Beispiele für die Benutzung der rvs® Client API
`com.gedas.rvs.data.config`	Datenklassen für Jobstart- und Parameterverwaltung
`com.gedas.rvs.data.user`	Datenklassen für Benutzerverwaltung
`com.gedas.rvs.data.station`	Datenklassen für Stationsverwaltung
`com.gedas.rvs.data.net`	Datenklassen für Netzwerk
`com.gedas.rvs.data.job`	Datenklassen für Jobverwaltung
`com.gedas.rvs.data.log`	Wird nicht benutzt
`com.gedas.rvs.data.query`	Datenklassen für Abfragen

4.1.3 Unterscheidung Datenklassen/Serverfunktionalität

In diesem Kapitel möchten wir die Begriffe, die im Handbuch verwendet werden und mit der Struktur der rvs® Client API zusammenhängen, näher erläutern.

Die zwei wichtigsten Begriffe sind Serverfunktionalität und Datenklassen.

Unter Serverfunktionalität sind die Klassen aus dem Paket `com.gedas.rvs.client.api` zu verstehen, die den Zugriff auf die Funktionalität der rvs® Middleware ermöglichen. Die Serverfunktionalität benötigt als Parameter-Rückgabewerte die Datenklassen.

Die Datenklassen befinden sich in den Unterpaketen von `com.gedas.rvs.data`. Sie sind gegliedert in die Bereiche: Stationen (`station`), Netzwerk (`net`), Benutzer (`user`), Jobs (`job`), Jobstarts und Parameter (`config`), Logs (`log`) und Abfragen (`query`). Objekte dieser Klassen werden von den Methoden aus `com.gedas.rvs.client.api` geliefert bzw. erwartet.

Die Serverfunktionalität ihrerseits unterteilt sich in die Bereiche Basisfunktionalität und spezialisierte Funktionalität.

Unter Basisfunktionalität sind Login, Logout, Starten und Stoppen von rvs® sowie Zugriff auf die spezialisierten Bereiche Stationsverwaltung, Benutzerverwaltung, Jobverwaltung, Jobstart- und Parameterverwaltung zu verstehen.

Unter spezialisierter Funktionalität sind Stations-, Benutzer-, Job-, Jobstarts und Parameterverwaltung zu verstehen.

Einige der wichtigen Klassen der Serverfunktionalität aus dem Paket `com.gedas.rvs.client.api` sind: `Rvs` und `UserProfile`.

Die Klasse `Rvs` beinhaltet Methoden für den Zugriff auf Basisfunktionalität von rvs®.

Die Klasse `UserProfile` beschreibt die Zugriffsrechte auf rvs® Server. Sie entspricht der SessionID und wird bei jedem erfolgreichen Login erzeugt. Sie ist notwendig für den Zugriff auf jede Serverfunktionalität.

In den folgenden Kapiteln werden zuerst die Basisfunktionalität und danach spezialisierte Bereiche gemeinsam (Datenklassen; Serverfunktionalität und entsprechende Beispiele) vorgestellt

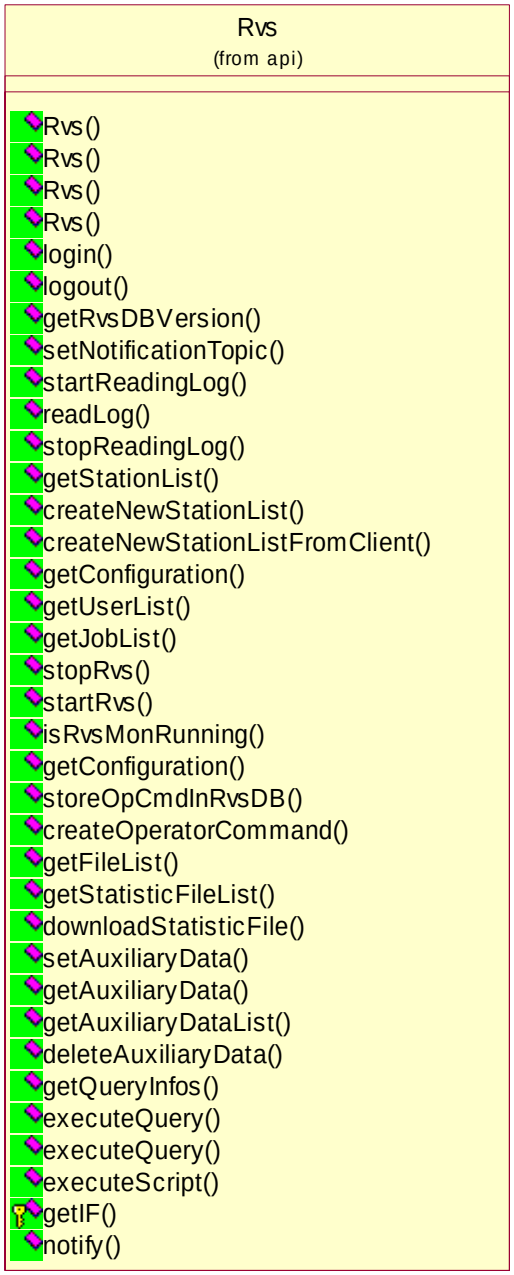
4.2 Basisfunktionalität

Die Basisfunktionalität ermöglicht das Ein- und Ausloggen eines Clients auf der rvs® Middleware sowie den Zugriff auf zentrale Serverfunktionen wie Starten und Stoppen von rvs® und Zugriff auf Konfigurationsinformationen.

Darüber hinaus stellt sie den Zugriff auf die spezialisierten Bereiche Stations-, Benutzer-, Job-, Jobstart- und Parameterverwaltung zur Verfügung.

Die Methoden der Basisfunktionalität sind in der Klasse `Rvs` im Paket `com.gedas.rvs.client.api` zu finden.

Diagramm der Klasse Rvs:



4.2.1 Überblick

Im Folgenden werden ausgewählte Bereiche der Basisfunktionalität vorgestellt, die für den Zugriff auf die Stations-, Benutzer-, Job-, Jobstarts und Parameterverwaltung wichtig sind.

Konstruktor

Es existiert ein Konstruktor mit Initialisierung wichtiger Verbindungsparameter. Der Konstruktor löst die angegebene rvs® Middleware beim RMI-Namensdienst auf.

Definition:

```
Rvs(  String hostName,
      String rvsName,
      RvsNotificationCallback brokenConnectionListener)
```

Parameter:

`hostName`	Hostname der rvs Middleware
`rvsName`	Name der laufenden rvs Middlewareinstanz
`brokenConnectionListener`	Callback für Benachrichtigungen über Verbindungsabbruch (optional)

Ein- und Auslogfunktionalität

Die Ein- und Auslogfunktionalität stellt das Einloggen und Ausloggen zur Verfügung.

Definitionen:

```
UserProfile login(String login,
                  String password,
                  RvsNotificationCallback notifyReference)
logout(UserProfile user)
```

Parameter `login`:

`login`	Loginname des Benutzers
`password`	Passwort des Benutzers
`notifyReference`	Nur für interne Zwecke, der Benutzer soll es auf „null“ setzen.

Rückgabewert `login`:

`UserProfile`	Sie entspricht einer SessionID und wird bei jedem erfolgreichen Login erzeugt, bei fehlerhaftem Login wird `RvsException` geworfen

Konfigurationsdaten und Datenbank-Version

Über `getConfiguration` können Konfigurationsinformationen (die Werte der vorhandenen Variablen) abgefragt werden, wie sie in der serverseitigen Datei `rvsenv.dat` gespeichert sind.

Die Methode `getRvsDBVersion` gibt die Version der verwendeten rvs® Datenbank zurück.

Definitionen:

```
String getConfiguration(UserProfile user, String name)
String getRvsDBVersion(UserProfile user)
```

Parameter `getConfiguration` und `getRvsDBVersion`:

`user`	SessionID des Benutzers
`name`	Name der Variablen aus der Datei `rvsenv.dat`, deren Wert abgefragt wird z.B. PATH, DB, SERVER,…

Rückgabewert `getConfiguration`:

`String`	Wert der Variablen aus der rvs®-Umgebungsdatei `rvsenv.dat`.

Rückgabewert `getRvsDBVersion`:

`String`	der Datenbank-Versions-String

Hinweis: Es existiert noch eine Methode `getConfiguration` mit einem anderen Ergebnistyp (`Configuration` statt `String`), die den Zugriff auf die Konfigurationsinformationen für die Jobstart- und Parameterverwaltung ermöglicht. Siehe letzten Abschnitt dieses Kapitels.

Zugriff auf die Stationsverwaltung

Über die Zugriffsfunktionen auf die Stationsverwaltung wird ein Objekt zurückgeliefert, über das die Stationsverwaltung verfügbar wird. Mögliche Operationen sind: eine Stationsliste holen

(`getStationList`), alte Stationsliste in rvs® löschen und eine neue erzeugen (`createNewStationList` oder `createNewStationListFromClient`).

Definition:

```
StationList getStationList (UserProfile user)
```

Rückgabewert `getStationList`:

`StationList`	das Objekt über das die Funktionalität Stationsverwaltung zur Verfügung steht

Definition:

```
StationList createNewStationList (String filename,
      UserProfile user)
```

Parameter `createNewStationList`:

`filename`	Dateiname der Stationsliste z.B. `rdstat.dat`
`user`	SessionID des Benutzers

Rückgabewert `createNewStationList`:

`StationList`	das Objekt über das die Funktionalität Stationsverwaltung zur Verfügung steht

Diese Methode löscht die alte Stationsliste in rvs® und erstellt eine neue aus der Datei `filename`.

Definition:

```
StationList createNewStationListfromClient
      (FileReaderServer filereader, UserProfile user)
```

Parameter `createNewStationListfromClient`:

`filereader`	Dateiname der clientseitige Stationsliste
`user`	SessionID des Benutzers

Rückgabewert `createNewStationListfromClient`:

`StationList`	das Objekt über das die Funktionalität Stationsverwaltung zur Verfügung steht

Diese Methode löscht auch die alte Stationsliste in rvs®. Als Input-Datei für die neue Stationsliste dient die clientseitige Stationsliste.

Zugriff auf die Benutzerverwaltung

Über diese Zugriffsfunktion wird ein Objekt zurückgeliefert, über das die Benutzerverwaltung verfügbar wird.

Definition:

```
UserList getUserList (UserProfile user)
```

Parameter `getUserList`:

`user`	SessionID des Benutzers

Rückgabewert `getUserList:`

`UserList`	das Objekt, über das die Funktionalität Benutzerverwaltung zur Verfügung steht

Zugriff auf die Jobverwaltung

Über diese Zugriffsfunktion wird ein Objekt zurückgeliefert, über das die Jobverwaltung verfügbar wird.

Definition:

```
JobList getJobList (UserProfile user)
```

Parameter `getJobList`:

`user`	SessionID des Benutzers

Rückgabewert `getJobList:`

`JobList`	das Objekt, über das die Funktionalität Benutzerverwaltung zur Verfügung steht

Zugriff auf die Jobstart- und Parameterverwaltung

Die Methode `getConfiguration` ermöglicht den Zugriff auf die Konfigurationsinformationen für die Jobstart- und Parameterverwaltung.

Definitionen:

```
Configuration getConfiguration(UserProfile user)
```

Parameter `getConfiguration`:

`user`	SessionID des Benutzers

Rückgabewert `getConfiguration`:

`Configuration`	Das Objekt, über das die Funktionalität Jobstart- und Parameterverwaltung zur Verfügung steht.

4.2.2 Wie starte ich den rvs®-Server?

In diesem Kapitel wird als Beispiel das Programm zum Starten des rvs®-Servers vorgestellt.

Beispiel (`StartServer.java`):

```
package com.gedas.rvs.client.api.tools;


import com.gedas.rvs.client.api.Rvs;
import com.gedas.rvs.client.api.UserProfile;


public class StartServer
{
    // runtime data
    Rvs rvs = null;
    UserProfile user = null;
    Login loginHelper = null;
    String jobnumber = null;

    // configuration data (set once)
    int mode = 0;
    boolean assumeRMIshutdown = false;
    boolean verbose = false;
    boolean writeLog = false;


       public void work() throws Exception
       {
        if (verbose) System.out.println("WORK");

        try
        {
            rvs.startRvs(mode,user);
        }
        catch (Exception e)
        {
            if (!assumeRMIshutdown)
            {
                throw e;
            }
        }

       if (verbose) System.out.println("Send start command to server. ");

       if (verbose) System.out.println("Finished. ");
        }



    /**
     * Performs the login.
     */
    public void login()
    {
        //Login
        try
        {
            // establish connection to rvs middleware (rvs server)
            loginHelper = new Login(verbose);
            rvs = loginHelper.login();
            user = loginHelper.getUserProfile();

            if (rvs==null)
            {
                System.err.println("ERROR: Connection failed.");
                if (verbose) System.out.println("** ABORT **");
                System.exit(-1);
            }
```

```
            else
            {
                if (verbose) System.out.println("Connected. ");
            }
        }
        catch (Exception e)
        {
            System.err.println("Exception during login: "+e);
            e.printStackTrace();
            if (verbose) System.out.println("** ABORT **");
            System.exit(-1);
        }
    }

    /**
     * Performs the logout.
     */
    public void logout()
    {
        //Logout
        if (!assumeRMIshutdown)
        {
            try
            {
                loginHelper.logout();
            }
            catch (Exception e)
            {
                    System.err.println("Exception during logout: "+e);
                    e.printStackTrace();
                    if (verbose) System.out.println("** ABORT **");
                    System.exit(0);
            }
        }
    }

       /**
        * Reads the command line arguments into the data structures.
        */
       private boolean readArguments(String[] args)
       {
           if (verbose) System.out.println("** READ ARGUMENTS **");

        int i=0;


          // read parameters
         while (i<args.length)
         {

            if (args[i].compareTo("-verbose")==0)
                {
                    // set verbose output mode
                    verbose = true;
                    i++;
                }
                else       if       (args[i].compareTo("-?")==0       ||
args[i].compareTo("/?")==0)
                {
                    // call usage
                    printUsage();
                    return false;
                }
                else       if       (args[i].compareTo("-help")==0       ||
args[i].compareTo("-h")==0)
                {
                    // call help
                    printHelp();
                    return false;
                }
                else // default with abort
                {
                    System.err.println("Error:  parameter  "+args[i]+"  not
supported!");
                    printUsage();
```

```
                    throw new IllegalArgumentException("Error: parameter
"+args[i]+" not supported!");
                }
            }
            if (verbose) System.out.println("** READ ARGUMENTS finished
**");

            return true;
        }



        /**
         * Prints the help on system out.
         */
        static private void printHelp()
        {
            System.out.println("");
            System.out.println("Starts the server.");
            System.out.println("Using rvs Middleware: rvs Monitor will be
started.");
            System.out.println("");
               printUsage();
        }

        /**
         * Prints the usage on system out.
         */
        static private void printUsage()
        {
            System.out.println("");
            System.out.println("Usage: StartServer");
            System.out.println("");
            System.out.println("\tParameter :");
            //System.out.println("\t-m  <mode>:  sets  the  shutdown  mode
(integer value, default=0)");
            //System.out.println("\t-r           :  set  RMI  shutdown  (RMI
errors will be ignored after starting)");
            System.out.println("\t-verbose : verbose output");
            System.out.println("\t-help    : prints help");
            System.out.println("\t-?       : prints usage");
            System.out.println("\t");
        }



        public static void main(String args[])
        {
               try
               {
             StartServer gj = new StartServer();

             if (gj.readArguments(args))
             {
                 gj.login();
                 gj.work();
                 gj.logout();
             }
               }
               catch (Exception e)
               {
                      System.out.println("Cought Exception:"+e);
                      e.printStackTrace();
               }
         finally
         {
             System.exit(0);
         }
        }
}
```

4.2.3 Wie zeige ich den Monitor-Log an?

Das folgende Beispiel zeigt Ihnen, wie man sich den Monitor-Log von rvs® anzeigen lassen kann.

Beispiel (`ShowMonitorLog.java`):

```
package com.gedas.rvs.client.api.tools;


import java.util.Vector;

import com.gedas.rvs.RvsException;
import com.gedas.rvs.RvsNotificationCallback;
import com.gedas.rvs.RvsNotificationTopic;
import com.gedas.rvs.client.api.Rvs;
import com.gedas.rvs.client.api.UserProfile;


public class ShowMonitorLog implements RvsNotificationCallback
{
    // runtime data
    Rvs rvs = null;
    UserProfile user = null;
    Login loginHelper = null;

    // configuration data (set once)
    boolean verbose = false;


        public void work() throws Exception
        {
        if (verbose) System.out.println("WORK");

        try
        {
            RvsNotificationTopic topic = new RvsNotificationTopic();
            topic.currentLogLines = true;
            rvs.setNotificationTopic(topic,user);
        }
        catch (Exception e)
        {
            throw e;
        }

        if  (verbose)  System.out.println("Work  finished  but  keeps  on
looping endlessly. ");

        boolean keepRunning = true;
        while(keepRunning)
        {
            try
            {
                Thread.sleep(1000);
            }
            catch (Exception e)
            {
                e.printStackTrace();
                keepRunning = false;
            }
        }
        }

//      public void init() throws Exception
//      {
//             config       =       new       PropertyResourceBundle(new
FileInputStream(configFilename));
//      }
```

```
    /**
     * Performs the login.
     */
    public void login()
    {
        //Login
        if (verbose) System.out.println("** LOGIN **");

        try
        {
           // establish connection to rvs middleware (rvs server)
            loginHelper = new Login(verbose);
            rvs = loginHelper.loginWithCallback(this);
            user = loginHelper.getUserProfile();

            if (rvs==null)
            {
                System.err.println("ERROR: Connection failed.");
                if (verbose) System.out.println("** ABORT **");
                System.exit(-1);
            }
            else
            {
                if (verbose) System.out.println("Connected. ");
            }
        }
        catch (Exception e)
        {
            System.err.println("Exception during login: "+e);
            e.printStackTrace();
            if (verbose) System.out.println("** ABORT **");
            System.exit(-1);
        }
    }

    /**
     * Performs the logout.
     */
    public void logout()
    {
        //Logout
        try
        {
            loginHelper.logout();
        }
        catch (Exception e)
        {
                System.err.println("Exception during logout: "+e);
                e.printStackTrace();
                if (verbose) System.out.println("** ABORT **");
                System.exit(0);
        }
    }

       /**
        * Reads the command line arguments into the data structures.
        */
       private boolean readArguments(String[] args)
       {
           if (verbose) System.out.println("** READ ARGUMENTS **");

        int i=0;


              // read parameters
           while (i<args.length)
           {
            if (args[i].compareTo("-verbose")==0)
                {
                    // set verbose output mode
                    verbose = true;
                    i++;
                }
                else        if        (args[i].compareTo("-?")==0        ||
args[i].compareTo("/?")==0)
                {
```

```
                    // call usage
                    printUsage();
                    return false;
                }
                else      if      (args[i].compareTo("-help")==0      ||
args[i].compareTo("-h")==0)
                {
                    // call help
                    printHelp();
                    return false;
                }
                else // default with abort
                {
                    System.err.println("Error:  parameter  "+args[i]+"  not
supported!");
                    printUsage();
                    throw  new  IllegalArgumentException("Error:  parameter
"+args[i]+" not supported!");
                }
            }
            if  (verbose)  System.out.println("**  READ  ARGUMENTS  finished
**");

            return true;
        }


        /**
         * Prints the help on system out.
         */
        static private void printHelp()
        {
            System.out.println("");
            System.out.println("Shows the current monitor log lines.");
            System.out.println("");
               printUsage();
        }

        /**
         * Prints the usage on system out.
         */
        static private void printUsage()
        {
            System.out.println("");
            System.out.println("Usage: ShowMonitorLog [-verbose]");
            System.out.println("");
            System.out.println("\tParameter :");
            System.out.println("\t-verbose : verbose output");
            System.out.println("\t-help    : prints help");
            System.out.println("\t-?       : prints usage");
            System.out.println("\t");
        }


    /**
     * Exists the process with the given return code.
     */
//    static private void exit(int returnCode)
//    {
//        if (verbose) System.out.println("** ABORT **");
//        System.exit(returnCode);
//    }

        public static void main(String args[])
        {
                try
                {
             ShowMonitorLog gj = new ShowMonitorLog();

             if (gj.readArguments(args))
             {
                 gj.login();
                 gj.work();
                 gj.logout();
             }
```

```
                }
                catch (Exception e)
                {
                        System.out.println("Cought Exception:"+e);
                        e.printStackTrace();
                }
            finally
            {
                System.exit(0);
            }
           }

    /**
     * The callback method that has to be implemented by the listener.
     *
     * @param reason    the reason code as defined above
     * @param param     the object sending the notification, or any other
     *                  parameter, as created by the notification sender
     */
    public void notify(int reason, Object param) throws RvsException
    {
        if   (verbose)   System.out.println("got   notification:"+reason+":
"+param);

        if (reason==RvsNotificationCallback.RVS_LOG_LINE_CREATED)
        {
            printLogLines((String[])param);
        }
        else if (reason==RvsNotificationCallback.RVS_EVENT_LIST)
        {
            Vector v = (Vector)param;

            for (int i=0;i<v.size();i++)
            {
                RvsNotificationCallback.Notification n
                    = (RvsNotificationCallback.Notification)
                        v.elementAt(i);

              if (n.reason==RvsNotificationCallback.RVS_LOG_LINE_CREATED)
                {

                    printLogLines((String[])n.parameter);
                }
              else
                {
                  if (verbose) System.out.println("element in event list
was no log line ("+n.reason+")=> gets ignored");
                }
            }
        }
        else
        {
            if (verbose) System.out.println("notification was no log line
=> gets ignored");
        }
    }

    private void printLogLines(String[] lines)
    {
        for (int i=0;i<lines.length;i++)
        {
            System.out.println(lines[i]);
        }
    }
}
```

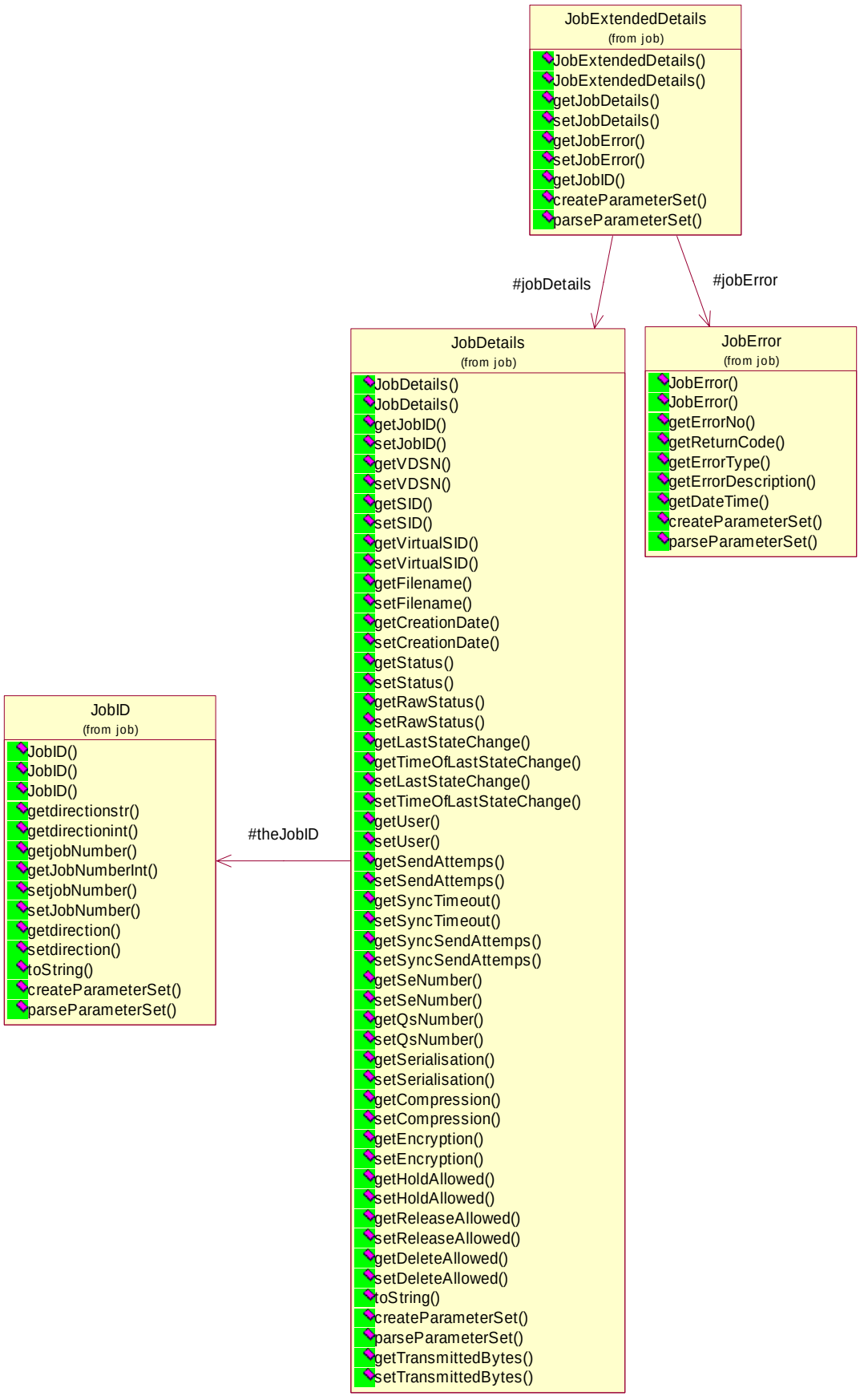
4.3 Jobverwaltung

In diesem Kapitel werden die Datenklassen für die Jobverwaltung, Methoden der Serverfunktionalität und anschließend die Programm-Beispiele (Erzeugen eines Sendeauftrags und Anzeigen der Jobliste) vorgestellt.

4.3.1 Datenklassen

In diesem Kapitel werden die wichtigsten Klassen aus dem Paket `com.gedas.rvs.data.job` vorgestellt.

Diagramm der Job-Klassen:



Die wichtigste Klasse in diesem Paket ist die Klasse `JobDetails`. Diese Klasse stellt die kompletten Daten (wie z.B. `JobID`, `Filename`, `VDSN`, `Virtual SID`, `SID`, `Status`, `Compression`,...) eines Sende- oder Empfangsjobs zur Verfügung. Diese Daten können Sie in der rvs® Client/Server-Oberfläche mit einem Doppelklick auf die gewünschte Jobzeile in der Übersichtstabelle anzeigen. Die Klasse `JobDetails` wird nicht benutzt, um einen neuen Job zu erzeugen, sondern um einen Job mit allen dazugehörigen Informationen darzustellen. Beispiel `CreateSendEntry` im Kapitel 4.3.3 zeigt Ihnen, wie man einen Sendejob erzeugen kann.

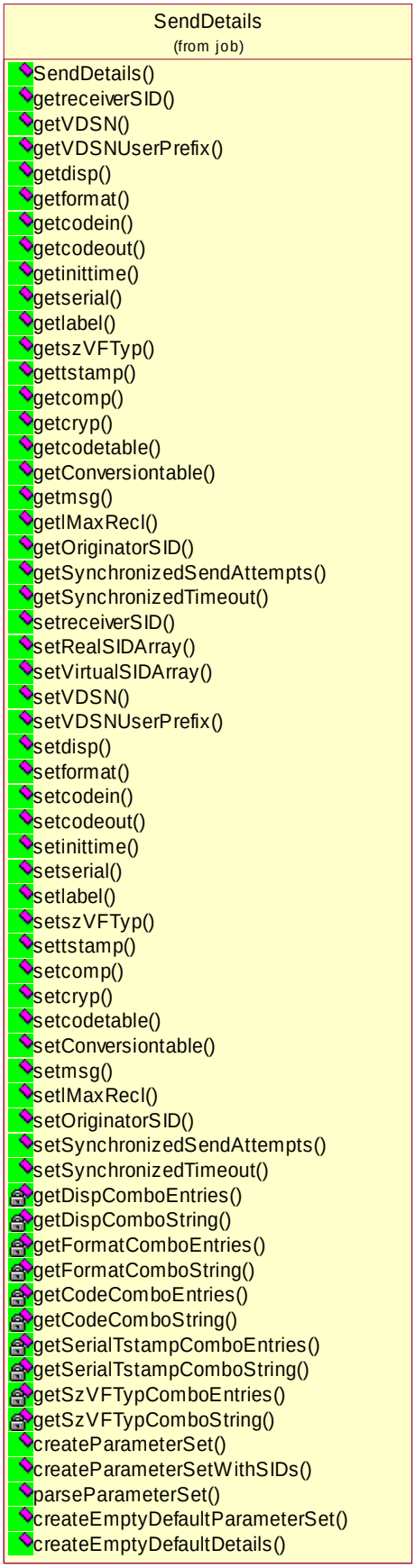
Die Basisinformationen (`JobID`, `VDSN`, `SID`, `Status`) zu einem Job sind analog wie bei Stationen in der Klasse `JobOverview` zu finden. Diese Informationen werden in der rvs® Client/Server-Oberfläche in der jeweiligen Jobzeile in der Übersichtstabelle abgebildet.

Die Klasse `InboxDetails` enthält die Informationen über die Dateien aus dem **Inbox**-Ordner. Folgende Informationen stehen zur Verfügung: Dateiname, Dateigröße und Datum. Diese Informationen werden Ihnen angezeigt, wenn Sie eine Datei aus dem Inbox-Ordner in der rvs® Client/Server-Oberfläche anklicken.

Die Klasse `OutboxDetails` enthält analog zur Klasse `Inbox-Details` die Informationen über die Dateien aus dem Outbox-Ordner.

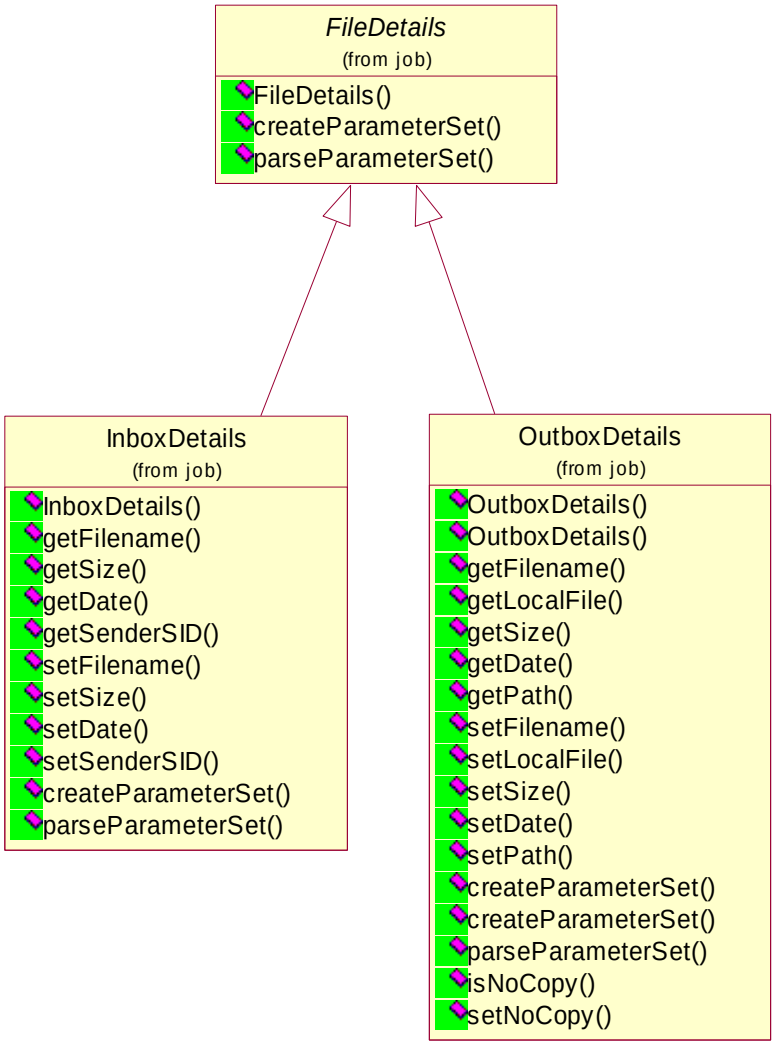
In der Klasse `SendDetails` sind die Informationen enthalten, die zusammen mit der Klasse `OutboxDetails` gebraucht werden, um einen Sendeauftrag zu erzeugen (siehe Beispiel aus Tools `CreateSendEntry`). Dies spiegelt sich in der rvs® Client/Server-Oberfläche im Fenster **Create New Transmission** wieder. Dieses Fenster erscheint, wenn Sie eine Datei im Outbox-Ordner anklicken.

Diagramm der Klasse `SendDetails`:



Die Klasse `FileDetails` ist die Basisklasse von den Klassen `InboxDetails` und `OutboxDetails`, die die Gemeinsamkeiten der beiden Klassen vereinigt. Diese Klasse wird dann verwendet, wenn es nicht klar ist, ob sich die gewünschte Datei im **Inbox**- oder **Outbox**-Ordner befindet.

Diagramm der Klasse `FileDetails`:

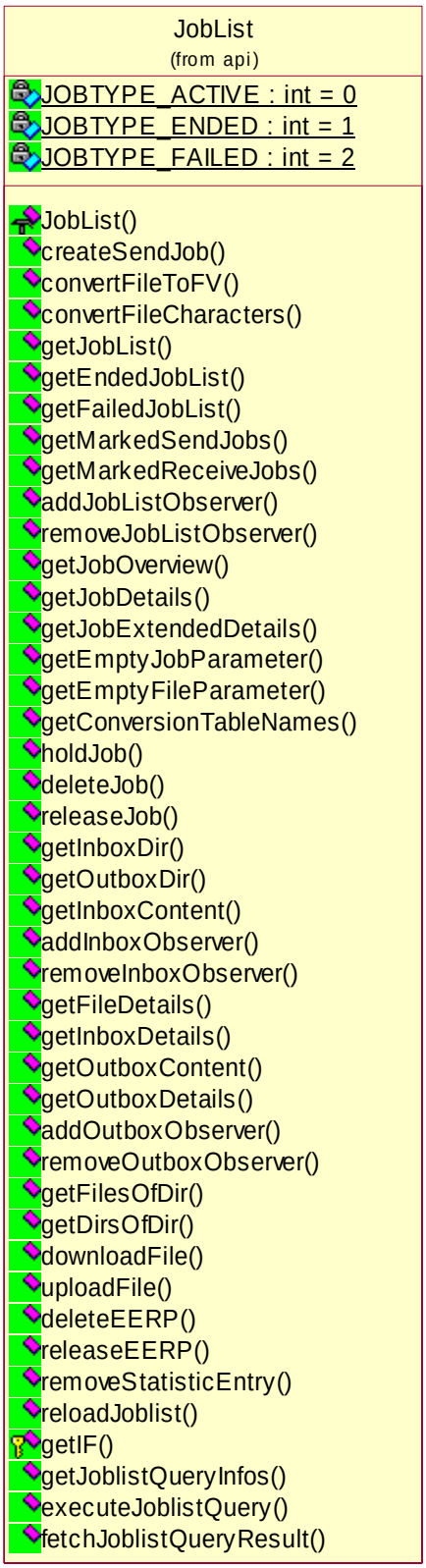


Die Klasse `JobError` enthält die Informationen über einen Jobfehler wie z.B. Fehlernummer, Return Code, Typ und Datum mit Uhrzeit. `JobExtendedDetails` beinhaltet die Informationen aus der Klasse `JobError` und der Klasse `JobDetails`.

4.3.2 Serverfunktionalität

In diesem Kapitel wird die Klasse `JobList` vorgestellt. Diese Klasse enthält die Methoden für die Serverfunktionatität der Jobverwaltung.

Diagramm der Klasse `JobList`:


JobList
(from api)
JOBTYPE_ACTIVE : int = 0
JOBTYPE_ENDED : int = 1
JOBTYPE_FAILED : int = 2
JobList()
createSendJob()
convertFileToFV()
convertFileCharacters()
getJobList()
getEndedJobList()
getFailedJobList()
getMarkedSendJobs()
getMarkedReceiveJobs()
addJobListObserver()
removeJobListObserver()
getJobOverview()
getJobDetails()
getJobExtendedDetails()
getEmptyJobParameter()
getEmptyFileParameter()
getConversionTableNames()
holdJob()
deleteJob()
releaseJob()
getInboxDir()
getOutboxDir()
getInboxContent()
addInboxObserver()
removeInboxObserver()
getFileDetails()
getInboxDetails()
getOutboxContent()
getOutboxDetails()
addOutboxObserver()
removeOutboxObserver()
getFilesOfDir()
getDirsOfDir()
downloadFile()
uploadFile()
deleteEERP()
releaseEERP()
removeStatisticEntry()
reloadJoblist()
getIF()
getJoblistQueryInfos()
executeJoblistQuery()
fetchJoblistQueryResult()

Konstruktor

Der in `JobList` definierte Konstruktor wird von der Klasse `Rvs` aufgerufen und sollte nicht vom Anwendungsprogrammierer verwendet werden. Er wird daher hier nicht weiter behandelt.

Lesen der Jobliste

Die aktuelle Jobliste kann über die Methode `getJobList` gelesen werden. Es wird ein Objekt vom Typ `Vector` zurückgegeben, gefüllt mit Elementen vom Typ `JobOverview`. Die Klasse `JobOverview` enthält wichtigste Informationen über alle Sende- und Empfangsjobs (wie z.B. JobID, VDSN, StationID, Status, Zeit der Erstellung).

Definition:

```
Vector getJobList(UserProfile user)
```

Lesen der Jobdetails

Mit der Methode `getJobDetails` können Detailinformationen eines Jobs mit einer bestimmten JobID (`jobID` als Eingabe-Parameter) gelesen werden. Es wird ein Objekt der Klasse `JobDetails` zurückgegeben. Die Klasse `JobDetails` enthält alle möglichen Job-Informationen (Parameter).

Definition:

```
JobDetails getJobDetails(JobID jobID, UserProfile user)
```

Parameter:

`jobID`	ID des Jobs, für welchen die Informationen gelesen werden.

Erzeugen eines Sendeauftrags (Jobs)

Ein neuer Sendeauftrag kann mit der Methode `createSendJob` erzeugt werden. Diese Methode liefert ein Objekt der Klasse `JobDetails`, das alle Informationen über den erzeugten Job beinhaltet.

Definition:

```
JobDetails createSendJob(OutboxDetails outboxDetails,
SendDetails sendDetails, UserProfile user)
```

Parameter:

`outboxDetails`	Parameter (Details) der Datei, die gesendet werden soll; z.B. Dateiname, Pfad und Dateigröße
`sendDetails`	Parameter (Details) des Sendejobs, der erzeugt werden soll; z.B. StationID des Empfängers, VDSN, Codein, Codeout,….

Einen Job anhalten

Mit `holdJob` kann ein Job auf den Status `HELD` gesetzt werden. **Hinweis**: Erst nachdem ein Job im Status `HELD` ist, darf er gelöscht oder wieder freigegeben werden.

Definition:

```
void holdJob(JobID jobID, UserProfile user)
```

Parameter:

`jobID`	ID des Jobs, der angehalten werden sollte.

Einen Job löschen

Mit `deleteJob` wird der Job mit der JobID `jobID` aus rvs® gelöscht. **Hinweis**: Der Job, der gelöscht werden soll, muss vorher mit der Methode `holdJob` in Status `HELD` gesetzt werden (siehe `holdJob`).

Definition:

```
void deleteJob(JobID jobID, UserProfile user)
```

Parameter:

`jobID`	ID des Jobs, der gelöscht werden soll.

EERP freigeben

Ein EERP (End-to-End-Response; Empfangsquittung), der sich im Status `HELD` befindet, kann mit der Methode `releaseEERP` freigegeben werden.

Definition:

```
void releaseEERP(JobID jobID, UserProfile user)
```

Parameter:

`jobID`	ID des Jobs, dessen EERP freigegeben werden sollte.

EERP löschen

Mit `deleteEERP` wird der EERP des Jobs mit der JobID `jobID` aus rvs® gelöscht. **Hinweis**: Der EERP, der gelöscht werden soll, muss sich im Status `HELD` befinden.

Definition:

```
void deleteEERP(JobID jobID, UserProfile user)
```

Parameter:

`jobID`	ID des Jobs, dessen gelöscht werden soll.

4.3.3 Wie erzeuge ich einen Sendeauftrag?

Das folgende Beispiel zeigt Ihnen wie ein Sendeauftrag erzeugt werden kann. Das Programm heißt `CreateSendEntry.java` und befindet sich im Paket `com.gedas.rvs.client.api.tools`.

Beispiel:

```
package com.gedas.rvs.client.api.tools;


import com.gedas.rvs.client.api.JobList;
import com.gedas.rvs.client.api.Rvs;
import com.gedas.rvs.client.api.UserProfile;
import com.gedas.rvs.data.job.JobDetails;
import com.gedas.rvs.data.job.OutboxDetails;
import com.gedas.rvs.data.job.SendDetails;


/*********************************************************
* CreateSendEntry allows creation of a send job.
*/
public class CreateSendEntry
{
    // runtime data
    Rvs rvs = null;
    UserProfile user = null;
    JobList joblist = null;
    Login loginHelper = null;

    // these variables must be set via parameters
    String dsn      = null;
    String sid      = null;

    // these variables may be set via parameters
    String     vdsn = null;
    String     outputFormat  = null;
    String  inputCode   = null;
    String  outputCode  = null;
    int        recordLength = 0;

    // configuration data (set once)
    boolean verbose = false;
    boolean writeLog = false;

  /**
   *   Reads   the   command   line   arguments   and   sets   the   internal
   * data.
   */
    public void readArguments(String[] args)
    {
        int i=0;
        while (i<args.length)
        {
            if (args[i].compareTo("-d")==0)
            {
                // data set name
                dsn = args[i+1];
                i+=2;
            }
            else if (args[i].compareTo("-s")==0)
            {
                // stations id of receiver
                sid = args[i+1];
                i+=2;
            }
            else if (args[i].compareTo("-v")==0)
```

```
            {
                // virtual data set name
                vdsn = args[i+1];
                i+=2;
            }
            else if (args[i].compareTo("-f")==0)
            {
                // output format
                outputFormat = args[i+1];
                i+=2;
            }
            else if (args[i].compareTo("-l")==0)
            {
                // record length
                recordLength = Integer.parseInt(args[i+1]);
                i+=2;
            }
            else if (args[i].compareTo("-i")==0)
            {
                // input code (A/E)
                inputCode = args[i+1];
                i+=2;
            }
            else if (args[i].compareTo("-o")==0)
            {
                // output code (A/E)
                outputCode = args[i+1];
                i+=2;
            }
            else if (args[i].compareTo("-verbose")==0)
            {
                // set verbose output mode
                verbose = true;
                i++;
            }
            else         if         (args[i].compareTo("-?")==0         ||
args[i].compareTo("/?")==0)
            {
                // call usage
                printUsage();
                return;
            }
            else // default with abort
            {
                System.err.println("error:   parameter   "+args[i]+"   not
supported!");
                printUsage();
            }
        }

        if (verbose) System.out.println("** READ PARAMETER **");


        if ( dsn==null || sid==null)
        {
            System.err.println("error: required parameter are missing");
            printUsage();
        }
    }

    /**
     * Performs the login.
     */
    public void login()
    {
        //Login
        try
        {
            // establish connection to rvs middleware (rvs server)
            loginHelper = new Login(verbose);
            rvs = loginHelper.login();
            user = loginHelper.getUserProfile();
            joblist = rvs.getJobList(user);

            if (rvs==null)
            {
```

```
                System.err.println("ERROR: Connection failed.");
                if (verbose) System.out.println("** ABORT **");
                System.exit(-1);
            }
            else
            {
                if (verbose) System.out.println("Connected. ");
            }
        }
        catch (Exception e)
        {
            System.err.println("Exception during login: "+e);
            e.printStackTrace();
            if (verbose) System.out.println("** ABORT **");
            System.exit(-1);
        }
    }

    /**
     * Performs the logout.
     */
    public void logout()
    {
        //Logout
        try
        {
            loginHelper.logout();
        }
        catch (Exception e)
        {
            System.err.println("Exception during logout: "+e);
            e.printStackTrace();
            if (verbose) System.out.println("** ABORT **");
            System.exit(-1);
        }
    }

    /**
     * Performs the send entry creation.
     */
    public void createSendEntry()
    {
        try
        {
            if (verbose) System.out.println("** CREATE SEND ENTRY **");

            if (vdsn==null || vdsn.equals(""))
            {
                vdsn=dsn;
            }

            // get outbox details for the file
            OutboxDetails od = joblist.getOutboxDetails(dsn,user);

            // create send details for the job
            SendDetails sd = new SendDetails(sid,vdsn);
            // keep the file after successfull sending
            sd.setdisp("K");
            if (outputFormat!=null) sd.setformat(outputFormat);
            if (inputCode !=null) sd.setcodein(inputCode);
            if (outputCode!=null) sd.setcodeout(outputCode);
            sd.setlMaxRecl(recordLength);

            // create the job in rvs
            JobDetails jd = joblist.createSendJob(od,sd,user);

            if (verbose)
            {
                System.out.println("*********************************");
                System.out.println("** SEDJOB SUCCESSFULLY CREATED ***");
                System.out.println("*********************************");
            }
               }
        catch (Exception e)
        {
```

```
            System.err.println("Exception during create send entry: "+e);
            e.printStackTrace();
            if (verbose) System.out.println("** ABORT **");
            System.exit(-1);
        }
    }

    /**
     * The main method.
     */
    static public void main(String[] args)
    {
        CreateSendEntry cs = new CreateSendEntry();
        cs.readArguments(args);
        cs.login();
        cs.createSendEntry();
        cs.logout();

        System.exit(0);
    }

    /**
     * Prints the usage.
     */
    static private void printUsage()
    {
        System.out.println("");
        System.out.println("Usage:   CreateSendEntry   -d   filename   -s
receiver sid [-viofl?][-verbose]");
        System.out.println("");
        System.out.println("\tRequired send parameter:");
        System.out.println("\t-d: data set name");
        System.out.println("\t-s: stations id of receiver");
        System.out.println("\t");
        System.out.println("\tOptional send parameter:");
        System.out.println("\t-v: virtual data set name");
        System.out.println("\t-i: input code (A,E)");
        System.out.println("\t-o: output code (A,E)");
        System.out.println("\t-f: format (T,U,F,V)");
        System.out.println("\t-l: record length");
        System.out.println("\t");
        System.out.println("\tOptional parameter:");
        System.out.println("\t-verbose: verbose output");
        System.out.println("\t-?: shows this usage");
        System.out.println("\t");
              System.exit(-1);
    }
}
```

4.3.4 Wie zeige ich die Liste der aktiven Jobs an?

Mit diesem Beispiel (`GetJobList.java`) kann die Liste der aktiven Jobs angezeigt werden.

```
package com.gedas.rvs.client.api.tools;


import java.util.Vector;

import com.gedas.rvs.client.api.JobList;
import com.gedas.rvs.client.api.Rvs;
import com.gedas.rvs.client.api.UserProfile;
import com.gedas.rvs.data.job.JobOverview;


public class GetJobList
{
    // runtime data
    Rvs rvs = null;
    UserProfile user = null;
    JobList joblist = null;
    Login loginHelper = null;

    // configuration data (set once)
    boolean readFromCache = false;
    boolean longOutput = false;
    boolean verbose = false;
    boolean writeLog = false;

    // helping data
    String[] direction = {"SND","RCV"};


       public void work() throws Exception
        {
        if (verbose) System.out.println("WORK");
        Vector list = null;

        if (verbose) System.out.println("Getting job list. ");
        if (!readFromCache) joblist.reloadJoblist(user);
        list = joblist.getJobList(user);
        if (verbose) System.out.println("Got job list. ");

        printVector(list);
        if (verbose) System.out.println("Work finished. ");
       }

//     public void init() throws Exception
//     {
//             config       =       new       PropertyResourceBundle(new
FileInputStream(configFilename));
//     }


    /**
     * Performs the login.
     */
    public void login()
    {
        //Login
        try
        {
            // establish connection to rvs middleware (rvs server)
            loginHelper = new Login(verbose);
            rvs = loginHelper.login();
            user = loginHelper.getUserProfile();
            joblist = rvs.getJobList(user);

            if (rvs==null)
            {
                System.err.println("ERROR: Connection failed.");
```

```
                if (verbose) System.out.println("** ABORT **");
                System.exit(-1);
            }
            else
            {
                if (verbose) System.out.println("Connected. ");
            }
        }
        catch (Exception e)
        {
            System.err.println("Exception during login: "+e);
            e.printStackTrace();
            if (verbose) System.out.println("** ABORT **");
            System.exit(-1);
        }
    }

    /**
     * Performs the logout.
     */
    public void logout()
    {
        //Logout
        try
        {
            loginHelper.logout();
        }
        catch (Exception e)
        {
            System.err.println("Exception during logout: "+e);
            e.printStackTrace();
            if (verbose) System.out.println("** ABORT **");
            System.exit(-1);
        }
    }

    private void printVector(Vector v)
    {
        if   (verbose)   System.out.println("Joblist   will   be   printed.
size="+v.size());
        System.out.println("************************");

        for(int i=0;i<v.size();i++)
        {
            if (verbose) System.out.println("Joblist element #"+i);
            printJOV((JobOverview)v.elementAt(i));
        }
        System.out.println("************************");
        System.out.println("List ended. size="+v.size());
        System.out.println("************************");
    }

    private void printJOV(JobOverview jed)
    {
        if (jed==null)
        {
            if (verbose)
            {
                      System.out.println("JOV == NULL");
            }
        }
        else
        {
            if (longOutput)
            {
             System.out.println("List              job              number
"+jed.getJobID().getjobNumber()
 +" ("+direction[jed.getJobID().getdirection()]+")"
 +":");
System.out.println("SID          : "+jed.getSID());
System.out.println("VDSN         : "+jed.getVDSN());
System.out.println("State        : "+jed.getStatus());
System.out.println("************************");
            }
            else // short output
            {
```

```
System.out.println("job "+jed.getJobID().getjobNumber()
+" ("+direction[jed.getJobID().getdirection()]+")"
+": state="+jed.getStatus());
            }
         }
    }

        /**
         * Reads the command line arguments into the data structures.
         */
        private boolean readArguments(String[] args)
        {
            if (verbose) System.out.println("** READ ARGUMENTS **");

         int i=0;


               // read parameters
            while (i<args.length)
            {
                if (args[i].compareTo("-a")==0)
                {
                    // set long output mode
                    longOutput = true;
                    i++;
                }
             else if (args[i].compareTo("-c")==0)
                {
                    //  set  cache  modus  (joblist  will  not  be  reloaded
before listing)
                    readFromCache = true;
                    i++;
                }
             else if (args[i].compareTo("-verbose")==0)
                {
                    // set verbose output mode
                    verbose = true;
                    i++;
                }
                else        if        (args[i].compareTo("-?")==0        ||
args[i].compareTo("/?")==0)
                {
                    // call usage
                    printUsage();
                    return false;
                }
                else       if       (args[i].compareTo("-help")==0       ||
args[i].compareTo("-h")==0)
                {
                    // call help
                    printHelp();
                    return false;
                }
                else // default with abort
                {
                    System.err.println("Error:  parameter  "+args[i]+"  not
supported!");
                    printUsage();
                    throw  new  IllegalArgumentException("Error:  parameter
"+args[i]+" not supported!");
                }
            }
            if  (verbose)  System.out.println("**  READ  ARGUMENTS  finished
**");

            return true;
        }



        /**
         * Prints the help on system out.
         */
        static private void printHelp()
        {
```

```
            System.out.println("");
            System.out.println("GetJobList   shows   the   list   of   active
jobs.");
            System.out.println("");
               printUsage();
        }

        /**
         * Prints the usage on system out.
         */
        static private void printUsage()
        {
            System.out.println("");
            System.out.println("Usage: GetJobList [-a][-c][-verbose]");
            System.out.println("");
            System.out.println("\tParameter :");
//          System.out.println("\t-s       : list the marked send jobs");
//          System.out.println("\t-r           :  list  the  marked  receive
jobs");
          System.out.println("\t-a           :  prints  all  available  job
data");
          System.out.println("\t-c        : reads joblist from middleware
cache");
          System.out.println("\t               (no  joblist  reload  will  be
done)");
          System.out.println("\t-verbose : verbose output");
          System.out.println("\t-help    : prints help");
          System.out.println("\t-?       : prints usage");
          System.out.println("\t");
}


    /**
     * Exists the process with the given return code.
     */
//    static private void exit(int returnCode)
//    {
//        if (verbose) System.out.println("** ABORT **");
//        System.exit(returnCode);
//    }

        public static void main(String args[])
        {
               try
               {
            GetJobList gjl = new GetJobList();

            if (gjl.readArguments(args))
            {
                gjl.login();
                gjl.work();
                gjl.logout();
            }
               }
               catch (Exception e)
               {
                      System.out.println("Cought Exception:"+e);
                      e.printStackTrace();
               }
         finally
         {
             System.exit(0);
         }
        }
}
```

4.4 Stationsverwaltung

Die Stationsverwaltung ermöglicht das Lesen der Stationsliste und das Anlegen, Ändern und Löschen von einzelnen Stationen in rvs®. Darüberhinaus ist das Exportieren und Importieren von Stationslisten im XML-Format möglich. Diese Stationslisten können auch zum und vom Client exportiert und importiert werden. Für das Bearbeiten von Stationen können diese gesperrt werden.

4.4.1 Datenklassen: Stationen

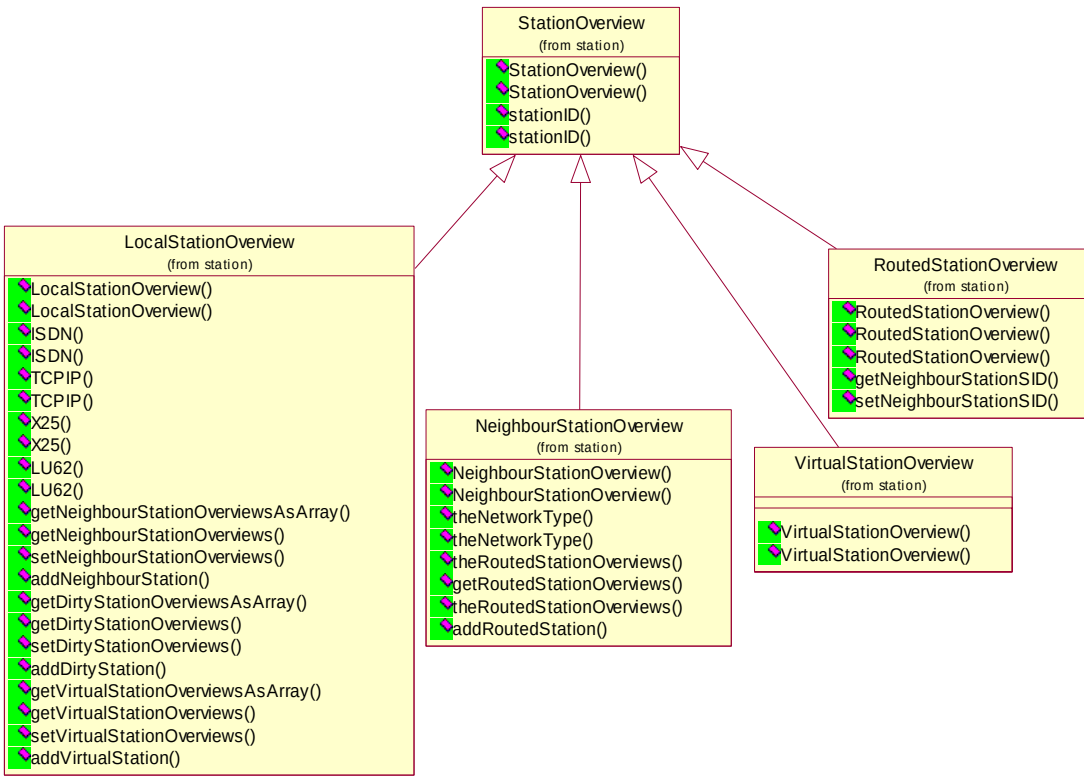
Die Datenklassen des Bereichs Stationen befinden sich in dem Paket `com.gedas.rvs.data.station`. Unterscheiden lassen sich die Gruppen Overview (abgeleitet von `StationOverview`), Details (`StationDetails`) und Station (abgeleitet von `Station`). In der Gruppe Station wird neben der allgemeinen Klasse `Station`, noch zwischen der lokalen Station (Klasse `LocalStation`), der Nachbarstation (Klasse `NeighbourStation`), der virtuellen Station (Klasse `VirtualStation`) und der gerouteten Station (Klasse `RoutedStation`) unterschieden. Des Weiteren befinden sich in diesem Paket die von der Klasse `Station` benutzten Hilfsklassen `Contact` und `OdetteID`.

Overview-Klassen

Die Overview-Klassen enthalten Basisinformationen über Stationen. Zu den Basisinformationen zählen u.a. die SID (Stations ID), für Nachbarstationen über welches Netzwerk sie erreichbar sind und für geroutete Stationen über welche Nachbarstation sie geroutet werden. Daneben erhält die lokale Station eine Liste aller Nachbarstationen und diese eine Liste aller über sie gerouteten Stationen. Damit lässt sich über die Overviews ein Stationsbaum erstellen. Über ein Stationsbaum lässt sich kontrollieren, welche Stationen direkte Nachbarn und welche geroutet sind.

Die Liste der DirtyStations (aus `LocalStationOverwiew`) enthält geroutete Stationen, denen eine gültige Nachbarstation fehlt.

Diagramm Overview-Klassen:



Station-Klassen

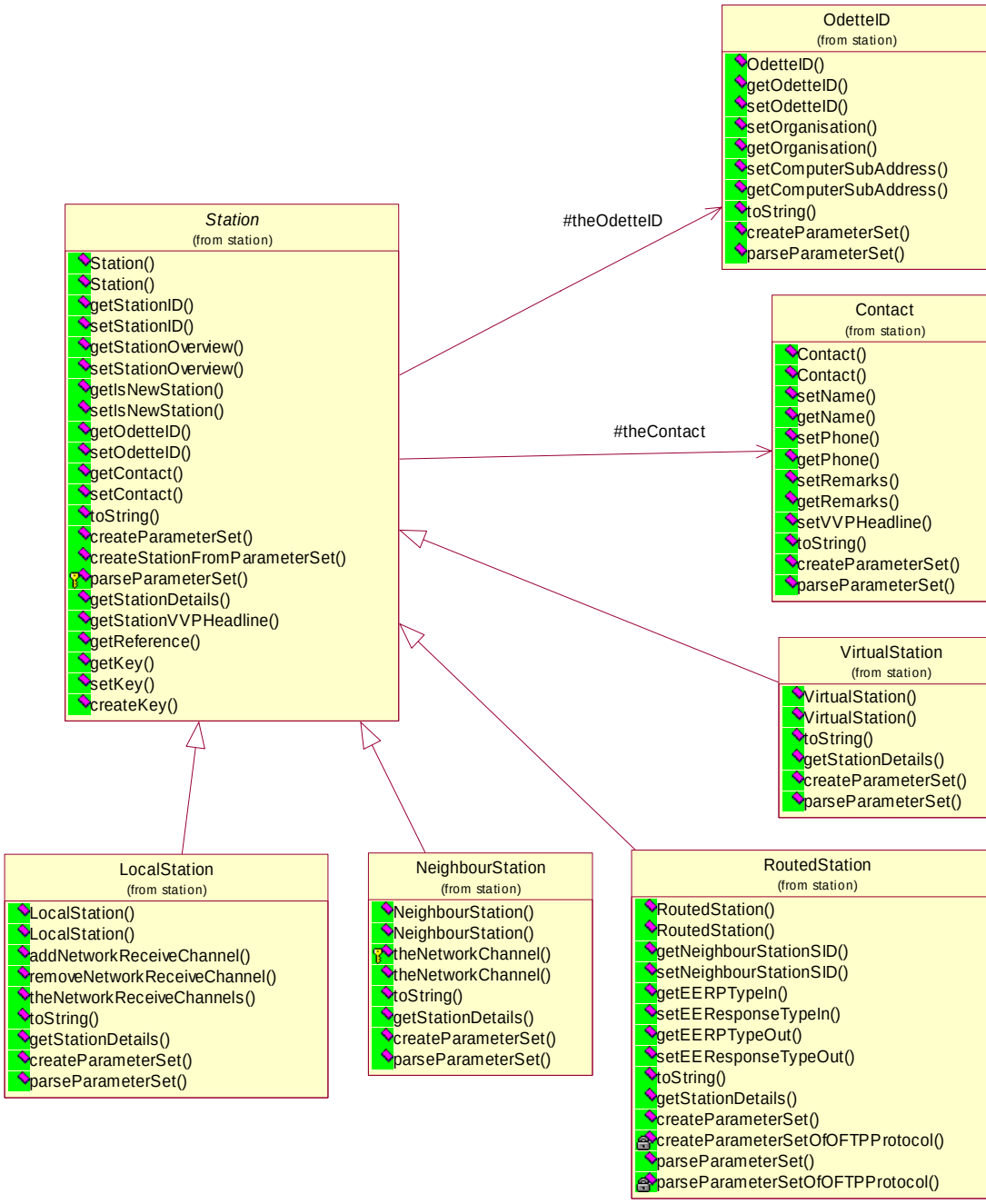
Die Stations-Klassen enthalten, im Gegensatz zu den Overview-Klassen, die kompletten Daten einer Station. Das ist die Basisklasse für die verschiedenen Ausprägungen von Stationen. Abgeleitet hiervon sind die konkreten Klassen: `LocalStation`, `RoutedStation`, `NeighbourStation`, `VirtualStation`.

Die lokale Station und Nachbarstationen enthalten in den Stationsdaten auch Angaben über das verwendete Netzwerk und Parameter hierzu. Bei gerouteten Stationen entfällt dies, da sie über eine Nachbarstation geroutet werden.

Bei der lokalen Station entsprechen die eingestellten Netzwerk-Parameter denen der Empfangskanäle. Der Netzwerk-Parameter einer Nachbarstation entspricht einem Sendekanal.

Die Datenklassen der Netzwerke werden im Kapitel 4.4.2 "Datenklassen: Netzwerk" besprochen.

Diagramm der Station-Klassen:



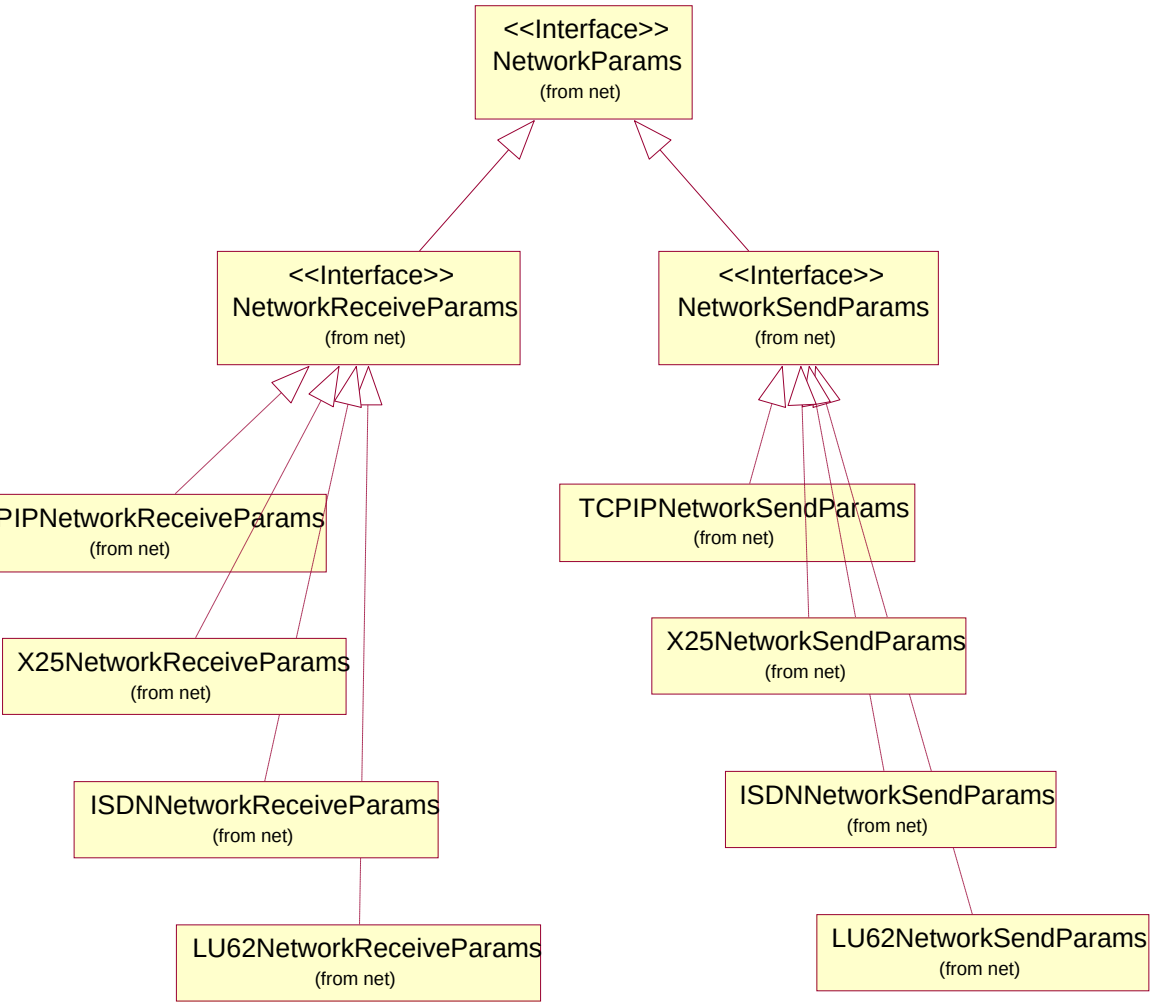
4.4.2 Datenklassen: Netzwerk

Die Datenklassen des Bereichs Netzwerk befinden sich im Paket `com.gedas.rvs.data.net`. Sie werden von den Stations-Klassen verwendet, um die benutzten Netzwerk-Parameter abzubilden. Die Klassenstruktur für Netzwerk ist die komplexeste der konkreten Datenstrukturen.

Es existieren jeweils zwei sich überlappende Kategorien. Zum einen wird zwischen Empfangs- und Senderichtung unterschieden, zum anderen die verschiedenen Netzwerkarten TCP/IP, X.25, ISDN und LU6.2.

Deneben werden OFTP, EERP und die Devices für X.25 und ISDN separat abgebildet.

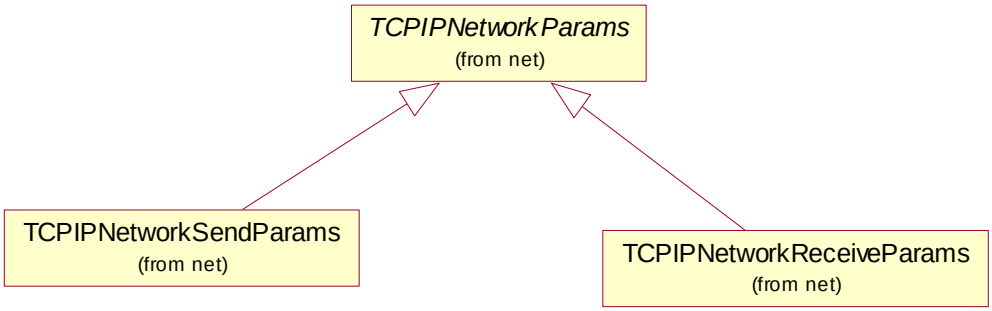
Interface `NetworkParams`



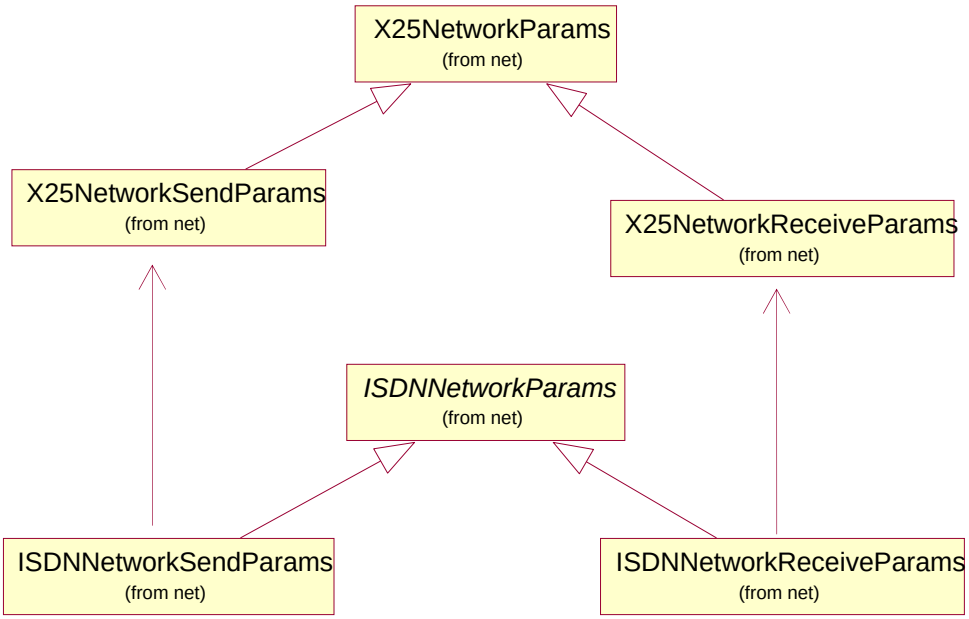
Mit den Interfaces `NetworkReceiveParams` und `NetworkSendParams` wird Empfangs- und Senderichtung unterschieden. Diese beiden Interfaces werden von den Netzwerk-spezifischen Parametersätzen implementiert.

Basisklasse `TCPIPNetworkParams`

`TCPIPNetworkParams` ist die Basisklasse der jeweiligen Empfangs- und Sendeparametersätze.



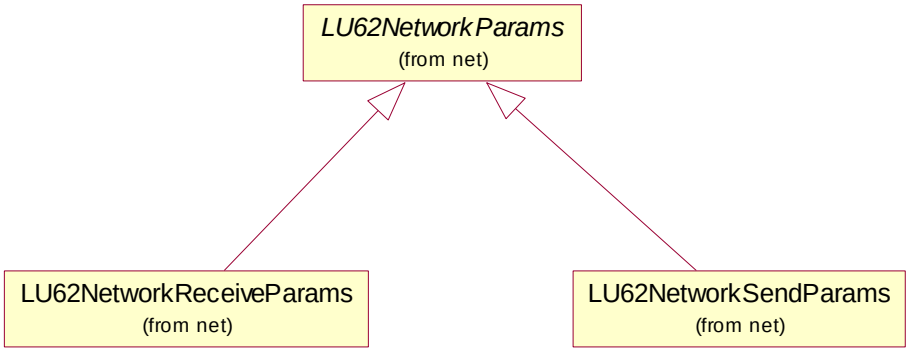
Basisklassen `X25NetworkParams` und `ISDNNetworkParams`



`X25NetworkParams` ist die Basisklasse der jeweiligen Empfangs- und Sendeparametersätze. Dies gilt ebenso für `ISDNNetworkParams`. Darüber hinaus enthalten `ISDNNetworkSendParams` und `ISDNNetworkReceiveParams` Referenzen auf ein Objekt der jeweiligen X25-Parameter. Grund dafür ist, dass X.25 über ISDN zum Einsatz kommt, also eine ISDN-Verbindung auch immer eine X25-Verbindung beinhaltet.

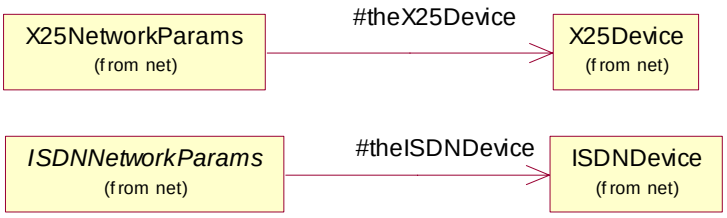
Basisklasse `LU62NetworkParams`

`LU62NetworkParams` ist die Basisklasse der jeweiligen Empfangs- und Sendeparametersätze.



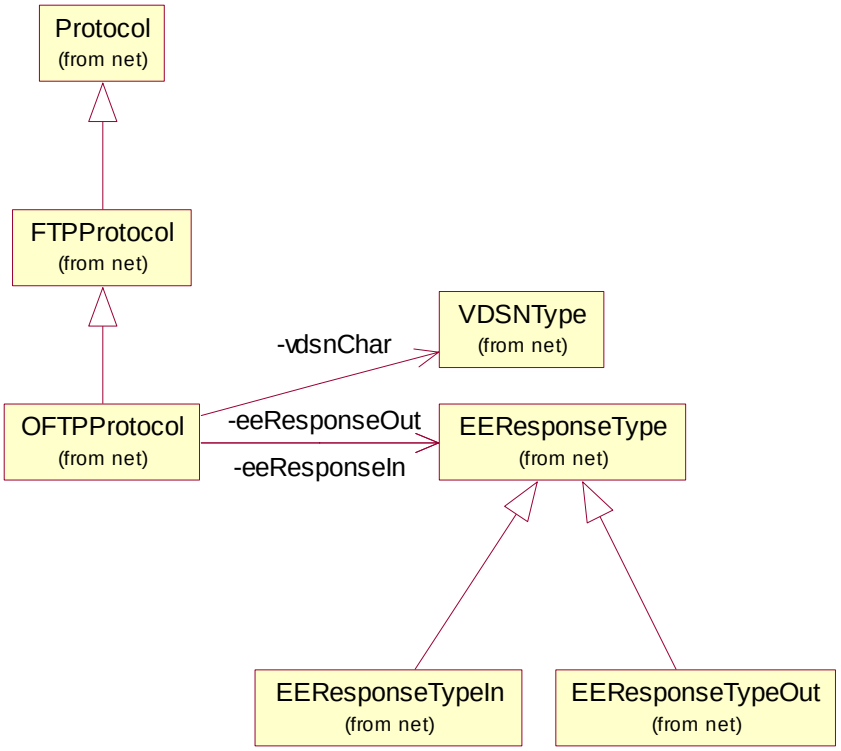
Devices für X.25 und ISDN

`X25NetworkParams` und `ISDNNetworkParams` enthalten jeweils eine Referenz zu den zugehörigen Devices. Ein Device beschreibt die Hardware, die für den Zugang zum Netzwerk benutzt wird.



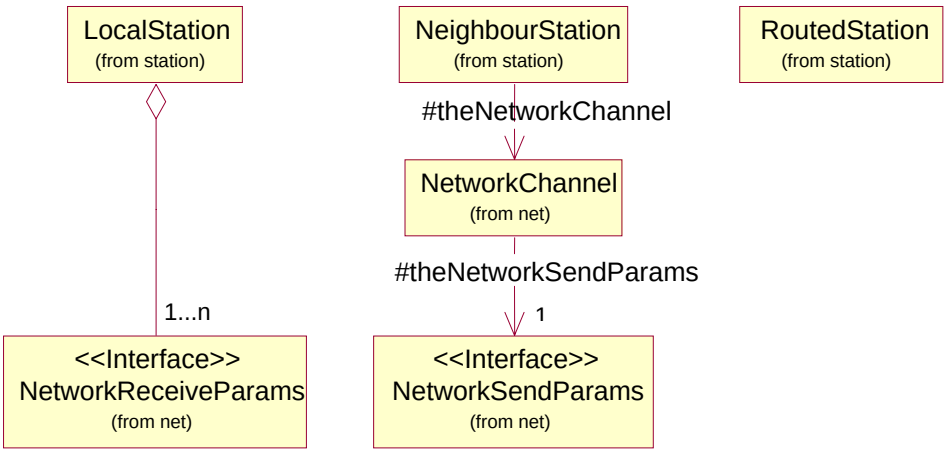
`OFTPProtocol` und zugehörige Klassen

Die Klasse `OFTPProtocol` mit der Basisklasse `Protocol` enthält Referenzen auf die Klassen `EEResponseType` und `VDSNType`. Beim `EEResponseType` wird zwischen Ein- und Ausgangsrichtung unterschieden.



Benutzung der Netzwerkklassen durch die Stationsklassen

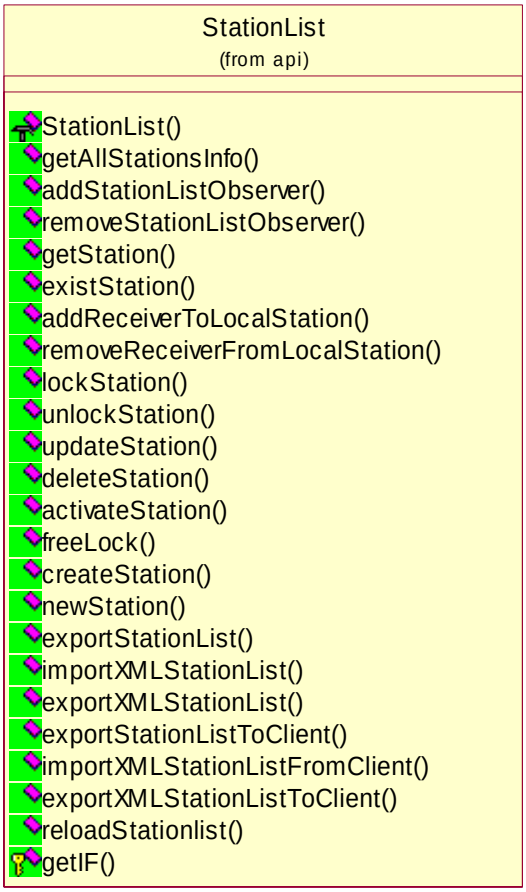
Die Stationsklassen (Package `com.gedas.rvs.data.station`) benutzen die Netzwerkklassen. Die lokale Station enthält eine Sammlung von `NetworkReceiveParams` und Nachbarstationen einen `NetworkChannel`, der wiederum `NetworkSendParams` enthält. Geroutete Stationen enhalten keine Netzwerkklassen.



4.4.3 Serverfunktionalität

Die Methoden der Stationsverwaltung sind in der Klasse `StationList` im Paket `com.gedas.rvs.client.api` zu finden.

Diagramm der Klasse `StationList`:



Konstruktor

Der in `StationList` definierte Konstruktor wird von der Klasse `Rvs` aufgerufen und sollte nicht vom Anwendungsprogrammierer verwendet werden. Er wird daher hier nicht weiter behandelt.

Lesen der Stationsliste

Die Stationsliste kann über die Methode `getAllStationsInfo` gelesen werden. Sie gibt ein Objekt der Klasse `LocalStationOverview` zurück, in dem eine Liste mit Nachbarstationen (`NeighbourStationOverview`) gespeichert sind. Jede Nachbarstation enthält wiederum eine Liste der über sie gerouteten Stationen.

Definition:

```
LocalStationOverview getAllStationsInfo()
```

Lesen der Stationsdetails

Mit der Methode `getStation` können Detailinformationen einer Station mit der StationID `aStationID` als Eingabe-Parameter gelesen werden. Es wird ein Objekt der Klasse `Station` zurückgegeben.

Definition:

```
Station getStation(String aStationID, UserProfile user)
```

Parameter:

`aStationID`	StationID der Station, für welche die Stationsdetails gelesen werden.

Erzeugen einer Station

Eine neue Station kann mit der Methode `createStation` erzeugt werden.

Definition:

```
void createStation(Station station, UserProfile user)
```

Parameter:

`station`	neue Station, ein Objekt der Klasse Station
`user`	SessionID des Benutzers

In der rvs® Client API Version 1 war diese Funktionaltität mittels der Methoden `getStation` und `updateStation` realisiert.

Ändern einer Station

Mit `updateStation` werden die Detailinformationen der übergebenen Station in rvs® gespeichert. Grundlage für Änderungen an einer Station sollten immer die Detailinformationen sein, die vorher mit `getStation` geholt wurden.

Definition:

```
void updateStation(Station station, UserProfile user)
```

Parameter:

`station`	Station, die geändert werden soll; ein Objekt der Klasse Station

Löschen einer Station

Mit `deleteStation` wird die Station mit der StationsID `aStationID` aus rvs® gelöscht.

Definition:

```
void deleteStation(String aStationID, UserProfile user)
```

Parameter:

`aStationID`	StationsID der zu löschenden Station

Aktivieren einer Station

Für die Station mit der übergebenen SID `aStationID` wird eine Verbindung aufgebaut.

Definition:

```
void activateStation(String aStationID, UserProfile user)
```

Parameter:

`aStationID`	StationsID der zu aktivierenden Station

Empfänger für die lokale Station hinzufügen

Diese Methode fügt zur lokalen Station einen Empfänger hinzu.

Definition:

```
LocalStation      addReceiverToLocalStation(int      port,
UserProfile user)
```

Parameter:

`port`	Art des hinzuzufügenden Netzwerkes; erlaubte Werte: NetworkType.PORT_X25 NetworkType.PORT_ISDN NetworkType.PORT_TCPIP

NetworkType.PORT_LU62

Empfänger von der lokalen Station löschen

Diese Methode entfernt einen Empfänger von der lokalen Station.

Definition:

```
LocalStation removeReceiverFromLocalStation(String name,
UserProfile user)
```

Parameter:

`name`	Name des Empfängers, der entfernt werden soll.

Station sperren und entsperren

Diese Methoden sperren eine Station und geben sie wieder frei (`lockStation` und `unlockStation`). Eine Station muss vor dem Update oder vor dem Löschen immer gesperrt werden. Der Anwender, der eine Station gesperrt hat, soll diese auch entsperren. Falls der Server die Entsperr-Aktion nicht mitbekommen hat, kann ein Administrator eine gesperrte Station auch ohne Zutun des Anwenders, der sie gesperrt hatte, mit `freeLock` wieder freigeben.

Definition:

```
void lockStation(String aStationID, UserProfile user)
void unlockStation(String aStationID, UserProfile user)
void freeLock (String aStationID, UserProfile user)
```

Parameter:

`aStationID`	die StationsID der Station, die gesperrt oder entsperrt werden soll.

Stationsliste exportieren

Diese Methoden exportieren die Stationsliste in eine Datei. Zielformat ist entweder das rvs® eigene Format (`exportStationList`) oder XML (`exportXMLStationList`).

Abgelegt wird die Datei entweder serverseitig (`exportStationList`, `exportXMLStationList`) oder clientseitig (`exportStationListToClient`, `exportXMLStationListToClient`).

Definition:

```
void exportStationList(String filename, UserProfile user)
void exportXMLStationList(String filename, boolean
      fullPath, UserProfile user)
void exportStationListToClient(UserProfile user, String
      localFilename)
void exportXMLStationListToClient(UserProfile user, String
      localFilename)
```

Parameter:

`localFilename`	Name der Datei in die die Stationsliste geschrieben wird.
`fullPath`	Flag, ob Dateiname vollen Pfad enthält.

Stationsliste importieren

Diese Methoden importieren eine XML-Stationsliste. Die zu importierende Datei liegt entweder serverseitig (`importXMLStationList`) oder clientseitig (`importXMLStationListFromClient`).

Definition:

```
Vector importXMLStationList(String filename, boolean
      deleteBeforeImporting, UserProfile user)

Vector importXMLStationListFromClient(String localFilename,
      boolean deleteBeforeImporting, UserProfile user)
```

Parameter:

`localFilename`	Name der Datei aus der die neue Stationsliste gelesen wird
`deleteBeforeImporting`	Flag, ob die rvs-interne Stationsliste vor dem Import gelöscht wird

Rückgabewert:

`Vector`	Ein Vektor, der zwei Vektoren mit SID-Listen enthält; erstens die importierten, zweitens die abgelehnten Stationen.

4.4.4 Wie erzeuge ich eine neue Station?

In diesem Beispiel wird eine neue Station in der Stationsliste von rvs® angelegt.

Beispiel (`CreateNeighbourStation.java`):

```
package com.gedas.rvs.client.api.tools;


import com.gedas.rvs.RvsException;
import com.gedas.rvs.data.net.NetworkType;
import com.gedas.rvs.data.station.NeighbourStation;
import com.gedas.rvs.data.station.NeighbourStationOverview;
import com.gedas.rvs.data.net.TCPIPNetworkSendParams;
import com.gedas.rvs.data.net.X25NetworkSendParams;
import com.gedas.rvs.data.net.ISDNNetworkSendParams;
import com.gedas.rvs.data.net.LU62NetworkSendParams;
import com.gedas.rvs.client.api.UserProfile;
import com.gedas.rvs.client.api.StationList;
import com.gedas.rvs.client.api.Rvs;
import com.gedas.util.IPAddress;


/**********************************************************************
* CreateNeighbourStation allows the creation of a new rvs station.
*/
public class CreateNeighbourStation
{
    // runtime data
    Rvs rvs = null;
    UserProfile user = null;
    StationList stationlist  = null;
    Login loginHelper = null;

    String sid = null;

    int  intNetwork = 0;
    NetworkType networkType = null;

    // these variables may be set via parameters
    String odetteID   = null;
    String SID  = null;
    String strNetwork = null;

    // configuration data (set once)
    boolean verbose = false;

    /**
     * Reads the command line arguments and sets the internal data.
     */
    public void readArguments(String[] args)
    {
  // parse command line parameters
     int i=0;
        while (i<args.length)
        {
            if (args[i].compareTo("-?")==0)
            {
               //help
                printUsage();
                i+=2;
            }
                else if (args[i].compareTo("-sid")==0)
            {
                // Station ID
                SID = args[i+1];
                i+=2;
            }
```

```
                else if (args[i].compareTo("-verbose")==0)
            {
                // verbose mode
                verbose = true;
                i++;
            }
                else if (args[i].compareTo("-oid")==0)
            {
                // Odette ID
                odetteID = args[i+1];
                i+=2;
            }
                else if (args[i].compareTo("-nt")==0)
            {
                // Network Type
                strNetwork = args[i+1];
                i+=2;

                if (strNetwork.compareTo("X25")==0)
                    intNetwork = 1;
                else if (strNetwork.compareTo("LU62")==0)
                    intNetwork = 2;
                else if (strNetwork.compareTo("TCPIP")==0)
                    intNetwork = 3;
                else if (strNetwork.compareTo("ISDN")==0)
                    intNetwork = 4;

                switch (intNetwork)
                {
                   case 1: networkType = NetworkType.toNetworkType(1000);
                   break;
                   case 2: networkType = NetworkType.toNetworkType(1001);
                   break;
                   case 3: networkType = NetworkType.toNetworkType(1002);
                   break;
                   case 4: networkType = NetworkType.toNetworkType(1003);
                   break;
                 }
            }
        }

        if (SID==null && odetteID==null && strNetwork==null)
        {
          System.out.println("No  parameter  input:  using  default  values
now...");
        }

               // set default values
               if (SID==null)
                      SID = "NEWSTATION";
               if (odetteID==null)
                      odetteID ="Z123456789";
               if (strNetwork==null)
                      networkType = NetworkType.toNetworkType(1002);

        if (verbose)
        {
            System.out.println("READ ARGUMENTS:");
            System.out.println("SID         : "+SID);
            System.out.println("OID         : "+odetteID);
            System.out.println("Network     : "+strNetwork);
            System.out.println("Network Type: "+networkType);
            System.out.println();
        }
    }

    /**
     * Performs the login.
     */
    public void login()
    {
        //Login
        try
        {
           // establish connection to rvs middleware (rvs server)
```

```
            loginHelper = new Login(verbose);
            rvs = loginHelper.login();
            user = loginHelper.getUserProfile();
            stationlist = rvs.getStationList(user);

            if (rvs==null)
            {
                System.err.println("ERROR: Connection failed.");
                if (verbose) System.out.println("** ABORT **");
                System.exit(-1);
            }
            else
            {
                if (verbose) System.out.println("Connected. ");
            }
        }
        catch (Exception e)
        {
            System.err.println("Exception during login: "+e);
            e.printStackTrace();
            if (verbose) System.out.println("** ABORT **");
            System.exit(-1);
        }
    }

    /**
     * Performs the logout.
     */
    public void logout()
    {
        //Logout
        try
        {
            loginHelper.logout();
        }
        catch (Exception e)
        {
            System.err.println("Exception during logout: "+e);
            e.printStackTrace();
            if (verbose) System.out.println("** ABORT **");
            System.exit(-1);
        }
    }

    /**
     * Performs the station creation.
     */
    public void createStation()
    {
        //Create Station
        try
        {
            if (verbose) System.out.println("** CREATE STATION **");

           // create new StationOverview object
                      NeighbourStationOverview     overview     =     new
                    NeighbourStationOverview(SID, networkType);
                      if     (verbose)     System.out.println("**     Net
                    type:"+overview.theNetworkType().toString());

            // create the station object;
            NeighbourStation station = new NeighbourStation(overview);
            station.getOdetteID().setOdetteID(odetteID);

            // setting the required network parameter
            switch (intNetwork)
            {
                case 1:
                if (verbose) System.out.println("** X25");

((X25NetworkSendParams)(station.theNetworkChannel().getNetworkSendParams(
))).x25Device().setName("default");

((X25NetworkSendParams)(station.theNetworkChannel().getNetworkSendParams(
))).setAddress("default");
                    break;
```

```
                case 2:
                if (verbose) System.out.println("** LU62");

((LU62NetworkSendParams)(station.theNetworkChannel().getNetworkSendParams
())).name("default");

((LU62NetworkSendParams)(station.theNetworkChannel().getNetworkSendParams
())).tpName("default");

((LU62NetworkSendParams)(station.theNetworkChannel().getNetworkSendParams
())).mode("default");
                    break;
                case 3:
                if (verbose) System.out.println("** TCPIP");
                    IPAddress ip = new IPAddress("default");

((TCPIPNetworkSendParams)(station.theNetworkChannel().getNetworkSendParam
s())).setIPAddress(ip);

((TCPIPNetworkSendParams)(station.theNetworkChannel().getNetworkSendParam
s())).setPort(3305);
                    break;
                case 4:
               if (verbose) System.out.println("** ISDN");
                    // use as device type RCAPI, NETISDN or CAPI to get
identified as ISDN neighbour station

((ISDNNetworkSendParams)(station.theNetworkChannel().getNetworkSendParams
())).theISDNDevice().type("RCAPI");

((ISDNNetworkSendParams)(station.theNetworkChannel().getNetworkSendParams
())).isdnAddress("default");

((ISDNNetworkSendParams)(station.theNetworkChannel().getNetworkSendParams
())).theX25NetworkSendParams().setAddress("default");
                    break;
            }

            // create the new station in rvs
            stationlist.createStation(station,user);
            if (verbose)
            {
                System.out.println("**********************************");
                System.out.println("** STATION SUCCESSFULLY CREATED **");
                System.out.println("**********************************");
            }
            }
        catch (Exception e)
        {
            System.err.println("Exception during create station: "+e);

            if (e instanceof RvsException)
            {
                if (verbose) System.out.println("** ABORT **");
                System.exit(-1);
            }
            else
            {
                e.printStackTrace();
                if (verbose) System.out.println("** ABORT **");
                System.exit(-1);
            }
        }
    }

    /**
     * The main method.
     */
    static public void main(String[] args)
    {
        CreateNeighbourStation cs = new CreateNeighbourStation();
        cs.readArguments(args);
        cs.login();
        cs.createStation();
        cs.logout();
```

```
        System.exit(0);
    }

    /**
     * Prints the usage.
     */
    static private void printUsage()
    {
        System.out.println("");
        System.out.println("Usage:  CreateStation  -sid  StationID  -oid
ODETTEID -nt NETWORKTYPE [-verbose]");
        System.out.println("");
        System.out.println("\tRequired parameters:");
        System.out.println("\t-sid: StationID");
        System.out.println("\t-oid: OdetteID");
        System.out.println("\t-nt:   NetworkType   (TCPIP,   ISDN,   X25,
LU62)");
        System.out.println("\tOptional parameter:");
        System.out.println("\t-verbose: verbose output");
        System.out.println("\t");
        System.exit(-1);
    }
}
```

4.5 Benutzerverwaltung

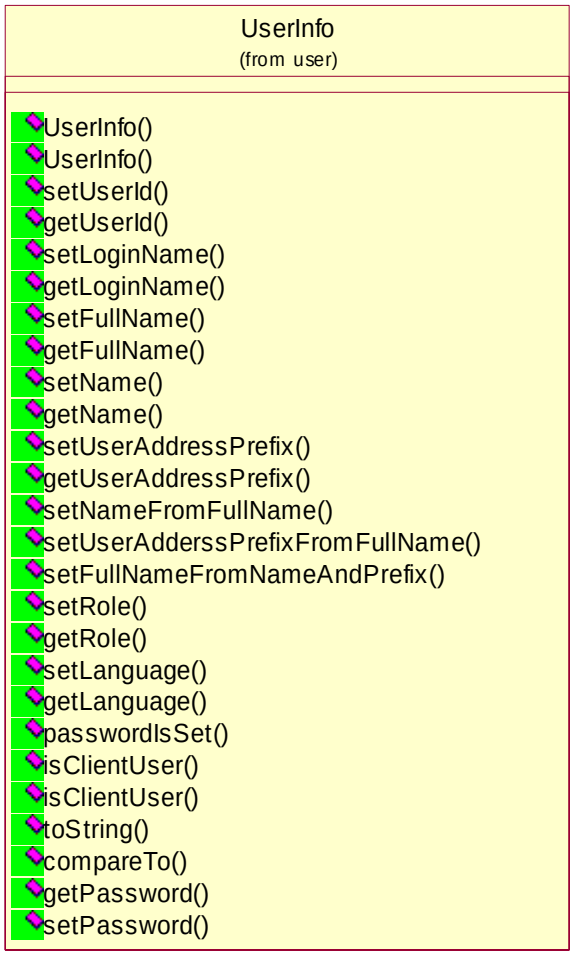
Die Benutzerverwaltung ermöglicht das Lesen der Benutzerliste und Anlegen, Ändern und Löschen von einzelnen Benutzern in rvs®. Darüberhinaus ist das Anzeigen aller Benutzerinformationen und Ändern des Benutzerpassworts möglich. Für das Bearbeiten der Benutzer können diese gesperrt (gelockt) werden.

4.5.1 Daten

Die Datenklassen für die Benutzerverwaltung befinden sich im Paket `com.gedas.rvs.data.user`.

Die wichtigste Klasse in diesem Paket ist die Klasse `UserInfo`. Diese Klasse enthält alle Informationen über einen Benutzer. Diese sind: Benutzername (account), voller Name, Passwort, Alias (Benutzerordner in Verzeichnissen `inbox`/`outbox`), Gruppe (Administrator/Operator/Benutzer), Sprache des Benutzers (Englisch/Deutsch) und Client-Benutzer (Ja/Nein).

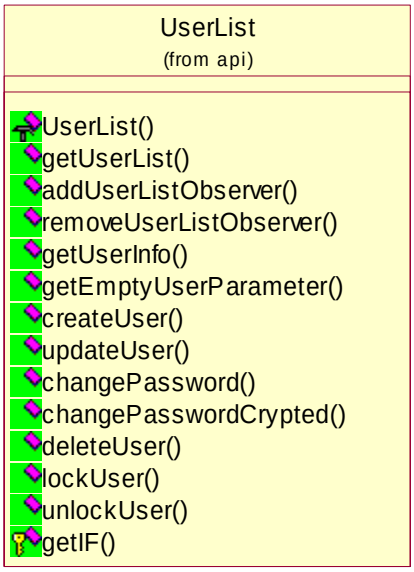
Diagramm der Klasse `UserInfo`:


UserInfo
(from user)
UserInfo()
UserInfo()
setUserId()
getUserId()
setLoginName()
getLoginName()
setFullName()
getFullName()
setName()
getName()
setUserAddressPrefix()
getUserAddressPrefix()
setNameFromFullName()
setUserAdderssPrefixFromFullName()
setFullNameFromNameAndPrefix()
setRole()
getRole()
setLanguage()
getLanguage()
passwordIsSet()
isClientUser()
isClientUser()
toString()
compareTo()
getPassword()
setPassword()

4.5.2 Serverfunktionalität

Die Methoden der Benutzerverwaltung sind in der Klasse `UserList` im Paket `com.gedas.rvs.client.api` zu finden.

Diagramm der Klasse `UserList`:



Konstruktor

Der in `UserList` definierte Konstruktor wird von der Klasse `Rvs` aufgerufen und sollte nicht vom Anwendungsprogrammierer verwendet werden. Er wird daher hier nicht weiter behandelt.

Lesen der Benutzerliste

Die Benutzerliste kann über die Methode `getUserList` gelesen werden. Sie gibt ein Objekt der Klasse `Vector` zurück. Dieses Objekt enthält seinerseits die Objekte der Klasse `UserInfo`.

Definition:

```
Vector getUserList(UserProfile user)
```

Lesen der Benutzerinformationen

Mit der Methode `getUserInfo` können komplette Daten (alle Parameterwerte) eines Benutzers mit der BenutzerID `login` gelesen werden. Es wird ein Objekt der Klasse `UserInfo` zurückgegeben.

Definition:

```
UserInfo getUserInfo(String login, UserProfile user)
```

Parameter:

`login`	ID des Benutzers, für welchen die kompletten Daten gelesen werden.

Erzeugen eines Benutzers

Ein neuer Benutzer kann mit der Methode `createUser` erzeugt werden. Als Eingabe-Parameter dienen die Daten aus der Klasse `UserInfo` und Rückgabewert der Methode ist: BenutzerID vom Typ String.

Definition:

```
String createUser(UserInfo userInfo, UserProfile user)
```

Parameter:

`userInfo`	Daten eines Benutzers, der erzeugt werden soll

Ändern eines Benutzers

Mit `updateUser` werden Parameterwerte eines Benutzers aktualisiert und in rvs® gespeichert.

Definition:

```
void updateUser(UserInfo userInfo, UserProfile user)
```

Parameter:

`userInfo`	Daten eines Benutzers, die aktualisiert werden soll.

Löschen eines Benutzers

Mit `deleteUser` wird ein Benutzer mit der BenutzerID `userID` aus rvs® gelöscht.

Definition:

```
void deleteUser(String userID, UserProfile user)
```

Parameter:

`userID`	UserID des zu löschenden Benutzers

Hinweis: Ein Benutzer muss immer vor dem Update oder vor dem Löschen mit der Methode `lockUser` gesperrt werden. Der Anwender, der einen Benutzer gesperrt hat, soll diesen auch mit der Methode `unlockUser` entsperren.

Passwort eines Benutzers ändern

Mit `changePassword` kann man das Passwort eines Benutzers ändern.

Definition:

```
void changePassword(String userID, String oldPassword,
String newPassword, UserProfile user)
```

Parameter:

`userID`	UserID des Benutzers, dessen Passwort geändert werden soll.
`oldPassword`	Altes Passwort
`newPassword`	Neues Passwort

4.5.3 Wie erzeuge ich einen neuen Benutzer?

In diesem verkürzten Beispiel wird gezeigt, wie mit dem Konstruktor UserInfo ein neuer Benutzer angelegt werden kann.

Beispiel (nur Methode):

```
// runtime data
    UserProfile userprofile    = null;
    UserList userlist = null;




public void createUser()
    {
       //Create User
        try
        {
            //create the user object with a constructor
            UserInfo              userinfo              =              new
UserInfo(null,"csh","CarlSchmidt","carl","gedas","A","E",false,"UZT8?*");


           // create the new user in rvs
            userlist.createUser(userinfo,userprofile);

        catch (Exception e)
        {
            System.err.println("Exception during create user: "+e);

            if (e instanceof RvsException)
            {
              System.exit(-1);
            }
            else
            {
                e.printStackTrace();
                System.exit(-1);
            }
        }
    }
```

Der erste Parameter `uid` beim Aufruf vom Konstruktor `UserInfo` wird auf `null` gesetzt, da er nur beim Update eines Benutzers von der Bedeutung ist.

4.6 Jobstart- und Parameterverwaltung (Configuration)

Die Jobstart- und Parameterverwaltung ermöglicht das Lesen der Jobstart- und Parameterliste. Darüber hinaus ist das Anlegen, Ändern und Löschen von einzelnen Jobstarts und Ändern von Parametern in rvs® möglich.

4.6.1 Daten

Die Datenklassen für die Jobstart- und Parameterverwaltung befinden sich im Paket `com.gedas.rvs.data.config`.

Die wichtigsten Klassen in diesem Paket sind `JobStartFilter` und `ParameterSet`.

Die Klasse `JobStartFilter` dient zum Erzeugen eines Jobstarts (Sende– oder Empfangsrichtung) mit allen dazugehörigen Parametern.

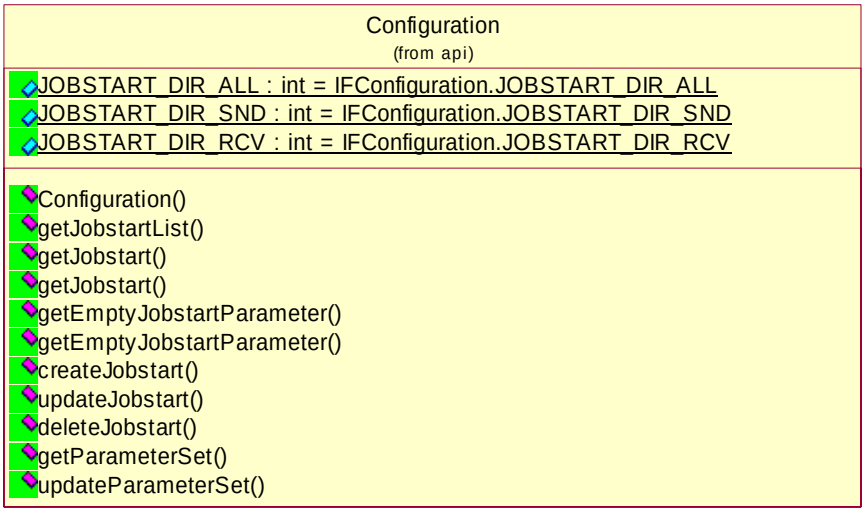
Die Klasse `ParameterSet` enthält eine Hash-Tabelle, in der alle alle rvs®-Parameter gespeichert sind. Als Schlüssel (key) wird der rvs®-Parametername verwendet.

Diagramm der Klasse `JobstartFilter`:

4.6.2 Serverfunktionalität

Die Methoden der Jobstart- und Parameterverwaltung sind in der Klasse `Configuration` in dem Paket `com.gedas.rvs.client.api` zu finden.

Diagramm der Klasse `Configuration`:



Konstruktor

Der in der Klasse `Configuration` definierte Konstruktor wird von der Klasse `Rvs` aufgerufen und sollte nicht vom Anwendungsprogrammierer verwendet werden. Er wird daher hier nicht weiter behandelt.

Lesen der Jobstartliste

Die Jobstartliste kann über die Methode `getJobstartList` gelesen werden. Sie gibt ein Objekt der Klasse `Vector` zurück.

Definition:

```
Vector getJobstartList(UserProfile user, int direction)
```

Parameter:

`direction`	Sie haben durch diesen Parameter die Wahl, welche Jobstarts Sie lesen möchten (Übergeben Sie die int-Konstante JOBSTART_DIR_ALL für beide Richtungen: (Senden(JS) und Empfangen (RE)) oder die Konstanten JOBSTART_DIR_SND; JOBSTART_DIR_RCV für nur eine Richtung.

Lesen eines Jobstarts

Mit der Methode `getJobStart` können die Daten eines einzigen Jobstarts mit einer bestimmten ID gelesen werden. Es wird ein Objekt der Klasse `JobstartFilter` zurückgegeben.

Definition:

```
JobstartFilter getJobstart(UserProfile user, String id)
```

Parameter:

`id`	ID des Jobstarts, für welchen die kompletten Daten gelesen werden.

Erzeugen eines Jobstarts

Ein neuer Jobstart kann mit der Methode `createJobstart` erzeugt werden. Als Eingabe-Parameter dienen die Daten enthalten in der Klasse `JobstartFilter` und Rückgabewert der Methode ist ein Jobstart vom Typ `JobstartFilter`.

Definition:

```
JobstartFilter createJobstart(JobstartFilter jobstart,
UserProfile user)
```

Parameter:

`jobstart`	Parameter eines Jobstarts, der erzeugt werden soll. Folgende Parameter sind in diesem Objekt enthalten: id des Jobstarts, direction (true=send; false=receive), SID, VDSN, Prozess.

Ändern eines Jobstarts

Mit `updateJobstart` werden die Parameterwerte eines Jobstarts aktualisiert und in rvs® gespeichert.

Definition:

```
void updateJobstart(UserProfile user, JobstartFilter
jobstart)
```

Parameter:

jobstart	Parameter eines Jobstarts, der geändert werden soll. Folgende Parameter sind in diesem Objekt enthalten: id des Jobstarts, direction (true=send; false=receive), SID, VDSN, Prozess.

Löschen eines Jobstarts

Mit `deleteJobstart` wird ein Jobstart aus rvs® gelöscht.

Definition:

```
void deleteJobstart(UserProfile user, JobstartFilter
jobstart)
```

Parameter:

`jobstart`	Parameter eines Jobstarts, der geändert werden soll. Folgende Parameter sind in diesem Objekt enthalten: id des Jobstarts, direction (true=send; false=receive), SID, VDSN, Prozess.

Lesen der Parameterliste

Die Parameterliste kann über die Methode `getParameterSet` gelesen werden. Sie gibt ein Objekt der Klasse `ParameterSet` zurück. Die einzelnen Parametername – Parameterwert – Einträge werden in einer Hash-Tabelle gespeichert, die mit `getCompleteSet()` verfügbar ist.

Definition:

```
ParameterSet getParameterSet(UserProfile user)
```

Ändern der Parameter

Mit `updateParameterSet` werden Parameterwerte von rvs® aktualisiert und gespeichert.

Definition:

```
ParameterSet updateParameterSet(UserProfile user,
ParameterSet params)
```

Parameter:

params	Parameter die geändert werden sollen.

4.6.3 Wie erzeuge ich einen residenten Empfangseintrag?

In dieser Methode wird ein neuer Empfangseintrag mittels des Konstruktors `JobstartReceiveFilter` erzeugt. Wenn eine Datei mit dem virtuellen Dateinamen `TEST2` von der Station `GFR11` eintrifft, wird die batch-Datei `C:\rvs\system\resentr.bat` ausgeführt.

```
/***
  * Performs the RE creation.
  */
    public void createRE()
    {
        //Create RE
        try
        {
            if (verbose) System.out.println("** CREATE RE **");


           //create the RE object with a constructor
            JobstartReceiveFilter         jobstartre         =         new
JobstartReceiveFilter("                     ",                    "GFR11",
"TEST2","C:\\rvs\\system\\resentr.bat");


           configuration.createJobstart(jobstartre, userprofile);

            if (verbose)
            {
                System.out.println("*********************************");
                System.out.println("** RE SUCCESSFULLY CREATED **");
                System.out.println("*********************************");
            }
               }
        catch (Exception e)
        {
            System.err.println("Exception during creating RE: "+e);

            if (e instanceof RvsException)
            {
                if (verbose) System.out.println("** ABORT **");
                System.exit(-1);
            }
            else
            {
                e.printStackTrace();
                if (verbose) System.out.println("** ABORT **");
                System.exit(-1);
            }
        }
    }
```