Technische Universität Wien

DIPLOMARBEIT

GenEdit – A Generic Editor and Tools for Questionnaires

Thema

Ausgeführt am Institut für

Institut für Softwaretechnik und Interaktive Systeme der Technischen Universität Wien

unter der Anleitung von

a.o. Univ. Prof. Silvia Miksch

durch

Martina Osztovits

Name

Langeg. 33
7162 Tadten

Anschrift

Datum

Unterschrift (Student)

Abstract

The objective of this master thesis is to design and develop a generic editor for questionnaires together with some useful add-on tools suitable for many different applications. This generic editor should be able to manage any questionnaires which collect time-oriented, highly structured data regardless of a specific problem. Hence, the questionnaires have to be generated during run-time on demand using various configuration files for the start-up of a specific application. The implementation aims at being as general as possible to be useful for as many problems as possible.
Finally, the generic editor is demonstrated on the case of questionnaires used during a clinical trial which takes place at the Department of Child and Adolescent Neuropsychiatry at the Medical University of Vienna. This psychotherapeutic study analyses alternative therapeutic processes of anorectic girls by collecting a huge amount of highly structured time-oriented data through questionnaires. The generic editor should simplify the management of the questionnaires as well as the collection the data necessary for this study.

Kurzfassung

Im Rahmen dieser Diplomarbeit soll ein generischer Editor für Fragebögen entwickelt werden. Zusätzliche Werkzeuge sollen einen Überblick über die, während der Arbeit mit dem generischen Editor erfassten zeitorientierten und strukturierten Daten, ermöglichen. Da der Editor mit Fragebögen verschiedenster Problemstellungen arbeiten können soll, ist es notwendig, dass die spezifische Anwendung, und auch die Fragebögen und deren Antworten aufgrund verschiedenster Konfigurationsfiles zur Laufzeit erzeugt werden. Es wird darauf abgezielt die Implementation so allgemein wie möglich zu halten, damit das Programm für möglichst viele Aufgabenstellungen eingesetzt werden kann.
Schließlich wird der Einsatz des entwickelten Programms am Fall von Fragebögen, die während einer klinischen Studie der Kinder- und Jugendpsychiatrie der Medizinischen Universität Wien an magersüchtige Mädchen gerichtet werden, gezeigt. Ziel ist es eine einfachere Eingabemöglichkeit, der, während dieser Studie gesammelten zeitorientierten, strukturierten Daten, und eine einfache Verwaltungsmöglichkeit der Fragebögen, zu ermöglichen.

Acknowledgements

This thesis is dedicated to my children, Amanda, Helene, Sebastian, and Sabrina. They taught me that it is vital to live my own life.
Special thanks to Klaus Hinum for his support and supervision and to Silvia Miksch, my supervisor.
I would like to thank my parents, my husband, and my friends for their patience and support. I am especially grateful to Bernd Dupal, who shared his experience in object oriented programming with me.
Many thanks also to Peter Votruba for his advices in Java Programming, especially for his very valuable information about TableLayout, JTables and creating executable JAR files.
Thanks to all, who contribute their knowledge by publicizing their ideas in the World Wide Web and especially to all the developers of free software and tools.
Special thanks to Herman Suppan for his advices and worldly wisdoms.

Contact Information

Martina Osztovits
Langeg. 33
7162 Tadten
Österreich

E-Mail: m.osz@gmx.at

Contents

Overview 7

I. PROBLEM ANALYSIS 8

1 Problem Description and Goal 8

2 Concepts and Definitions 8
2.1 Structural vs. Measurement Data vs. Parameter Sources 9
2.1.1 Parameter Source 9
2.1.2 Structural Data 9
2.1.2.1 Form Group 9
2.1.2.2 Form or Questionnaire 10
2.1.2.3 Parameter, Question 10
2.1.2.4 Allowed Values 10
2.1.3 Measurement Data 10
2.1.3.1 Answer Set Information 11
2.1.3.2 Answer Details 11

II. DESIGN AND IMPLEMENTATION 12

3 Structural Data 12

4 Measurement Data 13

5 Data Source 14
5.1 EER of the Data Source 14
5.1.1 The Form Group 15
5.1.2 The Form or Parameter Group 15
5.1.3 Parameter 16
5.1.4 Allowed Value 16
5.1.5 Additional Field 17
5.1.6 Allowed for Additional Fields 17
5.1.7 Parameter Source 18
5.1.8 Measurement 18

6 Required Functionality 19
6.1 Design of Forms 19
6.2 Maintenance of Parameter Sources 20
6.3 Recording Parameter Sets and Parameters 20
6.4 Quick Recording of Head Values without Opening an Record 20
6.5 Extras 20
6.6 Descriptive Statistical Data 20

7 Design Tradeoffs 21
7.1 Database vs. XML file format 21
7.2 Single User vs. Multiple User 22
7.3 Mutual Exclusiveness for Design and Record Data tasks 22
7.4 User Management 23
7.5 Programming Language and Tools Used 23
7.5.1 Consideration for Choosing an XML parser 24
7.5.2 Consideration for Choosing an API for XML processing 24
7.5.3 Document Validation 25
7.5.4 The User Interface 25

8 Packages Structure and Class Implementations 26
8.1 GenericEditor class 27

8.2 Configuration and Profile 28
8.2.1 The Default Time Granularity and How to Change It 30
8.3 Data Model 30
8.3.1 The Overall Data Administration Class – TDataAdmin 31
8.3.1.1 Ensuring Mutual Exclusiveness 33
8.3.2 The Data Administration Class for Form Group Specific Data – TAData 33
8.3.3 Internal Administration of Hierarchical Data 34
8.3.3.1 The Records – RecordData 34
8.3.3.2 The Tables – TableData 35
8.3.4 TableDataIterator 37
8.3.5 A Comparator for Records 38
8.3.6 Additional Head Fields 39
8.3.7 Templates for Setting up the Conversions from Internal to Displayed Representation 40
8.3.8 The value package 41
8.4 Input – Output – physical Data Storage 42
8.4.1 The Storage Manager Concept for Tables 42
8.4.1.1 Abstract Storage Manager 43
8.4.1.2 Storage Managers for Parameter Sources 45
8.4.1.3 Storage Managers for Measurement Data 46
8.4.1.4 The Storage Manager for Forms 48
8.4.2 The Importer 48
8.4.3 The Exporter 49
8.4.4 The GenEdit Print Utility 50
8.4.5 Helper Methods for IO Access 51
8.5 The User Interface 53
8.5.1 The Main Window 53
8.5.2 The Record Data Window 55
8.5.2.1 The Date Spinner 57
8.5.2.2 The Custom ListCellRenderer 58
8.5.3 The Design Form Window 58
8.5.4 The Parameter Source Maintenance Window 61
8.5.5 The Filter / Search Dialog 62
8.5.6 Export Dialog 65
8.5.6.1 Common Export Options 65
8.5.6.2 Export Dialog 66
8.5.6.3 Dialog for Exporting and Starting an External Program 67
8.5.6.4 Export Chooser 68
8.5.6.5 The Modified File Chooser 69
8.5.7 Import Dialog 70
8.5.8 Table Sorter 71
8.6 The Custom Event of the Generic Editor 72
8.6.1 Data Change Event 73
8.6.2 Data Change Listener 73
8.7 Utilities 73

III. DEMONSTRATION OF THE TOOL AND CONCLUSION 74

9 Illustration for a clinical trial about anorectic girls 74
9.1 Getting Started 74
9.1.1 Setting up the Configuration File 74
9.1.2 The Parameters of the Configuration File 74
9.1.3 The Default Configuration File in German 78
9.1.4 The English Configuration File Used for this Documentation 79
9.1.5 The Language Specific Translations 80
9.2 The Command Line Parameters 81
9.3 Starting the Program 81
9.4 Using the Generic Editor 82
9.4.1 Recording Data 82
9.4.1.1 Create a New Record 84
9.4.1.2 Select an Existing Record 84
9.4.1.3 Entering Answers 84

9.4.1.4 Searching for a Record / Specifying Values of Head Fields without Opening the Record 84
9.4.1.5 Save the Answers 85
9.4.1.6 Resetting the Answers 85
9.4.1.7 Deleting the Answers 85
9.4.1.8 Printing the Answers 85
9.4.1.9 Close a Record / Work with another Record 85
9.4.1.10 Sorting the Answers 85
9.4.1.11 Navigation through the Records 85
9.4.1.12 The Shortcuts and Icons for the Tasks 85
9.4.2 Designing Forms 86
9.4.2.1 Creating a new Form 88
9.4.2.2 Opening an Existing Form 88
9.4.2.3 Specifying the Form’s Parameters and the Questions 88
9.4.2.4 Searching for a Form / Specifying Values of Head Fields without Opening the Record 90
9.4.2.5 Save the Form 90
9.4.2.6 Resetting the Form 90
9.4.2.7 Copying a Form 90
9.4.2.8 Deleting the Form 90
9.4.2.9 Printing the Form 90
9.4.2.10 Previewing the Form 90
9.4.2.11 Close a Record / Work with Another Record 91
9.4.2.12 Navigation through the Records 91
9.4.2.13 Sorting the Forms 91
9.4.2.14 Deleting All Answers to a Form 91
9.4.2.15 The Shortcuts and Icons for the Tasks 91
9.4.3 Maintaining Parameter Sources 92
9.4.3.1 Creating a New Parameter Source 93
9.4.3.2 Delete 93
9.4.3.3 Saving Changes 94
9.4.3.4 Reset 94
9.4.3.5 Close 94
9.4.3.6 Delete All Answers of a Parameter Source 94
9.4.3.7 The Shortcuts and Icons for the Tasks 94
9.4.4 Extras 94
9.4.4.1 Exporting Data 94
9.4.4.2 Importing Data 98
9.4.4.3 Exporting and Automatically Start of an External Program 98
9.4.5 Searching / Filtering the Data 100
9.4.5.1 The Fields of the Filter Table 101
9.4.5.2 Assigning Values to Head Fields 101
9.4.5.3 Saving the table 101
9.4.5.4 Closing the Search/Filter Dialog 101
9.4.5.5 The Shortcuts and Icons for the Tasks 102

10 Future perspectives 102
10.1 User Management 102
10.2 Database Management System 102
10.3 Multi User 102
10.4 XML Document Validation 102
10.5 Profile Settings 102

11 Summary 103

IV. APPENDIX 104

1 Definitions 104
1.1 Anorexia nervosa 104
1.2 EER 104
1.3 Parser 104
1.4 Singleton 104

2 The File Formats **105**
2.1 Conflation 105
2.2 CSV (= CSV – multiple row) 105
2.3 CSV – single row 106
2.4 A Single AnswerFile 106
2.5 Multiple AnswerFiles in the AnswerDir subdirecotry 106
2.6 XML File Format for Forms 107
2.7 XML File Format for Parameter Sources 107

V. TABLES OF 107

1 Figures **107**

2 Tables **108**

3 Codes and Example Files **109**

VI. BIBLIOGRAPHY 110

Literature **110**

Internet Resources **111**

Overview

This thesis is divided into three parts. Part I covers the problem analysis, which includes not only a detailed description of the problem and the goal, but also analyzes the data involved. Part II deals with design and implementation issues. It covers in detail the package structure and the classes, the underlying data model and the user interface. Part III demonstrates the application which has been implemented within the framework of this thesis with questionnaires designed for a clinical study about anorectic girls. It focuses on setting up the Generic Editor for a specific task and on user interface details. Furthermore, it covers future perspectives, especially proposals for enhancements.

I. Problem Analysis

1 Problem Description and Goal

The purpose of this thesis is to develop a generic editor for questionnaires. It is generic in the sense that the questionnaires are not hard coded. Instead, the questionnaire definitions will be loaded during the program start. While the main field of operation is for questionnaires which are used during a study about anorectic girls at the Department of Child and Adolescent Neuropsychiatry at the Medical University of Vienna, the generic editor is designed sufficiently general to cover questionnaires for virtually any field of operation.
The generic editor has to cover three main tasks, which are:

- Maintenance of questionnaires
- Maintenance of parameter sources (i.e. repliers of questionnaires)
- Maintenance of answers

Regarding its use during our study about anorectic girls, it is a tool, which records the data needed in the in2vis project. In2vis is short for Interactive Information Visualization. The in2vis project aims at exploring and supporting human reasoning processes and takes place at the Institute of Software Technology and Interactive Systems. On 1st of July 2004 a team headed by Silvia Miksch started the in2vis project. During the in2vis project some tools were developed which aim at visualizing huge amounts of abstract, but highly structured data. One of these tools is Gravi++, which was developed by Klaus Hinum. In his thesis he proposes an interactive Information Visualization called Gravi++, which aims in visualizing highly structured, temporal, categorical data. “It integrates a spring-based core visualization to display the multidimensional data set.” [Hinum 2006 p.3] Further details on Gravi++ can be found in [Hinum 2005] and [Hinum 2006].
Even before the in2vis project, during the LinkVis project a tool was created which aimed at evaluating and visualizing psychotherapeutic processes. Both projects use a common data source, called the Conflation file format (see chapter IV.2.1 for information on the Conflation file format). Further details on the LinkVis project can be found in [Herzog 2004]. Furthermore, Stardinates were developed in 2003 by Monika Lanzenberger (see also [Lanzenberger 2003 1], [Lanzenberger 2003 2] and [Lanzenberger 2003 3]). “The Stardinates are a novel interactive Information Visualization (InfoVis) technique which aims at visualizing highly structured data. They represent some Gestalt principles very well, especially the principles of Closure and ’Prägnanz’. As a consequence, Stardinates form very distinct and memorable patterns which make abstraction and aggregation much easier” [Lanzenberger 2003 1]. Stardinates, LinkVis, and Gravi++ use common file formats. A main aim of the generic editor is to be able to read the existing file formats and export the data in the Conflation File Format, so that Gravi++ is capable of visualizing the recorded data.

Another aspect is that master data and variable master data of patients are conceptually handled the same way as ordinary questionnaires. Furthermore, there is a special focus that it is easy to enhance the editor.

2 Concepts and Definitions

We will now introduce the important concepts, which will be used throughout the rest of this thesis. The data concept introduced in [Herzog 2004] is enhanced by form groups. This is necessary to able to cover time-oriented data, not time-dependent data and data which requires additional information conceptually the same way.

2.1 Structural vs. Measurement Data vs. Parameter Sources

It is important to distinguish between structural and measurement data and parameter sources. This concept can also be found in [Herzog 2004]. According to his thesis, the data can be divided into two parts, on the one hand the master data, which describe the questions and questionnaires in detail and on the other hand the measurement data, which are the answers to these questions.
I want to stretch this concept and categorize the data in three parts, which are structural data, measurement data, and parameter sources. This is necessary to be able to treat all the parameters which are collected for each parameter source conceptually the same way and also to provide maintenance of parameter sources.

2.1.1 Parameter Source

As in [Herzog 2004], a parameter source keeps information about the source of measured values. This can be for example a person answering a questionnaire, but it is not restricted to this. In order to keep the program as general as possible, the program only stores a unique identifier for each parameter source. The kind of the parameter source needed for a concrete application is specified in the application’s profile. The profile configures the generic editor for a concrete application. For example, in the context of our study about anorectic girls, the parameter sources would be patients. The parameter source maintenance keeps only track of the parameter source ids. All other data which would normally be recorded to describe a parameter source are treated like all other parameters and are therefore, measurement data. E.g. the name of a patient is treated like any other measurement data and is not closely associated with the parameter source entity, i.e. it is not an attribute of the parameter source entity.

2.1.2 Structural Data

Structural data are all data which are necessary for keeping track of the structure of a questionnaire. Especially, these are the identifier of the form, optionally a description of the form and the number of parameters of the form and the details for each parameter which belong to this form. These data are likely to be set up only once and are not likely to be changed once they are set up. They reflect the structure of the form.

2.1.2.1 Form Group

A form group is a group of questionnaires, which have certain properties in common or which are arbitrarily grouped together by the user. But there is one restriction in arbitrarily grouping questionnaires: The questionnaires have to be similar in:

- Time-dependence and time granularity
- Requirement of the same additional information

To group forms into distinct form groups is merely of interest when recording answers. The definition of questionnaires does not depend on the form group the form belongs to. However, the problem is how to uniquely identify an answer (or measurement) set. E.g. master data is not time dependent. Therefore, the parameter source id and the form id are sufficient for identifying an answer set within a form group that contains non time dependent data. Other data is time dependent, e.g. questionnaires which are answered multiple times during a therapy. These answer sets require additionally, a time field to uniquely identify an answer set. Furthermore, it might be necessary to specify further key values for identifying an answer set uniquely – e.g. a parameter source flag to store whether the questionnaire has been answered by the patient, his father, his mother, or his therapist. And it might be useful to be able to specify further non – key values- e.g. for categorizing the answer sets. An example would be the qualitative time, which categorizes each answer into different qualitative times, e.g. pre for previous to the therapy, post for posterior to the therapy etc.

The number of form groups and the detailed specifications of each form group are set up during a configuration process. This ensures that the generic editor can be used in different fields of operation.

2.1.2.2 Form or Questionnaire

A form consists of a set of questions, which belong together. Each form is uniquely identified within a form group by its form identifier.
Furthermore, a description of the form can be specified.

2.1.2.3 Parameter, Question

A parameter is a criterion, which the user wants to observe. It is identified within a form by its parameter-id, also called question-id.
Besides the parameter-id, there is a number of other structural information necessary:

- Parameter text: the formulation of the question
- Data type of the answers
- Minimum value: optionally
- Maximum value: optionally
- Mean value: optionally

2.1.2.4 Allowed Values

In addition to specifying a minimum or a maximum value, the user can also restrict the data range of a question by specifying a list of allowed values. Each allowed value has the following fields:

- Id: The actually allowed value. It is unique within each parameter (question) and a key field.
- Description: optionally a description for each value.

2.1.3 Measurement Data

During the life time of the application, the most common task is to collect values to the parameters set up in the design phase. The values collected are the measurement data of the program. These data are also called "measurement data" in [Herzog 2004, p.16]. Considering the special case of patient data as described in [Herzog 2004, p.16], it is necessary to distinguish time oriented data (which is recorded frequently) and static data:

"The operative data consist of answers of the questionnaires and information about the patients. The information about the patients are more static and don't change every period. The answers of the questionnaires will be recorded frequently. The frequency of the data recording depends on the demands of the researcher." [Herzog 2004, p.16]

Time oriented data are collected multiple times for each patient, whilst static data is usually collected only once. For the editor this means that it must capable of either keeping the information when the record was recorded as a key field or omitting the date information. So it is necessary to divide measurement data into different groups which are called form groups. Further information on form groups can be found in section I.2.1.2.1.

Furthermore, one has to distinguish between measurement set information and collection of measurement details.

2.1.3.1 Answer Set Information

Similar to the form definition, some information is needed to identify a set of measurement values which belong together and possibly categorize or describe them. During this work, I call these information head fields. There are some head fields which are necessary to uniquely identify an answer set record. These are key fields to the answer set record. The number and kind of head fields which are necessary, depend on the form group. It is also possible to introduce additional key fields. Optionally, it should be possible to somehow describe or categorize each answer set. For this purpose the user of the application can introduce additional non-key head fields for a form group.

In the case of static data (e.g. master data of a patient), one needs the following key fields

- “Parameter Source Id” which specifies the parameter source (e.g. the patient’s id). This is actually a reference.
- “Form Id” which specifies the form used. This, too, is actually a reference to the form definition

In order to be able to collect time oriented data, the following field is necessary

- “date of recording”

Under some circumstances it is necessary to keep track of additional key fields. E.g. for our study about anorectic girls, each questionnaire can be either answered by a therapist, the mother, the father or the patient. So, one must be able to set up an additional key field called “Parameter Source Flag”. It additionally stores for each patient who answered the questionnaire (child, mother, father etc).

In order to be able to somehow categorize or describe each record, the program should be capable of defining further additional head fields which do not belong to the key of the record. In the case of our studies about anorectic girls, this can be for example a qualitative timestamp, which categorizes each measurement date into

• pre	previous to the therapy
• kat 1	first measurement during the therapy
• kat 2	second measurement during the therapy
• kat 3	third measurement during the therapy
• kat 4	fourth measurement during the therapy
• post	after the therapy

Such a categorization of data is convenient, e.g., to be able to better compare recorded data of different parameter sources or of different forms. First of all, the measurement dates need not be the same although they would belong to the same qualitative time category. Vise versa there can be measurements originated by different parameter sources which took place at the same date, but do not belong to the same category, e.g. a patient who has already finished the therapy and a patient who has not yet started the therapy answered one and the same questionnaire at the same date.

2.1.3.2 Answer Details

The answer details are the concrete measured values of parameters. It has one key which is the “parameter-id”, also called question-id, which is a reference to the parameter definition and a non-key field keeping the recorded value, which is the concrete measured value.

II. Design and Implementation

3 Structural Data

The current version of the program stores the structural data of the forms in single XML files. The name and the location of the files are to be specified in the application's profile. The details of the file format used for storing the form's structural information can be found in the appendix. The following details are needed and have a natural hierarchical structure, which is nearly directly mapped into the tags and attributes of the XML file. Details about the hierarchical structure can be found in the EER of the data source (see section 5.1). The format is the nearly the same which was already used by LinkVis (see [Herzog 2004]). However, it has been enhanced by an additional field to be able to keep a description of the form in addition to its form-id. Following the necessary information is listed.

- Form-id: must be unique
- Form description
- The questions:
    - The question-id
    - Optionally, the question-text
    - The data type of the answers: allowed values are: integer, float and string. The default value is integer.
    - Optionally, the allowed minimum value
    - Optionally, the allowed mean value: according to [Herzog 2004] this is a normal value. In future releases it might be used as default value during the recording of values. In the current version, it is only specifiable, but not used anywhere else in the program.
    - Optionally, the allowed maximum value
    - Optionally, a display-as information field: this field is new and specifies how the answer field should be displayed during the recording data of this form. Allowed values are: "radiobuttons", "combobox" or "textfield".
    - Optionally, a list of allowed values:
        - The value: It must be unique for each question.
        - Optionally, a description of the value

Structural data can be edited during a design process, which can be started at any time – unless it is blocked by another open window (e.g. if already a recording of measurement data takes place for the same form group). Following, a part of an XML file keeping structural information is listed:

```
<parameter_group count="1" description="" id="ASW">
<parameter data_type="integer" display_as="" id="1" max_value="4"
mean_value="" min_value="1" name="Wenn sich Widerstände auftun, finde ich
Mittel und Wege, mich durchzusetzen">
<allowed_values>
<allowed value="1"/>
<allowed value="2"/>
<allowed value="3"/>
<allowed value="4"/>
</allowed_values>
</parameter>
</parameter_group>
```

Code 3-1 Example of Parameter Definition

In the example of Code 3-1 a parameter group with one question (count="1") is identified. Its id is "ASW", the description is empty. The question allows only integer answers within the range 1 to 4 (min_value and max_value). Its normal value is undefined. The question name, which will be displayed during the recording of measurement data, is: “Wenn sich Widerstände auftun, finde ich Mittel und Wege, mich durchzusetzten”. Furthermore, it defines a list of allowed values which are 1, 2, 3, and 4.

4 Measurement Data

Currently, there are two options for storing measurement data. The first one is to use a single AnswerFile – one for each of the form groups. The other is to use a directory structure, which is common to the one used in former tools. Details to the format of the Answer file can be found in the appendix IV.2.4. It merely wraps another tag around the answer sets.
The directory method uses a single answer file for each of the answer sets. It is created in a subdirectory of the answer directory, which is specified in the profile. The subdirectories name is the replier-id (normalized to not contain characters invalid for the operating system or leading to a malformed URL). The file name is the concatenated the replier-id, the form-id, and the date in format DDMMYYYY. Of course, again the replier-id and the form-id must not contain any characters invalid for the operating system and need normalization. Should some values lead to duplicate filenames, the program tries to append an underscore followed by a number.

Each answer set has a hierarchical structure:

- The form-id of the form to which the questions belong
- The parameter source id
- Depending on the form group: a date
- Depending on the form group: additional head values
- The answers to the questions:
    - The question-id
    - The value

Both file formats reflect this hierarchical structure. Because the first solution stores the data in a single file, the advantages of it are, that there is no need to check for characters in form-ids or replier-ids which lead to for a specific operating system invalid file or directory name or malformed URL and that it is easy to save a backup copy of a single file. The file name used is set up during the configuration process (in section 8.2), so the user is responsible of choosing a valid filename. The disadvantage is that the file can get huge in a short time and might exceed the maximum allowed file size of an operating system. Furthermore, it is more difficult to include or exclude answers from being loaded by the editor. On the contrary, the directory structure solution provides an easy way for including or excluding answers – they simple need to be moved somewhere else in the file system and won’t be read any more during the initialization process of the program. But, some values for parameter sources or form ids might lead to invalid filenames or to duplicate filenames. In order to prevent problems regarding this issue, the generic editor, normalizes the filename, by removing all the invalid characters for a specific operating system. Should this lead to an empty string, the directory for the replier is hard coded to “EMPTY” and all files of such parameter sources will go in this subdirectory. Should this conversion lead to a duplicate filename the generic editor tries to add an underscore followed by a number. If the program does not succeed after 100 tries, the user is asked to specify a unique filename. Following is an example of a part of an answer file:

```
<linkvis_data_records parameter_group_id="TEST" parameter_source_id="0"
day="21" month="11" year="2006">
  <parameter id="1" value="5"/>
  <parameter id="2" value="6"/>
</linkvis_data_records>
```
Code 4-1 Example of Measurement Data

The example in Code 4-1 shows a single answer to the form with id “Test” (parameter_group_id). It was answered by the parameter source with the id “0” on 21st of November 2006 (day, month, year). The answer to question “1” was “5” and the answer to question “2” was “6”. There is no need to store further information on the parameter definition in the answer file, because the parameter_group_id and the id (of the parameter) refer uniquely to the parameter definition and it is clear from the location of the answer file, which form group the answered form belongs to. The location (subdirectory) of each form group’s answer files, when using the answer directory storage method for this form group, has been set up during the configuration process. Details about the configuration process can be found in section 8.2.

5 Data Source

The above concepts lead to a couple of tables. Each table can have simple records, or records, which are references to other tables. The following tables are necessary.

- Form group table
- Form table: keeps information about the available forms
- Question table: keeps the question detail information
- Allowed values table: keeps the values, which are allowed for the questions
- Parameter source table: keeps the available parameter sources
- Measurement table: keeps the available measurements

Depending on the form group, the kind and number of the answer set information varies. The goal is that the record data process does not need to know about its form group.

5.1 EER of the Data Source

Following, an entity relationship model (see Figure 5-1) is used to provide a high-level description of the conceptual data model. Obviously, the analyses of the needed data sources lead to nearly the same EER, which can be found in [Herzog 2004 p. 44].

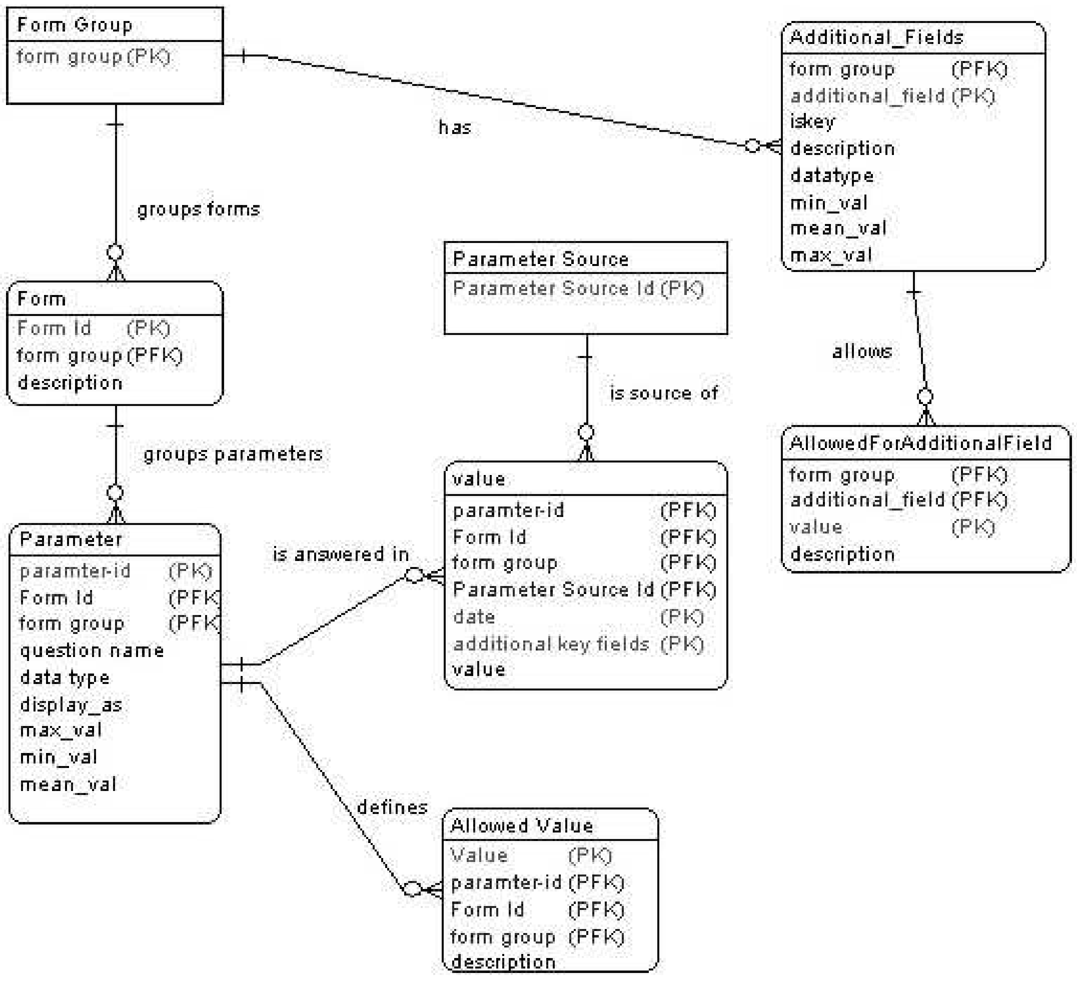


Figure 5-1 EER of the Data Source. The available data is grouped into different entities.

The entities themselves are similar to those described in [Herzog 2004]. Actually there is no change for the structural data, besides there can be multiple form entities, because forms are now a sub entity of form group.

5.1.1 The Form Group

The form group collects forms. Form is a synonym for parameter group as defined in [Herzog 2004]. Table 5-1 lists properties of the form group entity.

Entity			form_group
Field	**Key**	**Data Type**	**Description**
id	Primary	String	Identifier of the form group

Table 5-1 Form Group Entity. The form group entity keeps track of the form groups.

5.1.2 The Form or Parameter Group

Again as in [Herzog 2004 p.47]: “The parameter group collects one or more parameters. The parameter group is a synonym of the questionnaire and the parameter is a synonym of a question.” This entity stays nearly the same and in [Herzog 2004], however an additional form group identifier is needed:

Entity			parameter_group
Field	**Key**	**Data Type**	**Description**
form group	Primary	String	Identifier of the form group
id	Primary	String	Identifier of the parameter group
description		String	A description of the parameter group.

Table 5-2 Parameter Group Entity. The parameter group entity keeps track of the forms.

Again, the id is often referred to as parameter-group-id (or form-id). This entity was enhanced by a description for each form. Table 5-2 lists the properties of the parameter group entity.

5.1.3 Parameter

Table 5-3 lists the properties of the parameter entity:

Entity			parameter
Field	**Key**	**Data Type**	**Description**
id	Primary	String	Identifier of the parameter
group_id	Primary	String	Identifier of the parameter group
form group	Primary	String	Identifier of the form group
display_as		String	The way how data entry should be possible, when recording measurement values
data_type		String	Data type of the parameter
min		Depends on the data type, therefore, stored as String	Allowed minimum
max		Depends on the data type, therefore, stored as String	Allowed maximum
mean		Depends on the data type, therefore, stored as String	The normal value

Table 5-3 Parameter Entity. The parameter entity keeps track of the properties of a single parameter.

There is a difference to [Herzog 2004], which is, that the min, max and mean values are stored as strings instead of as decimals. This is necessary to be able to define minimum, mean, and maximum values for other data types than numbers, too. Everything else stayed the same than in [Herzog 2004], besides, that Herzog missed out the mean value in his table.

5.1.4 Allowed Value

This entity keeps track of the values allowed for a particular parameter. The allowed values need not be specified. If there are not any allowed values defined and no min and no max value defined (see section 5.1.3 about parameter) than all parameters of the specified data type are allowed.
The properties of this entity stay the same than in [Herzog 2004], besides, the data type of the value, and an additional field, which can be used to store a text description of the meaning of a parameter value (e.g. 1 = easy, 2 = medium, 3 = difficult for a question like: “Do you think, it easy to read?”) Table 5-4 lists the properties of the properties of the allowed_value entity.

Entity			allowed_value
Field	**Key**	**Data Type**	**Description**
form group	Primary	String	Identifier of the form group
group_id	Primary	String	Identifier of the parameter group
param_id	Primary	String	Identifier of the parameter
value		Depends on the data type, therefore, stored as String	One of the allowed values
description		String	A description of the value

Table 5-4 Allowed Values Entity. The allowed value entity keeps track of the allowed values

5.1.5 Additional Field

This entity defines the additional fields, which are needed for a particular form group. The fields defined in this entity enhance the number of necessary (key and non-key) fields of the measurement entity. It is comparable to a parameter definition (see section 5.1.3), but on the contrary to a parameter definition, it defines an attribute to a measurement set rather than to single measurement value. Therefore, it defines the same attributes than a parameter definition. The properties of the additional field entity are listed in Table 5-5:

Entity			additional_field
Field	**Key**	**Data Type**	**Description**
id	Primary	String	Identifier of the additional field
form group	Primary	String	Identifier of the form group
field name		String	A description of the field
is_key		Boolean	Whether, it is a key field to measurement sets of this form group
data_type		String	Data type of the parameter
display_as		String	The way how data entry should be possible, when recording measurement values
Min		Depends on data type, therefore, stored as String	Allowed minimum
Max		Depends on data type, therefore, stored as String	Allowed maximum
Mean		Depends on data type, therefore, stored as String	The normal value

Table 5-5 Additional Field Entity. Additional fields are used during the loading process to keep track of additional head fields. These are only used to set up the answer set table

5.1.6 Allowed for Additional Fields

Defines allowed values, if any restrictions of the data range are needed for an additional field. It has the same function than the allowed value entity (see 5.1.4). Its properties can be found in Table 5-6

Entity			allowed_for_additional
Field	**Key**	**Data Type**	**Description**
form group	Primary	String	Identifier of the form group

additional_field	Primary	String	Identifier of the parameter
Value		Depends on data type, therefore, stored as String	One of the allowed values
Description		String	A description of the value

Table 5-6 Allowed for Additional Entity. This entity keeps the values allowed for each additional field

5.1.7 Parameter Source

Unlike in [Herzog 2004], the entity parameter source only keeps track of the parameter source ids. If there is further key information necessary for the measurement entity than the further key fields have to be defined in the profile (see section 8.2). The properties of this entity are listed in Table 5-7:

Entity			**parameter_source**
Field	**Key**	**Data Type**	**Description**
id	Primary	String	Identifier of the parameter source

Table 5-7 Parameter Source Entity. The parameter source entity keeps a list of available parameter sources

Nevertheless, the parameter sources are stored separately to avoid insert, update or delete anomalies. An insert anomaly would be the need to keep at least one pseudo measurement in order to create a new parameter source. A delete anomaly would occur, if all measurement data for a parameter source table would be deleted, than also the parameter source id would be lost. An update anomaly could occur, if there is a need to update a single attribute which is dependent on the parameter source id multiple times, because it is stored with a number of records. The update anomaly can still occur, but only if the user designs the questionnaires badly. So the user is responsible for designing the questionnaires carefully.

5.1.8 Measurement

The measurement entity keeps hierarchical information. It keeps all records which are available for each person and each form and also the detailed values of all the other necessary parameters. Depending on the additional fields, it might have one or more additional (key) fields (see section 5.1.5). In the case of our study about anorectic girls the parameter_source_flag is such an additional key field. It does no longer belong to the parameter source entity, but instead became an attribute of the measurement. Furthermore, non time dependent parameters (e.g. the master record of a patient) do not require a date key field. The detailed definition of the measurement entity for a given form group is, therefore, unknown until the profile (see section 8.2) is read. Table 5-8 show the general outline of measurement entity:

Entity			**Measurement**
Field	**Key**	**Data Type**	**Description**
form group	Primary	String	Identifier of the form group
group_id	Primary	String	Identifier of the parameter group
param_id	Primary	String	Identifier of the parameter
source_id	Primary	String	Identifier of the parameter source

Entity			Measurement
Field	**Key**	**Data Type**	**Description**
additional_key 1	Primary	Depends on the data type definition of the additional field in the additional key entity, therefore, stored as String	
…	…		
additional_key n	Primary	Depends on the data type definition of the additional field in the additional key entity, therefore, stored as String	
additional_nonkey 1		Depends on the data type definition of the additional field in the additional key entity, therefore, stored as String	
…			
additional_nonkey 1		Depends on the data type definition of the additional field in the additional key entity, therefore, stored as String	
date		Date	The date of the measurement
description		String	A description of the value

Table 5-8 Measurement Entity. The measurement entity keeps the actual measurement values and refers to the structural data as well as to the parameter source.

6 Required Functionality

Three main functionalities are needed, which are the design of forms, the maintenance of parameter sources, and the recording of values. Additionally, the program comes with some extra functionality for exporting and importing data and displays some descriptive statistical data.

6.1 Design of Forms

The user is provided with options to:

- Create a new form: The user can either create a new form from scratch or copy from an existing form. Copying from an existing form will also copy all its questions, including their allowed values.
- Open an existing form
- Edit the form and question definitions. This includes also the creation and deletion of questions.

- Save a form
- Print a form
- Preview the recording mode using the form he currently designs.
- Delete a form

All these functions require that a list of available forms is dynamically created by reading the form definition from the data source. A design trade-off is that the form definitions are read from the data source only at program start up and are kept in an internal data structure afterwards. This has the advantage that the time demanding read from the physical storage is done only once. So, the performance is improved.

6.2 Maintenance of Parameter Sources

This only requires maintaining the parameter source ids, because all other information to parameter sources is to be handled conceptually the same than answers to questionnaires. E.g., the name of the patient would be a parameter which is answered by and recorded for this patient. So, the user can set up a form group keeping all static data (or master data) of a patient and a form which contains the patient's name as one of the questions. The answers to forms of this form group do not require a date field which is part of the key, because it stores static data.

6.3 Recording Parameter Sets and Parameters

The program enables the user to add, check and delete answer values. The answers to parameters form an n:m relation between the parameter sources and the parameter. Additionally, the interest is rather on answers to a specific form than to single questions. Therefore, it, again, makes sense to use the form to group the questions to be answered together. The record data window reflects this n:m relation.

6.4 Quick Recording of Head Values without Opening an Record

Sometimes it is useful to be able to quickly specify non – key head values for each record. Therefore, the user is provided with this functionality by displaying a list of all records without their details. The user can than key in all the non – key head values which are not explicitly read-only in this list, without explicitly selecting and opening the record for editing. This is a useful facility, so it has been implemented in a general way in order to be able to use it for each kind of hierarchical data. Therefore, the same functionality can be used during the recording of data as well as during the design of forms. An example of the use of this facility can be found in III.9.4.5.2.

6.5 Extras

Extras provides import and export facilities. The generic editor is able to import two different CSV formats and to export to the following file formats:

- The Conflation file format (see IV.2.1)
- CSV format multiple rows (see IV.2.2)
- The AnswerFile format: this has been implemented for convenience to be able to migrate from an AnswerDir setup to an AnswerFile setup. (see IV.2.4)

Furthermore, it is possible to export to a data format specified in the profile and start the program specified in the profile thereafter. This feature was required to ease the data analyzing for the clinical personal, which is done with Gravi++.

6.6 Descriptive Statistical Data

Some basic descriptive statistical data is calculated and displayed at program start up. These include the number of available parameter sources, the number of available form groups, the number of forms of each group, the number of answers of each group and the last changes of each of them. The parameter source maintenance also displays the number of available answer for each parameter source. Similarly, the design of forms, displays the number of answers available for a form. A screen shot of the start window can be found in III.9.3.

7 Design Tradeoffs

7.1 Database vs. XML file format

There is a trade-off between using XML files, flat file formats or databases as data sources.

Database Management Systems: “A DBMS is a complex set of software programs that control the organization, storage and retrieval of data in a database” [WWW-2]. They have the following advantages/disadvantages, though not all DBMS must have all the features listed below.

Advantages:

- **Concurrency** - DBMS provide various tools and techniques to deal with concurrency, e.g. transaction management and locking.
- **Backup and Replication**: DBMS usually provide backup and replication tools.
- **Rule Enforcement**: Most DBMS provide facilities to set up rules, which must be fulfilled by the attributes or records stored.
- **Security**: One can easily define access rights.
- **Computation**: Some common computations requested on attributes such as counting, summing, averaging, sorting, grouping, cross-referencing are provided by the DBMS, therefore, a computer application which uses a DBMS, does not need to implement these from scratch.
- **Change and Access Logging**: Logging services keep a record of access occurrences and changes.
- **Automated optimization**: If there are frequently occurring usage patterns or requests, some DBMS can adjust themselves to improve the speed of those interactions. In some cases the DBMS will merely provide tools to monitor performance, allowing a human expert to make the necessary adjustments after reviewing the statistics collected.
- **Meta-data Repository**: Meta-data e.g. descriptions of attributes or rules.
- **Modeling Tool**: A DBMS can also act as a modeling tool.

Disadvantages:

- **Installation:** A database management must be set up at the user site.
- **Configuration**: The database management system requires a complex configuration
- **Maintenance:** An expert is needed at the user site to maintain the DBMS
- **Data Inclusion/Exclusion:** There is no easy way to include/exclude data → Directory Storage manager for Answers (see section 8.4.1.3.)

A set of XML Files:

These have the following advantages:

- They are readable and editable by a simple Editor
- Existing data are available in XML file format
- Data Inclusion/Exclusion: is depending on the organization of the XML files relatively easy
- Still they provide a hierarchical structure.

- Robustness (compared to simple text files): In XML each datum is marked up with what it means. Therefore, it is robust against flipping around information. (see also [Harold 2002])
- Extensibility: One can easily provide additional information by simple adding an extra element. (see also [Harold 2002])
- Ease-of-Use: There are already a couple of XML parsers available. The parser shields from a lot of details, which are irrelevant for an application (see also [Harold 2002]), e.g.:
  - Encoding
  - Line separation character(s)
  - The way how reserved characters are escaped
  - Byte order of the underlying system
- No installation needed, if the default API of JAVA is used.
- Allows document validation by parser.

Disadvantages:
- Poor performance compared to DBMS
- All DBMS tools and mechanisms are missing. So these issues are to be programmed in the application, if needed.

A set of flat file formats, e.g. of CSV format:

Advantage:
- They are readable and editable by a simple Editor

Disadvantages:
- They don't provide any hierarchical structure
- They are not as robust, extensible and easy to use than XML files.

Former works led to existing data sources, which are in XML format. A main focus of this work is to be able to work with the existing data sources. Therefore, the current version of the generic editor uses a set of XML files, which are described in more detail in chapter IV.2. In order to be able to easily enhance the application, the objects providing data accesses have been bundled and encapsulated. For implementation details see section 8.4.1.
However, the use of XML files means that there have to be some compromises regarding concurrency.

7.2 Single User vs. Multiple User

The current version of the program is a single-user program, which means that only one user can work with the program at a time. In order to ease the program implementation, it has been dispensed with multi-user tasks, because the current application has to deal with XML files. Further releases, might work with a database management system, which makes it much easier to deal with concurrency. Because the database management system already provides mechanism for dealing with concurrency, there is no need to implement such mechanisms from scratch in the application. So, multiple user support is a future issue and not within the scope of this thesis.

7.3 Mutual Exclusiveness for Design and Record Data tasks

In order to prevent data inconsistencies and to avoid the need of locking on a record basis, some tasks may not be started at the same time. For example the design process for a certain form group must not be run at the same time as the recording of data for the same form group. The ensure this they have been made mutual exclusive by an easy locking mechanism. The instance of the TDataAdmin class takes care about the locks. Table 7-1 completely lists which tasks exclude which other task:

Excludes	Parameter Source Maintenance	Design for form group k	Recording of measurements for form group k	Export/Import
Parameter Source Maintenance	Yes	No	Yes	Yes
Design for form group i	No	i = k	i = k	Yes
Recording of measurements for form group i	Yes	i = k	i = k	Yes
Export/Import	Yes	Yes	Yes	Yes

Table 7-1 Mutual Exclusiveness. We list which tasks exclude each other.

The maintenance of parameter sources can be run at the same time as any form design, because they do not interfere. The parameter source maintenance as well as any design process prohibits a recording of data for the same form group. Of course, the user is allowed to open more design windows for different form groups and he is allowed to record data for one form group while designing forms for another form group. Before any of the windows is opened the program ensures, that it is not locked by any other window. If it can be opened, it locks the data according to the scheme of Table 7-1 and thereafter the window is displayed. When closed, the data are released again. Of course, there must not be more than one instance of those classes which implement one of the tasks mentioned above. So they are Singletons in this sense. Detailed information about the Singleton design pattern can be found in [Gamma 1995 pp.127].

7.4 User Management

The current version of the generic editor is a single user version. It has been dispensed with user management tasks, because the tool's main field of operation is during a special clinical study. During this study only one or two users are going to enter the data and design forms etc. Those users are well trained in both the maintenance tasks and the recording of measurement data. Therefore, it was not necessary to implement a user handling in this version of the program. Further releases might well need to deal with user management, because it could be possible, that the "parameter sources" key in their answers themselves instead of filling in a paper-questionnaire, which has to be inputted by somebody else. Of course it would not be desirable that the parameter sources are also allowed to change structural data or maintain parameter sources or look at other than their own answers. This is a future issue and not the scope of this thesis.

7.5 Programming Language and Tools Used

The generic editor was developed in Java, release 1.5. It uses one of the provided standard XML APIs and the parser which comes with Java 1.5. Among other advantages of Java, Java was chosen, because it is an object-oriented, high-level programming language which includes xml support and is platform independent. Further information on advantages of Java compared to other programming languages can be found in [WWW-22], [WWW-23], and [WWW-24]. One of the advantages of Java is also that it is free and that there are free tools

and utilities available. E.g. Eclipse UML Free (see [WWW-25]) is an integrated development environment, which was used during the development process and for creating the UML diagrams in this thesis. The Toad Data Modeler (see [WWW-26]), a free database modeler, was used for creating the EER diagram.
There are a couple of Java XML APIs and parser available, the following section analyzes some of them and explains the decision made.

7.5.1 Consideration for Choosing an XML parser

Choosing a parser library requires looking at many aspects. Parsers differ in how many features they implement, their costs, the APIs they implement, how correct and how fast they work. [Harold 2002] divides parsers into three categories:"

- Fully validating parsers
- Parsers that do not validate, but do read the external DTD subset and external DTD parameter entity references in order to supply entity replacement and assign attribute types
- Parsers that read only the internal DTD subset and do not validate." [Harold 2002]

There is a variety of parsers available, but the Xerces parser is bundled with the JDK 1.5 distribution. It supports both DOM and SAX. Because it is of advantage to be able to work with standard parsers and APIs (see section 7.5.2), Xerces was the parser of choice for the generic editor. Furthermore, Xerces is a validating parser with a very good conformance to XML 1.0. [Harold 2002]. The Definition of XML 1.0 can be found at [WWW-4].

7.5.2 Consideration for Choosing an API for XML processing

According to [Harold 2002] one of the most important decisions at the start of an XML project is choosing an application programming interface (API). [Harold 2002] states that while it is possible to swap in an alternative, if a specific parser causes troubles, often without recompilation the code, changes to the API may well involve redesigning and rebuilding the entire application from scratch. There is a variety of APIs for processing XML documents available. According to [Harold 2002] "there are two major standard APIs for processing XML documents with Java – the simple API for XML (SAX) and the Document Object Model (DOM) … In addition there are a host of other, somewhat idiosyncratic APIs including JDOM, dom4j, ElectricXML, and XMLPULL. Finally, each specific parser generally has a native API that it exposes below the level of the standard API. … However, picking such an API for XML limits your choice of parser, and indeed may even tie you to one particular version of the parser, since parser vendors tend not to worry a great deal about maintaining naïve compatibility between releases." [Harold 2002].
Following, we will only consider SAX and DOM, as JAXP bundles them and some factory classes and the TrAX XSLT API together. Furthermore, JAXP is a standard part of Java 1.4 and later and there is a big advantage in using standard packages: There is no need to install any further software in addition to the JAVA runtime environment at the user's site.

- SAX: SAX is short for Simple API for XML. It is event-driven. "The SAX classes and interfaces model the parser, the stream from which the document is read, and the client application receiving data from the parser. However, no class models the XML document itself. Instead the parser feeds content to the client application through a callback interface … This makes SAX very fast and very memory efficient (since it doesn't need to store the entire document in memory)." [Harold 2002].
  The drawback is that it is event-driven and that it does not read the whole document at once.
- DOM: DOM is short for Document Object Model. Unlike SAX it can be used to read and write documents. It represents each XML document as a Document object and the

Document object's methods allow searching and updating the XML document. Therefore, DOM is much more convenient than SAX when random access to widely separated parts of the original document is required. The drawback is that it is quite memory intensive compared to SAX and "not nearly as well suited to streaming applications." [Harold 2002]

The application uses Dom for reading the XML documents into an internal structure, because DOM reads the entire document at once and because it easy to extract the required information and because an event-driven API like SAX was not convenient. SAX did not prove to be an option, because it does not read the whole XML file at once. It rather reads from top to bottom and signals the host program whenever it detects something of interest. So the host program would need to keep a stack of all interesting occurrences and build an internal data structure out of these afterwards. This is obviously more complicated than using DOM, which reads all tags, values, and attributes into a tree like structure, which can be investigated by the host program much easier.

However, the current version of the generic editor does not use DOM for writing XML files. It proved to be faster to use standard IO mechanisms to write the XML files rather than building a DOM tree and serializing it to disk by using DOM functions. In particular serializing the DOM structure to disk caused problems with large amounts of data when writing the Conflation file format. The drawback of this approach is that the program has to take care of writing well-formed and correctly encoded XML files rather than leaving this to DOM.

7.5.3 Document Validation

The current release does not perform any document validation. Of course it makes some assumptions about the content of the XML files, but these are currently not checked through an XML specific validation. Most of the current parsers support a validation against DTD only, if any. Xerces also supports a validation against Schema. Nevertheless, the current release does neither use DTD nor Schema validation.

7.5.4 The User Interface

The user interface classes make extensive use of SWING classes. According to [Loy 2002 p. 1], it is part of a larger family of Java products known as the Java Foundation Classes (JFC). [Loy 2002 p.2] defines JFC as follows: "The FC (Foundation Classes) is a suite of libraries designed to assist programmers in creating enterprise applications with Java." The JFC consists of:

- AWT = Abstract Windows Toolkit
- Accessibility: support for users which have trouble with traditional user interfaces
- 2DAPI
- Drag and Drop
- Swing

Swing is not a replacement for AWT. It is actually built on top of the core AWT libraries. However, AWT provided only a minimum amount of functionality necessary to create a windowing application. Furthermore, in the contrary to AWT components, which are heavy-weight components, SWING components are light-weight components. While AWT components rely on native widgets, SWING components are written entirely in Java and have a consistent Look and Feel across platforms. Following, we list the most important Swing features:

- Pluggable Look-and-Feels: The look and feel can be changed during runtime.
- Lightweight Components: These are not dependent on native peers to render themselves.
- A lot of additional components and features.

Further details on Swing can be found in [Loy 2002].

Most of the generic editor's user interface classes use JTables and some TableLayout, to place the components. Details about TableLayout can be found at [WWW-11]. During the program development I used the hints on JTables which can be found at [WWW-12] to [WWW-18]. Regarding AWT's layout manager I used [Zukowski 1997] as my main source of information.

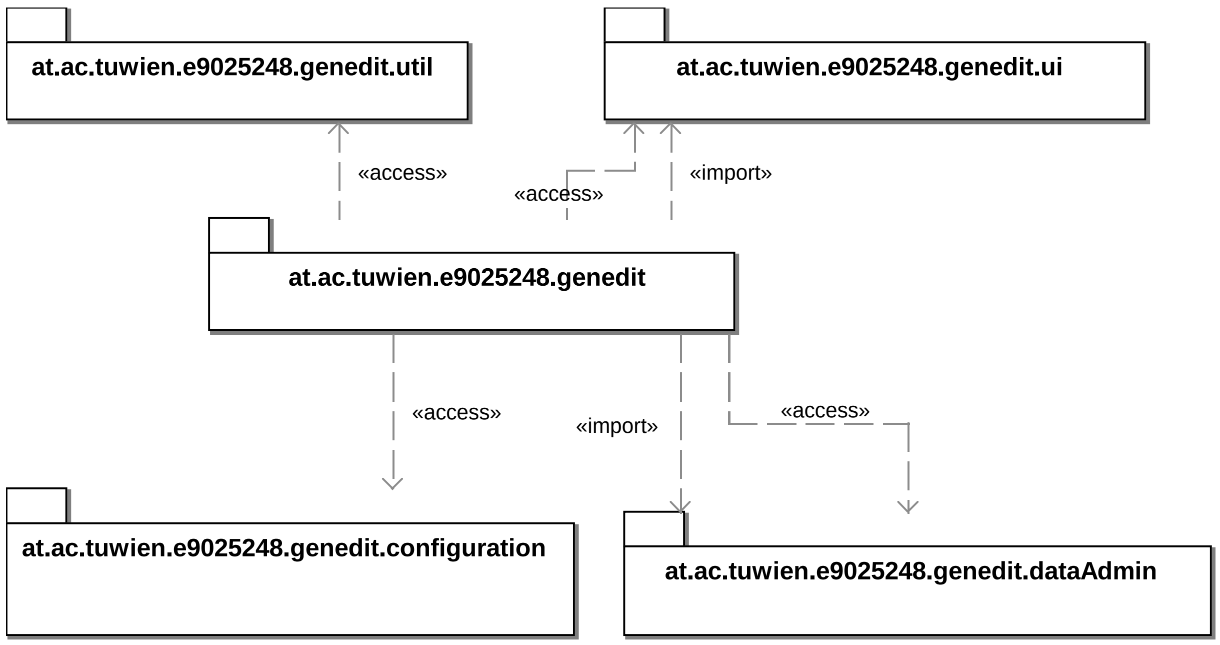
8 Packages Structure and Class Implementations

The base package name of the generic editor's classes is at.ac.tuwien.e9025248.genedit. This package name follows the java naming convention: It starts with the reversely read name of the university's domain, followed by my user id, followed by the project name.
The classes were divided into some sub packages, which reflect their use. This resulted in the following package structure, listed in Table 8-1:

Package Name	Package Description
at.ac.tuwien.e9025248.genedit	Keeps the main class
at.ac.tuwien.e9025248.genedit.configuration	Keeps all configuration details
at.ac.tuwien.e9025248.genedit.dataAdmin	Keeps all data administration issues of the internal data structure
at.ac.tuwien.e9025248.genedit.dataAdmin.value	A package for keeping and converting from internal representation to displayed graphical user interface elements.
at.ac.tuwien.e9025248.genedit.genEditEvents	Defines two classes for progressing data change events
at.ac.tuwien.e9025248.genedit.io	All physical storage issues
at.ac.tuwien.e9025248.genedit.ui	Windows and UI elements
at.ac.tuwien.e9025248.genedit.util	Keeps a small class with some useful static methods.

Table 8-1 Package Structure. We split the classes into different packages.

The packages were carefully designed in a modular way, to make extensions or modification easy and to keep local changes local without the need of changing anything globally. We emphasized data encapsulation and modular design. Figure 8-1 shows the overall packages in an UML diagram.


at.ac.tuwien.e9025248.genedit.util
at.ac.tuwien.e9025248.genedit.ui
«access»
«access»
«import»
at.ac.tuwien.e9025248.genedit
«access»
«import»
«access»
at.ac.tuwien.e9025248.genedit.configuration
at.ac.tuwien.e9025248.genedit.dataAdmin
Figure 8-1 Package Structure. The packages depend and access each other.

The Generic Editor class (see chapter 8.1), which implements the main method, is the only class, which resides in at.ac.tuwien.e9025248.genedit.

8.1 GenericEditor class

The GenericEditor class implements the main method. This method is called at program startup. The program start up performs the following tasks.

1. The profile is read for configuring the generic editor
2. The language specific translations are read.
   If there is an error in steps (1) or (2), an error message will be displayed and the application will be closed again.
3. The parameter sources are loaded.
4. The forms of each form group are loaded.
5. The answers for each group of forms are loaded. If there are parameter-sources not yet defined through step 3, they will be automatically created during the loading process of the answers.
6. The main menu is displayed.

Upon a successful program startup all the necessary data is loaded into the internal data structures, which are implemented by the TableData class and the RecordData class. Figure 8-2 shows the UML diagram of the GenericEditor class.

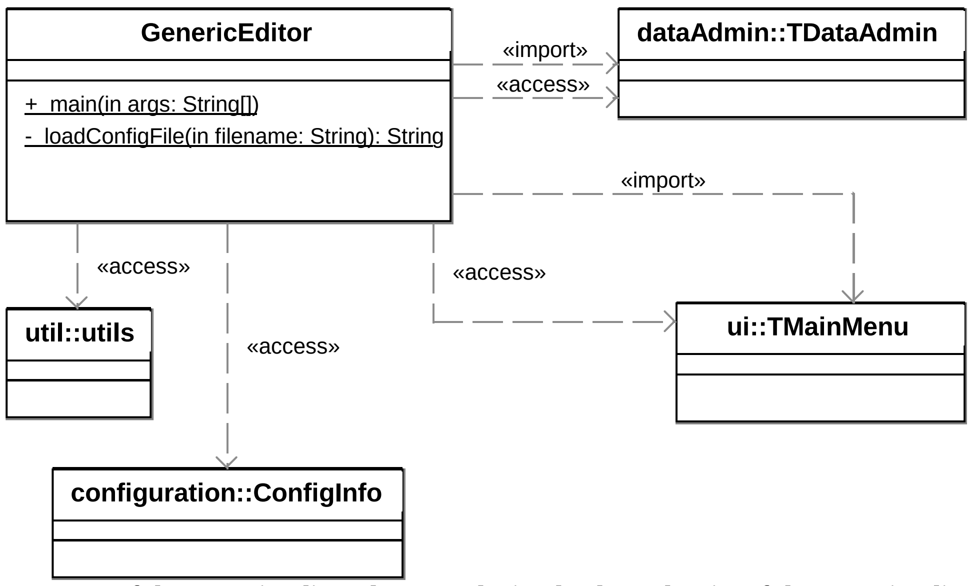


Figure 8-2 UML of the GenericEditor class. We depict the dependencies of the GenericEditor class.

8.2 Configuration and Profile

The configuration tasks are performed by the classes in the configuration package. In this package there are three classes. While the classes ExportFileFormat and UIConstants merely define some user interface layout and export file format standards in static variables, it is the task of the ConfigInfo class to read the profile and the language file and to provide the application with this information. After the configuration process the details listed in Table 8-2 can be accessed by the rest of the program by accessing ConfigInfo’s static methods or variables.

Name of the variable or method	**Description**
ERRORLOG	The filename of the error log file
HISTORY	The filename of the history file
ResourceFile	The path to the file containing the language specific translations of menu entries, error messages etc.
title	Title of the Application
numFormFiles	The number of form groups (equal to the number of form definition files)
FormFiles	A String array which keeps the path to the form information
FormIds	A String array keeping the Name of the Form group, e.g. "Questionnaire"
SingleForm	A Boolean Array which keeps track of whether a form group is for not-time dependent data (single) or for time dependent data.
ShortCut	An array keeping the shortcuts for starting the record data task of each form group
AnswerFile	An array with the filenames, if the answers to a specific group of questionnaires are stored in a single file. It is an alternative to specifying an AnswerDir for a specific form group

Name of the variable or method	**Description**
AnswerDir	The subdirectory of PersDir for each form group, if the answers to a specific group of questionnaires are stored in subdirectories of PersDir. It is an alternative to specifying an AnswerFile
AddOnHeadFields	An array of arrays of AddOnHeadFields, The class AddOnHeadField keeps track of additional fields needed for recording the values of the parameter sets. The first dimension is for each form group, while the second dimension of the array is for each additional head field for within a form group.
PersonDir	The Directory containing the answers of specific persons. It must be specified, if any questionnaire group uses the AnswerDir option.
PersonFile	The file used to keep the available persons, if it is not available, it will be created through reading the subdirectories of PersDir. PersonFile is given precedence over PersonDir!
PersonType	The type of the replier, e.g. patient.
PersonShortCut	The shortcut to the parameter source maintenance
ExternalName	The displayed name of an external program
ExternalCmd	External program to start after export through a special menu option
ExternalFile	The file to export to before starting the external program
ExternalFileFormat	The file format to write for the external program
lastChanged	The data of the last changes to the configuration file (=profile).
getLastChanged()	Returns the date of the profile as a String for displaying.
loadResourceFile(String rscFile)	Loads the language file
loadConfigFile(String FileName)	Loads the profile (configuration file)
getLanguageSpecificName(int index)	Returns a String for a language specific translation.

Table 8-2 ConfigInfo Methods. We list the methods of the ConfigInfo class

Nearly all other classes make use of the ConfigInfo's getLanguageSpecificName() method. Some other variables are only used during the program initialization. E.g. the AdditionalHeadFields are read in and stored in AdditionalHeadFields, but once the internal data tables are set up, they are of no further use. So, they simple provide – together with the SingleForm array - the information for setting up the answer set tables for each form group. Figure 8-3 depicts the UML diagram of the classes of configuration package.

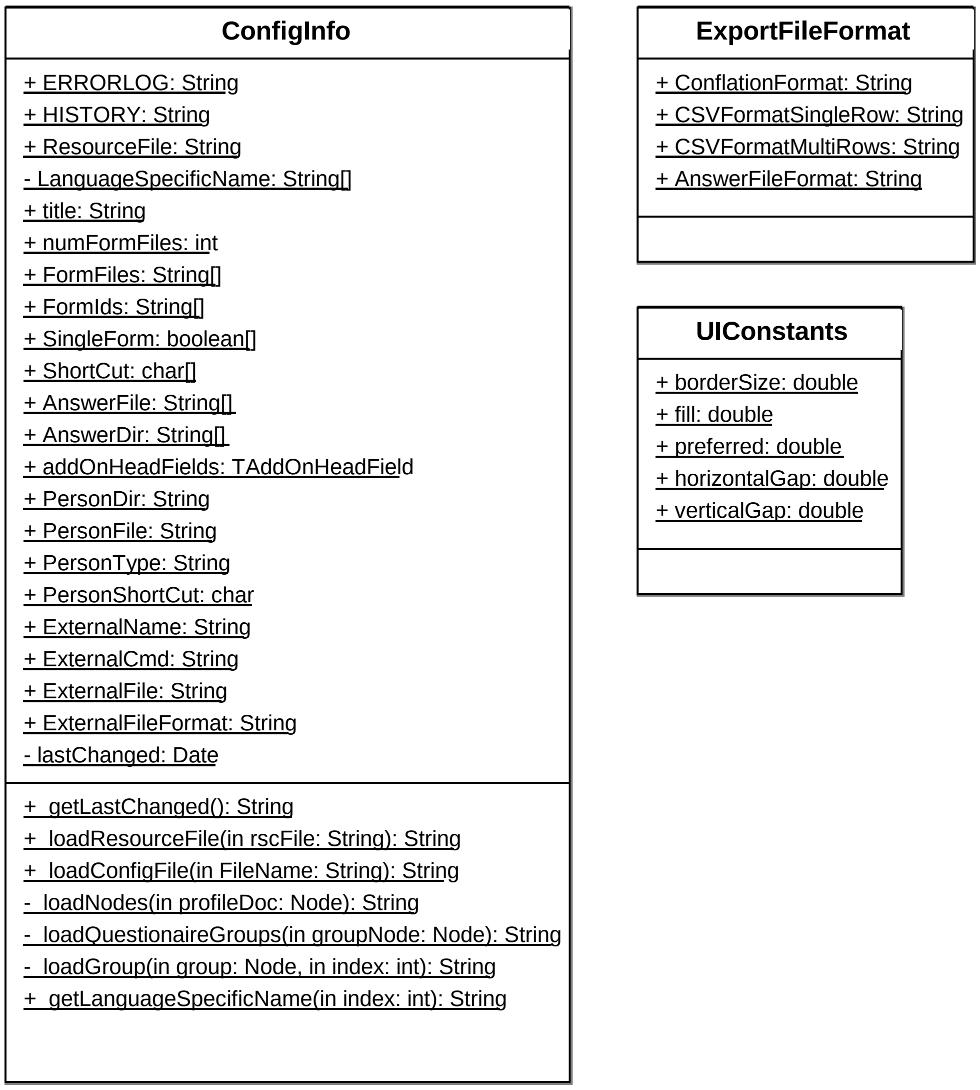


Figure 8-3 UML of the configuration package's classes. We depict the dependencies of the configuration classes

8.2.1 The Default Time Granularity and How to Change It

Besides recording non-time dependent measurement data, it is also possible use another than the default time granularity for the measurement data of a form group. The default time granularity for measurement data is in the current release the day, i.e. there can be a different measurement for every parameter source and every form every day. The easiest way to achieve this is to define a form group to keep non-time dependent data and set up additional key fields with date data types. E.g. datatype = "date:0:YYYY". This colon separated string specifies, that a field 0 is defined, with data type date, which keeps the year. Further fields can be defined for date field 0 keeping the month, the day, the hour, the minute etc. All fields with the same identifying number are combined together for displaying in the record data window, as long as they consecutively and have the same value in iskey. If iskey = "true", a field belongs to the key fields of measurement records of the form group they are defined for.

8.3 Data Model

All internal data administration classes reside in the package at.ac.tuwien.e9025248.genedit.dataAdmin. It consists of a couple classes for data administration and a sub-package for converting the internal string values to their graphical user interface representations. The following chapters describe the classes of the dataAdmin package in detail.

8.3.1 The Overall Data Administration Class – TDataAdmin

Regarding the internal data structures, the TDataAdmin class implements all data handling issues. It not only keeps the required structural and measurement data, it also keeps track of making data access mutual exclusive and it provides the rest of the application with some final static variables for accessing the data and for consistent naming. Therefore, other classes access these variables instead of using hard-coded strings for common terms. This leads to an easy to maintain program, because such terms need to be changed only in TDataAdmin, while the rest of the classes need not be touched, in case changes should become necessary in the future.

The TDataAdmin class is a singleton, in the sense that there may only be one copy of the data administration object. Details about the singleton design pattern can be found in [Gamma 1995] pp. 127, a short definition can be found in the appendix (see IV.1.4.). The TDataAdmin keeps another singleton, which is the parameter source table and an array of TAData objects (see section 8.3.2) and it provides functionality for various queries against the data.
Figure 8-4 UML of TDataAdmin shows the UML diagram of the TDataAdmin class. The UML diagram shows only package internal dependences, because of the size of the diagram. Further to the depicted dependences, the TDataAdmin class also uses configuration.ConfigInfo and io.AbstractStorageManager

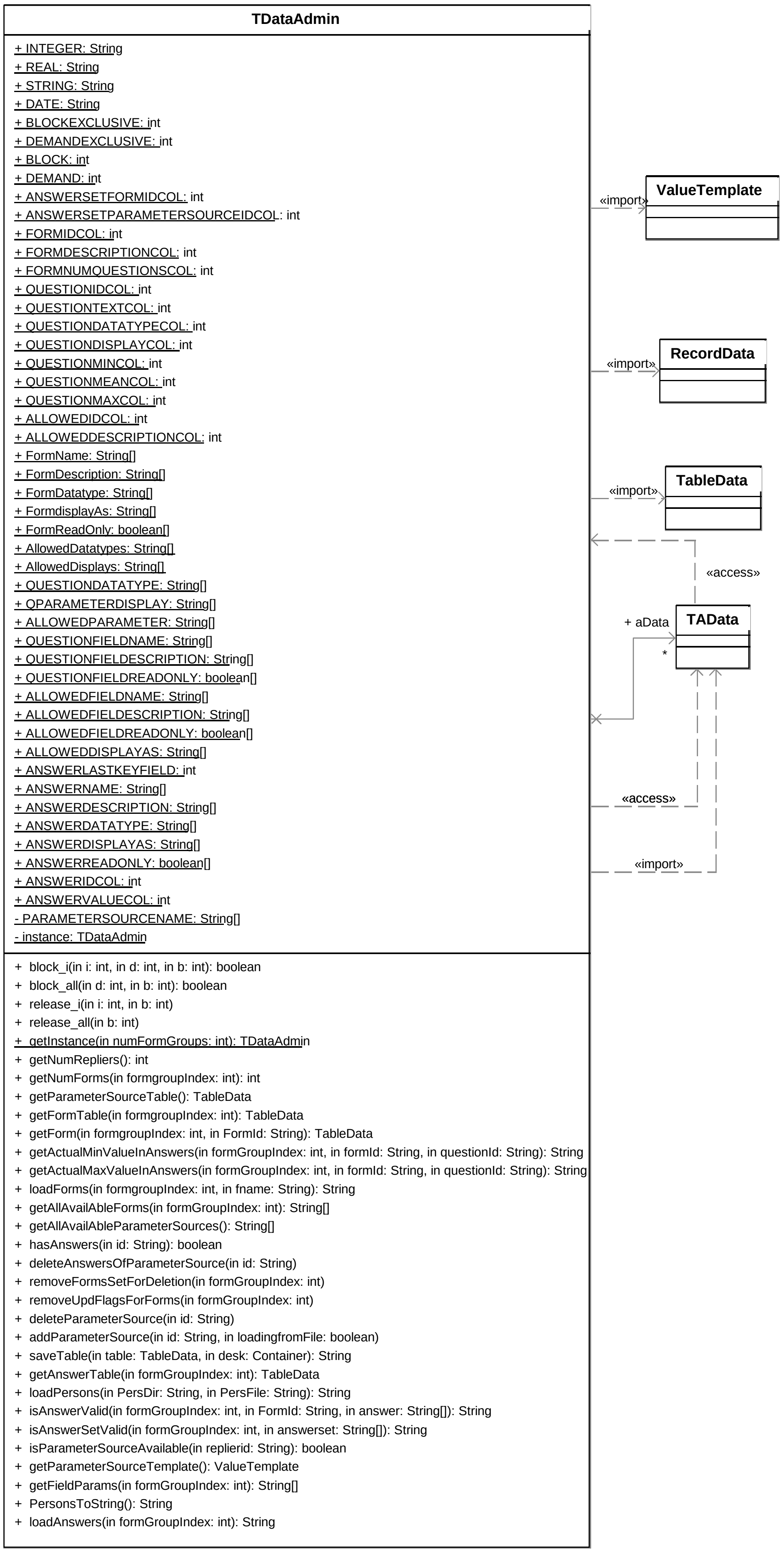

TDataAdmin
+ INTEGER: String
+ REAL: String
+ STRING: String
+ DATE: String
+ BLOCKEXCLUSIVE: int
+ DEMANDEXCLUSIVE: int
+ BLOCK: int
+ DEMAND: int
+ ANSWERSETFORMIDCOL: int
+ ANSWERSETPARAMETERSOURCEIDCOL: int
+ FORMIDCOL: int
+ FORMDESCRIPTIONCOL: int
+ FORMNUMQUESTIONSCOL: int
+ QUESTIONIDCOL: int
+ QUESTIONTEXTCOL: int
+ QUESTIONDATATYPECOL: int
+ QUESTIONDISPLAYCOL: int
+ QUESTIONMINCOL: int
+ QUESTIONMEANCOL: int
+ QUESTIONMAXCOL: int
+ ALLOWEDIDCOL: int
+ ALLOWEDDESCRIPTIONCOL: int
+ FormName: String[]
+ FormDescription: String[]
+ FormDatatype: String[]
+ FormdisplayAs: String[]
+ FormReadOnly: boolean[]
+ AllowedDatatypes: String[]
+ AllowedDisplays: String[]
+ QUESTIONDATATYPE: String[]
+ QPARAMETERDISPLAY: String[]
+ ALLOWEDPARAMETER: String[]
+ QUESTIONFIELDNAME: String[]
+ QUESTIONFIELDDESCRIPTION: String[]
+ QUESTIONFIELDREADONLY: boolean[]
+ ALLOWEDFIELDNAME: String[]
+ ALLOWEDFIELDDESCRIPTION: String[]
+ ALLOWEDFIELDREADONLY: boolean[]
+ ALLOWEDDISPLAYAS: String[]
+ ANSWERLASTKEYFIELD: int
+ ANSWERNAME: String[]
+ ANSWERDESCRIPTION: String[]
+ ANSWERDATATYPE: String[]
+ ANSWERDISPLAYAS: String[]
+ ANSWERREADONLY: boolean[]
+ ANSWERIDCOL: int
+ ANSWERVALUECOL: int
- PARAMETERSOURCENAME: String[]
- instance: TDataAdmin
+ block_i(in i: int, in d: int, in b: int): boolean
+ block_all(in d: int, in b: int): boolean
+ release_i(in i: int, in b: int)
+ release_all(in b: int)
+ getInstance(in numFormGroups: int): TDataAdmin
+ getNumRepliers(): int
+ getNumForms(in formgroupIndex: int): int
+ getParameterSourceTable(): TableData
+ getFormTable(in formgroupIndex: int): TableData
+ getForm(in formgroupIndex: int, in FormId: String): TableData
+ getActualMinValueInAnswers(in formGroupIndex: int, in formId: String, in questionId: String): String
+ getActualMaxValueInAnswers(in formGroupIndex: int, in formId: String, in questionId: String): String
+ loadForms(in formgroupIndex: int, in fname: String): String
+ getAllAvailAbleForms(in formGroupIndex: int): String[]
+ getAllAvailAbleParameterSources(): String[]
+ hasAnswers(in id: String): boolean
+ deleteAnswersOfParameterSource(in id: String)
+ removeFormsSetForDeletion(in formGroupIndex: int)
+ removeUpdFlagsForForms(in formGroupIndex: int)
+ deleteParameterSource(in id: String)
+ addParameterSource(in id: String, in loadingfromFile: boolean)
+ saveTable(in table: TableData, in desk: Container): String
+ getAnswerTable(in formGroupIndex: int): TableData
+ loadPersons(in PersDir: String, in PersFile: String): String
+ isAnswerValid(in formGroupIndex: int, in FormId: String, in answer: String[]): String
+ isAnswerSetValid(in formGroupIndex: int, in answerset: String[]): String
+ isParameterSourceAvailable(in replierid: String): boolean
+ getParameterSourceTemplate(): ValueTemplate
+ getFieldParams(in formGroupIndex: int): String[]
+ PersonsToString(): String
+ loadAnswers(in formGroupIndex: int): String
«import»
ValueTemplate
«import»
RecordData
«import»
TableData
«access»
+ aData
TAData
*
«access»
«import»

Figure 8-4 UML of TDataAdmin
We depict the dependencies of the TDataAdmin class.

8.3.1.1 Ensuring Mutual Exclusiveness

In order to ensure mutual exclusiveness of some tasks (see also mutual exclusiveness in chapter 7.3) the TDataAdmin class keeps track of which form group is currently in use by the private integer array named block. Four methods are provided to block or release the data (see Table 8-3):

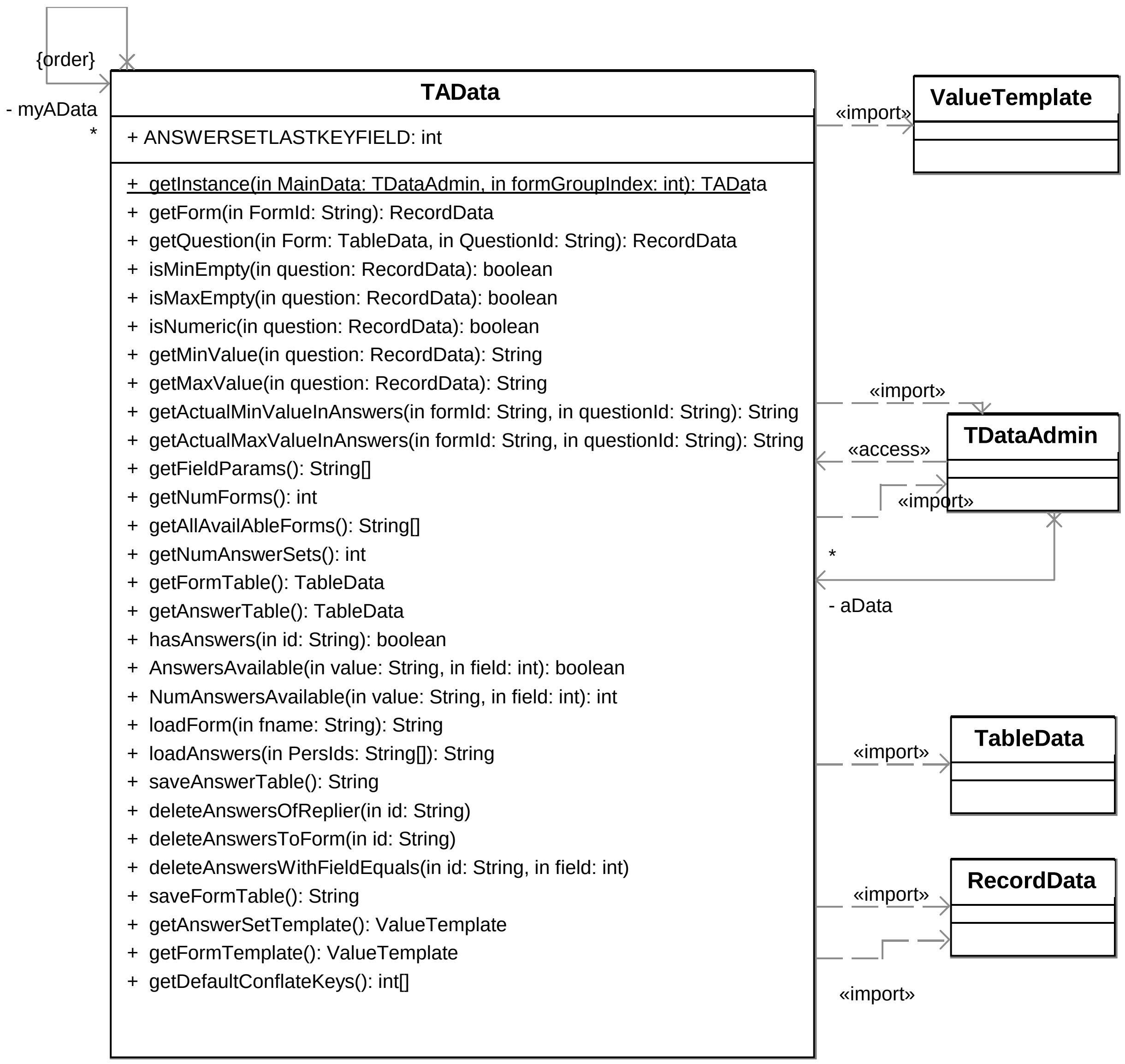
Method	Description
block_i()	Blocks form group i
block_all()	Blocks all form groups
release_i()	Releases form group i
release_all()	Releases all form groups

Table 8-3 Mutual Exclusiveness. We list the methods ensuring mutual exclusiveness

All these method provide synchronized access to the block array. Furthermore, the TDataAdmin class defines four final static integers for blocking or demanding the data. Each of the methods mentioned in Table 8-3 works in a similar way. First they check whether they can block the data by checking the current value(s) of the block array against the demanded value and if successful – they thereafter block the data by incrementing the value(s) of the block array by the block value passed as a parameter.

8.3.2 The Data Administration Class for Form Group Specific Data – TAData

While there is only one parameter source table for all form groups, there has to be a form table and a measurement table for each of the form groups. The TAData class implements an object which keeps track of the data needed for a single form group. The TDataAdmin class keeps references to the instances of TAData objects for each form group (for details see section 8.3.1). The main task of the TAData class is to set up the form table and the measurement (set) table according to the information found in the profile, especially in the two-dimensional AddOnHeadField array and the single array, which informs about time dependence. This task is performed during object creation only. After having set up the measurement table, the number and type of additional head fields and whether they belong to the index of the measurement (set) table or not, is completely transparent to the rest of the application. At the time all the measurement (set) tables have been set up, the two-dimensional AddOnHeadField table is no longer used, because all the structural and measurement data is thereafter solely accessed by using the TableData (see section 8.3.3.2) and RecordData (see section 8.3.3.1) classes.
Figure 8-5 shows the UML diagram of the TAData class. The UML diagram shows only package internal dependences, because of the size of the diagram. Further to the depicted dependences, the TAData class also uses configuration.ConfigInfo and io.AbstractStorageManager.


{order}
- myAData
*
TAData
+ ANSWERSETLASTKEYFIELD: int
+ getInstance(in MainData: TDataAdmin, in formGroupIndex: int): TAData
+ getForm(in FormId: String): RecordData
+ getQuestion(in Form: TableData, in QuestionId: String): RecordData
+ isMinEmpty(in question: RecordData): boolean
+ isMaxEmpty(in question: RecordData): boolean
+ isNumeric(in question: RecordData): boolean
+ getMinValue(in question: RecordData): String
+ getMaxValue(in question: RecordData): String
+ getActualMinValueInAnswers(in formId: String, in questionId: String): String
+ getActualMaxValueInAnswers(in formId: String, in questionId: String): String
+ getFieldParams(): String[]
+ getNumForms(): int
+ getAllAvailAbleForms(): String[]
+ getNumAnswerSets(): int
+ getFormTable(): TableData
+ getAnswerTable(): TableData
+ hasAnswers(in id: String): boolean
+ AnswersAvailable(in value: String, in field: int): boolean
+ NumAnswersAvailable(in value: String, in field: int): int
+ loadForm(in fname: String): String
+ loadAnswers(in PersIds: String[]): String
+ saveAnswerTable(): String
+ deleteAnswersOfReplier(in id: String)
+ deleteAnswersToForm(in id: String)
+ deleteAnswersWithFieldEquals(in id: String, in field: int)
+ saveFormTable(): String
+ getAnswerSetTemplate(): ValueTemplate
+ getFormTemplate(): ValueTemplate
+ getDefaultConflateKeys(): int[]
«import»
ValueTemplate
«import»
«access»
TDataAdmin
«import»
*
- aData
«import»
TableData
«import»
RecordData
«import»
Figure 8-5 UML of TAData. We depict the dependencies of the TAData class.

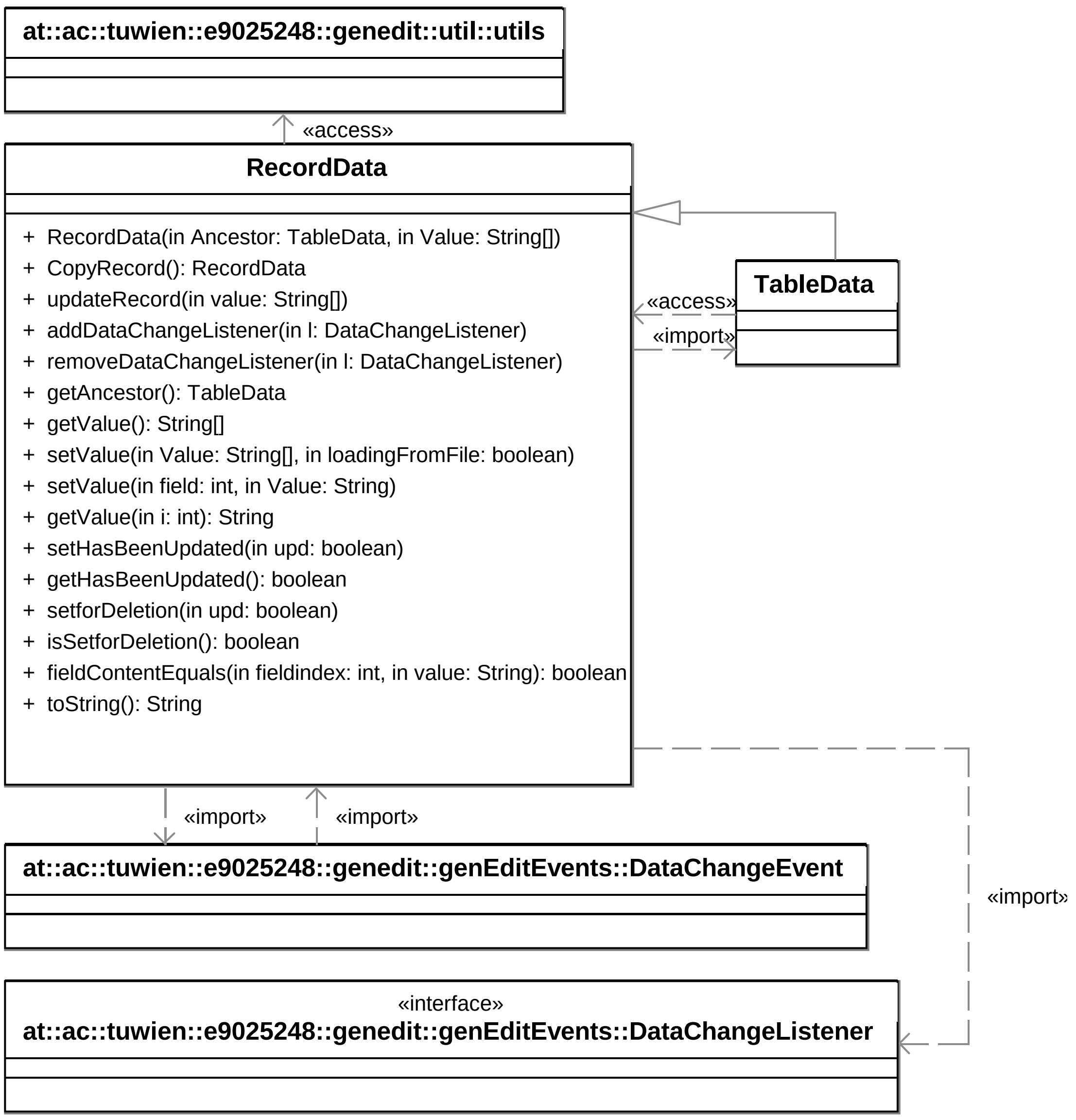
8.3.3 Internal Administration of Hierarchical Data

Both structural and measurement data have a hierarchical structure. We developed a unified internal representation for hierarchical data. Similar to database systems we use the concept of tables. A table is a set of values that are organized in rows and columns. “The columns are identified by name, and the rows are identified by the values appearing in a particular column subset which has been identified as candidate key” (see [WWW-3]). In our implementation the rows are kept in a list of RecordData. The TableData class keeps the structural information about the data and the content information of each field, e.g. the name of the columns etc. - and keeps a list of records (rows), whilst the instances of the RecordData class represents the rows of the table, which can again have a hierarchical structure, i.e. be of type TableData. Therefore, the implementation is split into two classes: The RecordData class and the TableData class.

8.3.3.1 The Records – RecordData

RecordData is the class representing the table records. Because TableData (see section 8.3.3.2) extends RecordData, each record can again be a sub table. A record keeps track of the values of a table row. It knows the table it belongs to, which is called the records ancestor. On

changes the hasbeenupdated flag is set, similarly, on delete requests, the setForDeletion flag is set. Upon having successfully physically stored the record or deleted the record the hasbeenupdated flag will be reset or the record will be deleted.
In order to notify other objects listening to changes of the data, RecordData keeps track of a Vector of dataChangeListeners. In the current release only the instance of the TMainMenu class registers itself as dataChangeListener (see section 8.6.2) to the RecordData objects. Figure 8-6 UML of RecordData shows the UML diagram of the RecordData class.



at::ac::tuwien::e9025248::genedit::util::utils
«access»
RecordData
+ RecordData(in Ancestor: TableData, in Value: String[])
+ CopyRecord(): RecordData
+ updateRecord(in value: String[])
+ addDataChangeListener(in l: DataChangeListener)
+ removeDataChangeListener(in l: DataChangeListener)
+ getAncestor(): TableData
+ getValue(): String[]
+ setValue(in Value: String[], in loadingFromFile: boolean)
+ setValue(in field: int, in Value: String)
+ getValue(in i: int): String
+ setHasBeenUpdated(in upd: boolean)
+ getHasBeenUpdated(): boolean
+ setforDeletion(in upd: boolean)
+ isSetforDeletion(): boolean
+ fieldContentEquals(in fieldindex: int, in value: String): boolean
+ toString(): String
TableData
«access»
«import»
«import»
«import»
at::ac::tuwien::e9025248::genedit::genEditEvents::DataChangeEvent
«import»
«interface»
at::ac::tuwien::e9025248::genedit::genEditEvents::DataChangeListener

Figure 8-6 UML of RecordData. We depict the dependencies of the RecordData class.

8.3.3.2 The Tables – TableData

TableData keeps structural information and possibly value information, because it extends RecordData (see section 8.3.3.1) and a list of records. The root table, which is also the root of some hierarchical data (e.g. a form or a parameter (set) table) does not contain any values and does not have an ancestor table. Table 8-4 TableData Fields lists the information, TableData keeps track of:

Name	**Description**
name	An array of field names

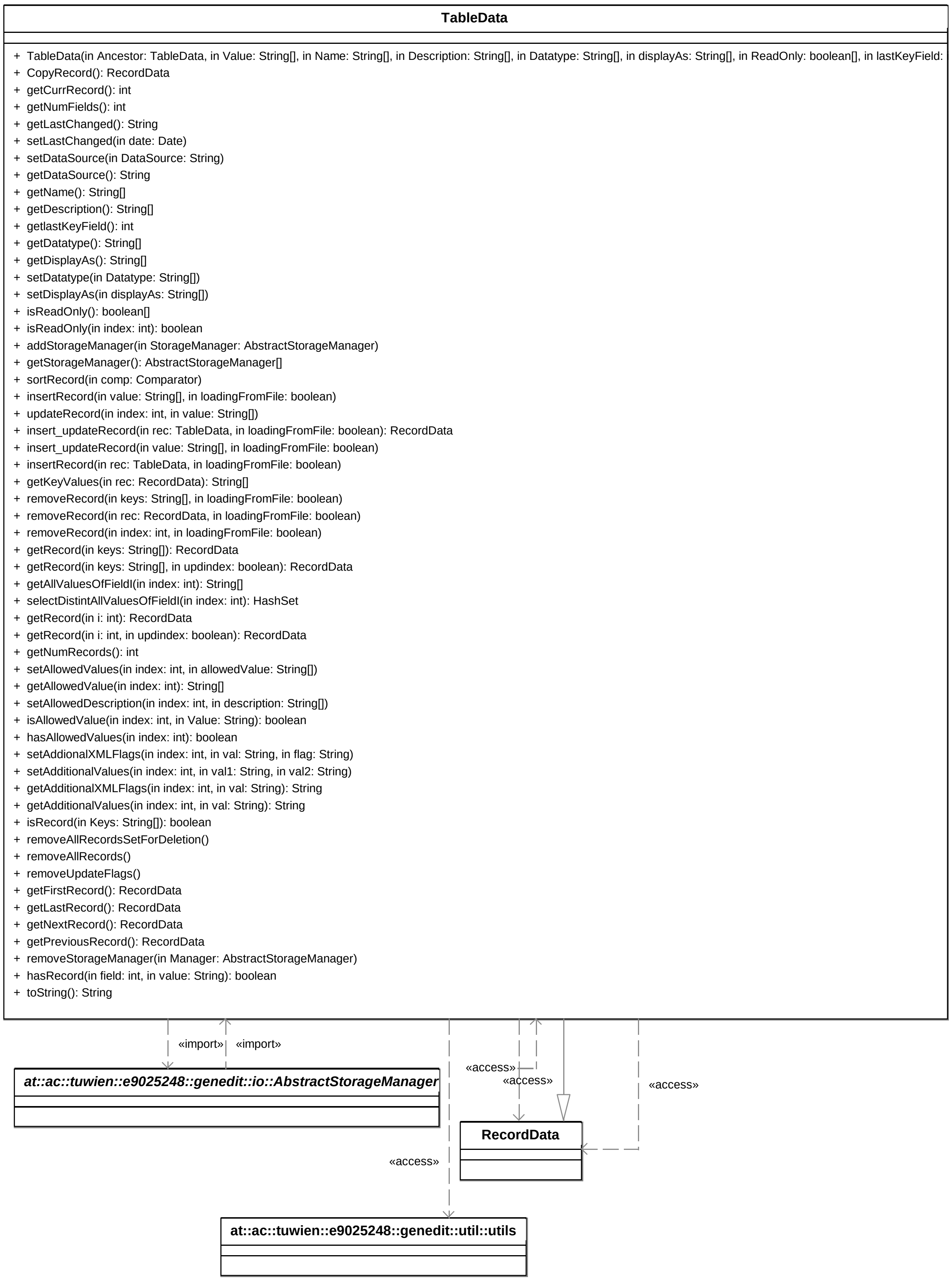
Name	**Description**
description	An array of field descriptions
datatype	An array of the field’s data types
displayAs	An array which keeps information about how to display each field in the record data user interface
readOnly	An array, which keeps track whether a field is read only
lastKeyField	The index of the last key field
StorageManager	A list of Storage Managers (see 8.4.1)

Table 8-4 TableData Fields. We list the properties a table has to keep track of.

A couple of methods provide - besides the usual getter and setter methods - functionality for:

- Browsing through the tables records
- Copying an record
- Sorting the records by means of a Comparator (see section 8.3.5)
- Inserting, updating and deleting records
- Searching for records
- Verifying whether the table contains a specific record

Figure 8-7 shows the UML diagram of the TableData class.


TableData
at::ac::tuwien::e9025248::genedit::io::AbstractStorageManager
RecordData
at::ac::tuwien::e9025248::genedit::util::utils
«import»
«import»
«access»
«access»
«access»
«access»

```
+ TableData(in Ancestor: TableData, in Value: String[], in Name: String[], in Description: String[], in Datatype: String[], in displayAs: String[], in ReadOnly: boolean[], in lastKeyField:
+ CopyRecord(): RecordData
+ getCurrRecord(): int
+ getNumFields(): int
+ getLastChanged(): String
+ setLastChanged(in date: Date)
+ setDataSource(in DataSource: String)
+ getDataSource(): String
+ getName(): String[]
+ getDescription(): String[]
+ getlastKeyField(): int
+ getDatatype(): String[]
+ getDisplayAs(): String[]
+ setDatatype(in Datatype: String[])
+ setDisplayAs(in displayAs: String[])
+ isReadOnly(): boolean[]
+ isReadOnly(in index: int): boolean
+ addStorageManager(in StorageManager: AbstractStorageManager)
+ getStorageManager(): AbstractStorageManager[]
+ sortRecord(in comp: Comparator)
+ insertRecord(in value: String[], in loadingFromFile: boolean)
+ updateRecord(in index: int, in value: String[])
+ insert_updateRecord(in rec: TableData, in loadingFromFile: boolean): RecordData
+ insert_updateRecord(in value: String[], in loadingFromFile: boolean)
+ insertRecord(in rec: TableData, in loadingFromFile: boolean)
+ getKeyValues(in rec: RecordData): String[]
+ removeRecord(in keys: String[], in loadingFromFile: boolean)
+ removeRecord(in rec: RecordData, in loadingFromFile: boolean)
+ removeRecord(in index: int, in loadingFromFile: boolean)
+ getRecord(in keys: String[]): RecordData
+ getRecord(in keys: String[], in updindex: boolean): RecordData
+ getAllValuesOfFieldI(in index: int): String[]
+ selectDistintAllValuesOfFieldI(in index: int): HashSet
+ getRecord(in i: int): RecordData
+ getRecord(in i: int, in updindex: boolean): RecordData
+ getNumRecords(): int
+ setAllowedValues(in index: int, in allowedValue: String[])
+ getAllowedValue(in index: int): String[]
+ setAllowedDescription(in index: int, in description: String[])
+ isAllowedValue(in index: int, in Value: String): boolean
+ hasAllowedValues(in index: int): boolean
+ setAddionalXMLFlags(in index: int, in val: String, in flag: String)
+ setAdditionalValues(in index: int, in val1: String, in val2: String)
+ getAdditionalXMLFlags(in index: int, in val: String): String
+ getAdditionalValues(in index: int, in val: String): String
+ isRecord(in Keys: String[]): boolean
+ removeAllRecordsSetForDeletion()
+ removeAllRecords()
+ removeUpdateFlags()
+ getFirstRecord(): RecordData
+ getLastRecord(): RecordData
+ getNextRecord(): RecordData
+ getPreviousRecord(): RecordData
+ removeStorageManager(in Manager: AbstractStorageManager)
+ hasRecord(in field: int, in value: String): boolean
+ toString(): String
```

Figure 8-7 UML of TableData. We depict the dependencies of the TableData class.

8.3.4 TableDataIterator

The class TableDataIterator implements the Iterator interface for generic access to the data. "An Iterator abstracts the traversal algorithm and shields clients from the internal structure of the objects they traverse." [Gamma 1995 p.70] Therefore, an Iterator helps us gain flexibility and reusability. Further details on the Iterator pattern can be found in [Gamma 1995, pp. 70].

Details and Examples about using an Iterator and implementing a custom Iterator in Java can be found in [Darwin 2005 pp.196 and pp. 214].
In the current release, the TableDataIterator is used only in the Exporter class as well as in the TAData class. It is of use everywhere, where there is a need to traverse all records of a table. Future releases might enhance the use of the TableDataIterator by removing calls to TableData's getRecord(k) everywhere, where a complete transversal of all records is needed, e.g. in the storage managers (see section 8.4.1.). The TableDataIterator implements Java's standard interface: java.util.Iterator. Figure 8-8 depicts the UML diagram of the TableDataIterator class.

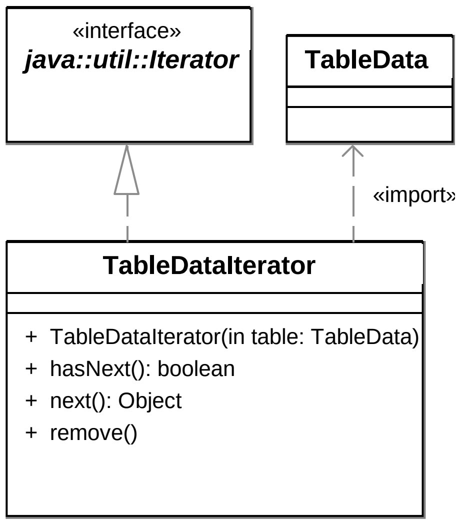


Figure 8-8 UML TableDataIterator. We depict the dependencies of the TableDataIterator class.

8.3.5 A Comparator for Records

The FieldComparator class provides functionality for comparing the values of records. For this purpose it implements the java.util.Comparator interface, which is a Java standard interface. It may use a template to convert the values of each record to possibly combined values (e.g. a Date possibly consists of more than one value field, because it for example consists of fields for a day, a month and a year). The purpose of the template is to provide a unique conversion between internal and external representation for all compared records. Some tables are simple enough to not require such a template, because they have only fields which can be compared without the need of a conversion from internal to external representation.
A FieldComparator object needs to know, in what sequence to compare the field values in any case. Therefore, it is compulsory to specify the sequence in the constructor.
Finally, the FieldComparator implements the inherited abstract compare() method. First of all, it checks, whether the objects to compare are both of type RecordData, than it either uses simple comparison of fields without using any template, or complex comparison by using a template for converting the internal representation to an external representation. If a conversion takes place the compareTo() method of the class, implementing the internal to external conversion is used. This class inherits from AbstractValue and can be either a SValue or a DateValue in the current release. Figure 8-9 depicts the UML diagram of the FieldComparator.

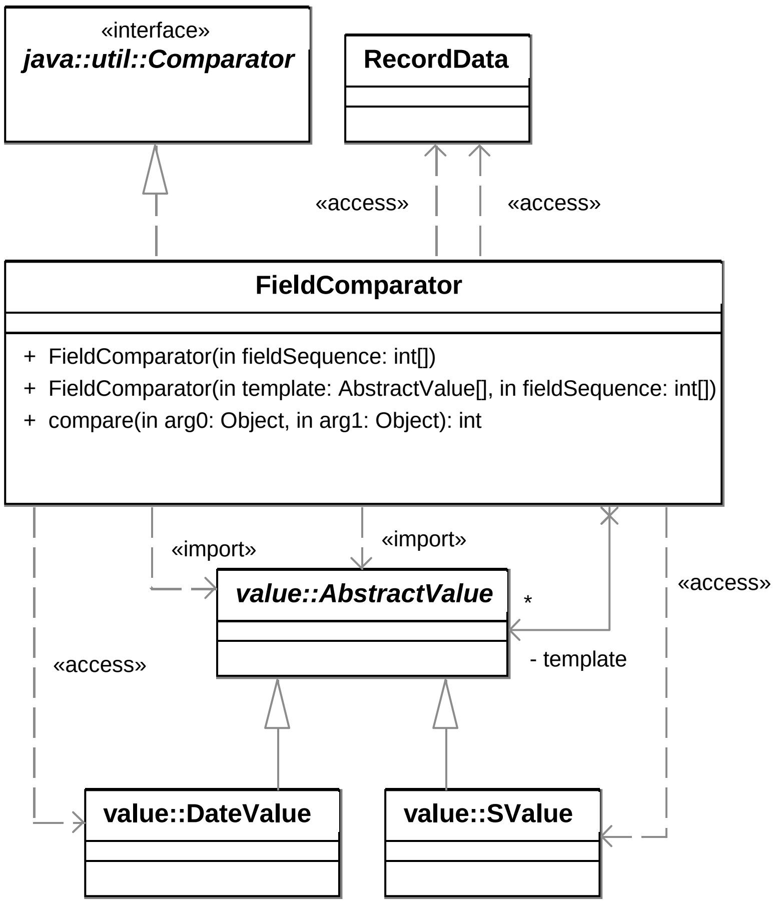


Figure 8-9 UML of FieldComparator. We depict the dependencies of the FieldComparator class.

8.3.6 Additional Head Fields

Additional head fields are read into an object of class TAddOnHeadField. There is one object for each additional field of each form group. Therefore, the ConfigInfo class keeps references to these objects in a two-dimensional array (one index for the form group, one index for the additional field). An additional head field consists of a name, a description, an XMLFlag to store it, a data type, information on how it wants to be displayed, and whether it belongs to the keys of the measurement sets of the form group it belongs to or not. It can have a list of allowed values, together with descriptions of each allowed value. Furthermore, each allowed value can be stored also through additional XML flags and by in the profile pre-configured values (see also section 8.2 about Configuration and Profile and chapter III.9.1.2 for the parameters of the configuration file). Additional XML flags and according values can be useful, if one wants to store a certain value in different ways. E.g., in the case of our study about anorectic girls, it can be used to provide information on how to sort the qualitative time stamps in the answer file. Figure 8-10 shows the UML diagram of the TAddOnHeadField class.

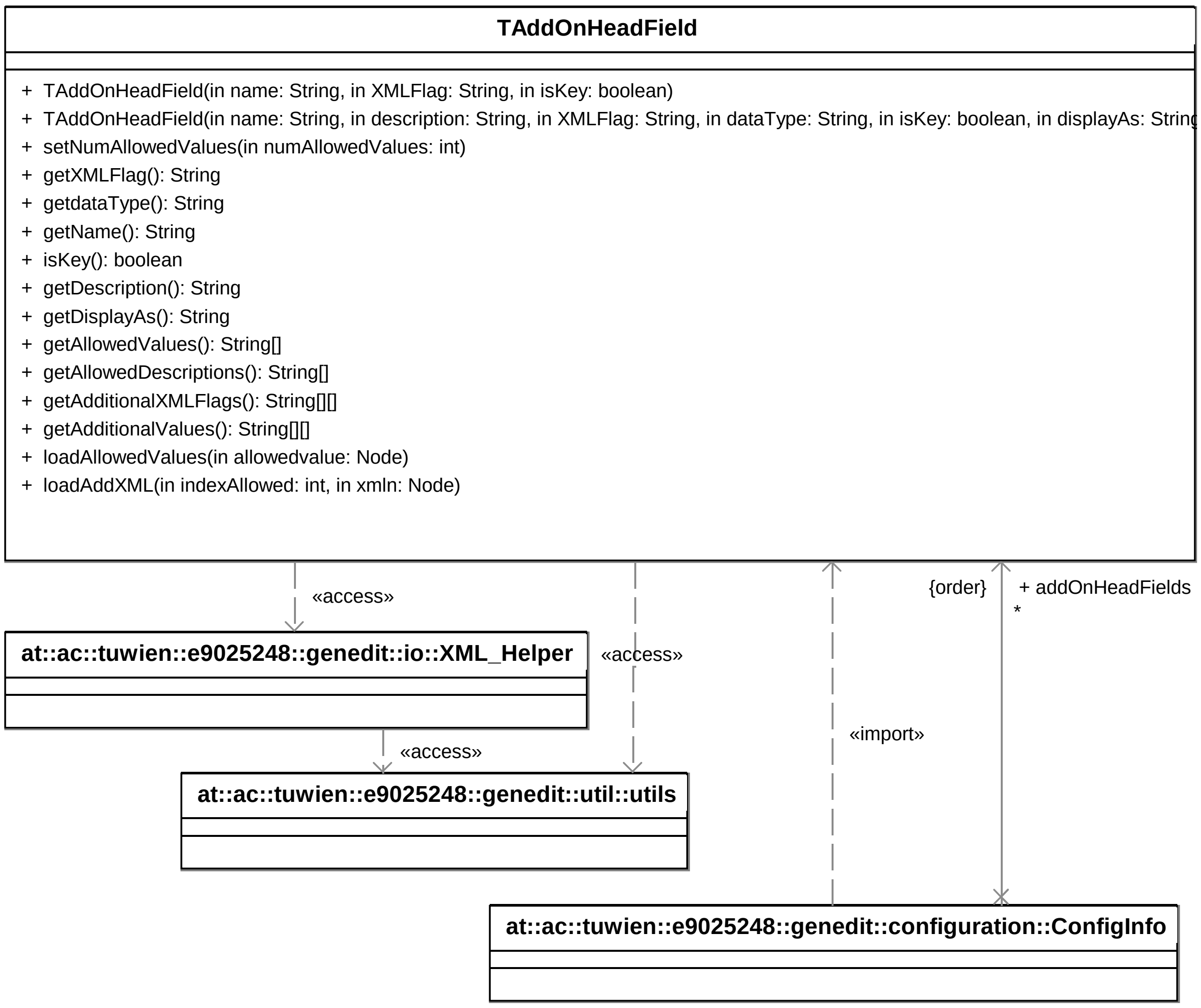

TAddOnHeadField
+ TAddOnHeadField(in name: String, in XMLFlag: String, in isKey: boolean)
+ TAddOnHeadField(in name: String, in description: String, in XMLFlag: String, in dataType: String, in isKey: boolean, in displayAs: String
+ setNumAllowedValues(in numAllowedValues: int)
+ getXMLFlag(): String
+ getdataType(): String
+ getName(): String
+ isKey(): boolean
+ getDescription(): String
+ getDisplayAs(): String
+ getAllowedValues(): String[]
+ getAllowedDescriptions(): String[]
+ getAdditionalXMLFlags(): String[][]
+ getAdditionalValues(): String[][]
+ loadAllowedValues(in allowedvalue: Node)
+ loadAddXML(in indexAllowed: int, in xmln: Node)
«access»
{order}
+ addOnHeadFields
*
at::ac::tuwien::e9025248::genedit::io::XML_Helper
«access»
«access»
«import»
at::ac::tuwien::e9025248::genedit::util::utils
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo

Figure 8-10 UML of TAddOnHeadField. We depict the dependencies of the TAddOnHeadField class.

8.3.7 Templates for Setting up the Conversions from Internal to Displayed Representation

The class ValueTemplate provides a template for setting up conversions from internal to external representations of field values. It maps the indices of the internal value string array of the RecordData to an external representation. E.g., a date field is represented externally as one field, while it is stored in RecordData by using possibly more than one field, e.g., one for the day, one for the month and one for the year. Therefore, it is useful to use a template for setting up this mapping. Then the mapping needs only be created once and can be reused anywhere in the application. Figure 8-11 depicts the UML diagram of the ValueTemplate class.

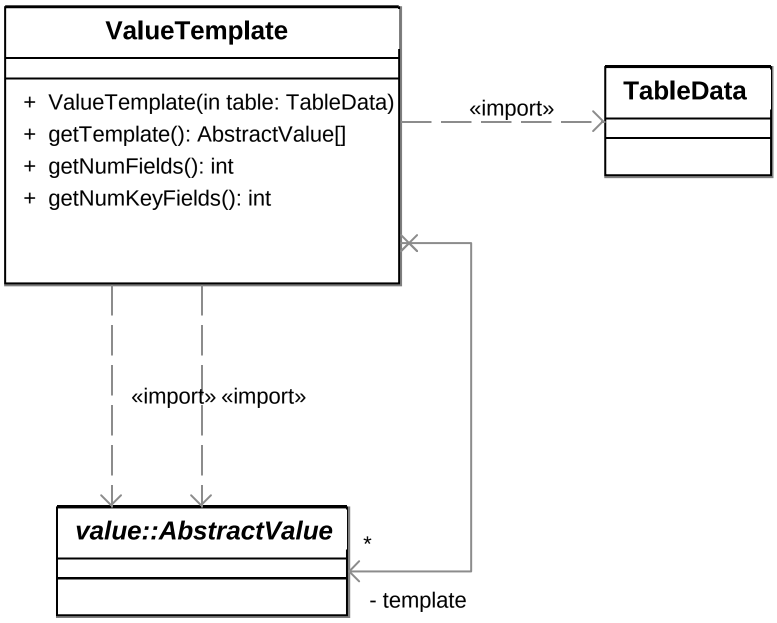


Figure 8-11 UML of ValueTemplate. We depict the dependencies of the ValueTemplate class.

8.3.8 The value package

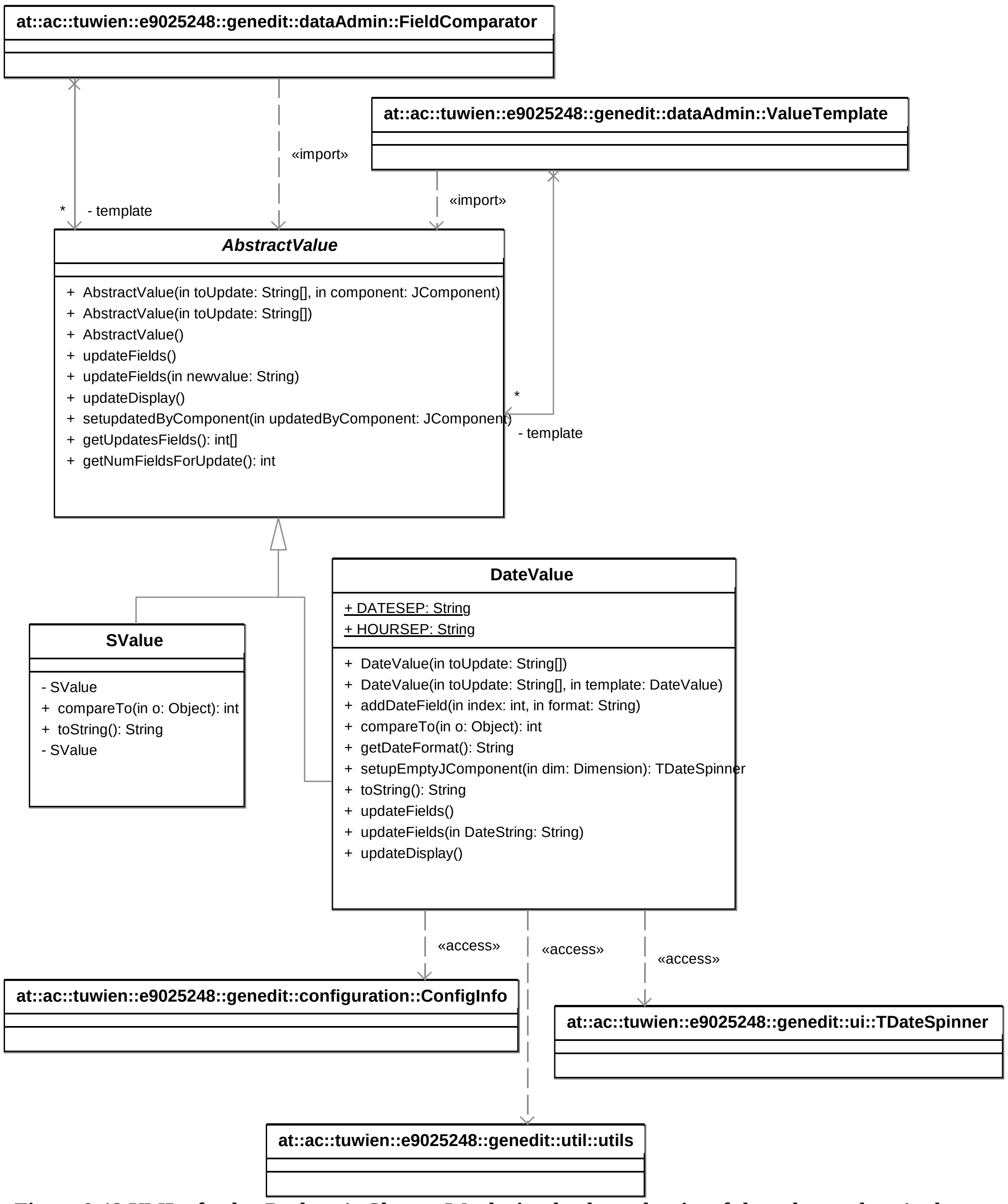
The value package consists of classes for providing the external representation of any String array. This can be for example the value array of a RecordData object. An abstract super class ensures that the classes of this package need to implement common methods. See Table 8-5 for details:

Method	Description
updateFields()	Updates the values of the fields of the String array mapped to by the currently displayed value of the according graphical user interface component
updateFields(String)	Updates the values of the fields of the String array mapped to by the value of the String
updateDisplay()	Updates the displayed value, by the values of the fields of the String array mapped to

Table 8-5 Common Methods of All Objects of the value Package. We list the common methods of all classes of the value package.

In the current release there are only two classes available, which concretely implement such a mapping. These are SValue for Strings and DateValue for dates. All sub-classes of AbstractValue have to implement the compareTo() method, which they inherit from the AbstractValue class. The AbstractValue class forces its sub-classes to implement the Comparable interface. This is where there is another advantage of using a SValue instead of a normal Sting value, because the SValue uses another comparison algorithm, which is to first sort numerically, then alphanumerically, and to have number values always before alphanumeric values.

Figure 8-12 depicts the UML diagram of the value package's classes



at::ac::tuwien::e9025248::genedit::dataAdmin::FieldComparator
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::ValueTemplate
«import»
*
- template
AbstractValue
+ AbstractValue(in toUpdate: String[], in component: JComponent)
+ AbstractValue(in toUpdate: String[])
+ AbstractValue()
+ updateFields()
+ updateFields(in newvalue: String)
+ updateDisplay()
+ setupdatedByComponent(in updatedByComponent: JComponent)
+ getUpdatesFields(): int[]
+ getNumFieldsForUpdate(): int
*
- template
DateValue
+ DATESEP: String
+ HOURSEP: String
+ DateValue(in toUpdate: String[])
+ DateValue(in toUpdate: String[], in template: DateValue)
+ addDateField(in index: int, in format: String)
+ compareTo(in o: Object): int
+ getDateFormat(): String
+ setupEmptyJComponent(in dim: Dimension): TDateSpinner
+ toString(): String
+ updateFields()
+ updateFields(in DateString: String)
+ updateDisplay()
SValue
- SValue
+ compareTo(in o: Object): int
+ toString(): String
- SValue
«access»
«access»
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo
at::ac::tuwien::e9025248::genedit::ui::TDateSpinner
at::ac::tuwien::e9025248::genedit::util::utils

Figure 8-12 UML of value Package's Classes. We depict the dependencies of the value package's classes.

8.4 Input – Output – physical Data Storage

Following, we will look in detail at the way the tables are stored physically.

8.4.1 The Storage Manager Concept for Tables

Matters of physically storing the data have been detached from the internal representation. Instead, storage managers are responsible for the task of physically storing the tables.

8.4.1.1 Abstract Storage Manager

The AbstractStorageManager class forces a common interface for all Storage Managers. All sub-classes of AbstractStorageManager need to implement the inherited abstract methods (see Table 8-6):

Method	**Description**
loadTable(TableData)	To load the table from the physical storage into the internal representation.
storeTable(TableData)	To store the table physically.

Table 8-6 Public Methods of AbstractStorageManager. We list the common methods of the storage managers.

A concrete storage manager can register itself as storage manager of a table. After registration of at least one storage manager, the table uses all of its registered storage managers to store the table physically. Therefore, each storing process requires three steps:

- Setting a record for update or deletion (added records are set to be updated)
- Physically storing the table by all its storage managers
- Removing the records set for deletion and removing all the update flags.

Figure 8-13 depicts the UML diagram of the AbstractStorageManager class. The storage managers for measurement data inherit form AbstractXMLAnswerManager and are not included in this figure.

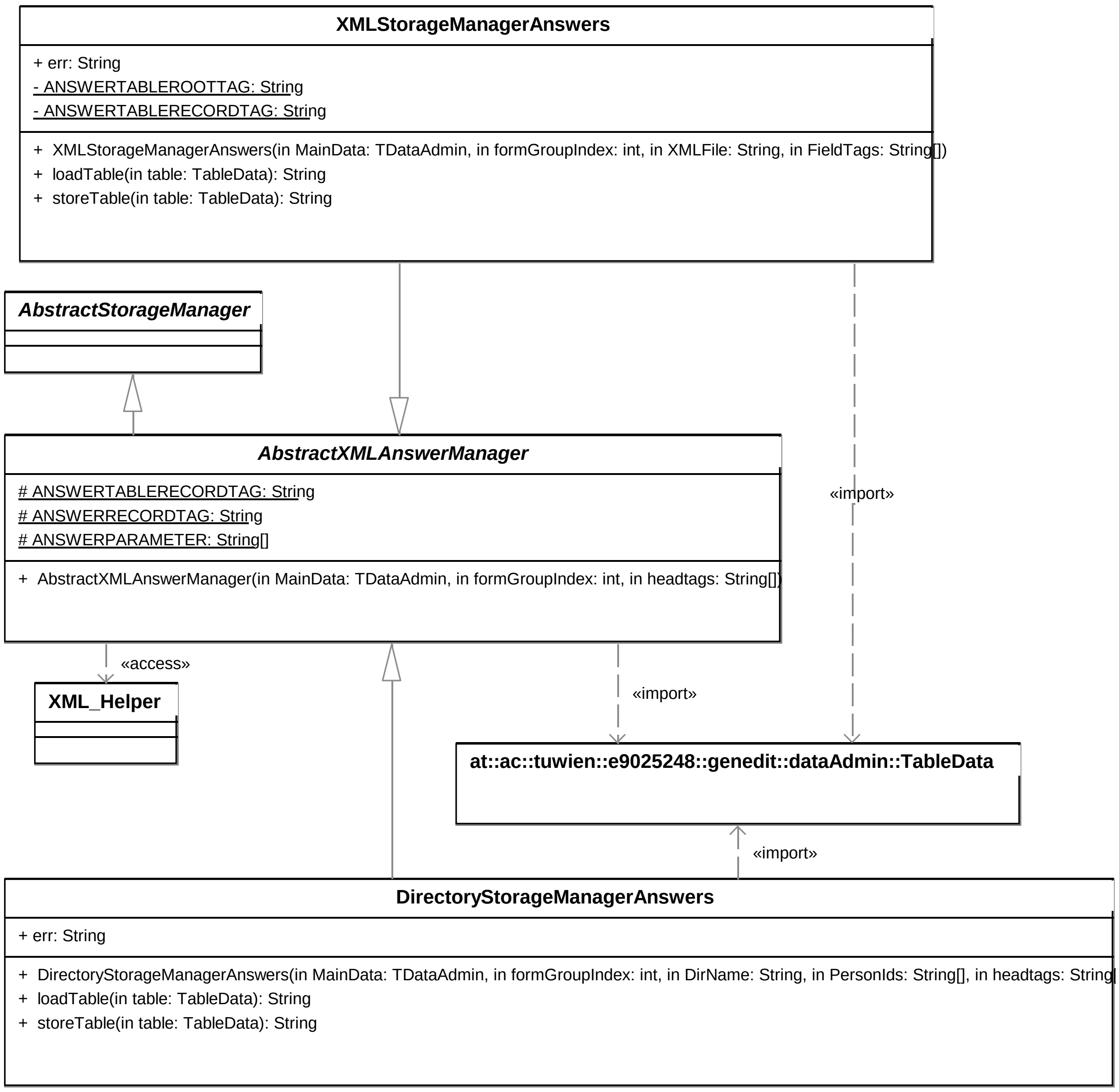

XMLStorageManagerAnswers
+ err: String
- ANSWERTABLEROOTTAG: String
- ANSWERTABLERECORDTAG: String
+ XMLStorageManagerAnswers(in MainData: TDataAdmin, in formGroupIndex: int, in XMLFile: String, in FieldTags: String[])
+ loadTable(in table: TableData): String
+ storeTable(in table: TableData): String
AbstractStorageManager
AbstractXMLAnswerManager
# ANSWERTABLERECORDTAG: String
# ANSWERRECORDTAG: String
# ANSWERPARAMETER: String[]
+ AbstractXMLAnswerManager(in MainData: TDataAdmin, in formGroupIndex: int, in headtags: String[])
«import»
«access»
XML_Helper
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::TableData
«import»
DirectoryStorageManagerAnswers
+ err: String
+ DirectoryStorageManagerAnswers(in MainData: TDataAdmin, in formGroupIndex: int, in DirName: String, in PersonIds: String[], in headtags: String
+ loadTable(in table: TableData): String
+ storeTable(in table: TableData): String

Figure 8-13 UML of the Storage Manager Classes. We depict the dependencies of the storage manager classes.

8.4.1.2 Storage Managers for Parameter Sources

In the current release there are two storage managers available for parameter sources. The first one is needed only once, at the first start up of the program, and only if there is data from former applications available. It is the DirectoryStorageManagerReplier. It reads the available parameter sources from the PersDir (see section 8.2): These are all subdirectories which have a numerical name. After having completed, the XMLStorageManagerReplier is used to store the parameter source table and the DirectoryStorageManagerReplier is removed from the list of the parameter source table's storage managers. During all further start ups, only the XMLStorageManagerReplier is used to read in the available parameter sources. There is only one exception and that is if the PersDir has been moved to somewhere else, in this case the DirectoryStorageManagerReplier is used again to read the additional parameter sources from the new directory. Details on the XML file format the XMLStorageManagerReplier uses can be found in the appendix (see chapter IV.2.7). Figure 8-14 depicts the UML diagram of the XMLStorageManagerReplier class and Figure 8-15 of the DirectoryStorageManagerReplier.

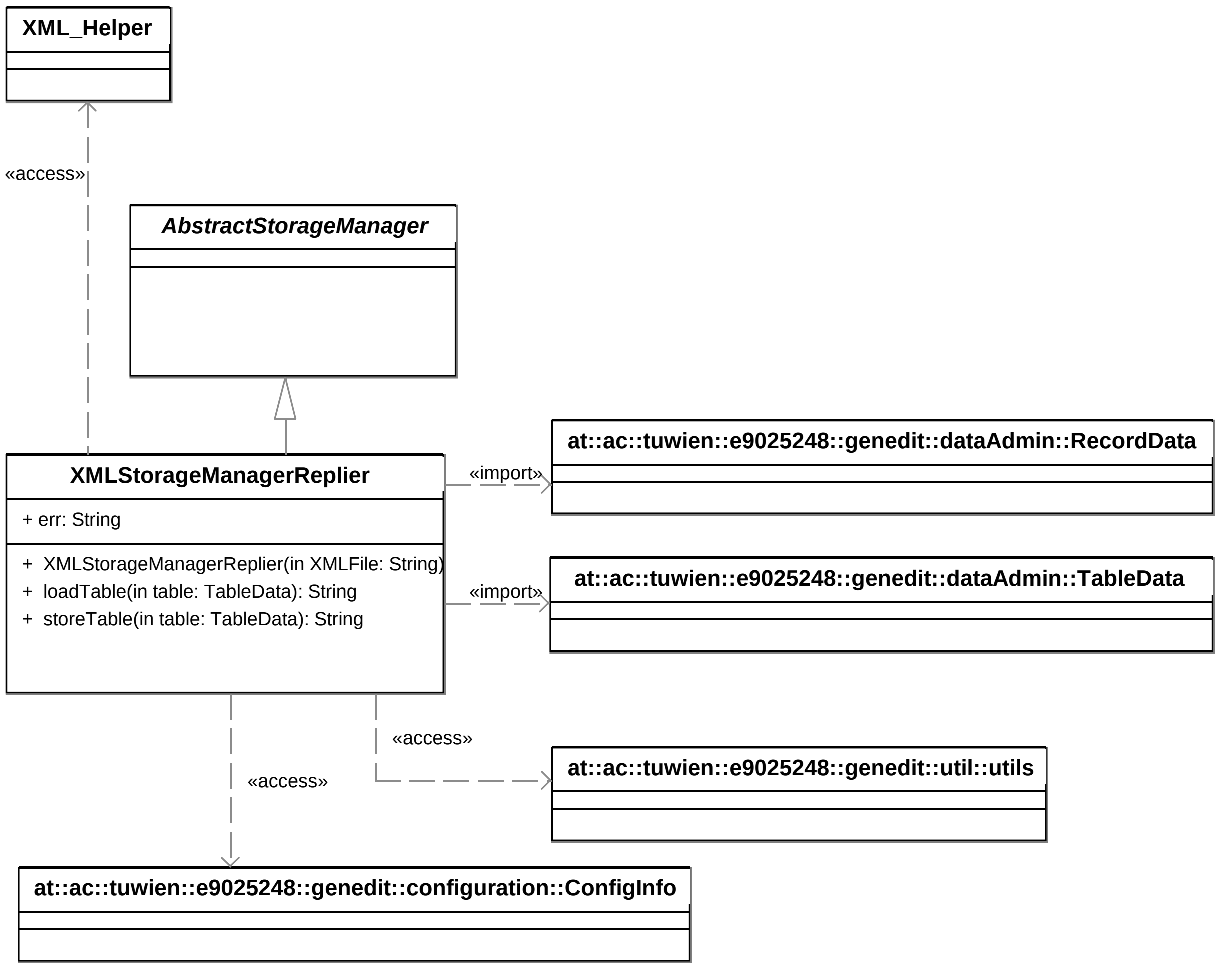

XML_Helper
«access»
AbstractStorageManager
XMLStorageManagerReplier
+ err: String
+ XMLStorageManagerReplier(in XMLFile: String)
+ loadTable(in table: TableData): String
+ storeTable(in table: TableData): String
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::RecordData
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::TableData
«access»
at::ac::tuwien::e9025248::genedit::util::utils
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo

Figure 8-14 UML of XMLStorageManagerReplier. We depict the dependencies of the XMLStorageManagerReplier class.

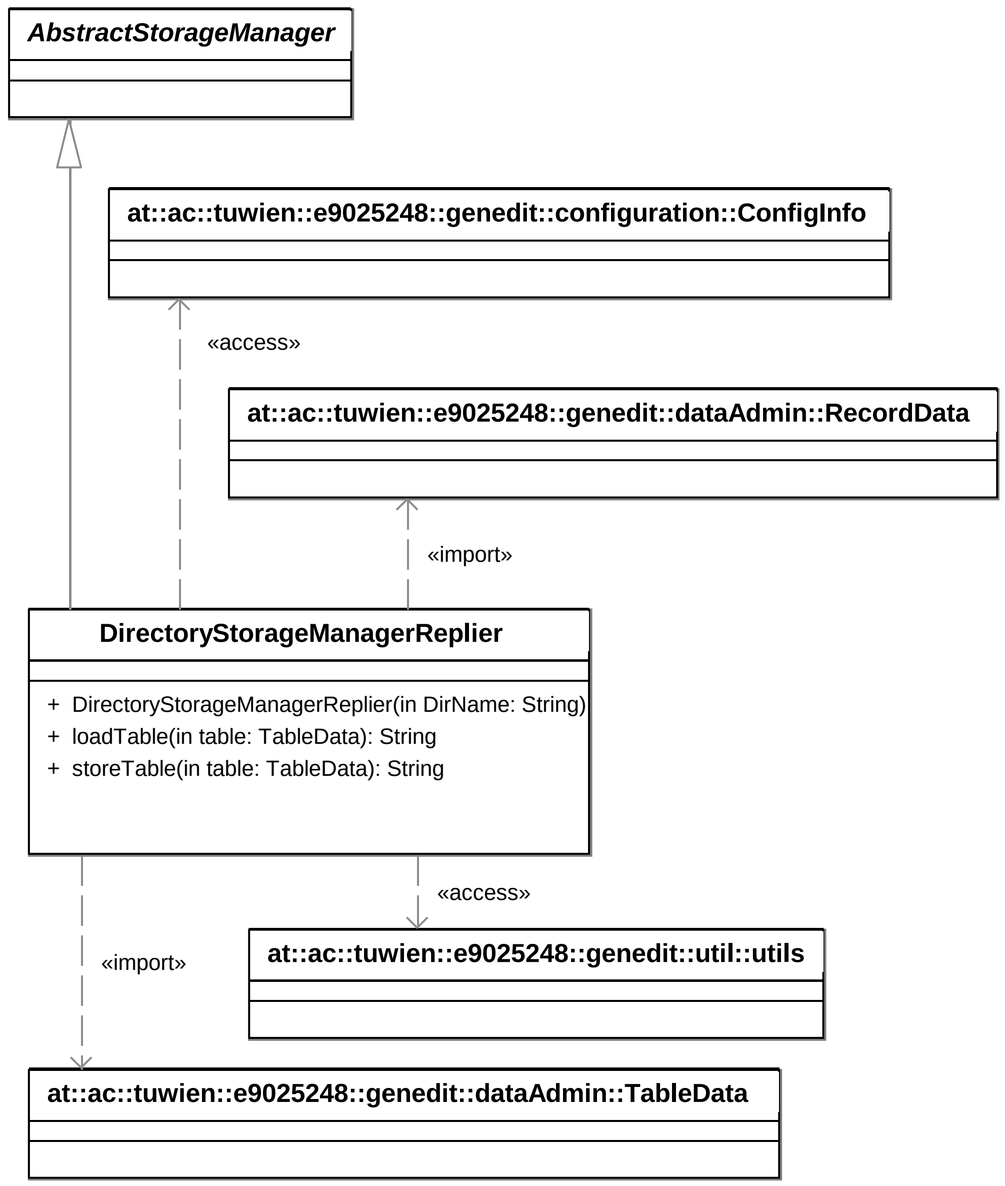

AbstractStorageManager
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo
«access»
at::ac::tuwien::e9025248::genedit::dataAdmin::RecordData
«import»
DirectoryStorageManagerReplier
+ DirectoryStorageManagerReplier(in DirName: String)
+ loadTable(in table: TableData): String
+ storeTable(in table: TableData): String
«access»
«import»
at::ac::tuwien::e9025248::genedit::util::utils
at::ac::tuwien::e9025248::genedit::dataAdmin::TableData

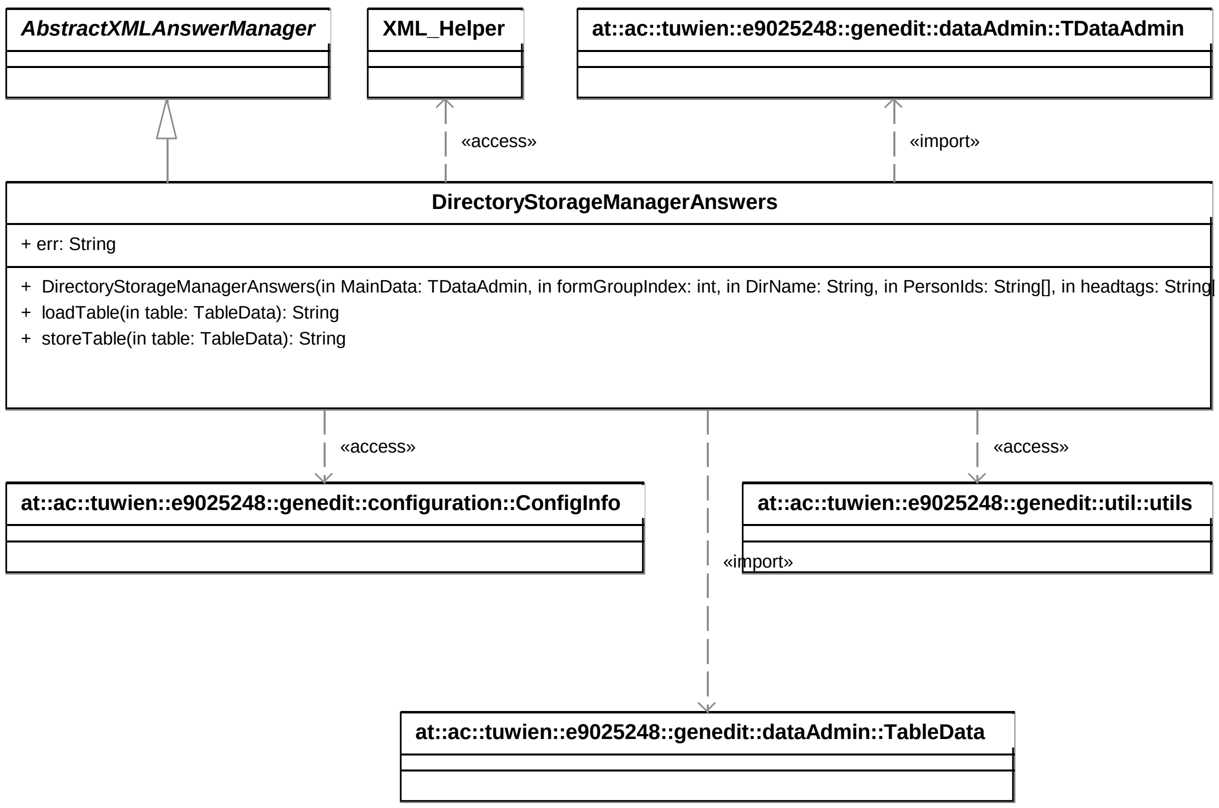
Figure 8-15 UML of DirectoryStorageManagerReplier. We depict the dependencies of the DirectoryStorageManagerReplier class.

8.4.1.3 Storage Managers for Measurement Data

There are also two storage managers available for measurement data, the DirectoryStorageManagerAnswers and the XMLStorageManagerAnswers.
The DirectoryStorageManagerAnswers reads and stores each measurement set in a separate file. The location of the file is in the subdirectory AnswerDir (see section 8.2), which is a subdirectory of the parameter source's directory, which is in turn a subdirectory of the PersDir (see section 8.2). The filename consists of the parameter source id, the form id and the date, possibly followed by an underscore and a number, if the same file name was already used to store another measurement set. Duplicate filename can occur, because the filename, as well as the directory name need to be normalized, in the sense that all characters, which could offend the local file system or which would lead to a malformed URL are removed. E.g.:

Parameter source	**Form-Id**	**Date**	**⇒ Filename**
x?	t	01.12.2006	⇒ xt01122006.xml
x	*t	01.12.2006	⇒ xt01122006.xml
*	§§	01.12.2006	⇒ EMPTY01122006.xml

Of course, removing characters can also lead to an empty subdirectory for a parameter source. To avoid this, all empty parameter source subdirectories are replaced by “EMPTY”. Similarly, an empty concatenation of a normalized form id with a normalized parameter source id is also replaced by “EMPTY”. Details on the XML file format the DirectoryStorageManagerAnswers uses can be found in the appendix (see chapter IV.2.5).
On the contrary, the XMLStorageManagerAnswers stores all measurement sets of a form group into a single file. The filename is determined by the AnswerFile variable of the ConfigInfo class (see section 8.2). There is only one difference in the file format used that is, that all measurement sets are wrapped by another XML tag for providing a single root tag. This is required by the XML standard. Details of the XML format can be found in [WWW-4] and [WWW-5]. Details on the XML file format the XMLStorageManagerAnswers uses can be found in the appendix (see chapter IV.2.4). An XML tutorial can be found at [WWW-10].
So the XMLStorageManagerAnswers and the DirectoryStorageManagerAnswers have some functionality in common: Therefore, they have a common abstract super class, which is the AbstractXMLAnswerManager.
Figure 8-16 and Figure 8-17 UML of XMLStorageManagerAnswers depicts the UML diagram of the answer storage manager. Further dependences which are not included in the figure are the use of the utils.util and configuration.ConfigInfo classes.



AbstractXMLAnswerManager
XML_Helper
at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
«access»
«import»
DirectoryStorageManagerAnswers
+ err: String
+ DirectoryStorageManagerAnswers(in MainData: TDataAdmin, in formGroupIndex: int, in DirName: String, in PersonIds: String[], in headtags: String
+ loadTable(in table: TableData): String
+ storeTable(in table: TableData): String
«access»
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo
at::ac::tuwien::e9025248::genedit::util::utils
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::TableData

Figure 8-16 UML of DirectoryStorageManagerAnswers. We depict the dependencies of the DirectoryStorageManagerAnswers class.

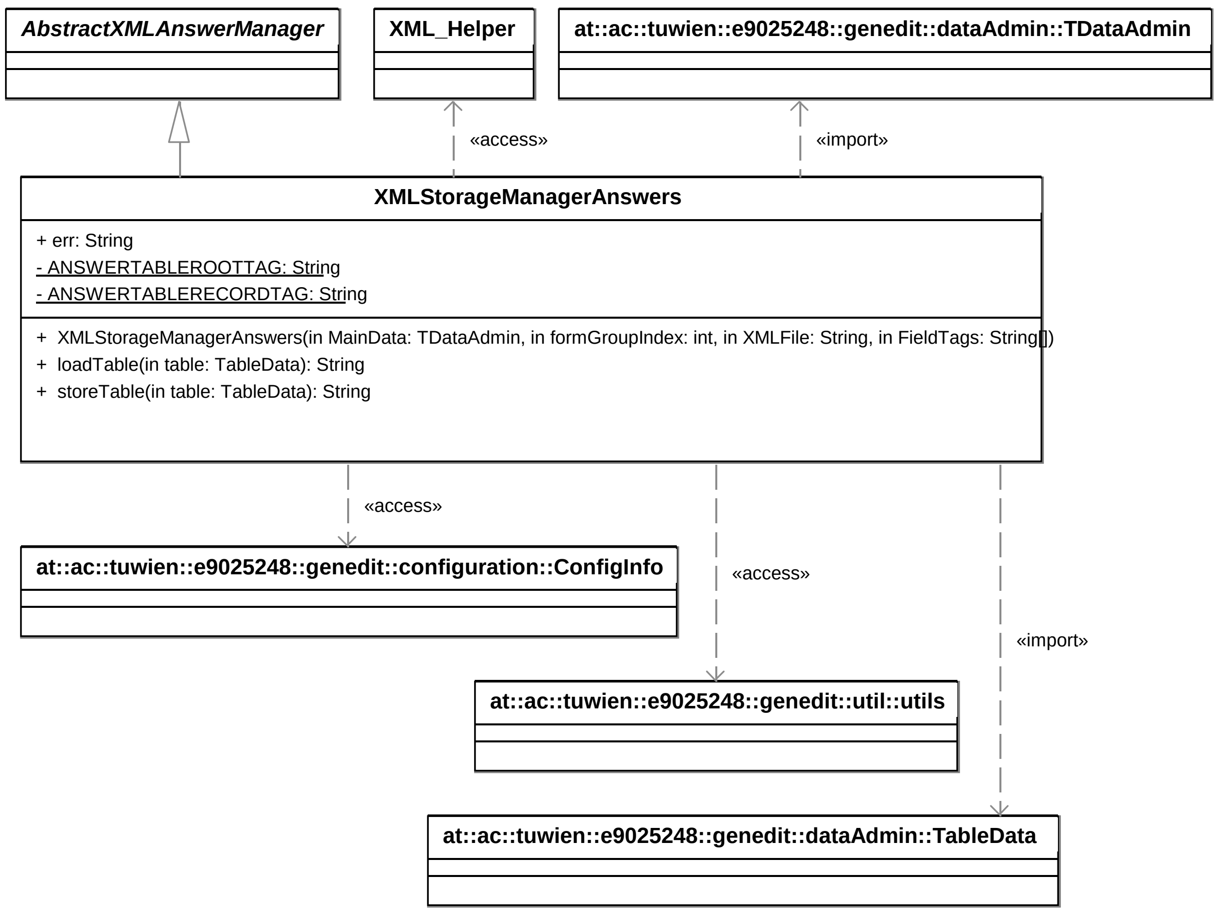

AbstractXMLAnswerManager
XML_Helper
at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
«access»
«import»
XMLStorageManagerAnswers
+ err: String
- ANSWERTABLEROOTTAG: String
- ANSWERTABLERECORDTAG: String
+ XMLStorageManagerAnswers(in MainData: TDataAdmin, in formGroupIndex: int, in XMLFile: String, in FieldTags: String[])
+ loadTable(in table: TableData): String
+ storeTable(in table: TableData): String
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo
«access»
«import»
at::ac::tuwien::e9025248::genedit::util::utils
at::ac::tuwien::e9025248::genedit::dataAdmin::TableData

Figure 8-17 UML of XMLStorageManagerAnswers. We depict the dependencies of the XMLStorageManagerAnswers class.

During the loading process the measurement data is checked for:

- Existence of the parameter source
- Existence of the form
- Existence of the question within the read form
- Validity of the value (→ required data type of the answer, within a possibly defined range, value is allowed, if there is a list of allowed values for a specific question)

If any measurement sets with unknown parameter sources are read, these parameter sources will automatically be created in the parameter source table and the parameter source table will be stored physically upon the completion of the loading process. If there are any other problems detected, the application displays a warning message.

8.4.1.4 The Storage Manager for Forms

In the current release only one storage manager is implemented for storing forms. It is the XMLStorageManagerForms. The filename is determined by the profile (see section 8.2). The format is the same as used in related programs. Details of the format can be found in the appendix (see chapter IV.2.6).

8.4.2 The Importer

The importer is capable of importing measurement data in two different CSV file formats. Details about the formats can be found in the appendix (see chapter IV.2.2 and chapter IV.2.3 respectively). During the import process the measurement data is checked for:

- Existence of the parameter source
- Existence of the form

- Existence of the question within the read form
- Validity of the value (→ required data type of the answer, within a possibly defined range, value is allowed, if there is a list of allowed values for a specific question)

If any measurement sets with unknown parameter sources are read, these parameter sources will be automatically created in the parameter source table and the parameter source table will be stored physically upon completion of the import. If there are any other problems detected, the application displays a warning message. Figure 8-18 depicts the UML diagram of the Importer class.

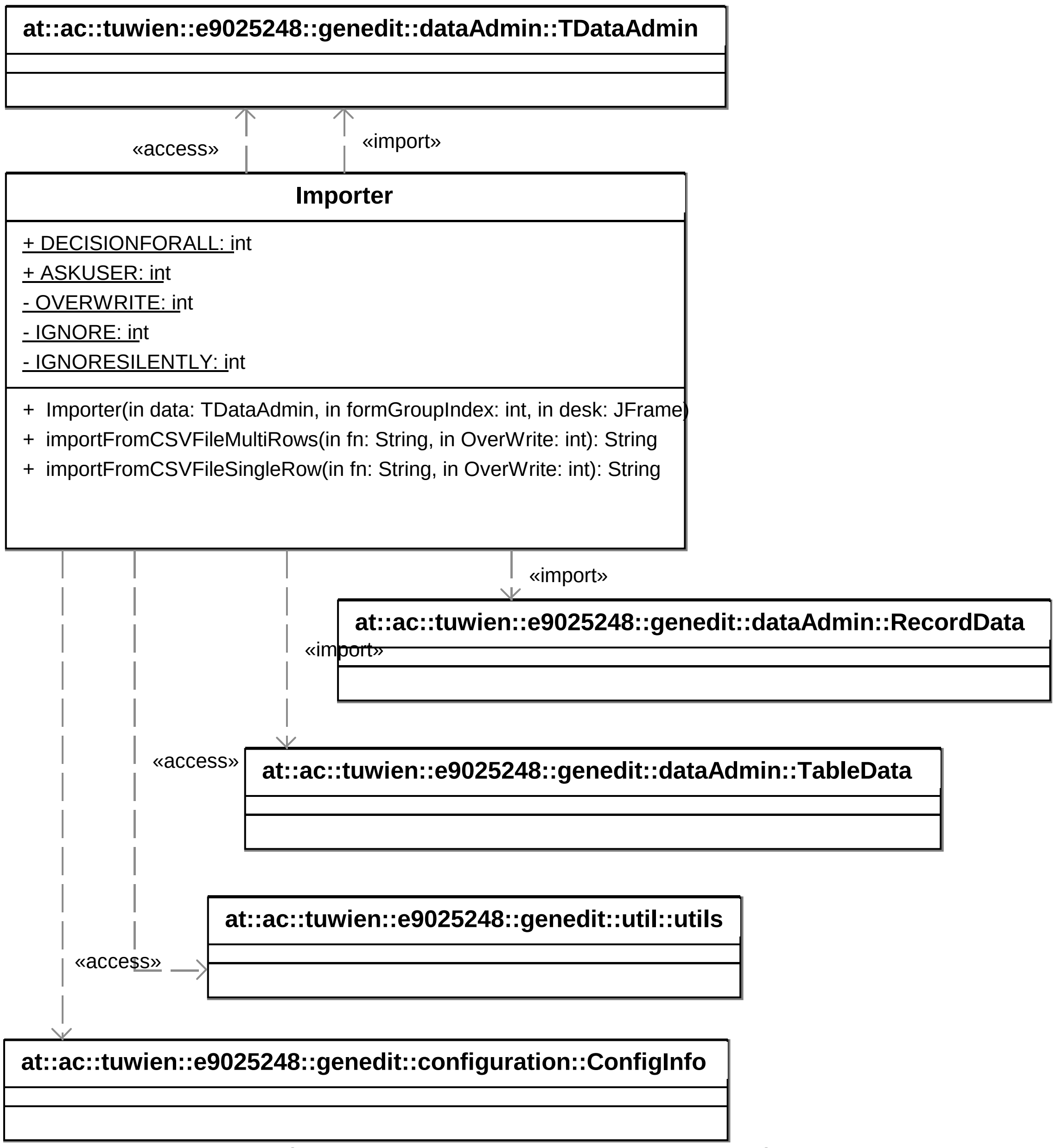

at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
«access»
«import»
Importer
+ DECISIONFORALL: int
+ ASKUSER: int
- OVERWRITE: int
- IGNORE: int
- IGNORESILENTLY: int
+ Importer(in data: TDataAdmin, in formGroupIndex: int, in desk: JFrame)
+ importFromCSVFileMultiRows(in fn: String, in OverWrite: int): String
+ importFromCSVFileSingleRow(in fn: String, in OverWrite: int): String
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::RecordData
«import»
«access»
at::ac::tuwien::e9025248::genedit::dataAdmin::TableData
at::ac::tuwien::e9025248::genedit::util::utils
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo

Figure 8-18 UML of the Importer. We depict the dependencies of the Importer class.

8.4.3 The Exporter

The Exporter is capable of exporting the measurement data to different file formats. These are the Conflation file format, the CSV-format multiple row and the AnswerFile format. It is possible to export only a subset of all records. For this purpose the exporter is provided with a list of indices, which conform to the filter settings. Furthermore, the Exporter is capable of scaling the measurement values. Both scaling and filtering is not supported with the

AnswerFile format, because it is only for being able to migrate measurement data stored in a directory structure to a single answer file (one for each form group). Further details about the exporter can be found in chapter III.9.4.4.1.
Figure 8-19 depicts the UML diagram of the Exporter class.

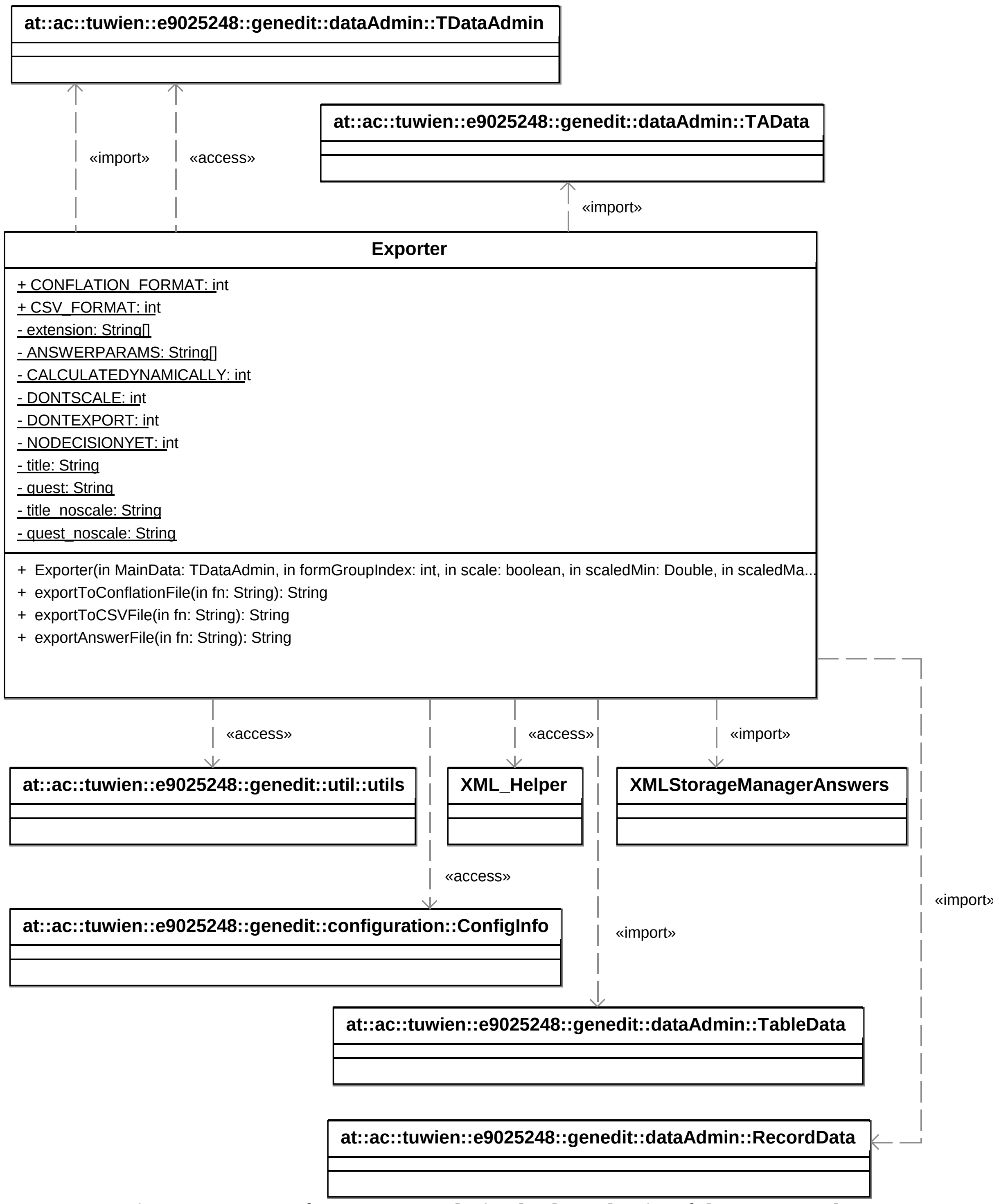

at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
«import»
«access»
at::ac::tuwien::e9025248::genedit::dataAdmin::TAData
«import»
Exporter
+ CONFLATION_FORMAT: int
+ CSV_FORMAT: int
- extension: String[]
- ANSWERPARAMS: String[]
- CALCULATEDYNAMICALLY: int
- DONTSCALE: int
- DONTEXPORT: int
- NODECISIONYET: int
- title: String
- quest: String
- title_noscale: String
- quest_noscale: String
+ Exporter(in MainData: TDataAdmin, in formGroupIndex: int, in scale: boolean, in scaledMin: Double, in scaledMa...
+ exportToConflationFile(in fn: String): String
+ exportToCSVFile(in fn: String): String
+ exportAnswerFile(in fn: String): String
«access»
«access»
«import»
at::ac::tuwien::e9025248::genedit::util::utils
XML_Helper
XMLStorageManagerAnswers
«access»
«import»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::TableData
at::ac::tuwien::e9025248::genedit::dataAdmin::RecordData

Figure 8-19 UML of Exporter. We depict the dependencies of the Exporter class.

8.4.4 The GenEdit Print Utility

The class GenEditPrintUtility provides functionality to print a Vector of a Vector of graphical components. The first Vector keeps track of the lines, the second Vector of the Components of this line. A head is added at the beginning of each page, and a footer, which shows the page number information at the end of each page. The lines are automatically split into pages. For each page, a JFrame, which covers the header, the components to be printed and the footer, is created. The core of the print utility is a modification of the Marty Hall's simple PrintUtility, which was downloaded from [WWW-6]. Marty Hall's utility was modified so that:

- It prints a couple of JComponents, which are retrieved from a Vector called LinesToPrint.
- It creates pages (JFrames), which contain these lines
- The page breaks are calculated according to the height of the lines.
- The printing process is starting in an own thread.
- It is restricted to printing JComponents which are placed in a JFrame. Top-level user interface windows cannot be placed in a JFrame. Therefore, one can't use JApplet, JFrame, JDialog or JWindow as a component in a line. Should such Components exist in a line the utility skips them quietly.

It is used from the record data window, to print a questionnaire together with its values and from the design form window to print empty forms. It is called from the record data window only. Still the user can print empty forms from the design window. For this purpose a record data window is created in preview mode by the design form window. Figure 8-20 depicts the UML diagram of the GenEditPrintUtility class.

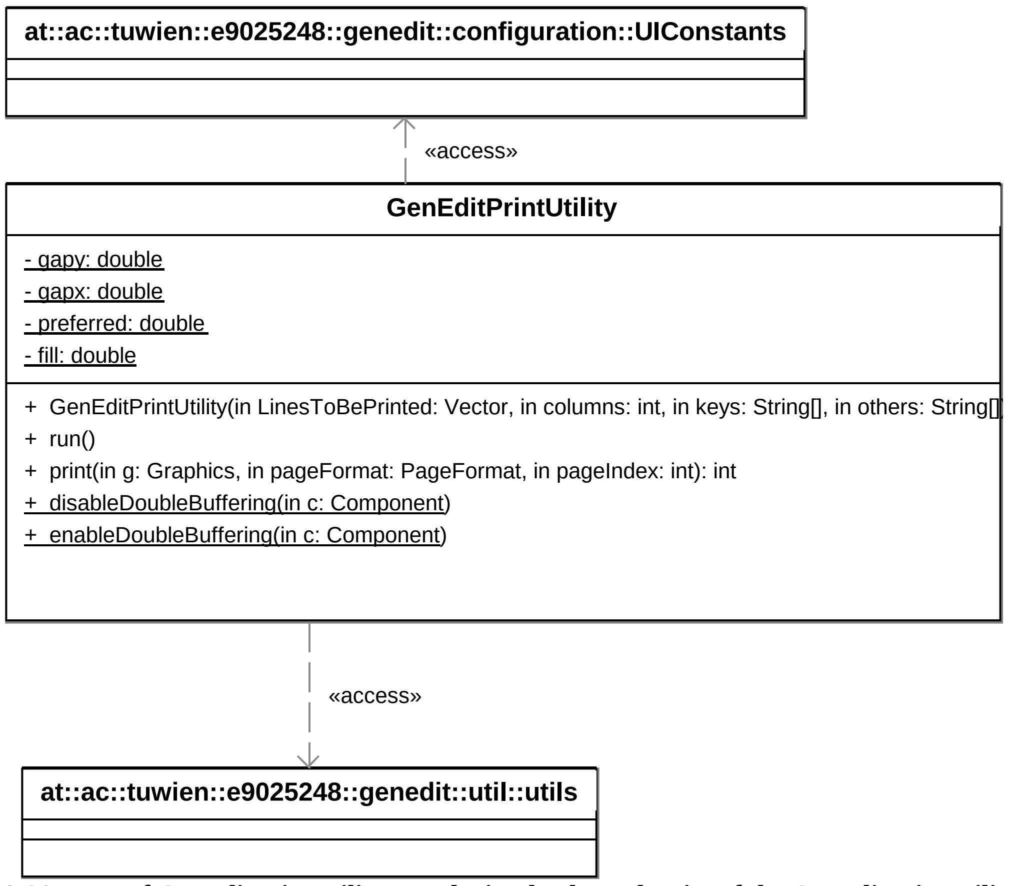

at::ac::tuwien::e9025248::genedit::configuration::UIConstants
«access»
GenEditPrintUtility
- gapy: double
- gapx: double
- preferred: double
- fill: double
+ GenEditPrintUtility(in LinesToBePrinted: Vector, in columns: int, in keys: String[], in others: String[]
+ run()
+ print(in g: Graphics, in pageFormat: PageFormat, in pageIndex: int): int
+ disableDoubleBuffering(in c: Component)
+ enableDoubleBuffering(in c: Component)
«access»
at::ac::tuwien::e9025248::genedit::util::utils

Figure 8-20 UML of GenEditPrintUtility. We depict the dependencies of the GenEditPrintUtility class.

8.4.5 Helper Methods for IO Access

Some static methods are provided by the class XML_Helper for file input and output, most of them for reading and writing XML Files. These are listed in Table 8-7:

Method	Description
closeFile(PrintWriter)	Closes the PrintWriter
closeTag(PrintWriter, boolean)	Closes the tag, if the boolean argument is true, there is still some text to come
createEmptyFile(String)	Creates an empty file with the given filename
getAttribValue(Node, String)	Extracts the value of the given argument and returns it.
makeValidArgument(String)	Replaces all invalid characters from a string and ensures correct ISO-8859-1 encoding
openXMLFile(String)	Opens an XML File for Reading
openXMLFileForWriting(String)	Opens an XML File for writing by using standard IO methods
openXMLforStoring()	Creates an empty DOM structure for output (no longer used)
serializeDOMtoFile(Document, String)	Serializes a DOM Document to disk (no longer used)
writeArgument(PrintWriter, String, String)	Writes an Argument using standard IO methods
writeClosingTag(PrintWriter, int, String)	Writes the closing tag
writeTagAndArguments(PrintWriter, int, String, String[], String[])	Writes a tag together with arguments and values
writeXMLHeader()	Writes the XML header information

Table 8-7 Methods for I/O Access. We list the helper methods for I/O access.

These static methods are used at many places in the code. Figure 8-21 UML of XML_Helper depicts the UML diagram of the XML_Helper class.

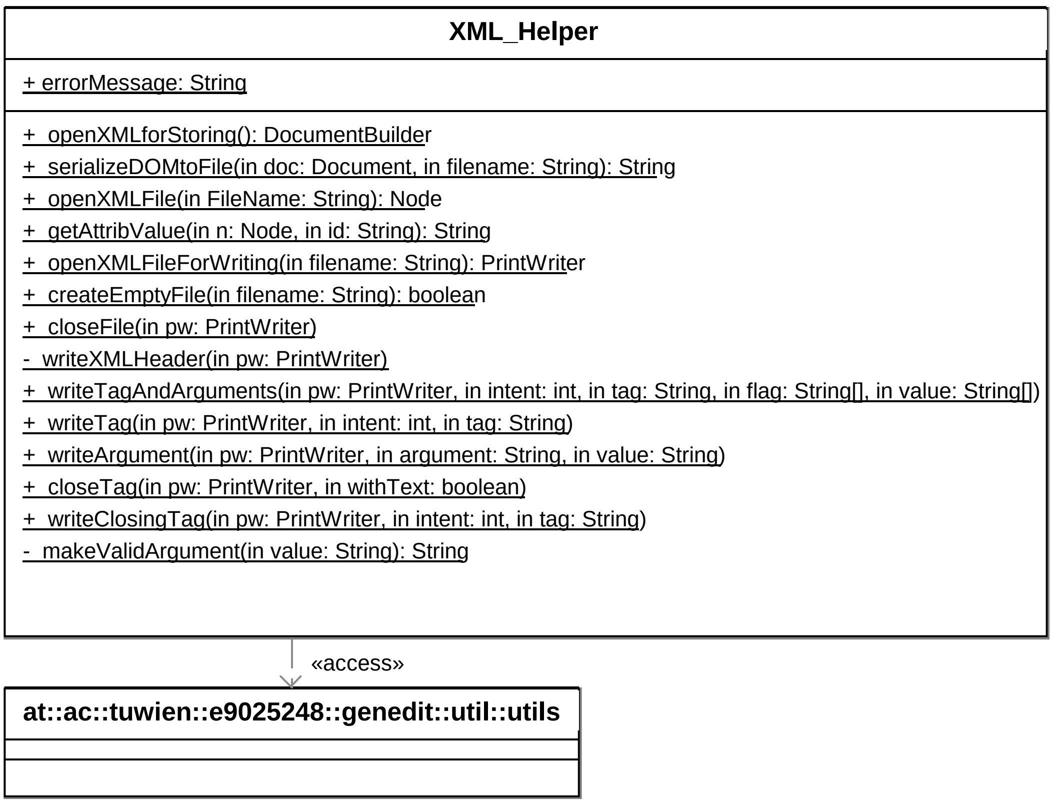

XML_Helper
+ errorMessage: String
+ openXMLforStoring(): DocumentBuilder
+ serializeDOMtoFile(in doc: Document, in filename: String): String
+ openXMLFile(in FileName: String): Node
+ getAttribValue(in n: Node, in id: String): String
+ openXMLFileForWriting(in filename: String): PrintWriter
+ createEmptyFile(in filename: String): boolean
+ closeFile(in pw: PrintWriter)
- writeXMLHeader(in pw: PrintWriter)
+ writeTagAndArguments(in pw: PrintWriter, in intent: int, in tag: String, in flag: String[], in value: String[])
+ writeTag(in pw: PrintWriter, in intent: int, in tag: String)
+ writeArgument(in pw: PrintWriter, in argument: String, in value: String)
+ closeTag(in pw: PrintWriter, in withText: boolean)
+ writeClosingTag(in pw: PrintWriter, in intent: int, in tag: String)
- makeValidArgument(in value: String): String
«access»
at::ac::tuwien::e9025248::genedit::util::utils

Figure 8-21 UML of XML_Helper. We depict the dependencies of the XML_Helper class.

8.5 The User Interface

All user interface classes reside in at.ac.tuwien.e9025248.genedit.ui. All classes implementing user interface windows have one or more inner classes for event handling of the menu actions. These inner classes inherit from AbstractAction. The AbstractAction class is in turn an abstract implementation of the Action interface. “An action allows the programmer to bundle a commonly used procedure and its bound properties (such as its name and an image to represent it) into a single class. This construct comes in handy if an application needs to call upon a particular function from multiple sources.” [Loy 2002 p. 41]
This is particularly useful, because nearly all the actions of each window can be invoked either from the menu or from a toolbar. So, if an action has to be disabled for some reason, the action itself can be disabled and the menu and the toolbar objects are automatically notified. Further details about actions can be found in [Loy 2002].
On the contrary, each window listens to actions which are triggered by single elements by implementing the ActionListener interface. In order to react to the closing of a window and to be able to ask the user, if he wants to store the changes he made, another inner class is used in most of the user interface classes.

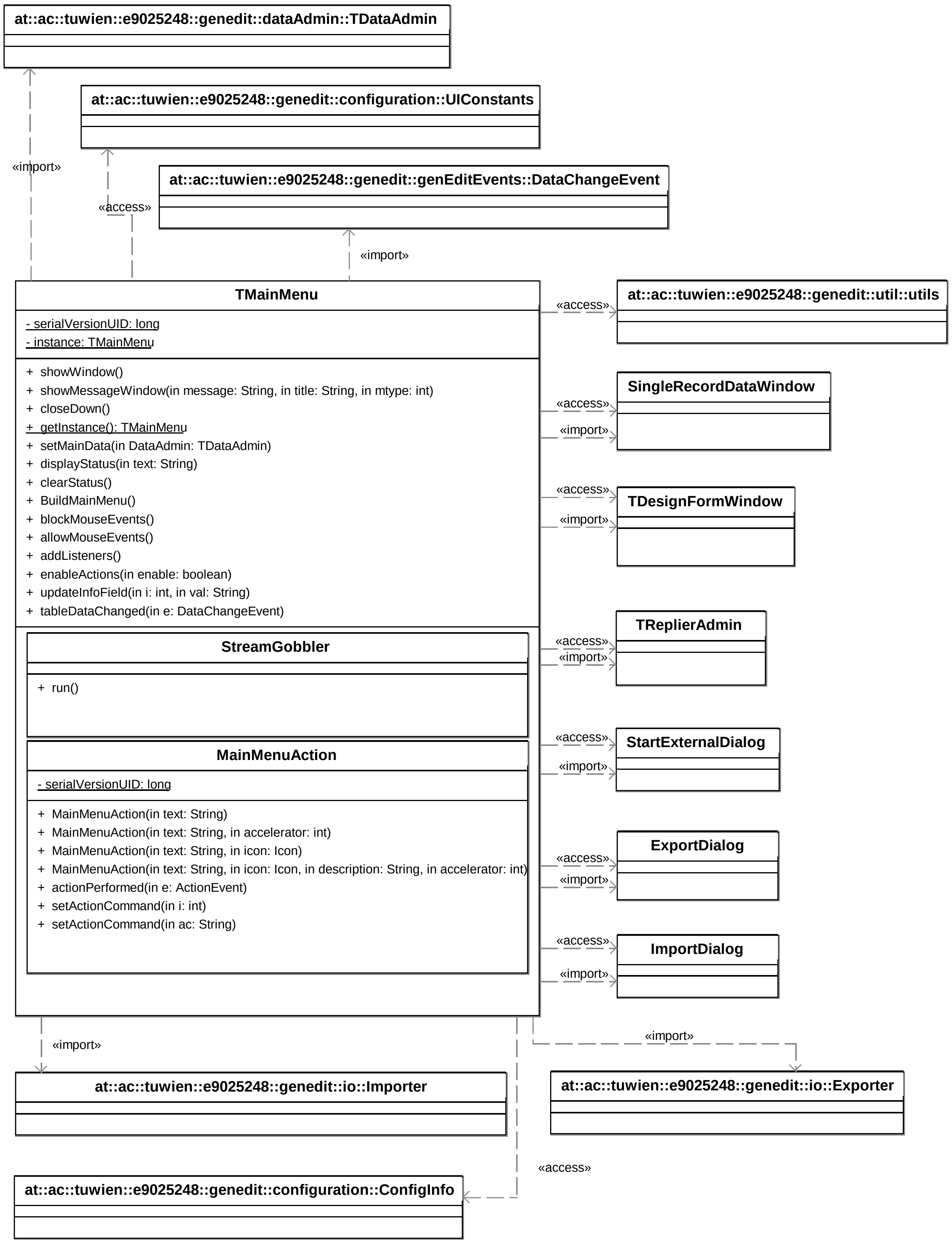
8.5.1 The Main Window

The main menu is displayed at program startup, to be able to assign a root window to error messages, which might need to be displayed during the program startup. Once the profile is completely read, the main window is updated, so that the main window’s menus reflect the form groups and the title of the window reflects the purpose of the concrete application. The content pane of the main menu is used to display some basic descriptive statistics about the stored data. The fields for displaying them are also created immediately after the profile (see section 8.2) has been read, because the number of fields necessary is evident at that time. The values of the fields are updated each time further information is available. The class implementing the main window is called TMainMenu. Table 8-8 describes the main windows public methods in detail.

Method	**Description**
allowMouseEvents()	Allows mouse events
blockMouseEvents()	Blocks mouse events
BuildMainMenu()	Builds the main menu according to the profile
clearStatus()	Clears the text displayed in the status field, which is displayed at the bottom of the window
displayStatus(String)	Displays the given string in the status field
enableActions(Boolean)	Depending on the Boolean value, the actions are either enabled or disabled
setMainData(TDataAdmin)	Provides the application with the main data
showMessageWindow(String, String, int)	Displays a given warning or error message, with a given title, the type of the window depends on the int argument
showWindow()	Displays the main window
updateInfoField(int, String)	Displays the given String in the info field with index int → the info fields display the statistical values.

Table 8-8 Public Methods of TMainMenu. We describe the public methods of the TMainMenu class.

Figure 8-22 depicts the UML diagram of the TMainMenu class.



at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
at::ac::tuwien::e9025248::genedit::configuration::UIConstants
at::ac::tuwien::e9025248::genedit::genEditEvents::DataChangeEvent
«import»
«access»
«import»
TMainMenu
- serialVersionUID: long
- instance: TMainMenu
+ showWindow()
+ showMessageWindow(in message: String, in title: String, in mtype: int)
+ closeDown()
+ getInstance(): TMainMenu
+ setMainData(in DataAdmin: TDataAdmin)
+ displayStatus(in text: String)
+ clearStatus()
+ BuildMainMenu()
+ blockMouseEvents()
+ allowMouseEvents()
+ addListeners()
+ enableActions(in enable: boolean)
+ updateInfoField(in i: int, in val: String)
+ tableDataChanged(in e: DataChangeEvent)
StreamGobbler
+ run()
MainMenuAction
- serialVersionUID: long
+ MainMenuAction(in text: String)
+ MainMenuAction(in text: String, in accelerator: int)
+ MainMenuAction(in text: String, in icon: Icon)
+ MainMenuAction(in text: String, in icon: Icon, in description: String, in accelerator: int)
+ actionPerformed(in e: ActionEvent)
+ setActionCommand(in i: int)
+ setActionCommand(in ac: String)
«access»
at::ac::tuwien::e9025248::genedit::util::utils
«access»
«import»
SingleRecordDataWindow
«access»
«import»
TDesignFormWindow
«access»
«import»
TReplierAdmin
«access»
«import»
StartExternalDialog
«access»
«import»
ExportDialog
«access»
«import»
ImportDialog
«import»
«import»
at::ac::tuwien::e9025248::genedit::io::Importer
at::ac::tuwien::e9025248::genedit::io::Exporter
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo

Figure 8-22 UML of TMainMenu. We depict the dependencies of the TMainMenu class.

Following, there are some notes to some of the methods:
The methods allowMouseEvents() and blockMouseEvents() work as follows: First a non-opaque JPanel is created. Then an empty mouse listener is associated with it and the glass pane of the window is set to this JPanel (see Code 8-1)

```
// for blocking mouse events overwrite the glass pane by glass
  glass = new JPanel(new GridLayout(0,1));
  glass.setOpaque(false);
  glass.addMouseListener(new MouseAdapter() {});
  glass.addMouseMotionListener(new MouseMotionAdapter() {});
  myDesc.setGlassPane(glass);
```

Code 8-1 Blocking Mouse Events. We list an example code for blocking mouse events.

Thereafter, mouse events can be blocked by setting the JPanel glass visible, and allowed by setting the JPanel glass invisible again. This works, because mouse events are sent to the top component, if components are positioned on top of each other, and because the glass pane is the first component in the container, meaning that it will be painted last and therefore, be top. So, if the top component has registered mouse listeners, the events are not sent to the covered components. In order to block mouse events one simple needs to create a new JPanel to use as the glass pane. This panel will listen for all mouse events and do nothing with them. Once the glass pane is made visible, none of the main window's actions can be started by mouse clicks. Further details on blocking mouse events this way can be found in [Loy 2002 pp. 231]. In addition to blocking the mouse events, also all actions are disabled during the data loading process. This is necessary to ensure that the data has been loaded completely, before starting to work with the application.

Once the data loading process has completed the user can trigger action for

- Recording measurement data
- Designing forms
- Maintenance of parameter sources and
- Importing or exporting data

Most of the user actions in the main window lead to opening of other windows, in which the work is done. Exceptions are the export and import facilities. If the user selects one of these, a dialog is started for choosing the options. Once the dialog is closed, and if the user chose the approve button, the instance of the TMainMenu class invokes the actual import or export process and possibly starts an external program (depending on the action chosen). Regarding the start of the external program, I used the hints found on [WWW-19] to get it working. According to JDK's Javadoc, problems might occur on some platforms: "The Runtime.exec methods may not work well for special processes on certain native platforms, such as native windowing processes, daemon processes, Win16/DOS processes on Win32, or shell scripts. The created subprocess does not have its own terminal or console. All its standard io (i.e. stdin, stdout, stderr) operations will be redirected to the parent process through three streams (Process.getOutputStream(), Process.getInputStream(), Process.getErrorStream()). The parent process uses these streams to feed input to and get output from the subprocess. Because some native platforms only provide limited buffer size for standard input and output streams, failure to promptly write the input stream or read the output stream of the subprocess may cause the subprocess to block, and even deadlock." (see [WWW-20]). The solution to this problem is to empty the standard error and the standard output stream at best at once as can be found in [WWW-19]. The TMainMenu's inner class StreamGlobbler is responsible for emptying the I/O streams as described above.

8.5.2 The Record Data Window

The windows for recording measurement data are implemented by the SingleRecordDataWindow class. There is exactly one record data window for each of the form groups, so the record data window is an oligoton is this sense. The record data window is opened upon user request, if the data is not blocked by any other window: Each instance of

the SingleRecordDataWindow checks whether the recording of measurement data of the selected form group is blocked and if it is not, it displays the record data window. The data is released again, if the window is closed.
The layout of the window reflects the n:m relation of measurements to parameter sources and to parameters, meaning that each parameter can be measured many times and that each parameter source can answer many times. Furthermore, the layout of each record data window reflects the layout of the measurement table of the selected form group. It at least allows the selection of a parameter source and a form to record data for these. Depending on the layout of the according measurement table, it can also display a date field, and additional key fields for selecting a measurement set. The user first has to specify all the key values, because they are necessary to uniquely identify a measurement set, thereafter he can start recording the data by pressing the button right to the key fields.
Afterwards, the questions are loaded from the form table of the selected group and the recording process is started. The user can now specify all the non key head values and all the values of the parameters, but is no longer allowed to change the key values, because this would lead to specifying another measurement set. In case the specified measurement set did not exist, it is finally created by selecting the save action.
The user can only indicate that he wants to work with another record, by choosing one of the following actions:

- New selection = close
- Search
- First, previous, next or last record.

Searching is covered in detail in chapter 8.5.5.
Table 8-9 lists the public methods of the SingleRecordDataWindow class.

Method	Description
getInstance(int, TDataAdmin, String)	Returns an instance of the SingleRecordDataWindow class, there can be only one instance per form group (the int argument)
preview(int, TDataAdmin, String)	Previews a record data window. It is used for previewing the edit mode during the design process. If in preview mode, the user cannot enter anything in the displayed fields
print()	Prints the questionnaire
showWindow()	Tries to block the data and if successful, displays the record data window
propertyChange(PropertyChangeEvent)	For noticing property changes of JFormattedTextFields
actionPerformed(ActionEvent)	Implements the actionPerformed method to react to non-menu actions.

Table 8-9 Public Methods of SingleRecordDataWindow. We describe the public methods of the SingleRecordDataWindow

Figure 8-23 depicts the UML diagram of the record data window.

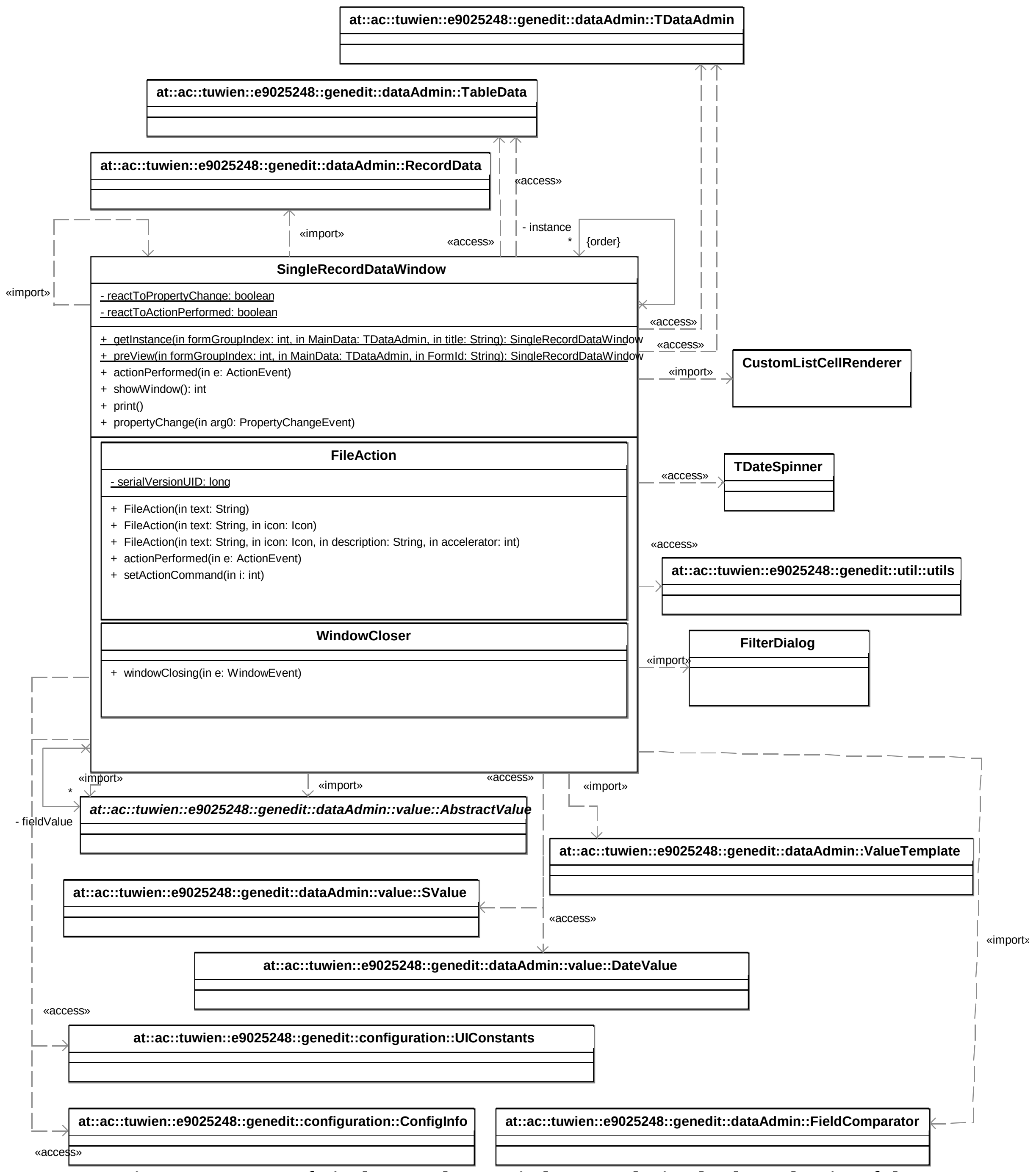

at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
at::ac::tuwien::e9025248::genedit::dataAdmin::TableData
at::ac::tuwien::e9025248::genedit::dataAdmin::RecordData
«access»
«import»
«access»
- instance
*
{order}
«import»
SingleRecordDataWindow
- reactToPropertyChange: boolean
- reactToActionPerformed: boolean
+ getInstance(in formGroupIndex: int, in MainData: TDataAdmin, in title: String): SingleRecordDataWindow
+ preView(in formGroupIndex: int, in MainData: TDataAdmin, in FormId: String): SingleRecordDataWindow
+ actionPerformed(in e: ActionEvent)
+ showWindow(): int
+ print()
+ propertyChange(in arg0: PropertyChangeEvent)
«access»
«access»
«import»
CustomListCellRenderer
FileAction
- serialVersionUID: long
+ FileAction(in text: String)
+ FileAction(in text: String, in icon: Icon)
+ FileAction(in text: String, in icon: Icon, in description: String, in accelerator: int)
+ actionPerformed(in e: ActionEvent)
+ setActionCommand(in i: int)
«access»
TDateSpinner
«access»
at::ac::tuwien::e9025248::genedit::util::utils
WindowCloser
+ windowClosing(in e: WindowEvent)
«import»
FilterDialog
«import»
*
«import»
«access»
«import»
- fieldValue
at::ac::tuwien::e9025248::genedit::dataAdmin::value::AbstractValue
at::ac::tuwien::e9025248::genedit::dataAdmin::ValueTemplate
at::ac::tuwien::e9025248::genedit::dataAdmin::value::SValue
«access»
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::value::DateValue
«access»
at::ac::tuwien::e9025248::genedit::configuration::UIConstants
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo
at::ac::tuwien::e9025248::genedit::dataAdmin::FieldComparator
«access»

Figure 8-23 UML of SingleRecordDataWindow. We depict the dependencies of the SingleRecordDataWindow class.

8.5.2.1 The Date Spinner

For displaying dates a custom date spinner is used. It is set up according to the date formats specified. The default time granularity is the day, this means that the date spinner displays the day, the month and the year and its step is the day. The user can modify the time granularity be specifying additional key head fields. So the date spinner is flexible enough to cover all allowed date formats (but the separators between the date elements are fixed: “/” for dates, “:” for time). Figure 8-24 depicts the UML diagram of the TDateSpinner class.

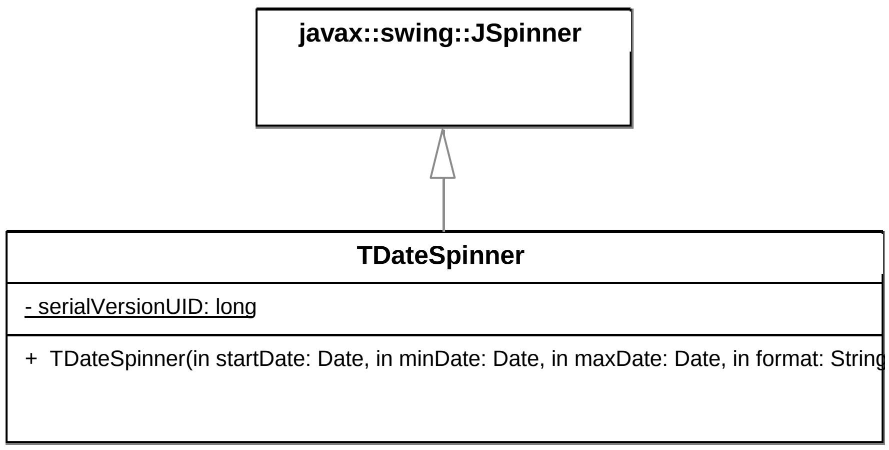

javax::swing::JSpinner
TDateSpinner
- serialVersionUID: long
+ TDateSpinner(in startDate: Date, in minDate: Date, in maxDate: Date, in format: Strin

Figure 8-24 UML of TDateSpinner. We depict the dependencies of the TDateSpinner class.

8.5.2.2 The Custom ListCellRenderer

In order to be able to provide tool tips for each item of a JCombobox a custom ListCellRenderer, which inherits from JLabel and implements the ListCellRenderer interface is implemented in class CustomListCellRenderer. Tool tips per item are used in combo boxes in the record data window, if there is a description to a specific allowed value available. The description is i.e. the meaning of a certain value, e.g., 1 = very good, 2 = good, 3 = average, 4 = bad. The descriptions to allowed values are set up during the design process. Figure 8-25 UML of the CustomListCellRenderer depicts the UML diagram of the CustomListCellRenderer. The implementation of the class used the solution found at [WWW-27].

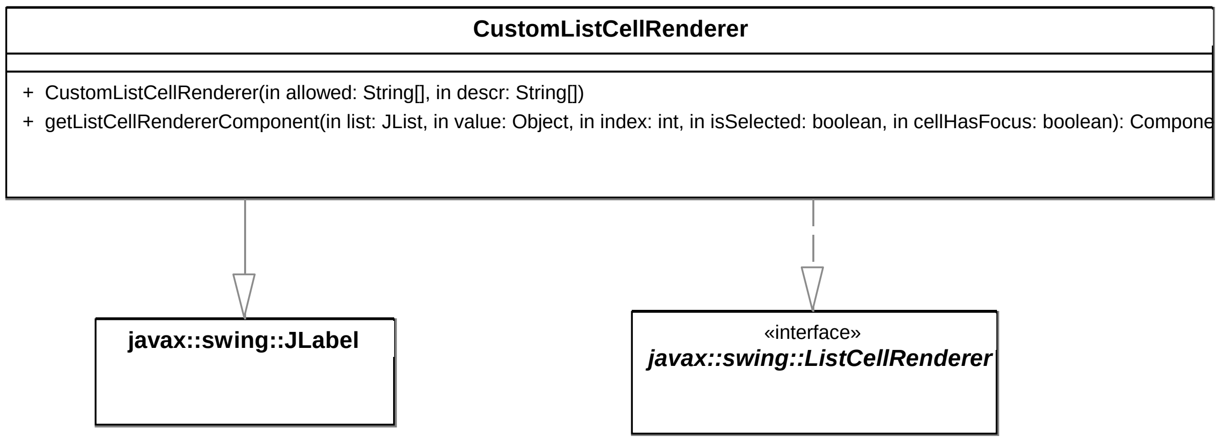

CustomListCellRenderer
+ CustomListCellRenderer(in allowed: String[], in descr: String[])
+ getListCellRendererComponent(in list: JList, in value: Object, in index: int, in isSelected: boolean, in cellHasFocus: boolean): Compone
javax::swing::JLabel
«interface»
javax::swing::ListCellRenderer

Figure 8-25 UML of the CustomListCellRenderer. We depict the dependencies of the CustomListCellRenderer class.

8.5.3 The Design Form Window

The TDesignFormWindow class implements the design process. Again, there can be only one design window for each form group. So the TDesignFormWindow is an oligoton is this sense. The start of the design process works similar to the record data process. First, it is checked whether the data can be blocked, and afterwards the design window is displayed upon success. The data is released again on window closing.

Each form has only one key field, which is the form-id, which is unique within a form group. So it is sufficient to key in a (possibly new) form-id to start the design process. Once the design process has been started, it is possible to add questions, change the questions' properties and add allowed values etc. Similar to the record data process the user cannot

change the form-id as long as he is in the design mode. The question details are displayed in a JTable with a custom TableModel, which is an instance of the class TFormModel. The allowed values are only displayed upon request and in a window of their own. Again a JTable is used to display the allowed values. Its TableModel is implemented by the class TAllowedModel. Both TableModels are implemented as inner classes of the TDesignFormWindow class, because they are only used there.
In total, the design form window reflects the hierarchical structure of the form definitions.
To indicate that he wants to work with another record, the user has the same options as described in the recording data section (compare section 8.5.2):

- New selection = close
- Search
- First, previous, next or last record.

Searching is covered in detail in section 8.5.5.
Table 8-10 lists the public methods of the TDesignFormWindow class along with a description.

Method	Description
getInstance(int, TDataAdmin, String)	Returns an instance of the TDesignFormWindow class. There can be only one instance per form group.
showWindow()	Tries to block the data and displays the design window upon success.

Table 8-10 Public Methods of TDesignFormWindow. We describe the public methods of the TDesignFormWindow.

Figure 8-26 depicts the UML diagram of the TDesignFormWindow. It only shows the dependences within the ui package, because the diagram is too big for displaying all the dependences of the project.

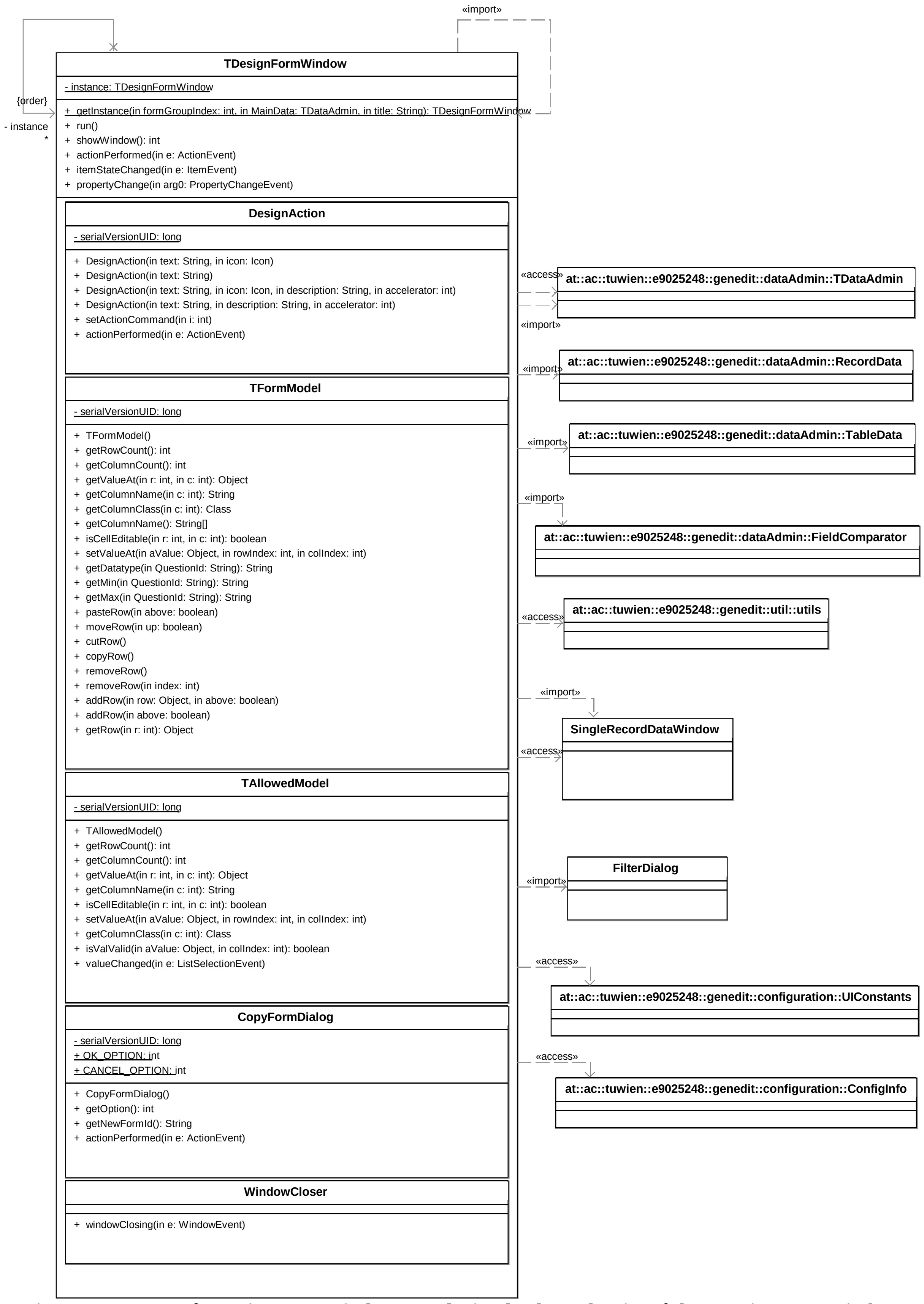

«import»
TDesignFormWindow
- instance: TDesignFormWindow
{order}
- instance
*
+ getInstance(in formGroupIndex: int, in MainData: TDataAdmin, in title: String): TDesignFormWindow
+ run()
+ showWindow(): int
+ actionPerformed(in e: ActionEvent)
+ itemStateChanged(in e: ItemEvent)
+ propertyChange(in arg0: PropertyChangeEvent)
DesignAction
- serialVersionUID: long
+ DesignAction(in text: String, in icon: Icon)
+ DesignAction(in text: String)
+ DesignAction(in text: String, in icon: Icon, in description: String, in accelerator: int)
+ DesignAction(in text: String, in description: String, in accelerator: int)
+ setActionCommand(in i: int)
+ actionPerformed(in e: ActionEvent)
TFormModel
- serialVersionUID: long
+ TFormModel()
+ getRowCount(): int
+ getColumnCount(): int
+ getValueAt(in r: int, in c: int): Object
+ getColumnName(in c: int): String
+ getColumnClass(in c: int): Class
+ getColumnName(): String[]
+ isCellEditable(in r: int, in c: int): boolean
+ setValueAt(in aValue: Object, in rowIndex: int, in colIndex: int)
+ getDatatype(in QuestionId: String): String
+ getMin(in QuestionId: String): String
+ getMax(in QuestionId: String): String
+ pasteRow(in above: boolean)
+ moveRow(in up: boolean)
+ cutRow()
+ copyRow()
+ removeRow()
+ removeRow(in index: int)
+ addRow(in row: Object, in above: boolean)
+ addRow(in above: boolean)
+ getRow(in r: int): Object
TAllowedModel
- serialVersionUID: long
+ TAllowedModel()
+ getRowCount(): int
+ getColumnCount(): int
+ getValueAt(in r: int, in c: int): Object
+ getColumnName(in c: int): String
+ isCellEditable(in r: int, in c: int): boolean
+ setValueAt(in aValue: Object, in rowIndex: int, in colIndex: int)
+ getColumnClass(in c: int): Class
+ isValValid(in aValue: Object, in colIndex: int): boolean
+ valueChanged(in e: ListSelectionEvent)
CopyFormDialog
- serialVersionUID: long
+ OK_OPTION: int
+ CANCEL_OPTION: int
+ CopyFormDialog()
+ getOption(): int
+ getNewFormId(): String
+ actionPerformed(in e: ActionEvent)
WindowCloser
+ windowClosing(in e: WindowEvent)
«access»
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::RecordData
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::TableData
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::FieldComparator
«access»
at::ac::tuwien::e9025248::genedit::util::utils
«import»
SingleRecordDataWindow
«access»
«import»
FilterDialog
«access»
at::ac::tuwien::e9025248::genedit::configuration::UIConstants
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo

Figure 8-26 UML of TDesignFormWindow. We depict the dependencies of the TDesignFormWindow class.

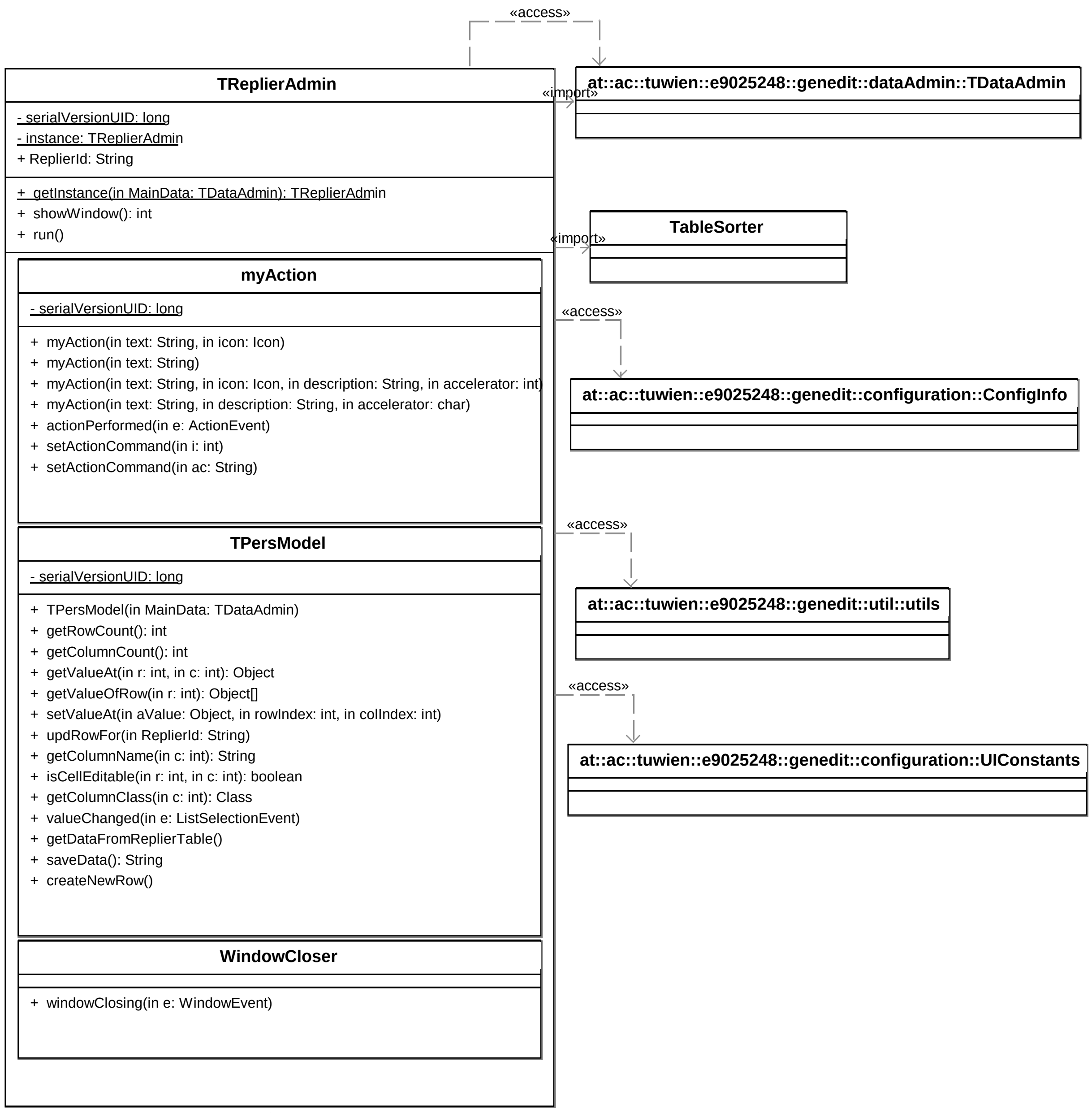
8.5.4 The Parameter Source Maintenance Window

The parameter source maintenance process is implemented by the TReplierAdmin class. It is simple compared to TDesignFormWindow or SingleRecordDataWindow, because the parameter sources do not have a hierarchical structure. So, in contrary to the record data window and the design form window, all the available parameter sources are loaded in a table, which displays all the available parameter sources together with the number of answers already given. The table is implemented as sortable JTable with a custom TableModel, which is implemented in the inner class TPersModel. The class TableSorter enhances JTable and allows sorting of the table contents. For information on the Table Sorter see section 8.5.8.
Table 8-11 lists the public methods of the TReplierAdmin class:

Method	Description
getInstance(int, TDataAdmin,String)	Returns an instance of the TReplierAdmin class. There can be only one instance.
showWindow()	Tries to block the data and displays the parameter source maintenance window upon success.

Table 8-11 Public Methods of TReplierAdmin. We describe the public methods of the TReplierAdmin class.

Figure 8-27 depicts the UML diagram of the TReplierAdmin class.


«access»
TReplierAdmin
- serialVersionUID: long
- instance: TReplierAdmin
+ ReplierId: String
+ getInstance(in MainData: TDataAdmin): TReplierAdmin
+ showWindow(): int
+ run()
myAction
- serialVersionUID: long
+ myAction(in text: String, in icon: Icon)
+ myAction(in text: String)
+ myAction(in text: String, in icon: Icon, in description: String, in accelerator: int)
+ myAction(in text: String, in description: String, in accelerator: char)
+ actionPerformed(in e: ActionEvent)
+ setActionCommand(in i: int)
+ setActionCommand(in ac: String)
TPersModel
- serialVersionUID: long
+ TPersModel(in MainData: TDataAdmin)
+ getRowCount(): int
+ getColumnCount(): int
+ getValueAt(in r: int, in c: int): Object
+ getValueOfRow(in r: int): Object[]
+ setValueAt(in aValue: Object, in rowIndex: int, in colIndex: int)
+ updRowFor(in ReplierId: String)
+ getColumnName(in c: int): String
+ isCellEditable(in r: int, in c: int): boolean
+ getColumnClass(in c: int): Class
+ valueChanged(in e: ListSelectionEvent)
+ getDataFromReplierTable()
+ saveData(): String
+ createNewRow()
WindowCloser
+ windowClosing(in e: WindowEvent)
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
«import»
TableSorter
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo
«access»
at::ac::tuwien::e9025248::genedit::util::utils
«access»
at::ac::tuwien::e9025248::genedit::configuration::UIConstants
Figure 8-27 UML of TReplierAdmin. We depict the dependencies of the TReplierAdmin class.

8.5.5 The Filter / Search Dialog

All filter and search dialogs are implemented solely by the FilterDialog class. Its constructor takes an option argument for selecting whether the purpose of the instance is to search, and select a single record, or to filter and provide a list of indices of the records which conform to the filter criteria. Another argument is used for specifying the purpose of the table. The current release defines and supports:

- FilterDialog.ANSWERTABLE
- FilterDialog.FORMTABLE and
- FilterDialog.PARAMETERSOURCETABLE.

Of course it also needs a link to the data source and to know the form group, if filtering or searching is set up for a form table or an answer table. The filter dialog is displayed upon user request from the design form window, from the record data window and from the export dialogs. Obviously, the purpose of the filter dialog is searching for records and selecting a single record, if triggered from the design form window or the record data window. Whilst the

purpose is filtering the data in order to export only a subset of the data, if started from one of the export dialogs.
The filter window displays two JTables. One of them displays all the records which conform to the filter criteria; the other one displays the filter criteria.
It supports two filter views, an advanced filter and a simple filter. Both have custom table models, which are implemented by inner classes, which inherit from another inner class, the AbstractFilterModel. The AbstractFilterModel ensures that both concrete filter models implement some functionality, which is used during the filtering process. So it is transparent during the actual filtering process, whether a simple or an advanced filter model is used.
The filtering is started on the fly as soon as there are changes to the filter criteria. This is done by listing to TableModelEvents.
Additionally, the filter dialog allows assigning values to the records' additional head fields, which do not belong to the key fields without the need of opening a record explicitly.

Figure 8-28 depicts the UML diagram of the FilterDialog class. The figure misses out the public static variables and method for the inner classes WindowCloser and myAction, because of the available space. However, these have the same tasks than in the other classes, which implement ui windows.

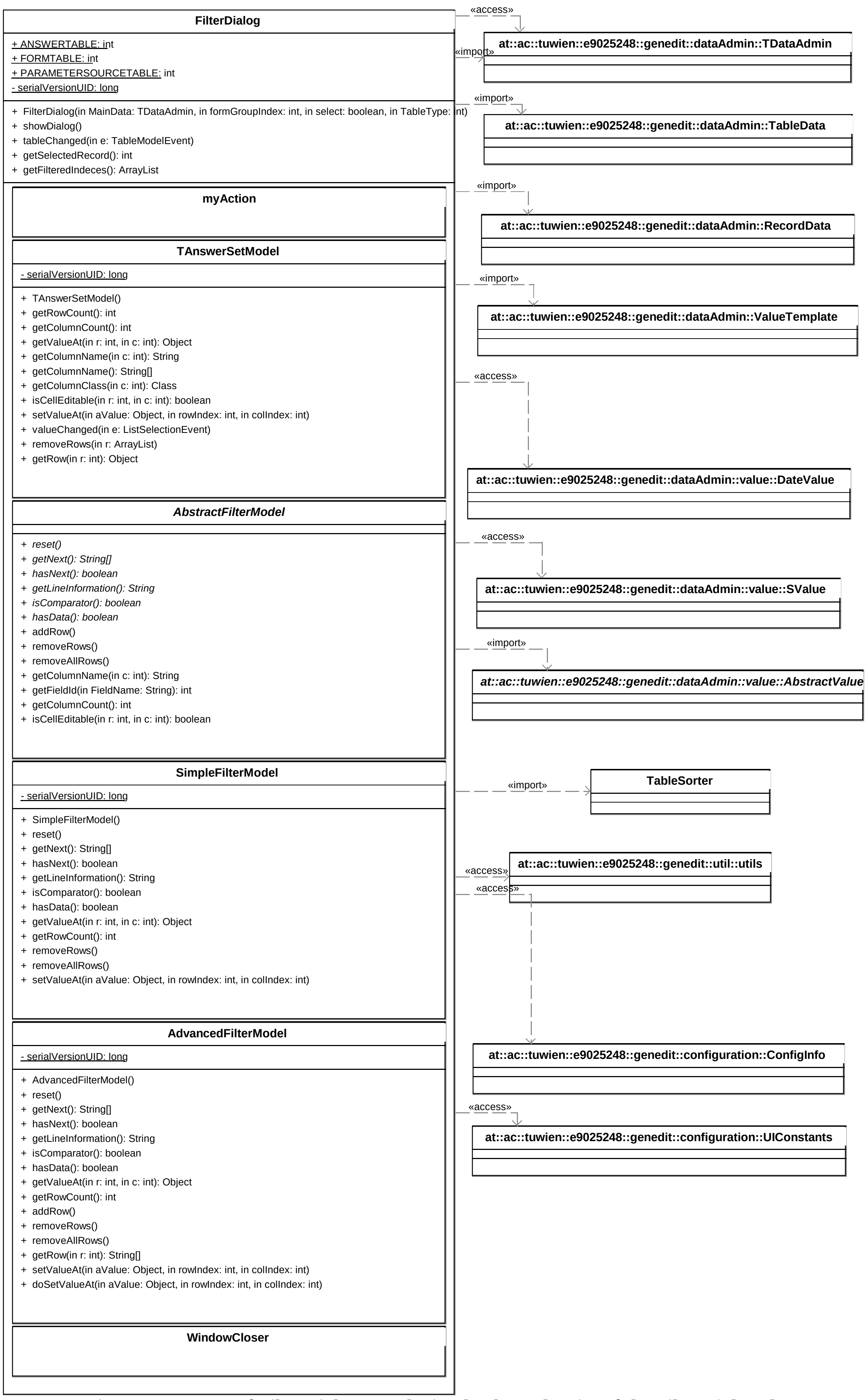

FilterDialog
+ ANSWERTABLE: int
+ FORMTABLE: int
+ PARAMETERSOURCETABLE: int
- serialVersionUID: long
+ FilterDialog(in MainData: TDataAdmin, in formGroupIndex: int, in select: boolean, in TableType: int)
+ showDialog()
+ tableChanged(in e: TableModelEvent)
+ getSelectedRecord(): int
+ getFilteredIndeces(): ArrayList
myAction
TAnswerSetModel
- serialVersionUID: long
+ TAnswerSetModel()
+ getRowCount(): int
+ getColumnCount(): int
+ getValueAt(in r: int, in c: int): Object
+ getColumnName(in c: int): String
+ getColumnName(): String[]
+ getColumnClass(in c: int): Class
+ isCellEditable(in r: int, in c: int): boolean
+ setValueAt(in aValue: Object, in rowIndex: int, in colIndex: int)
+ valueChanged(in e: ListSelectionEvent)
+ removeRows(in r: ArrayList)
+ getRow(in r: int): Object
AbstractFilterModel
+ reset()
+ getNext(): String[]
+ hasNext(): boolean
+ getLineInformation(): String
+ isComparator(): boolean
+ hasData(): boolean
+ addRow()
+ removeRows()
+ removeAllRows()
+ getColumnName(in c: int): String
+ getFieldId(in FieldName: String): int
+ getColumnCount(): int
+ isCellEditable(in r: int, in c: int): boolean
SimpleFilterModel
- serialVersionUID: long
+ SimpleFilterModel()
+ reset()
+ getNext(): String[]
+ hasNext(): boolean
+ getLineInformation(): String
+ isComparator(): boolean
+ hasData(): boolean
+ getValueAt(in r: int, in c: int): Object
+ getRowCount(): int
+ removeRows()
+ removeAllRows()
+ setValueAt(in aValue: Object, in rowIndex: int, in colIndex: int)
AdvancedFilterModel
- serialVersionUID: long
+ AdvancedFilterModel()
+ reset()
+ getNext(): String[]
+ hasNext(): boolean
+ getLineInformation(): String
+ isComparator(): boolean
+ hasData(): boolean
+ getValueAt(in r: int, in c: int): Object
+ getRowCount(): int
+ addRow()
+ removeRows()
+ removeAllRows()
+ getRow(in r: int): String[]
+ setValueAt(in aValue: Object, in rowIndex: int, in colIndex: int)
+ doSetValueAt(in aValue: Object, in rowIndex: int, in colIndex: int)
WindowCloser
«access»
at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
«import»
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::TableData
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::RecordData
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::ValueTemplate
«access»
at::ac::tuwien::e9025248::genedit::dataAdmin::value::DateValue
«access»
at::ac::tuwien::e9025248::genedit::dataAdmin::value::SValue
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::value::AbstractValue
«import»
TableSorter
«access»
at::ac::tuwien::e9025248::genedit::util::utils
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo
«access»
at::ac::tuwien::e9025248::genedit::configuration::UIConstants

Figure 8-28 UML of FilterDialog. We depict the dependencies of the FilterDialog class.

8.5.6 Export Dialog

There are two dialogs, which lead to an export of measurement data after the user clicked the approve button. Both of them behave similar, so the common methods and variables have been bundled together in another class called CommonExportOptions. The export dialogs only display the dialog and provide the host object (in our case the main window) with the selected export options. All the work of exporting (and program starting) is triggered by the main window upon approve. Therefore, the export dialogs lock the data. However the data is released by the main window after finishing the export.

8.5.6.1 Common Export Options

The class CommonExportOption provides functionality which is common for the standard export dialog (see section 8.5.6.2), and for the dialog which starts an external program after the export (see section 8.5.6.3).
Table 8-12 lists the public methods of the CommonExportOption class. They merely call the methods with the same name in their instance of the ExportChooser class:

Method	**Description**
doFilter()	Whether the user ticked the filter option
doScale()	Whether the user ticked the scale option
getConflateOver()	Returns the indices of the fields to conflate over
getFilteredIndeces()	Returns the indices of the records which conform to the filter criteria.
getMax()	The desired maximum for scaling
getMin()	The desired minimum for scaling

Table 8-12 Public Methods of the CommonExportOption. We describe the public methods common to all export classes.

Figure 8-29 depicts the UML diagram of the CommonExportOption class.

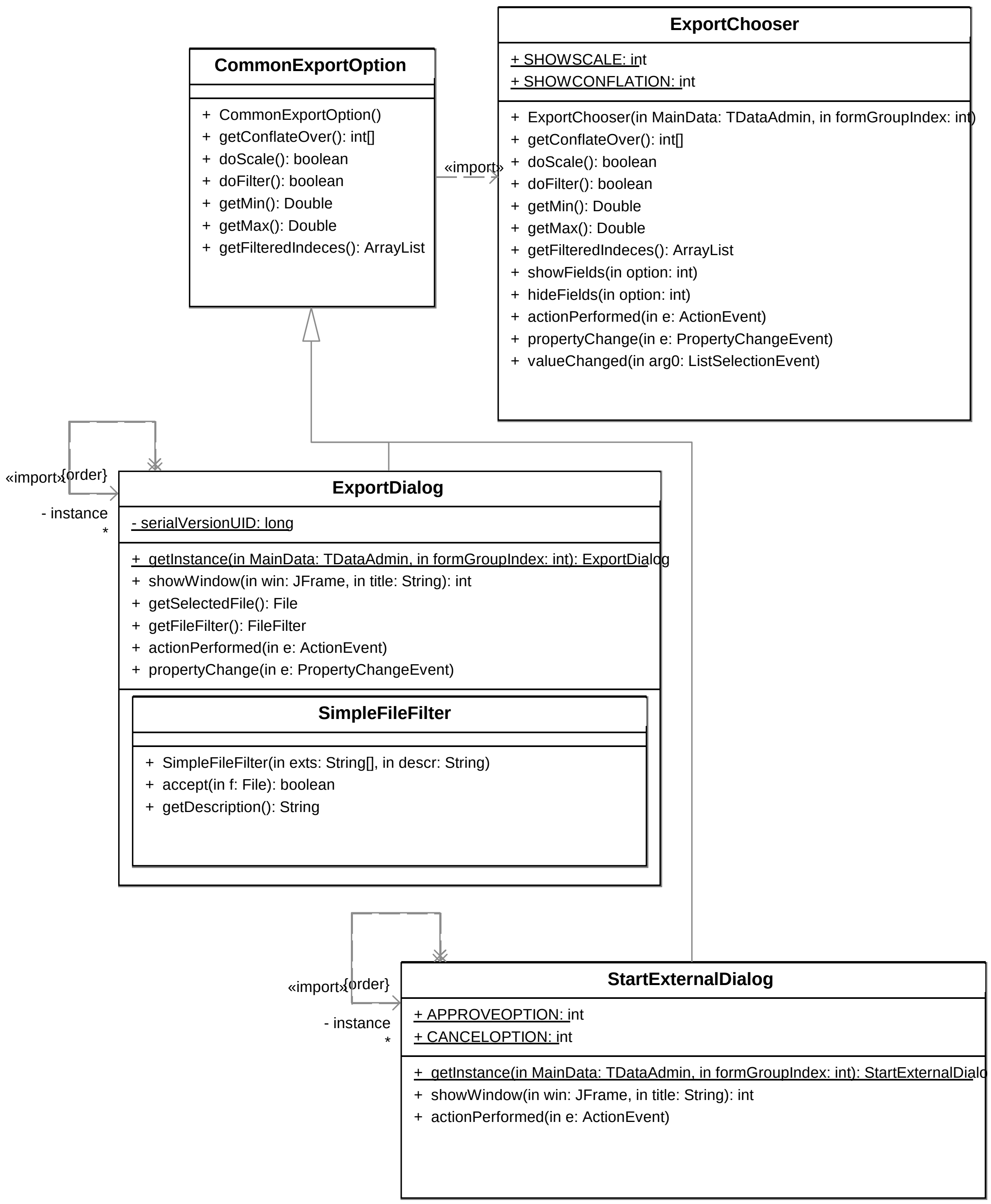
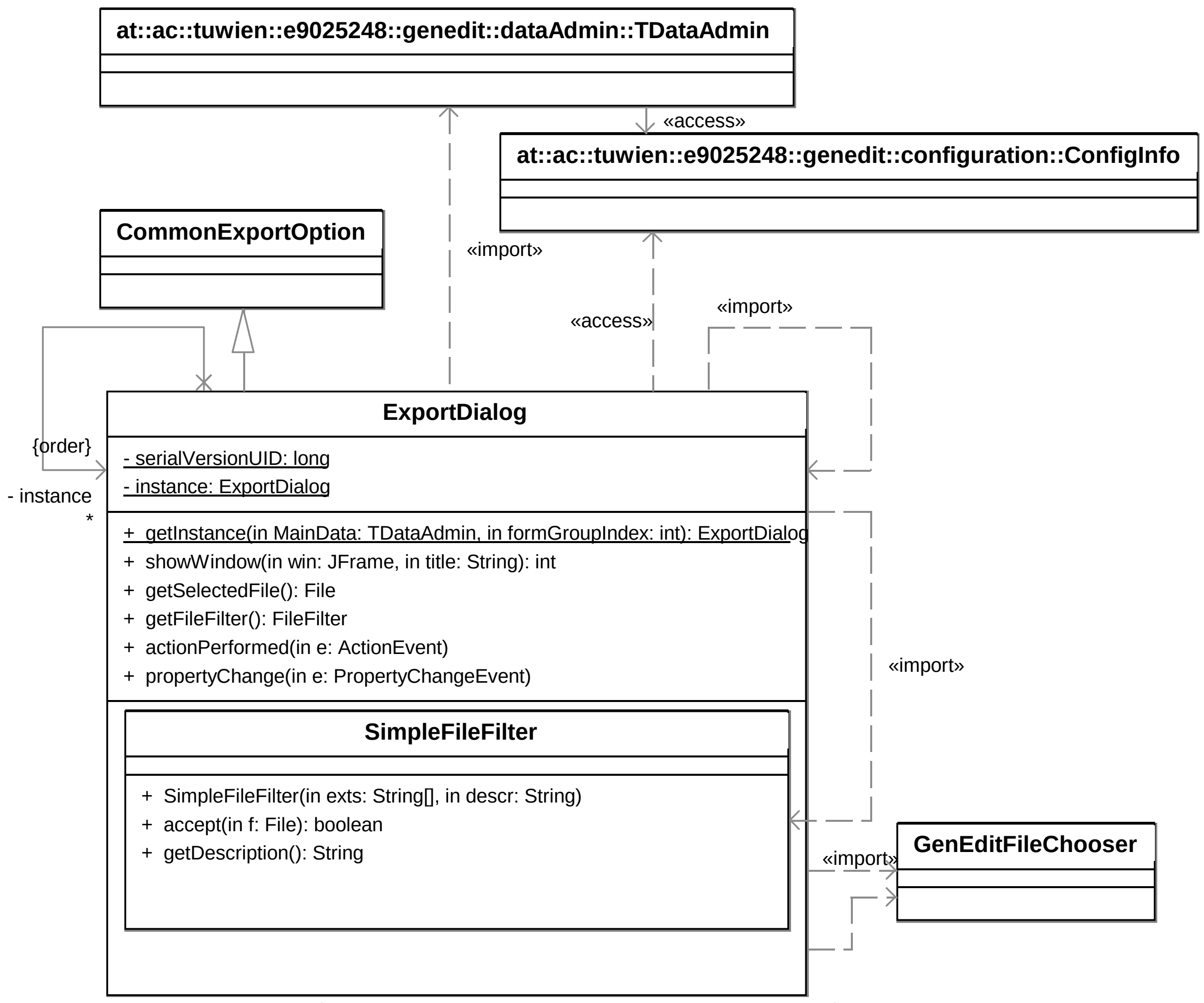

CommonExportOption
+ CommonExportOption()
+ getConflateOver(): int[]
+ doScale(): boolean
+ doFilter(): boolean
+ getMin(): Double
+ getMax(): Double
+ getFilteredIndeces(): ArrayList
«import»
ExportChooser
+ SHOWSCALE: int
+ SHOWCONFLATION: int
+ ExportChooser(in MainData: TDataAdmin, in formGroupIndex: int)
+ getConflateOver(): int[]
+ doScale(): boolean
+ doFilter(): boolean
+ getMin(): Double
+ getMax(): Double
+ getFilteredIndeces(): ArrayList
+ showFields(in option: int)
+ hideFields(in option: int)
+ actionPerformed(in e: ActionEvent)
+ propertyChange(in e: PropertyChangeEvent)
+ valueChanged(in arg0: ListSelectionEvent)
«import»
{order}
- instance
*
ExportDialog
- serialVersionUID: long
+ getInstance(in MainData: TDataAdmin, in formGroupIndex: int): ExportDialog
+ showWindow(in win: JFrame, in title: String): int
+ getSelectedFile(): File
+ getFileFilter(): FileFilter
+ actionPerformed(in e: ActionEvent)
+ propertyChange(in e: PropertyChangeEvent)
SimpleFileFilter
+ SimpleFileFilter(in exts: String[], in descr: String)
+ accept(in f: File): boolean
+ getDescription(): String
«import»
{order}
- instance
*
StartExternalDialog
+ APPROVEOPTION: int
+ CANCELOPTION: int
+ getInstance(in MainData: TDataAdmin, in formGroupIndex: int): StartExternalDialo
+ showWindow(in win: JFrame, in title: String): int
+ actionPerformed(in e: ActionEvent)

Figure 8-29 UML of CommonExportOption. We depict the dependencies of the CommonExportOption class.

8.5.6.2 Export Dialog

The ExportDialog inherits from CommonExportOptions. A JDialog with a GenEditFileChooser is displayed. The file chooser allows selecting the format and the file to export to. If the format is changed the file chooser's accessory needs to be updated accordingly. This is done by calling hideFields(int) or showFields(int) of the instance of the ExportChooser class, which is used as the file chooser's accessory. Further to the inherited public methods, it only provides methods for instantiating and displaying the window. Figure 8-30 depicts the UML diagram of the ExportDialog class.

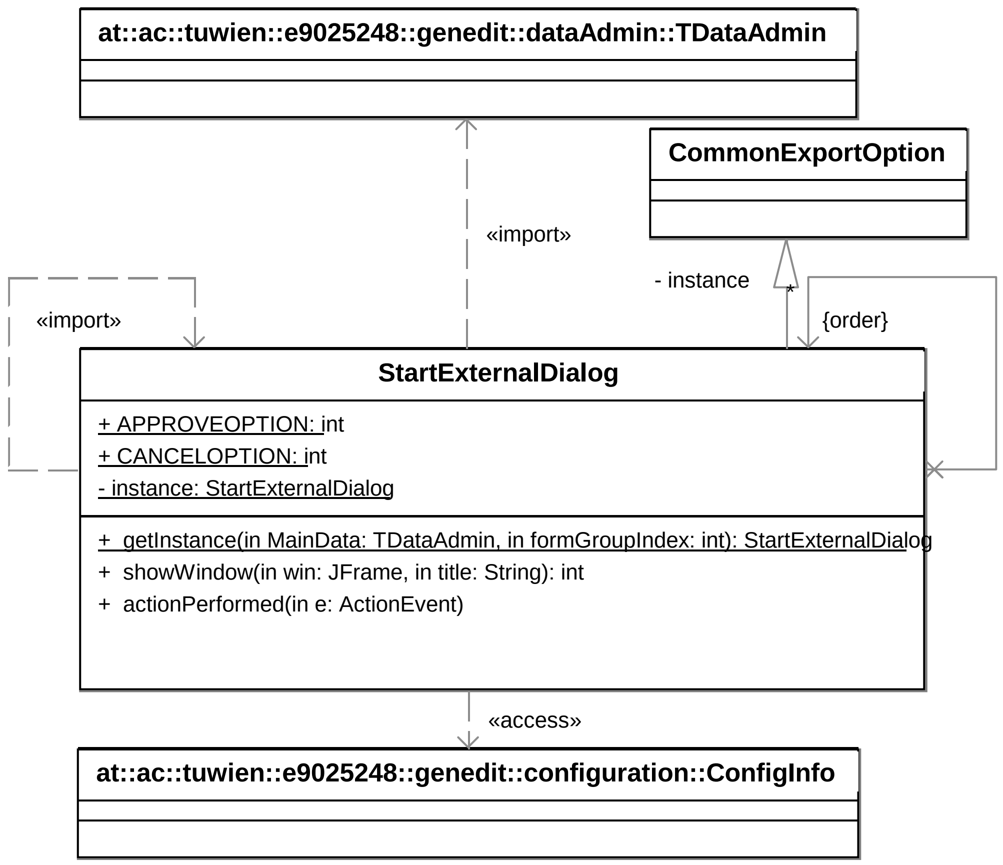


at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo
CommonExportOption
«import»
«access»
«import»
{order}
- instance
*
ExportDialog
- serialVersionUID: long
- instance: ExportDialog
+ getInstance(in MainData: TDataAdmin, in formGroupIndex: int): ExportDialog
+ showWindow(in win: JFrame, in title: String): int
+ getSelectedFile(): File
+ getFileFilter(): FileFilter
+ actionPerformed(in e: ActionEvent)
+ propertyChange(in e: PropertyChangeEvent)
«import»
SimpleFileFilter
+ SimpleFileFilter(in exts: String[], in descr: String)
+ accept(in f: File): boolean
+ getDescription(): String
«import»
GenEditFileChooser

Figure 8-30 UML of ExportDialog. We depict the dependencies of the ExportDialog class.

8.5.6.3 Dialog for Exporting and Starting an External Program

The class StartExternalDialog implements a dialog for exporting and starting an external program. It works similar the ExportDialog, but it does not require a file chooser. Instead the export file and the file format were defined in the profile. It has a button for starting the external program additionally to an Export button. Everything else works the same as in the ExportDialog, because it too uses an instance of the ExportChooser class for selecting the other export options (everything else besides file name and file format). Further to the inherited public methods it only provides methods for instantiating and displaying the window. Figure 8-31 depicts the UML diagram of the StartExternalDialog class.



at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
CommonExportOption
«import»
- instance
*
{order}
«import»
StartExternalDialog
+ APPROVEOPTION: int
+ CANCELOPTION: int
- instance: StartExternalDialog
+ getInstance(in MainData: TDataAdmin, in formGroupIndex: int): StartExternalDialog
+ showWindow(in win: JFrame, in title: String): int
+ actionPerformed(in e: ActionEvent)
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo

Figure 8-31 UML of StartExternalDialog. We depict the dependencies of the StartExternalDialog class.

8.5.6.4 Export Chooser

The class ExportChooser implements the accessories used in both of the export dialogs, the standard export dialog (see section 8.5.6.2) and for the dialog which starts an external program after the export (see section 8.5.6.3). It provides data format specific options. Depending on the chosen file format it displays options for scaling (not for AnswerFile format) or conflating values (conflation file format only). Its public methods are used to return the selected options and values. Table 8-13 lists the public methods of the ExportChooser:

Method	**Description**
doFilter()	Whether the user ticked the filter option
doScale()	Whether the user ticked the scale option
getConflateOver()	Returns the indices of the fields to conflate over
getFilteredIndeces()	Returns the indices of the records which conform to the filter criteria.
getMax()	The desired maximum for scaling
getMin()	The desired minimum for scaling
hideFields()	Hides the fields for options which are not available for certain file formats.
showFields()	Displays the fields for options which are needed for certain file formats.

Table 8-13 Public Methods of the ExportChooser. We describe the public methods used to choose the various export options.

Figure 8-32 depicts the UML diagram of the ExportChooser class.

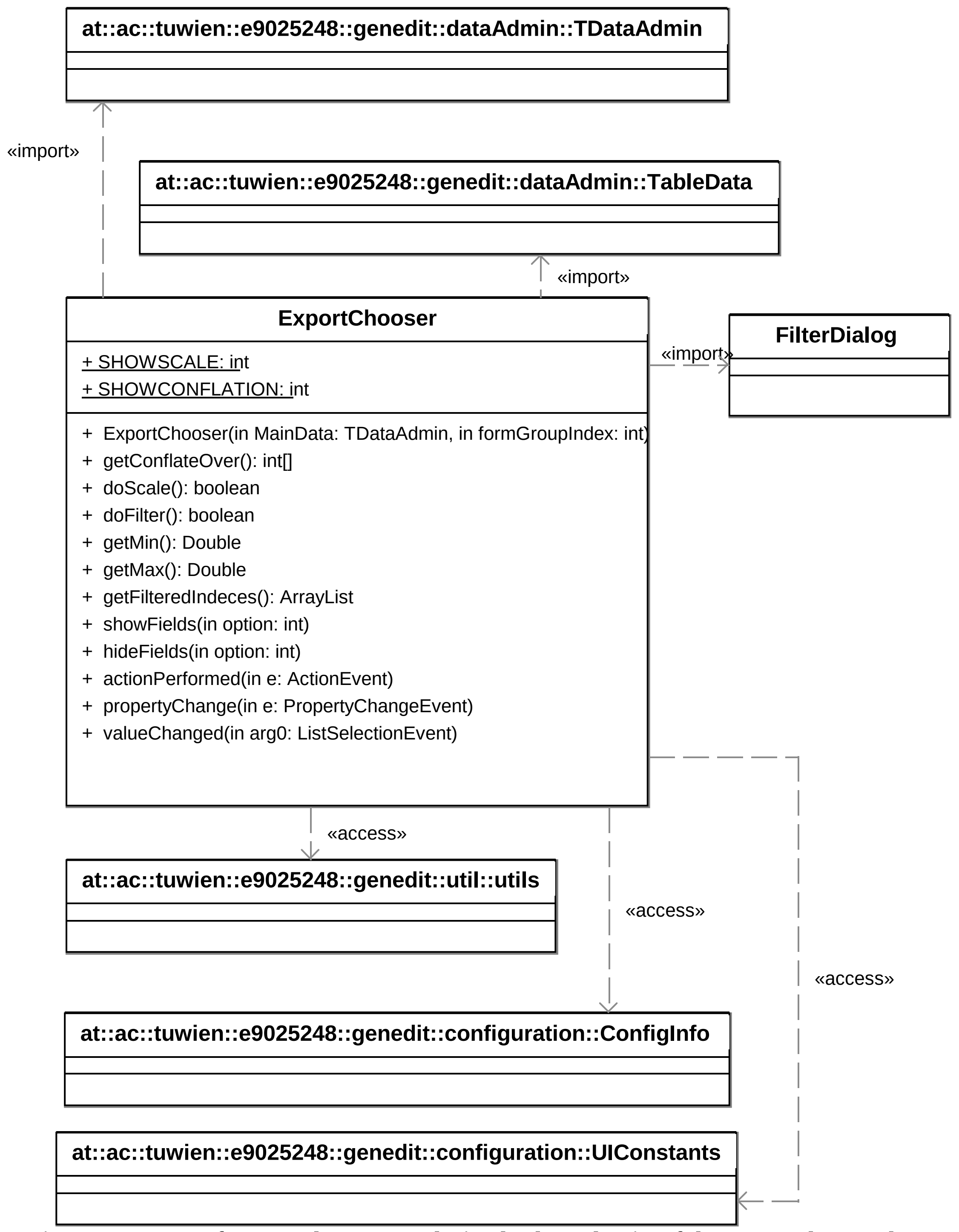

at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::TableData
«import»
ExportChooser
+ SHOWSCALE: int
+ SHOWCONFLATION: int
+ ExportChooser(in MainData: TDataAdmin, in formGroupIndex: int)
+ getConflateOver(): int[]
+ doScale(): boolean
+ doFilter(): boolean
+ getMin(): Double
+ getMax(): Double
+ getFilteredIndeces(): ArrayList
+ showFields(in option: int)
+ hideFields(in option: int)
+ actionPerformed(in e: ActionEvent)
+ propertyChange(in e: PropertyChangeEvent)
+ valueChanged(in arg0: ListSelectionEvent)
FilterDialog
«import»
«access»
at::ac::tuwien::e9025248::genedit::util::utils
«access»
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo
at::ac::tuwien::e9025248::genedit::configuration::UIConstants

Figure 8-32 UML of ExportChooser. We depict the dependencies of the ExportChooser class.

8.5.6.5 The Modified File Chooser

Wherever a file chooser is needed, a modified file chooser is used. It is implemented by the class GenEditFileChooser. This version of JFileChooser simply makes createDialog a public method. The JFileChooser behaves like this: When the user clicks on the approve button (Open or Save) or the Cancel button, an action event is fired. Unfortunately, if the user clicks on the close-dialog icon in the title bar, an event is not fired. In order to listen for this event, it is necessary to add a window event listener to the dialog. This means that JFileChooser.showDialog() cannot be used because it creates the dialog internally. The workaround is to override the createDialog() method to make it public and then call it to create the dialog. The idea to modify the behavior of the JFileChooser like this can be found in [WWW-8]. Figure 8-33 depicts the UML diagram of the GenEditFileChooser class.

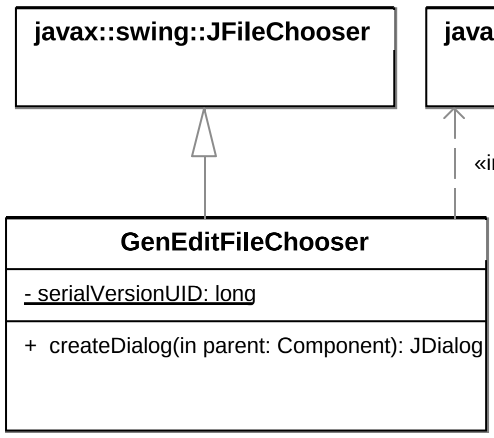

javax::swing::JFileChooser
javax::swing::JDialog
«import»
GenEditFileChooser
- serialVersionUID: long
+ createDialog(in parent: Component): JDialog

Figure 8-33 UML of GenEditFileChooser. We depict the dependencies of the GenEditFileChooser class.

8.5.7 Import Dialog

The import dialog only displays the dialog and provides the host object (in our case the main window) with the selected export options. All the work of importing is triggered by the main window upon approve. Therefore, the import dialog locks the data, but it is released by the main window after finishing the import. The ImportDialog uses the GenEditFileChooser (see section 8.5.6.5) for selecting the file to import. Figure 8-34 depicts the UML diagram of the ImportDialog class.

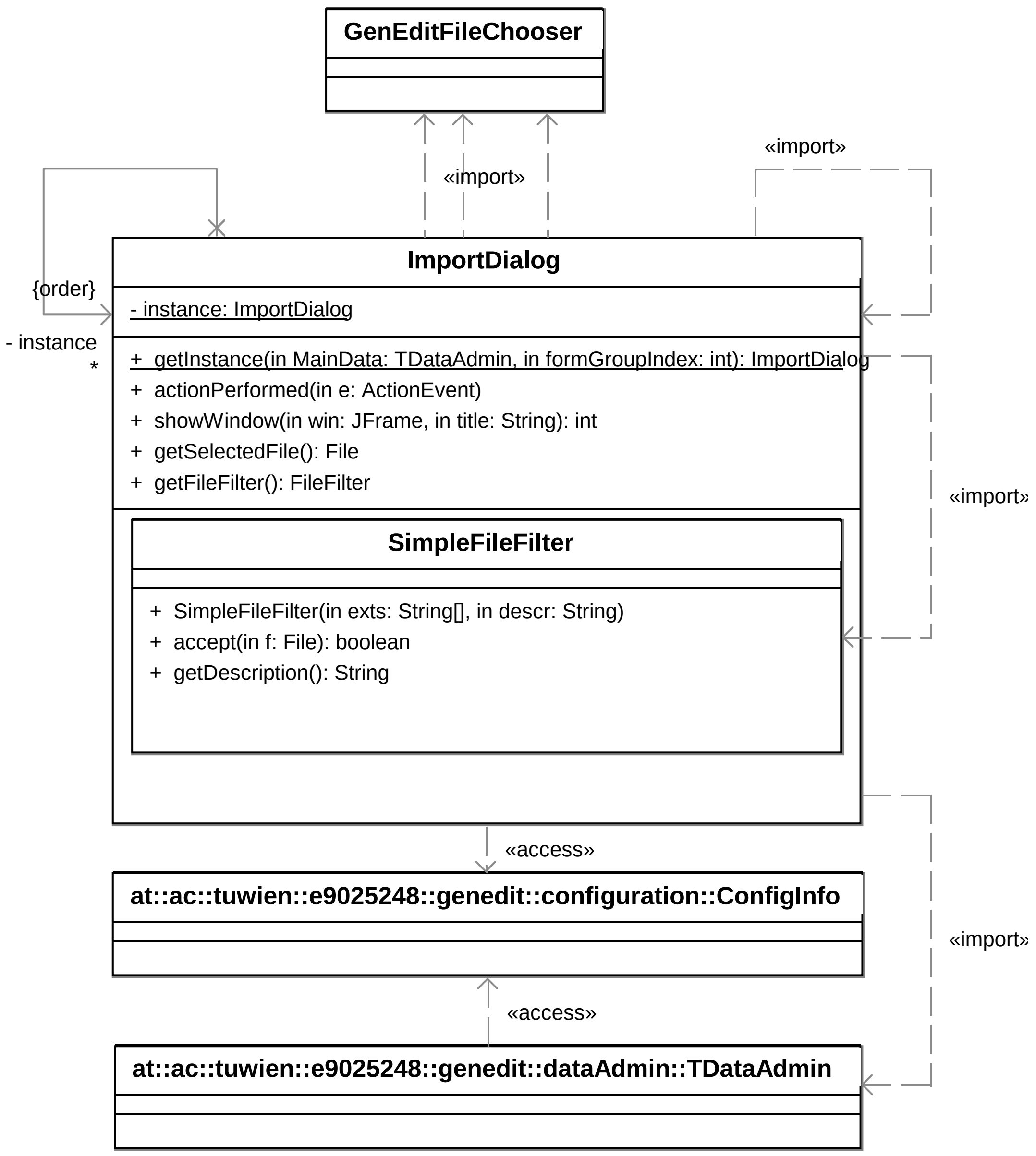

GenEditFileChooser
«import»
«import»
{order}
- instance
*
ImportDialog
- instance: ImportDialog
+ getInstance(in MainData: TDataAdmin, in formGroupIndex: int): ImportDialog
+ actionPerformed(in e: ActionEvent)
+ showWindow(in win: JFrame, in title: String): int
+ getSelectedFile(): File
+ getFileFilter(): FileFilter
«import»
SimpleFileFilter
+ SimpleFileFilter(in exts: String[], in descr: String)
+ accept(in f: File): boolean
+ getDescription(): String
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo
«import»
«access»
at::ac::tuwien::e9025248::genedit::dataAdmin::TDataAdmin

Figure 8-34 UML of ImportDialog. We depict the dependencies of the ImportDialog class.

8.5.8 Table Sorter

We downloaded the TableSorter class from [WWW-7] at 3rd of May 2006. It was implemented by Philip Milne, Brendon McLean, Dan van Enckevort and Parwinder Sekhon. The release used in this application is: 2.0 02/27/04.

The following description of the TableSorter class is from the documentation of the downloaded class: “TableSorter is a decorator for TableModels; adding sorting functionality to a supplied TableModel. TableSorter does not store or copy the data in its TableModel; instead it maintains a map from the row indexes of the view to the row indexes of the model. As requests are made of the sorter (like getValueAt(row, col)) they are passed to the underlying model after the row numbers have been translated via the internal mapping array. This way, the TableSorter appears to hold another copy of the table with the rows in a different order.

TableSorter registers itself as a listener to the underlying model, just as the JTable itself would. Events received from the model are examined, sometimes manipulated (typically widened), and then passed on to the TableSorter's listeners (typically the JTable). If a change to the model has invalidated the order of TableSorter's rows, a note of this is made and the sorter will resort the rows the next time a value is requested.

- When the tableHeader property is set, either by using the setTableHeader() method or the two argument constructor, the table header may be used as a complete UI for TableSorter. The default renderer of the tableHeader is decorated with a renderer that indicates the sorting status of each column. In addition, a mouse listener is installed with the following behavior:
- Mouse-click: Clears the sorting status of all other columns and advances the sorting status of that column through three values: {NOT_SORTED, ASCENDING, DESCENDING} (then back to NOT_SORTED again).
- SHIFT-mouse-click: Clears the sorting status of all other columns and cycles the sorting status of the column through the same three values, in the opposite order: {NOT_SORTED, DESCENDING, ASCENDING}.
- CONTROL-mouse-click and CONTROL-SHIFT-mouse-click: as above except that the changes to the column do not cancel the statuses of columns that are already sorting - giving a way to initiate a compound sort.”

8.6 The Custom Event of the Generic Editor

There is one custom event implemented. In the current release the instance of the TMainWindow is the only object listening to it. The technique implemented is described in detail in [WWW-9]. This article describes in detail when and how to make a Java class observable. Basically, the observer pattern (see [Gamma 1995]) is implemented.

Figure 8-35 depicts the UML diagram of the custom events.

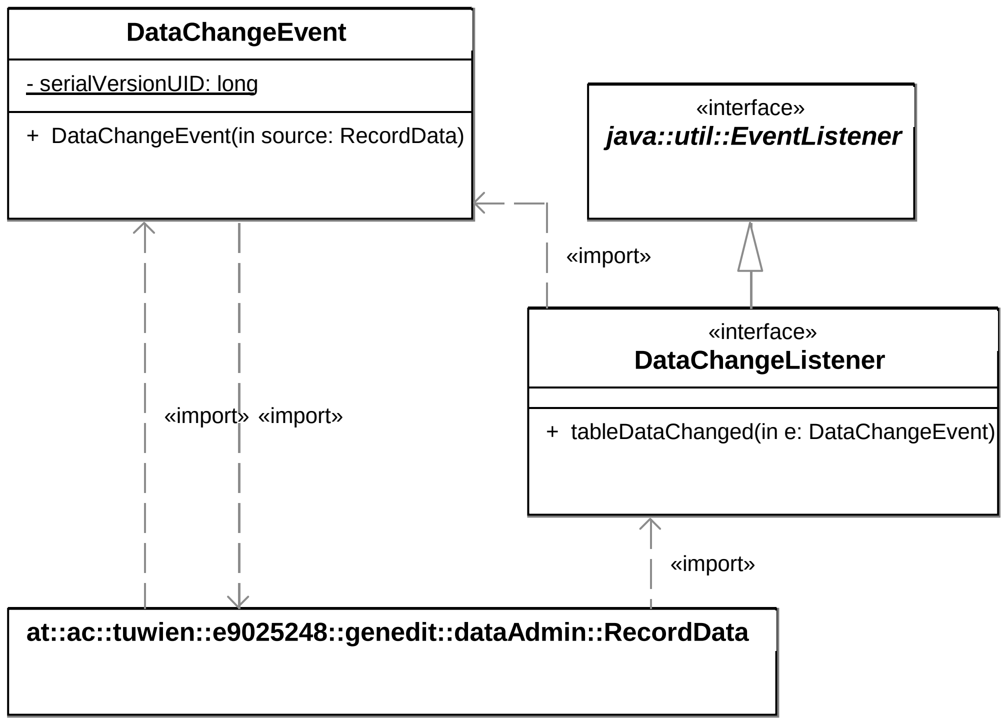

DataChangeEvent
- serialVersionUID: long
+ DataChangeEvent(in source: RecordData)
«interface»
java::util::EventListener
«import»
«interface»
DataChangeListener
+ tableDataChanged(in e: DataChangeEvent)
«import» «import»
«import»
at::ac::tuwien::e9025248::genedit::dataAdmin::RecordData

Figure 8-35 UML of the Custom Event. We depict the dependencies of the classes implementing custom event which indicates data changes.

8.6.1 Data Change Event

The DataChangeEvent is fired whenever there are changes to a record or table. The DataChangeEvent class inherits from java.util.EventObject.

8.6.2 Data Change Listener

The Data Change Listener is an interface which inherits from java.util.EventListener

8.7 Utilities

The class utils provides the application with a couple of static utility methods, which can be used anywhere in the program. Figure 8-36 depicts the UML diagram of this class.

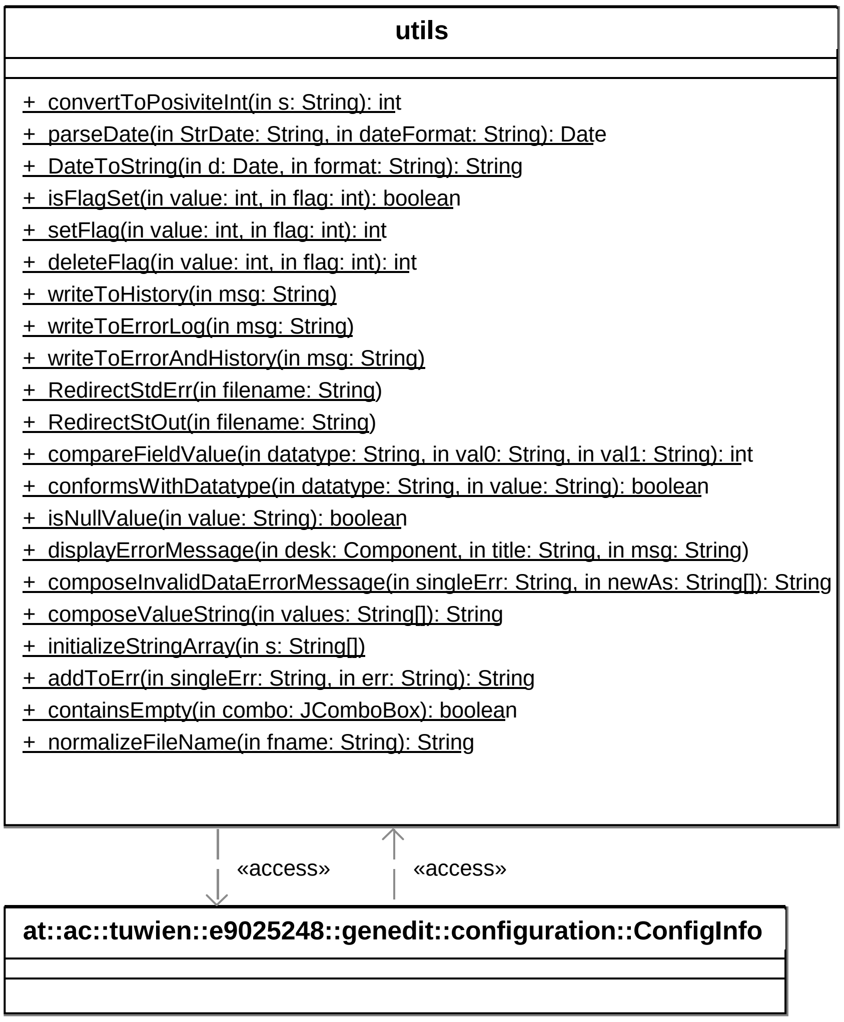

utils
+ convertToPosiviteInt(in s: String): int
+ parseDate(in StrDate: String, in dateFormat: String): Date
+ DateToString(in d: Date, in format: String): String
+ isFlagSet(in value: int, in flag: int): boolean
+ setFlag(in value: int, in flag: int): int
+ deleteFlag(in value: int, in flag: int): int
+ writeToHistory(in msg: String)
+ writeToErrorLog(in msg: String)
+ writeToErrorAndHistory(in msg: String)
+ RedirectStdErr(in filename: String)
+ RedirectStOut(in filename: String)
+ compareFieldValue(in datatype: String, in val0: String, in val1: String): int
+ conformsWithDatatype(in datatype: String, in value: String): boolean
+ isNullValue(in value: String): boolean
+ displayErrorMessage(in desk: Component, in title: String, in msg: String)
+ composeInvalidDataErrorMessage(in singleErr: String, in newAs: String[]): String
+ composeValueString(in values: String[]): String
+ initializeStringArray(in s: String[])
+ addToErr(in singleErr: String, in err: String): String
+ containsEmpty(in combo: JComboBox): boolean
+ normalizeFileName(in fname: String): String
«access»
«access»
at::ac::tuwien::e9025248::genedit::configuration::ConfigInfo

Figure 8-36 UML of utils. We depict the dependencies of the utils class.

III. Demonstration of the tool and Conclusion

9 Illustration for a clinical trial about anorectic girls

Although the editor is designed to be able to work with all kind of questionnaires, its main field of operation will be for clinical data.

9.1 Getting Started

Before starting the generic editor the first time, the configuration file must be adapted to the special requirements of each application. The generic editor is language-independent. It can be set up to run in basically every language, provided a specific language file is supplied. All error messages which pertain to errors occurring before loading the language file are in English.

9.1.1 Setting up the Configuration File

The configuration file uses a lot of parameters to configure the appearance of the application. The Generic Editor won't run without specifying the configuration file as a parameter to the program. One specific parameter specifies where to find the language translations. Should there be an error during the loading process of the configuration file or the language file the application will stop and display an according error message. In the current release the profile has to be edited by means of a text editor.

9.1.2 The Parameters of the Configuration File

The following table lists the various parameters and tags used by the configuration file. The file is in XML file format. The default configuration file is AnorexieGermanProfile.xml. For this thesis the Generic Editor is configured with the AnorexieEnglishProfile.xml configuration file to have the same languages in the user handbook and the application. Still, the questionnaires are in German as the form definitions (especially the question texts) have not been translated. Table 9-1 lists the tags of the profile and there parameters and explains their use.

Tag	**Explanation**	**Parameters**
editor_profile	The outermost tag which states that this xml file contains a profile for the editor	
title	The title of the application specified in the language which is used during the application[1]	
Questionnaire_Groups	This section specifies the different types of questionnaires which can be used in the specific application. The parameter count is mandatory and specifies the number of groups used during the application	

[1] translate, if setting up the Editor for a new Language

Tag	**Explanation**	**Parameters**
group	This subsection of Questionnaire_Groups specifies the options of a specific type of questionnaires	
id	…is a subsection of group and specifies the id of the group in the language used during the application. [2] It should be unique for the users to be able to distinguish the questionnaire groups. Examples for ids would be “Questionnaire” or “Master file data”	
path	…is a subsection of group and specifies the location of the xml file defining the questionnaire.	
answer_dir	…is a subsection of group and specifies the subdirectory used within Person_Dir to store the answers for a specific replier given at a specific time.	
answer_file	…is a subsection of group and an alternative to answer_dir. Only one of these two tags is allowed to be set for a specific group. If answer_file is set than all the answers to questionnaires of this groups are stored in a single file rather than in the directory structure below Person_Dir.	
single	…is a subsection of group and specifies whether there is only one set of answers allowed per replier (when set to true, i.e. static data are recorded) or whether there can be multiple answer sets given at different times (when set to false, i.e. time dependent data are recorded)	
ShortCut	…is a subsection of group and specifies the shortcut which starts the answering modus for the selected questionnaire group	

[2] translate, if setting up the Editor for a new Language

Tag	**Explanation**	**Parameters**
addon_head_fields	…is a subsection of group and specifies whether there are additional head fields for questionnaires of this group. A head field is a field which specifies a characteristic specific to the whole questionnaire, e.g. a qualitative time description about when the questions of the questionnaire had been posed.	count: the number of additional head fields
field	…is a subsection of addon_head_fields. It specifies a single head field	name: the name of the field xml_flag: the xml_flag: used to store it. description: for display only data_type: keyfield: if true, it is part of the key
allowed_values	…is a subsection of field and lists the allowed_values. It is empty, if any value is allowed.	count: the number of allowed fields.
allowed	the values which can be entered	value
additional_xml	…is a subsection of allowed and specifies a mapping to one or more different xml_flags used to store the same data.	flag: the xml_flag value: the value description: for display only data_type
Person_Dir	As a subsection of the editor_profile it specifies the directory where the answers of specific repliers can be found. The Person_Dir is optional, it must be used, in case of questionnaire groups, which keep the answers in a subdirectory of the Person_Dir.	

Tag	**Explanation**	**Parameters**
Person_File	As a subsection of the editor_profile it specifies the file, which should keep the ids of the available repliers. This tag is mandatory and has precedence before the directory structure under Person_Dir: If the Person_File exists, all available ids are read from the Person_File. If it does not exist, the Person_Dir is searched for available Replier-Ids and the Person_File is created according to the subdirectories of Person_Dir.	
Person_Type	… specifies, what kind of replier answers the question of the specific application, e.g. Patients	
language	As a subsection of the editor_profile, it specifies the name of the language used in a specific application. This is also the name of the language resource file.	
external_file	… specifies a default export file for exporting the data	<u>format</u> can be “Conflation”, “CSV” or “AnswerFile”
external_cmd	…a command which will be executed automatically after exporting the data to the file specified in external_file, when a special submenu option of the Extra Menu is chosen. The name of the submenu is either the one specified by the parameter name or the command name without the complete path (if one was specified)	<u>name</u> The <u>name</u> to display in the menu

Table 9-1 Tags and Parameters of the Profile. We describe the tags and the attributes of the profile.

9.1.3 The Default Configuration File in German

The default configuration file is AnorexieGermanProfile.xml (see Code 9-1). It can be found below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<editor_profile>
<title>Anorexie</title>
<Questionnaire_Groups count = "3">
  <group>
     <id>Fragebogen</id>
     <path>C:\Dokumente und Einstellungen\Martina\Eigene
Dateien\Diplom_Miksch\data_fuer_eingabetool2\parameter_master_data.xml</pa
th>
     <answer_dir>.</answer_dir>
     <single>false</single>
     <ShortCut>F</ShortCut>
     <addon_head_fields count = "2">
        <field name= "Qualitativer Zeitpunkt" xml_flag="kat"
description="Qualitativen Zeitpunt wählen" displayas = "combobox">
           <allowed_values count="7">
                 <allowed value="pre" count_addon_flags="1">
                       <additional_xml flag="katnr" value="100"/>
                 </allowed>
                 <allowed value="kat1" count_addon_flags="1">
                       <additional_xml flag="katnr" value="200"/>
                 </allowed>
                 <allowed value="kat2" count_addon_flags="1">
                       <additional_xml flag="katnr" value="300"/>
                 </allowed>
                 <allowed value="kat3" count_addon_flags="1">
                       <additional_xml flag="katnr" value="400"/>
                 </allowed>
                 <allowed value="kat4" count_addon_flags="1">
                       <additional_xml flag="katnr" value="500"/>
                 </allowed>
                 <allowed value="post" count_addon_flags="1">
                       <additional_xml flag="katnr" value="600"/>
                 </allowed>
                 <allowed value="eval2" count_addon_flags="1">
                       <additional_xml flag="katnr" value="700"/>
                 </allowed>
           </allowed_values>
        </field>
        <field name="Beantwortet von:" xml_flag="parameter_source_flag"
keyfield ="true" displayas="combobox">
           <allowed_values count="3">
                 <allowed value="C"     description = "Child"/>
                 <allowed value="M"     description = "Mother"/>
                 <allowed value="F"     description = "Father"/>
                 </allowed_values>
        </field>
     </addon_head_fields>
  </group>
  <group>
     <id>Variable Stammdaten</id>
     <path>C:\Dokumente und Einstellungen\Martina\Eigene
Dateien\Diplom_Miksch\data_fuer_eingabetool2\variable_stammdaten_master_da
ta.xml</path>
     <answer_file>..\var_masterfile.xml</answer_file>
     <single>false</single>
```

```
      <ShortCut>V</ShortCut>
   </group>
   <group>
      <id>Stammdaten</id>
      <path>C:\Dokumente_und_Einstellungen\Martina\Eigene
Dateien\Diplom_Miksch\data_fuer_eingabetool2\stammdaten_master_data.xml</p
ath>
      <answer_file>person_masterdata.xml</answer_file>
      <single>true</single>
      <ShortCut>S</ShortCut>
   </group>
</Questionnaire_Groups>
<Person_Dir>C:\Dokumente und Einstellungen\Martina\Eigene
Dateien\Diplom_Miksch\data_fuer_eingabetool2\data_records\</Person_Dir>
<Person_File>person.xml</Person_File>
<Person_Type>Patient</Person_Type>
<Person_ShortCut>P</Person_ShortCut>
<language>german</language>
<external_cmd name="Gravi++">C:\Dokumente und Einstellungen\Martina\Eigene
Dateien\Diplom_Miksch\data_fuer_eingabetool2\mybatch.bat</external_cmd>
<external_file format="Conflation">C:\Dokumente und
Einstellungen\Martina\Eigene
Dateien\Diplom_Miksch\data_fuer_eingabetool2\gravioutput.xml</external_fil
e>
</editor_profile>
```

Code 9-1 Default Configuration File. We list the default profile.

9.1.4 The English Configuration File Used for this Documentation

The English configuration file used for this thesis is AnorexieEnglishProfile.xml. It can be found in Code 9-2:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<editor_profile>
<title>Anorexia</title>
<Questionnaire_Groups count = "3">
   <group>
      <id>Questionnaire</id>

   <path>G:\Dokumente_und_Einstellungen\martina\Diplom_Miksch\data_fuer_ein
gabetool2\parameter_master_data.xml</path>
      <answer_dir>.</answer_dir>
      <single>false</single>
      <ShortCut>F</ShortCut>
      <addon_head_fields count = "2">
         <field name= "Qualitative Timestamp" xml_flag="kat" keyfield =
"false">
            <allowed_values count="6">
                  <allowed value="pre" count_addon_flags="1">
                        <additional_xml flag="katnr" value="100"/>
                  </allowed>
                  <allowed value="kat1" count_addon_flags="1">
                        <additional_xml flag="katnr" value="200"/>
                  </allowed>
                  <allowed value="kat2" count_addon_flags="1">
                        <additional_xml flag="katnr" value="300"/>
                  </allowed>
                  <allowed value="kat3" count_addon_flags="1">
                        <additional_xml flag="katnr" value="400"/>
                  </allowed>
                  <allowed value="kat4" count_addon_flags="1">
```

```
                    <additional_xml flag="katnr" value="500"/>
                </allowed>
                <allowed value="post" count_addon_flags="1">
                    <additional_xml flag="katnr" value="600"/>
                </allowed>
            </allowed_values>
        </field>
        <field name="Replied by:" xml_flag="parameter_source_flag">
            <allowed_values count="3" keyfield = "true">
                <allowed value="C"      description = "Child"/>
                <allowed value="M"      description = "Mother"/>
                <allowed value="F"      description = "Father"/>
                </allowed_values>
        </field>
      </addon_head_fields>
  </group>
  <group>
      <id>Variable Master File</id>

  <path>G:\Dokumente_und_Einstellungen\martina\Diplom_Miksch\data_fuer_ein
gabetool2\variable_stammdaten_master_data.xml</path>
      <answer_file>var_masterfile.xml</answer_file>
      <single>false</single>
      <ShortCut>V</ShortCut>
  </group>
  <group>
      <id>Master File</id>
  <path>G:\Dokumente_und_Einstellungen\martina\Diplom_Miksch\data_fuer_ein
gabetool2\stammdaten_master_data.xml</path>
      <answer_file>person_masterdata.xml</answer_file>
      <single>true</single>
      <ShortCut>S</ShortCut>
  </group>
</Questionnaire_Groups>
<Person_Dir>G:\Dokumente_und_Einstellungen\martina\Diplom_Miksch\data_fuer
_eingabetool2\data_records\</Person_Dir>
<Person_File>person.xml</Person_File>
<Person_Type>Patient</Person_Type>
<Person_ShortCut>P</Person_ShortCut>
<language>english</language>
<external_cmd name="Gravi++">C:\Dokumente und Einstellungen\Martina\Eigene
Dateien\Diplom_Miksch\data_fuer_eingabetool2\mybatch.bat</external_cmd>
<external_file format="Conflation">C:\Dokumente und
Einstellungen\Martina\Eigene
Dateien\Diplom_Miksch\data_fuer_eingabetool2\gravioutput.xml</external_fil
e>
</editor_profile>
```

Code 9-2 English Profile. We list the English profile used during this thesis.

9.1.5 The Language Specific Translations

The language specific translations of the program's menu entries, error messages and captions are found in a file relative to the root of the program installation in a subdirectory called resources. Its name is composed out of the content of the <language> settings in the configuration file. E.g. if it states **`<language>`**english**`</language>`** in the configuration file, than the file for the language specific translations is

{programroot}\resources\english.xml,

where {programroot} directory the Generic Editor is installed in.

9.2 The Command Line Parameters

The Generic Editor knows two command line parameters. The first is compulsory and specifies the location of the configuration file. The second disables the redirection of the outputs sent to STDERR to error.log and the outputs sent to STDOUT to history. The user can optionally choose this option by specifying 1 as a second command line parameter. This is an example showing how to invoke the program with redirected outputs:

```
 java -jar GenericEditor.jar .\AnorexieGermanProfile.xml
```

The following demonstrates how to configure the program for sending its messages to the console instead.

```
 java -jar GenericEditor.jar .\AnorexieGermanProfile.xml 1
```

9.3 Starting the Program

It is recommended to create a batch file for invoking the Generic Editor. The default batch file is: GenEdit.bat

The start up of the program is done in six steps:

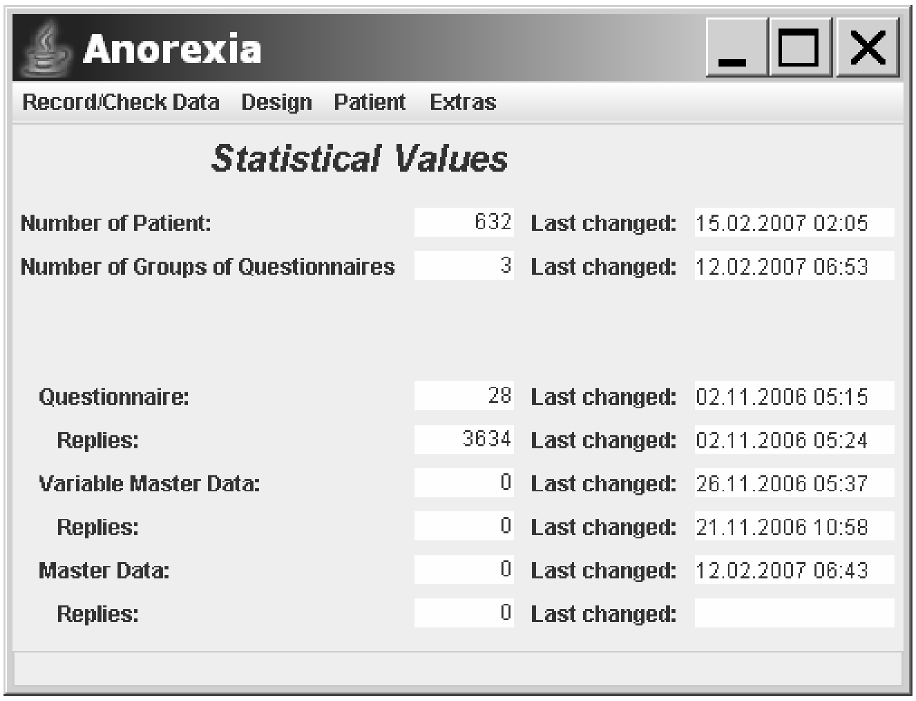
1. The profile is read for configuring the generic editor.
2. The language specific translations are read.

   If there is an error in steps (1) or (2) an error message will be displayed and the application will be closed again.
3. The parameter sources are loaded.
4. The forms of each form group are loaded.
5. The answers for each group of forms are loaded. If there are parameter-sources not yet defined through step 3, they will be automatically created during the loading process of the answers.
6. The main menu is displayed.

Following is the GenEdit.bat file, which has been used for creating this documentation:

```
java -jar GenericEditor.jar .\AnorexieGermanProfile.xml 1
```

Figure 9-1 depicts an example of a start up screen. It additionally displays basic descriptive statistics about the available parameter sources, forms and answers for each group.



Number of Patient:	632	Last changed:	15.02.2007 02:05
Number of Groups of Questionnaires	3	Last changed:	12.02.2007 06:53
Questionnaire:	28	Last changed:	02.11.2006 05:15
Replies:	3634	Last changed:	02.11.2006 05:24
Variable Master Data:	0	Last changed:	26.11.2006 05:37
Replies:	0	Last changed:	21.11.2006 10:58
Master Data:	0	Last changed:	12.02.2007 06:43
Replies:	0	Last changed:	

Figure 9-1 Main Window. We give an example of the main window.

Once the main window is displayed and the loading of the data is complete the user can start to work with the Generic Editor.

9.4 Using the Generic Editor

The Generic Editor supports four different tasks:

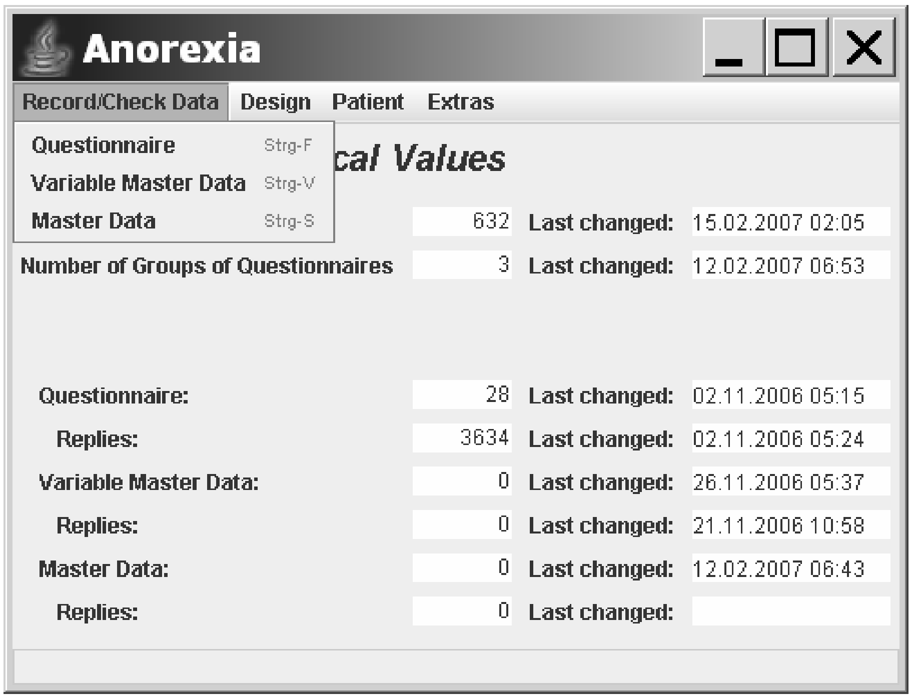
1. The recording of measurement data.
2. The design of forms
3. The maintenance of parameter sources
4. Export and Import of the measurement data.

The first three tasks reflect the fact that there are three basic things necessary to collect values for parameters:

1. Structural data: the definitions of the parameters. The parameters are grouped together in forms. The structural data is defined by task 2, the design of the forms.
2. Sources for the parameters. These are for example persons replying to a questionnaire. The parameter sources are defined by task 3, the maintenance of parameter sources.
3. Measurement data: These are the values of a parameter given by a parameter source (at a certain time). Measurement data are recorded by task 1.

9.4.1 Recording Data

The most frequent task is to record measurement data, which is to record the values to parameters of a certain form for an existing parameter source. These values are originated by a parameter source at a certain specifiable time. Some measurement data – e.g. master data - are not time-oriented and do not require specifying the time.
The first step is to select the group of questionnaires. The available groups of questionnaires are defined in the profile. Figure 9-2 shows the selection for the AnorexieEnglishProfile.xml



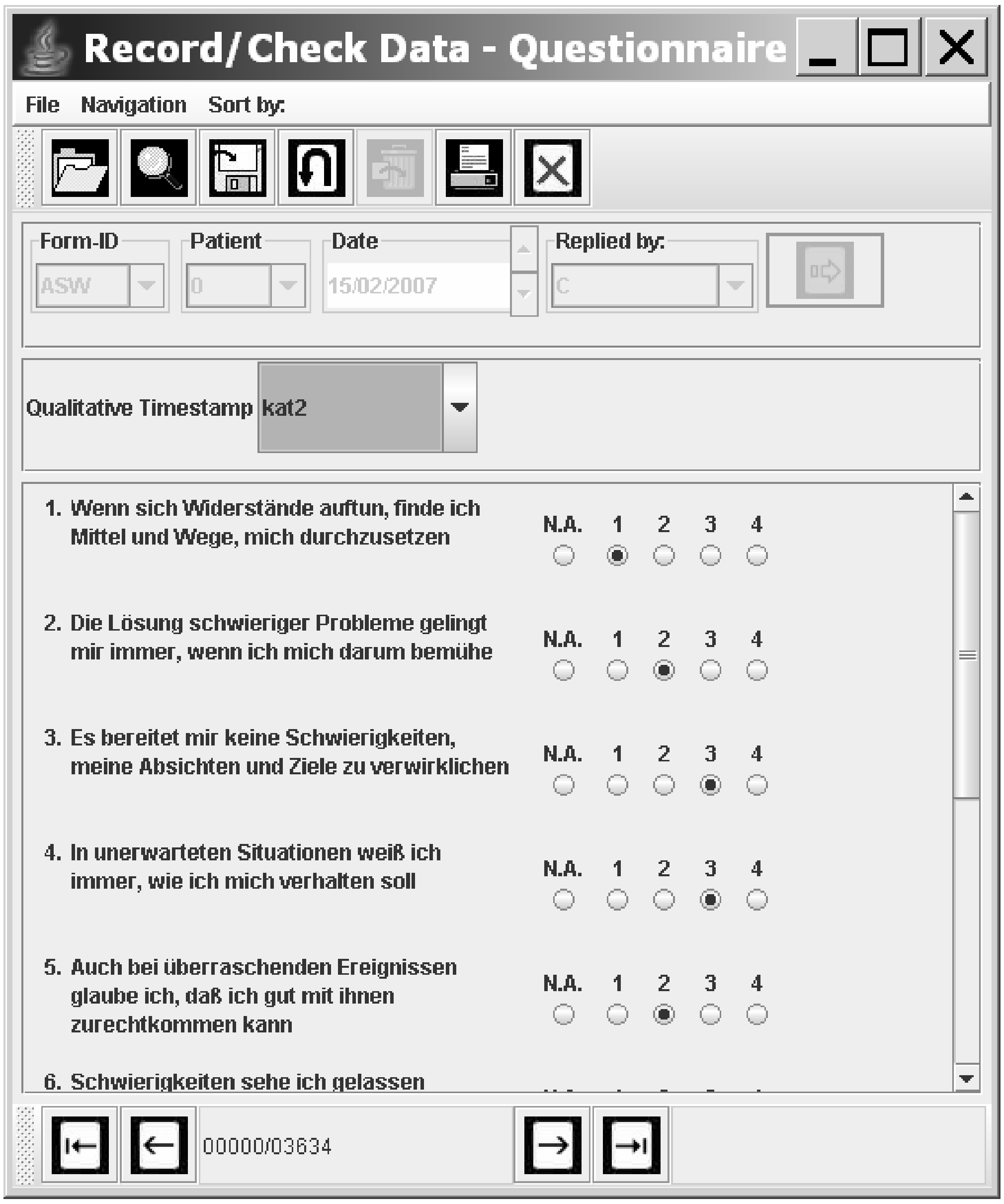
cal Values

	632	Last changed:	15.02.2007 02:05
Number of Groups of Questionnaires	3	Last changed:	12.02.2007 06:53
Questionnaire:	28	Last changed:	02.11.2006 05:15
Replies:	3634	Last changed:	02.11.2006 05:24
Variable Master Data:	0	Last changed:	26.11.2006 05:37
Replies:	0	Last changed:	21.11.2006 10:58
Master Data:	0	Last changed:	12.02.2007 06:43
Replies:	0	Last changed:	

Figure 9-2 Selection of the Form Group. Before the recording of parameter values can be started the user has to select the form group first.

If there are no parameter sources available or no forms defined for the selected form group, the program will display an error message instead of opening the record data window. It prompts the user to first define at least one parameter source or at least one form for this form group respectively.

Figure 9-3 shows, how the record data window looks like for questionnaires when using the AnorexieEnglishProfile.xml.



Form-ID: ASW
Patient: 0
Date: 15/02/2007
Replied by: C

Qualitative Timestamp: kat2

1. Wenn sich Widerstände auftun, finde ich Mittel und Wege, mich durchzusetzen — N.A. [✓ 1] 2 3 4
2. Die Lösung schwieriger Probleme gelingt mir immer, wenn ich mich darum bemühe — N.A. 1 [✓ 2] 3 4
3. Es bereitet mir keine Schwierigkeiten, meine Absichten und Ziele zu verwirklichen — N.A. 1 2 [✓ 3] 4
4. In unerwarteten Situationen weiß ich immer, wie ich mich verhalten soll — N.A. 1 2 [✓ 3] 4
5. Auch bei überraschenden Ereignissen glaube ich, daß ich gut mit ihnen zurechtkommen kann — N.A. 1 [✓ 2] 3 4
6. Schwierigkeiten sehe ich gelassen

00000/03634

Figure 9-3 Record Data Window. We give an example of a record data window.

After the record data window is displayed, the user can perform various tasks:

- File
    - New Selection = Close: Closes the current Selection.
    - Search: Opens the search/filter window and enables the user to select a record or assign values of head fields without explicitly opening the record.
    - Save: Saves the answers of the current questionnaire.
    - Reset: Resets the answers of the current questionnaire through reloading the answers
    - Delete: Deletes currently displayed answers
    - Print: Prints the displayed questionnaire.
- Navigation
    - First Record
    - Previous Record
    - Next Record

    - Last Record
- Sort by
    - Form Id
    - Parameter Source Id (in the case of the AnorexieEnglishProfile.xml these are Patient Ids.

After starting the record data window, the user can select the record he wants to work with. He can either select an existing record or create a new one.

9.4.1.1 Create a New Record

In order to create a new record the user needs to specify all the key parameters to these records. The key parameters consist of the form-id, the parameter source id, the date (if the form belongs to a form group which is time oriented) and possibly further key values as defined in the profile. After selecting the desired values, they user has to confirm his selection by clicking the button at the right of these fields or by pressing the enter key. Then the user can start keying in the answers given by the parameter source. For further information on this topic please read section 9.4.1.3.

9.4.1.2 Select an Existing Record

There are a couple of possibilities for selecting an existing record:

1. The user can enter the key values directly as described in section 9.4.1.1.
2. The user can search for an existing record by using the ***Search/Filter*** utility. The Search/Filter utility can be started either by choosing search from the file menu, or by clicking the magnifier or by using a shortcut. For further information on searching look at section 9.4.5.
3. The user can navigate trough the records by choosing ***First Record***, ***Previous Record***, ***Next Record*** or ***Last Record*** either from the Navigation menu or by clicking the icons at the bottom of the screen. It might be useful to ensure that the records are sorting in some way before using the navigation. For information about sorting look at section 9.4.1.10.

9.4.1.3 Entering Answers

After selection of a record the user can enter the answers. The component used for entering an answer can be either a text field or a combo box or a group of radio buttons. Which one is used, depends on the settings specified in the form definition and on heuristics. Please look into section 9.4.2 for further information on this topic.

9.4.1.4 Searching for a Record / Specifying Values of Head Fields without Opening the Record

In order to search for a record the user has to select the ***Search*** utility. The Record Data task as well as the Design task as well as the Export task use a common Search/Filter Dialog. Therefore, the Search/Filter Dialog is described in a section of its own. See section 9.4.5 for further information on this topic.

9.4.1.5 Save the Answers

The user can save changes at any time by choosing the ***Save*** action. If the user selects any task, which could result in closing the current record, the user will be prompted, whether he wants to save his changes, if there are any. The following actions result in closing the current record:

- First Record
- Previous Record
- Next Record
- Last Record
- New Selection / Close
- Search

Furthermore, the user is prompted to save his changes before sorting the records.

9.4.1.6 Resetting the Answers

In order to reset the entered answers to the last saved values, the user can select the ***Reset*** action.

9.4.1.7 Deleting the Answers

The answers currently displayed can be deleted by selecting the ***Delete*** action. The whole answer set will deleted. Afterwards, no record is displayed and the user can continue by selecting another record.

9.4.1.8 Printing the Answers

The questionnaire currently displayed is printed by selecting the ***Print*** action.

9.4.1.9 Close a Record / Work with another Record

In order to work with another record the user can select ***Close*** or ***New Selection***. Both are equivalent and result in the start situation.

9.4.1.10 Sorting the Answers

The answers can be sorted by using the ***Sort by:*** actions. The user can choose whether to primary sort by the form-id or the parameter source id. All other head fields are used for sub sorting the records. E.g., if the user chooses to primary sort for the form-id, the records are also sub sorted by the replier-id, the date etc.

9.4.1.11 Navigation through the Records

The user can navigate through the records by choosing first, previous, next or last record either from the menu or by clicking the according buttons on the bottom of the screen. The user can determine the sequence of the records by sorting them first. For further information on sorting the answers see section 9.4.1.10.

9.4.1.12 The Shortcuts and Icons for the Tasks

The shortcuts and icons for the tasks can be found in Table 9-2. META is short for the META key used by a specific platform. In the case of Windows XP this would be the CTRL key.

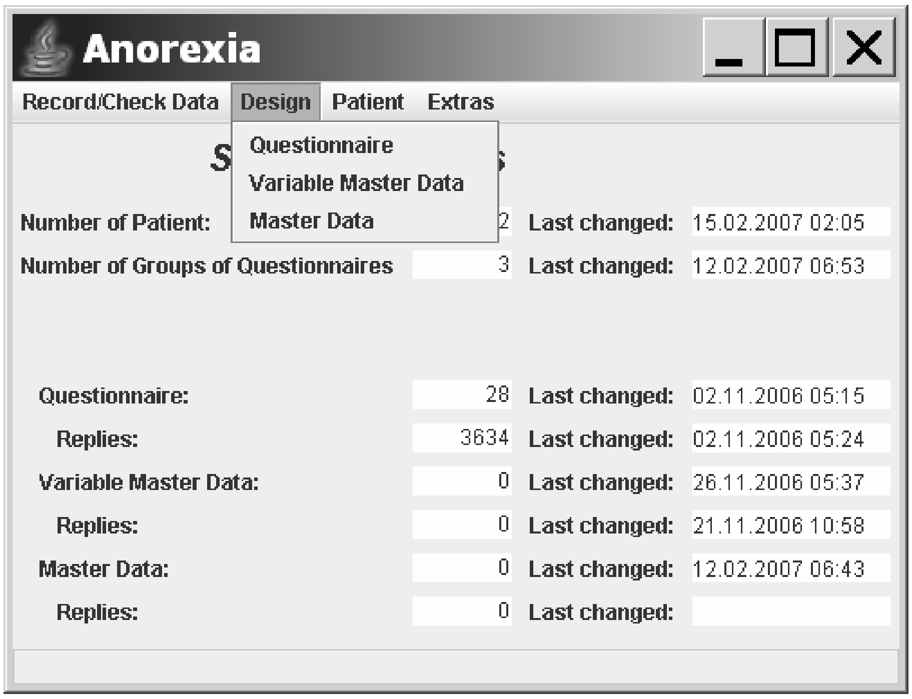
Task	**Shortcut**	**Icon**
New Selection/Close	META-F4	📂 ☒

Task	Shortcut	Icon
Search	META-F	🔍
Save	META-S	💾
Reset	META-Z	↶
Delete	META-D	🗑
Print	META-P	🖨
First Record	META-7	⇤
Previous Record	META-4	←
Next Record	META-6	→
Last Record	META-1	⇥

Table 9-2 Shortcuts in the Record Data Window. We list the actions, shortcuts and icons used in the record data window.

9.4.2 Designing Forms

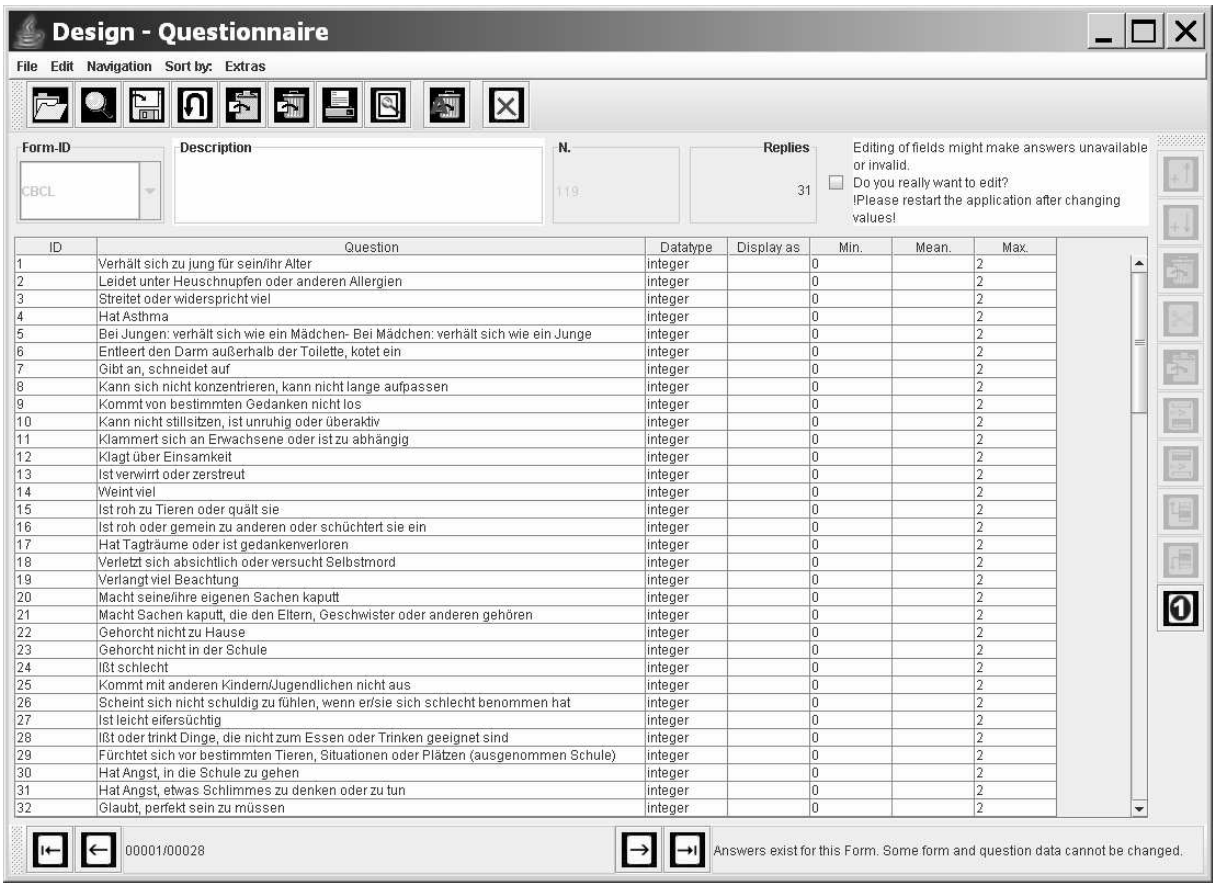
Before any measurement data can be recorded, the user needs to define the forms, for which measurement data is to be recorded.
The first step is to select the group of questionnaires. The available groups of questionnaires are defined in the profile. Figure 9-4 shows the selection for the AnorexieEnglishProfile.xml.



Number of Patient:	2	Last changed:	15.02.2007 02:05
Number of Groups of Questionnaires	3	Last changed:	12.02.2007 06:53
Questionnaire:	28	Last changed:	02.11.2006 05:15
Replies:	3634	Last changed:	02.11.2006 05:24
Variable Master Data:	0	Last changed:	26.11.2006 05:37
Replies:	0	Last changed:	21.11.2006 10:58
Master Data:	0	Last changed:	12.02.2007 06:43
Replies:	0	Last changed:	

Figure 9-4 Selection of the Form Group for Designing Forms. The user has to select for which form group he wants to start the design form process.

Figure 9-5 shows how the Design Form Window looks like:



Form-ID: CBCL

Description:

N.: 119

Replies: 31

Editing of fields might make answers unavailable or invalid.
Do you really want to edit?
!Please restart the application after changing values!

ID	Question	Datatype	Display as	Min.	Mean.	Max.
1	Verhält sich zu jung für sein/ihr Alter	integer		0		2
2	Leidet unter Heuschnupfen oder anderen Allergien	integer		0		2
3	Streitet oder widerspricht viel	integer		0		2
4	Hat Asthma	integer		0		2
5	Bei Jungen: verhält sich wie ein Mädchen- Bei Mädchen: verhält sich wie ein Junge	integer		0		2
6	Entleert den Darm außerhalb der Toilette, kotet ein	integer		0		2
7	Gibt an, schneidet auf	integer		0		2
8	Kann sich nicht konzentrieren, kann nicht lange aufpassen	integer		0		2
9	Kommt von bestimmten Gedanken nicht los	integer		0		2
10	Kann nicht stillsitzen, ist unruhig oder überaktiv	integer		0		2
11	Klammert sich an Erwachsene oder ist zu abhängig	integer		0		2
12	Klagt über Einsamkeit	integer		0		2
13	Ist verwirrt oder zerstreut	integer		0		2
14	Weint viel	integer		0		2
15	Ist roh zu Tieren oder quält sie	integer		0		2
16	Ist roh oder gemein zu anderen oder schüchtert sie ein	integer		0		2
17	Hat Tagträume oder ist gedankenverloren	integer		0		2
18	Verletzt sich absichtlich oder versucht Selbstmord	integer		0		2
19	Verlangt viel Beachtung	integer		0		2
20	Macht seine/ihre eigenen Sachen kaputt	integer		0		2
21	Macht Sachen kaputt, die den Eltern, Geschwister oder anderen gehören	integer		0		2
22	Gehorcht nicht zu Hause	integer		0		2
23	Gehorcht nicht in der Schule	integer		0		2
24	Ißt schlecht	integer		0		2
25	Kommt mit anderen Kindern/Jugendlichen nicht aus	integer		0		2
26	Scheint sich nicht schuldig zu fühlen, wenn er/sie sich schlecht benommen hat	integer		0		2
27	Ist leicht eifersüchtig	integer		0		2
28	Ißt oder trinkt Dinge, die nicht zum Essen oder Trinken geeignet sind	integer		0		2
29	Fürchtet sich vor bestimmten Tieren, Situationen oder Plätzen (ausgenommen Schule)	integer		0		2
30	Hat Angst, in die Schule zu gehen	integer		0		2
31	Hat Angst, etwas Schlimmes zu denken oder zu tun	integer		0		2
32	Glaubt, perfekt sein zu müssen	integer		0		2

00001/00028

Answers exist for this Form. Some form and question data cannot be changed.

Figure 9-5 Design Form Window. We give an example of a design form window.

After the design window is displayed, the user either needs to select an existing form, which he wants to modify or enter a new unique form-id for creating a new form.
Once a form-id has been chosen, the user can perform various tasks on the form. If there are already answers available for a form, the program prohibits the user to change some structural information, as long as the user does not enable changes explicitly.

The following tasks can be performed during the design process:

- File
  - New Selection
  - Search
  - Save
  - Reset
  - Copy
  - Delete
  - Print
  - Preview
- Edit
  - New
    - above
    - below
  - Delete
  - Cut
  - Copy

        - Paste
            - above
            - below
        - Move
            - up
            - down
        - Allowed Answers
- Navigation
    - First
    - Previous
    - Next
    - Last
- Sort By:
    - Form-Id
    - Description
- Extras
    - Delete Answers

9.4.2.1 Creating a new Form

A new form will be created automatically, if the user enters a new Form-Id in the combo box ***Form-ID***. After entering a new form-id an empty Form will be displayed. Thereafter, the user can add questions etc.

9.4.2.2 Opening an Existing Form

The user can either select the Form-Id from the List in the Combo Box ***Form-ID*** or enter it manually. Furthermore, he can choose a record by searching for it or by navigating through the records by selecting one of the navigation actions. After selecting an existing form-id the system checks for answers to this form. The number of answers which exist for a selected form is displayed in the information field ***Replies***. If there are answers to a form, some form properties can only be changed, if the user enables changing explicitly. The following changes are still possible without enabling changes explicitly, even if answers exist:

- Change of the question text
- Change of the way the answers can be entered
- Definition of minimum, maximum and mean values. The minimum and the maximum values are the minimum allowed and the maximum allowed values. So if the user specifies a new minimum or a new maximum value, the program checks whether the new minimum or maximum conforms to the existing data.
- Deletion of all allowed values. This is equal to allowing every value of the specified data type

9.4.2.3 Specifying the Form's Parameters and the Questions

After selection of a form the user can enter a description to the form and the questions.

9.4.2.3.1 Questions and Question's Properties

Questions can be added by choosing ***Add Above*** or ***Add Below*** either from the Edit Menu or by using the buttons or shortcuts.
Both actions will result in a new row in the question table where one can specify the parameters to be used for a question. The program will automatically assign a unique numeric question id to each new question. The question id can be overwritten and may also be a string, but it must be unique within a form. Also all other fields will contain default values for a new question.

From left to right the user can specify:

- The question id
- The question text
- The data type of the answer: This can be:
    - Integer (the default)
    - Float
    - String
- The way how the entries of the answers to a question can be done. This can be:
    - Empty -> the program uses a heuristic for displaying
    - Text field: a text field is displayed in any case
    - A group of radio buttons: radio buttons are displayed, if there are either allowed values for this question, or the data type of the answers is integer and the user specified a minimum and a maximum allowed value. If none of the above is applicable, the program will display a text field.
    - A combo box. Also the combo box will only be displayed, if there are allowed values defined or the data type of the question is integer and there is a minimum as well as a maximum allowed value defined. In any other circumstance a text field will be displayed instead.
- The minimum allowed value: This is the minimum value which may be entered by the user when recording answers to this question.
- The maximum allowed value: This is the maximum value which may be entered by the user when recording answers to this question.
- The mean value: The mean value defines a normal value for the answers. In the current version of the program it is only defined and stored but not used when recording answers. It could be used as a default value in future releases.

9.4.2.3.2 Defining allowed values for a questions

In order to define allowed values the user has to choose ***Edit/Allowed Values*** first. This will open another window, which displays and keeps track of the defined values of the marked question. To define allowed values, the user can thereafter use:

Edit/Add:	to add a new allowed value
Edit/Remove:	to delete the marked allowed value
Edit/Remove All:	to remove all allowed values, which means allowing any value of the data type chosen for this question.

9.4.2.3.3 Moving a question

The marked question can be moved up or down by choosing the actions ***Edit/Move Up*** or ***Edit/Move Down***. The sequence of the questions in the design window determines the sequence of the questions when recording answers for a particular form and the sequence of the answers when importing data using the “single row” CSV format. For further information on importing data refer to section 9.4.4.2.

9.4.2.3.4 Cut, Copy, Paste and Delete of Questions

Both ***Cut*** and ***Delete*** will remove a question from the form definition. In the contrary to ***Delete, Cut*** will keep a copy of the question in a temporary question. ***Copy*** too makes a copy of the currently marked question. Once there is a copy of a question available, the user can paste the question either above or below a currently marked question.

9.4.2.4 Searching for a Form / Specifying Values of Head Fields without Opening the Record

In order to search for a form the user has to select the ***Search*** utility either from the File Menu or by clicking the magnifier. The record data task as well as the design task as well as the export task use a common filter dialog. Therefore, the filter dialog is described in a section of its own. See section 9.4.5 for further information on searching and filtering data.

9.4.2.5 Save the Form

The user can save changes at any time by choosing ***Save*** from the File Menu, or by clicking the disk. If the user selects any task, which could result in closing the current form, the user will be asked, whether he wants to save his changes, if there are any. The following tasks result in closing the current form:

- First Record
- Previous Record
- Next Record
- Last Record
- New Selection / Close
- Search

Furthermore, the user is prompted to save his changes before sorting the records.

9.4.2.6 Resetting the Form

In order to reset the form to the last saved definition, the user can select ***Reset*** from the File menu or click the reset button.

9.4.2.7 Copying a Form

The user is enabled to quickly set up similar questionnaires by copying an existing form to a new form. In order to copy a form the user has to select the ***Copy*** action. All questions of the form including their allowed values will be copied to the new form if the user specifies a new unique form-id upon request.

9.4.2.8 Deleting the Form

The form currently displayed can be deleted by selecting ***Delete*** from the File menu. This will delete the form completely and results in the start situation. If there are answers available for the form the user will be asked, whether he wants to delete these answers, too. If he declines, the form will not be deleted.

9.4.2.9 Printing the Form

The form currently displayed is printed by selecting ***Print*** from the File menu or by clicking the Print button. Printing the form will lead to a printout of an empty form, which can be used as paper questionnaire.

9.4.2.10 Previewing the Form

One can preview the way how the form is displayed when recording answers for it by choosing the ***Preview*** action. The user has to store the form before running the preview. None of the actions available when recording data can be used in the preview mode.

9.4.2.11 Close a Record / Work with Another Record

In order to work with another record the user can select ***File/Close*** or ***File/New Selection***. Both are equivalent and result in the start situation.

9.4.2.12 Navigation through the Records

The user can navigate through the records by choosing ***first, previous, next*** or ***last record*** either from the menu or by clicking the according buttons at the bottom of the screen. The user can determine the sequence of the records by sorting them first. For further information on sorting the forms see section 9.4.2.13.

9.4.2.13 Sorting the Forms

The forms can be sorted by using the ***Sort by:*** Menu. The user can choose whether to primary sort by the form id or by the form description. All other form head fields are used for sub sorting the records. E.g. if the user chooses to primary sort by the form id, the records are also sub sorted by the form description and the number of questions.

9.4.2.14 Deleting All Answers to a Form

For convenience the user is provided with an option to delete all answers to a form. This action can be either started by selecting the ***Delete all Answers*** action explicitly or implicitly when the user decides to delete a form, for which there are still answers available. In the second case the system asks, whether the user wants to delete all the answers to a form and will only delete the answers and the form, if the user agrees.

9.4.2.15 The Shortcuts and Icons for the Tasks

Table 9-3 lists the shortcuts and icons for the design form task. META is short for the META key used by a specific platform. In the case of Windows XP this would be the CTRL key.

Task	**Shortcut**	**Icon**
New Selection/Close	META-F4	📂 ☒
Search	META-F	🔍
Save	META-S	💾
Reset	META-Z	↶
Copy	META-Y	📋
Delete	META-D	🗑
Print	META-P	🖨
Preview	META-L	🔍
First Record	META-7	⇤
Previous Record	META-4	←

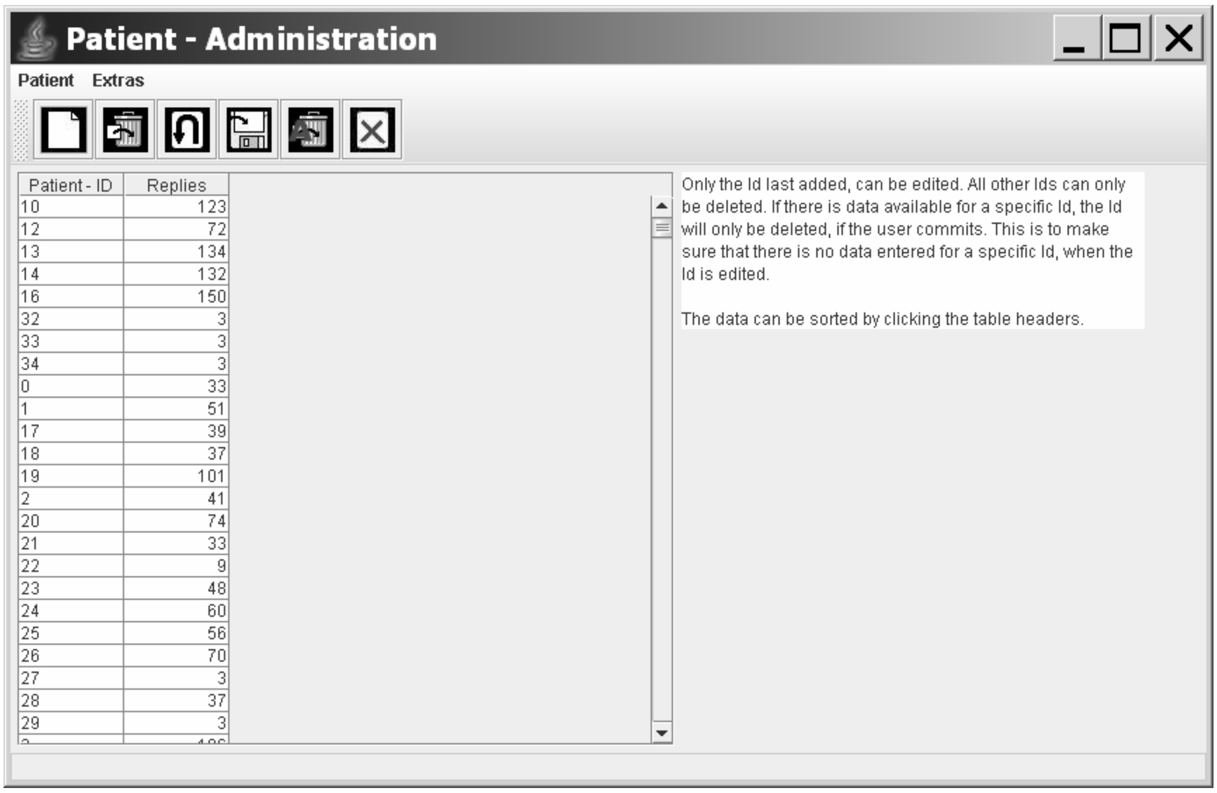
Task	**Shortcut**	**Icon**
Next Record	META-6	→
Last Record	META-1	→|
New above	META-+	+↑
New below	META-I	+↓
Delete	META-Del	🗑
Cut	META-B	✂
Copy (question)	META-K	🗑
Paste above	META-M	⇨
Paste below	META-J	⇨
Move up	META-8	↑
Move down	META-2	↓
Allowed Value	META-O	①
Add (allowed value)	META-1	①
Remove (allowed value)	META-BS	①
Remove all (allowed values)	META-2	①
Delete Answer	META-R	🗑

Table 9-3 Action of the Design Form Window. We list the actions, shortcuts and icons used in the design form window.

9.4.3 Maintaining Parameter Sources

The parameter sources are the sources of the values to the parameters. These can be for example the sources of the answers to questionnaires.
The type of the source is specified in the profile. In the case of the AnorexieEnglishProfile.xml the parameter sources are patients who participate at a clinical study about Anorexia.
In order to record data for a certain parameter source, the parameter source must be available. This means it must be created before being able to record data for a certain parameter source.
In the contrary to recording answers or designing forms, the maintenance of parameter sources requires only a single table, because the program does not record any details to a parameter source besides its id. Further information on parameter sources are handled conceptually like ordinary questionnaires. Therefore, the user has to design one or more forms for keeping track of the master data to a parameter source instead of directly entering them during the maintenance of parameter sources.

After selecting the maintenance of Parameter Source Ids - in the case of our AnorexieEnglishProfile.xml (see Code 9-2) these are patients – the following window is displayed (see Figure 9-6):



Patient - ID	Replies
10	123
12	72
13	134
14	132
16	150
32	3
33	3
34	3
0	33
1	51
17	39
18	37
19	101
2	41
20	74
21	33
22	9
23	48
24	60
25	56
26	70
27	3
28	37
29	3

Only the Id last added, can be edited. All other Ids can only be deleted. If there is data available for a specific Id, the Id will only be deleted, if the user commits. This is to make sure that there is no data entered for a specific Id, when the Id is edited.

The data can be sorted by clicking the table headers.

Figure 9-6 Parameter Source Maintenance Window. We give an example of the parameter source maintenance window.

The window displays a table of all available parameter sources – in our case patients – together with the number of answers available for each patient.

Now the following actions are possible:

- Patient
  - New
  - Delete
  - Save
  - Reset
  - Close
- Extras
  - Delete Answers

9.4.3.1 Creating a New Parameter Source

A new parameter source can be created by choosing the ***new*** action either from the parameter source menu or by using the shortcut or by clicking the new button

9.4.3.2 Delete

An existing parameter source can be deleted by marking it and choosing the ***delete*** action either from the parameter source menu or by using the shortcut or by clicking the delete button.

9.4.3.3 Saving Changes

Changes to the parameter source table are saved by demand through choosing the ***save*** action either from the parameter source menu or by using the shortcut or by clicking the save button. Furthermore, the user will be asked whether he wants to save his changes, if the parameter source maintenance window is closing.

9.4.3.4 Reset

The parameter source table can be reloaded from the disk by choosing the ***reset*** action either from the parameter source menu or by using the shortcut or by clicking the reset button.

9.4.3.5 Close

The parameter source maintenance window can be closed by choosing the ***close*** action either from the parameter source menu or by using the shortcut or by clicking the close button.

9.4.3.6 Delete All Answers of a Parameter Source

The user has the option to delete all answers which exist for the marked parameter source by choosing the ***delete answers*** action from the extras menu.

9.4.3.7 The Shortcuts and Icons for the Tasks

Table 9-4 lists the shortcuts and Icons of the Parameter Source maintenance:

Task	**Shortcut**	**Icon**
Close	META-F4	☒
New	META-+	🗋
Delete	META-D	🗑
Reset	META-Z	↶
Save	META-S	💾
Delete Answer	META-R	🗑

Table 9-4 Shortcuts of the Actions in the Parameter Source Maintenance Window. We list the actions, shortcuts and icons used in the parameter source maintenance window.

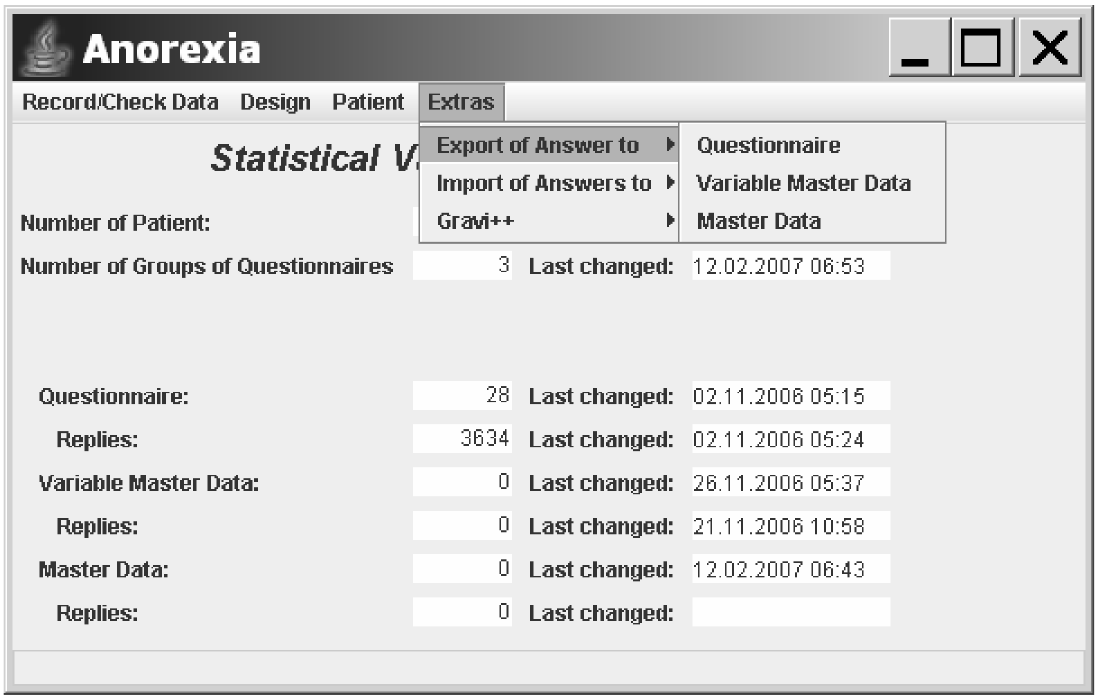
9.4.4 Extras

The Extras menu covers Import and Export facilities only in this version. Depending on the profile there can be a third option available that is export to a file defined in the profile and start of a program which is also defined in the profile. The Export and Import option will only export and import answers.

9.4.4.1 Exporting Data

In order to export data the user has to select the Export menu from the Extras menu. Each Export menu has a submenu to enable the user to select the desired group of questionnaires for which he intents to export the data. So the export requires two steps:

1. selection of the group of questionnaires (see Figure 9-7)
2. define the required parameters

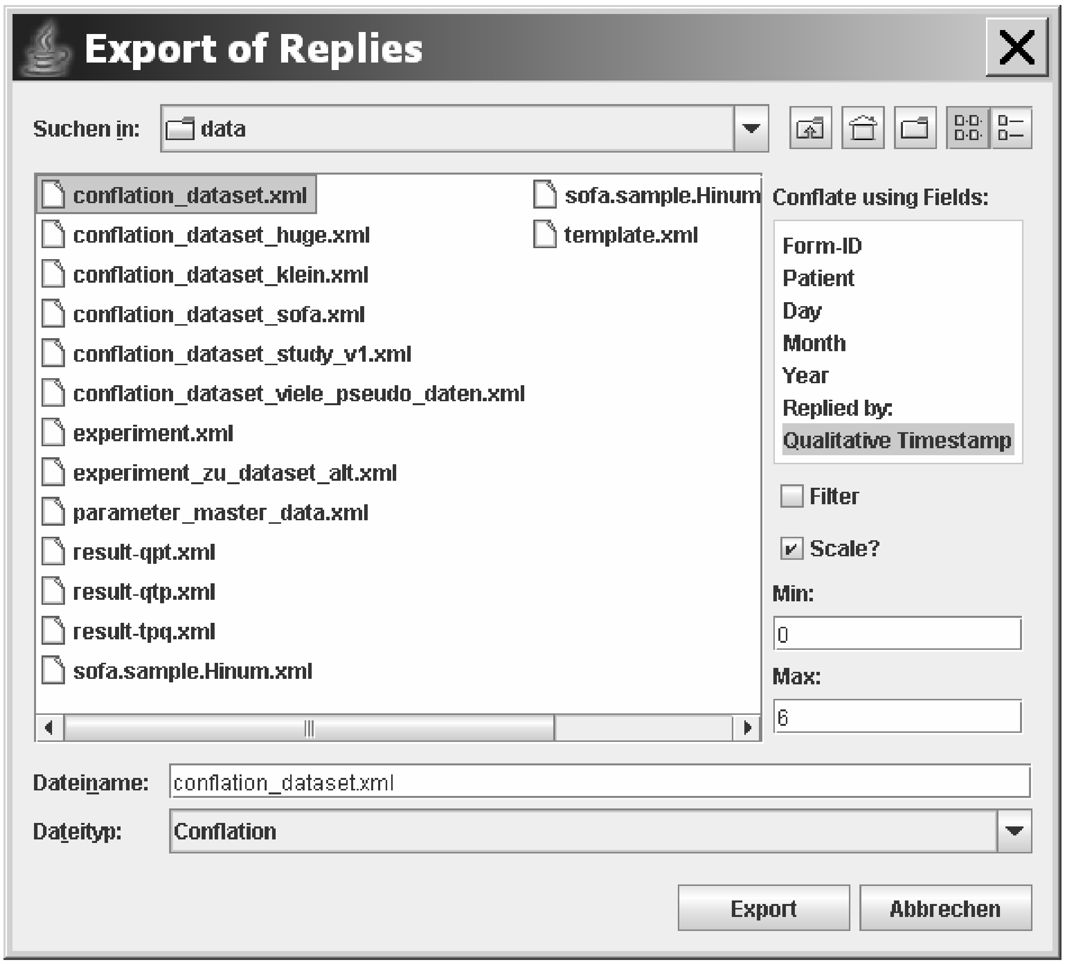


Number of Patient:			
Number of Groups of Questionnaires	3	Last changed:	12.02.2007 06:53
Questionnaire:	28	Last changed:	02.11.2006 05:15
Replies:	3634	Last changed:	02.11.2006 05:24
Variable Master Data:	0	Last changed:	26.11.2006 05:37
Replies:	0	Last changed:	21.11.2006 10:58
Master Data:	0	Last changed:	12.02.2007 06:43
Replies:	0	Last changed:	

Figure 9-7 Selection of the Form Group for Export. Before an export can be started the form group has to be selected.

9.4.4.1.1 Defining the desired parameters

After selecting the group of questionnaires a window like the following will be displayed. The details of the window depend on the profile (see section 9.1.2 for details on the configuration options). Figure 9-8 is an example of an export dialog.


Export of Replies

Suchen in: data

conflation_dataset.xml
conflation_dataset_huge.xml
conflation_dataset_klein.xml
conflation_dataset_sofa.xml
conflation_dataset_study_v1.xml
conflation_dataset_viele_pseudo_daten.xml
experiment.xml
experiment_zu_dataset_alt.xml
parameter_master_data.xml
result-qpt.xml
result-qtp.xml
result-tpq.xml
sofa.sample.Hinum.xml
sofa.sample.Hinum
template.xml

Conflate using Fields:
Form-ID
Patient
Day
Month
Year
Replied by:
Qualitative Timestamp

☐ Filter

☑ Scale?

Min: 0

Max: 6

Dateiname: conflation_dataset.xml

Dateityp: Conflation

Export Abbrechen

Figure 9-8 Export Dialog. We give an example of an export dialog.

In order to export data the user has to specify:

- The file to export to
- The format of the exported file. In the current version the following formats are available:
  - Conflation. The conflation file format is defined in chapter IV.2.1.
  - CSV: The CSV format is defined in chapter IV.2.2 and is equal to the multiple row CSV file format used in the Import file chooser
  - AnswerFile: This option is available for convenience. It makes it possible to migrate the answer data from a directory structure to a single file. See chapter IV.2.4 for details on the AnswerFile format.
- If the Conflation format is chosen the user has to select the field(s) to conflate over.
- If the export format is not AnswerFile, the user can also:
  - Scale the data for the export.
  - Filter the data and only export a subset of the data which conforms to the filter criteria

The export is started by clicking the export button. If the selected file already exists, the user needs to confirm that he wants to overwrite the existing file.

9.4.4.1.2 Scaling the data

By ticking the scale checkbox and specifying a desired minimum and maximum, the user can scale the data for the export. The scale function uses a linear function, which maps a defined minimum to the chosen minimum and a defined maximum to the chosen maximum. The following algorithm is used:

- Answers to question which have a defined allowed minimum and a defined allowed maximum: Here the defined minimum and the defined maximum are equal to the allowed minimum and the allowed maximum defined during the form definition process
- Answers to question with open scales. If either the allowed minimum or the allowed maximum is defined for this question this side is considered the defined minimum or maximum respectively. The other one (or both, if none exists in the question definition) are calculated dynamically upon user request. Should such question exist the program will display an option panel, so that the user can choose between:
    - Calculate the minimum or maximum dynamically. In this case it might happen that there are not enough data available for calculating a minimum or maximum or a minimum which differs from the maximum and vice versa. In this case another option panel will be displayed which enables the user to choose between:
        - Don’t scale the answers
        - Don’t export the answers
    - Don’t scale the answers to these questions
    - Don’t export these answers

  These options can be chosen only once and are valid during the whole export afterwards. So the user cannot choose not to export the answers of one question, to export non scaled data for another question and to calculate the existing minimums and maximums dynamically for a third question.

  However, there is a drawback in calculating the existing minimum and maximum dynamically, which is that it might be that the scaled data of exports done at different times cannot be compared to each other. This is because there might be answers added, changed or deleted in the meanwhile and this can lead to different calculated minimums and maximums at each export time.

9.4.4.2 Importing Data

The answer data can be imported by choosing the file format (file filter) and the file to import. The import utility knows the following file formats:

- CSV - single row
- CSV – multiple row

For details on the supported file formats please look at chapter IV.2. Should there be answers to a non-existing parameter source the parameter source is created automatically. The system checks for invalid answer values and displays an error message, if necessary.
Figure 9-9 shows an import dialog:

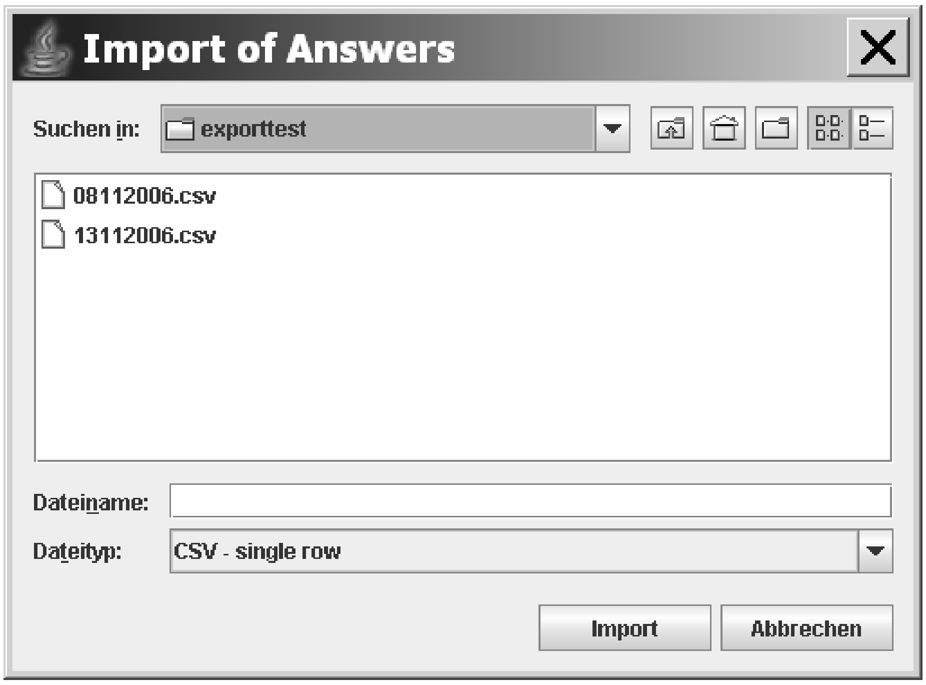
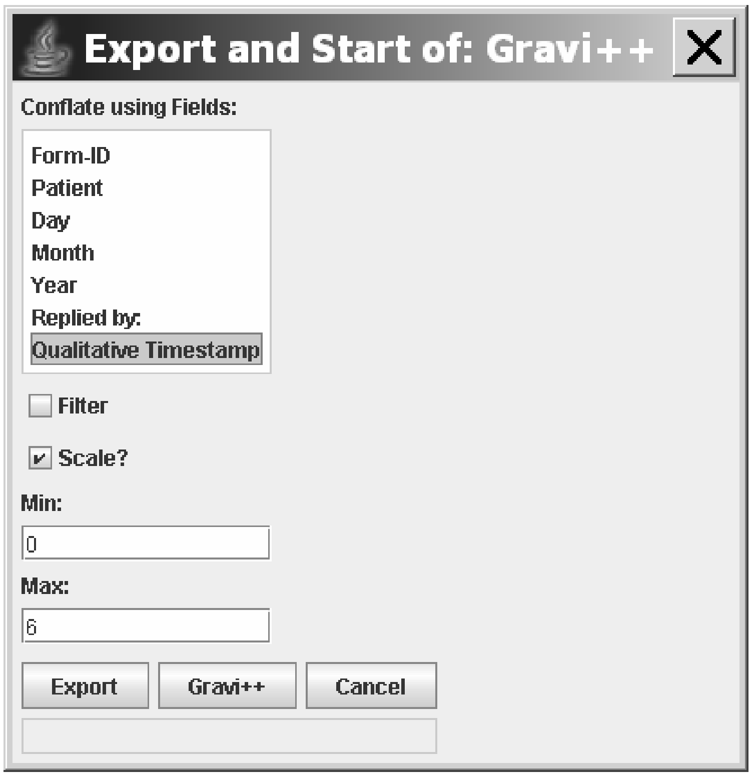


Figure 9-9 Import of Answers. We give an example of an import dialog.

9.4.4.3 Exporting and Automatically Start of an External Program

This option is only available, if the user specifies an external program and an external file in the profile (see section 9.1.2). It is pretty much the same than exporting the data to the file specified in the profile in the format specified in the profile. If the user does not specify the format in the profile it defaults to the multiple row CSV format. The name of the option is either determined by the parameter name to the external program in the profile, or, in case it is not specified, by the name of the called program without the path.
Therefore, this option does not require a file chooser. It uses the settings in the profile instead. The export as well as the external program will be started upon user request. Figure 9-10 is an example of a search/filter dialog.


Export and Start of: Gravi++

Conflate using Fields:

- Form-ID
- Patient
- Day
- Month
- Year
- Replied by:
- Qualitative Timestamp

☐ Filter

☑ Scale?

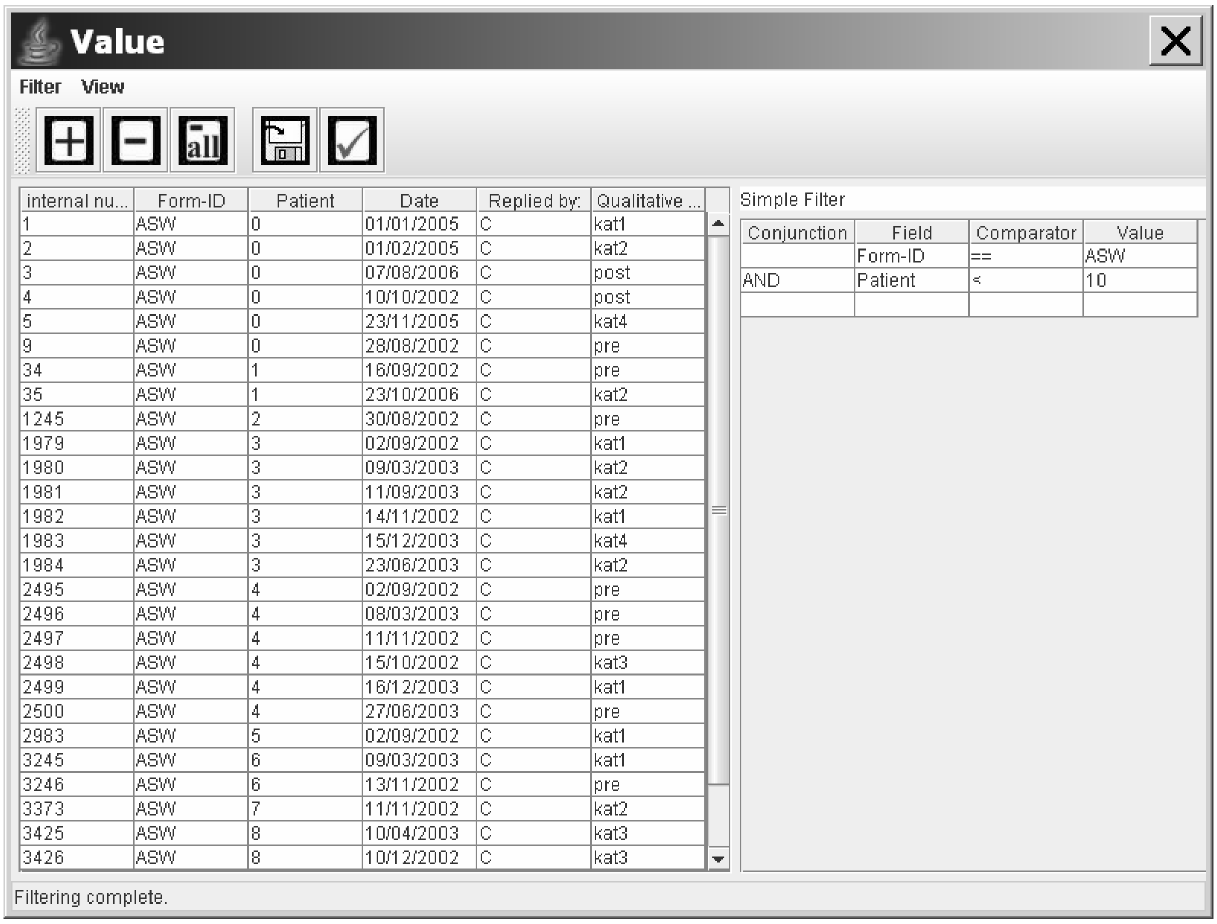
Min: 0

Max: 6

Figure 9-10 Export and Start of Gravi++ Dialog. We give an example of a dialog for exporting the data to the a file specified in the profile and start of an external program.

9.4.5 Searching / Filtering the Data

In order to be able to search for records or filter the data, the user can choose the search utility when recording data, as well as when exporting data, as well as when designing forms. There is only one search/filter utility which is configured by parameters to be capable to filter different tables and react slightly different when used for filtering than for searching.
Figure 9-11 is an example of a search/filter dialog:



internal nu...	Form-ID	Patient	Date	Replied by:	Qualitative ...
1	ASW	0	01/01/2005	C	kat1
2	ASW	0	01/02/2005	C	kat2
3	ASW	0	07/08/2006	C	post
4	ASW	0	10/10/2002	C	post
5	ASW	0	23/11/2005	C	kat4
9	ASW	0	28/08/2002	C	pre
34	ASW	1	16/09/2002	C	pre
35	ASW	1	23/10/2006	C	kat2
1245	ASW	2	30/08/2002	C	pre
1979	ASW	3	02/09/2002	C	kat1
1980	ASW	3	09/03/2003	C	kat2
1981	ASW	3	11/09/2003	C	kat2
1982	ASW	3	14/11/2002	C	kat1
1983	ASW	3	15/12/2003	C	kat4
1984	ASW	3	23/06/2003	C	kat2
2495	ASW	4	02/09/2002	C	pre
2496	ASW	4	08/03/2003	C	pre
2497	ASW	4	11/11/2002	C	pre
2498	ASW	4	15/10/2002	C	kat3
2499	ASW	4	16/12/2003	C	kat1
2500	ASW	4	27/06/2003	C	pre
2983	ASW	5	02/09/2002	C	kat1
3245	ASW	6	09/03/2003	C	kat1
3246	ASW	6	13/11/2002	C	pre
3373	ASW	7	11/11/2002	C	kat2
3425	ASW	8	10/04/2003	C	kat3
3426	ASW	8	10/12/2002	C	kat3

Conjunction	Field	Comparator	Value
	Form-ID	==	ASW
AND	Patient	<	10

Filtering complete.

Figure 9-11 Search and Filter Dialog. We give an example of a search and filter dialog.

The table displayed depends on the parameters passed to the search/filter utility. In our example it is an answer table. The following actions are possible:

- Filter
  - Add
  - Remove
  - Remove All
  - Save
  - Open
- View
  - Simple Filter
  - Advanced Filter

In the **simple filter view** the user merely needs to specify the filter criteria in the filter table on the right side. It is limited to up to three criteria. Choosing ***Filter/Add*** has no effect. ***Filter/Remove*** will remove the last criteria. ***Filter/Remove all*** will remove all filter criteria.

In the **advanced filter view** the user can specify as many filter criteria as he needs. A new filter criteria is added by choosing the ***add*** action. The marked filter criteria can be removed by choosing the ***remove*** action. All Filter criteria can be removed be choosing the ***Remove All*** action. In the advanced filter view each line of the filter table can contain only either a criteria or a conjunction. The advantage of the advanced filter criteria is that the user can specify any Boolean expression without the need of parenthesis, because the advanced filter uses postfix notation. Trailing conjunctions need not be specified as missing trailing conjunctions automatically default to AND.

Filtering is performed on the fly, each time there are changes to the filter criteria.

9.4.5.1 The Fields of the Filter Table

There are four fields in the filter table at the right side of the window.

- Conjunction: AND or OR can be selected from a list
- Field: Allows the selection of any of the head fields of the answer table or * for all fields. While there are some default head fields, the profile can define further fields.
- Comparator: This can be any of:
    - o == Exactly the same value, or all values, if Value equals *
    - o != Not the specified value, or none, if Value equals *
    - o ~~ The value matches the pattern specified in the Value field through a regular expression. (e.g., .* matches all)
    - o !~ Does not match the expression specified in the value field
    - o < Is smaller than the expression specified in the value field
    - o <= Is smaller than or equal to the expression specified in the value field
    - o > Is bigger than the expression specified in the value field
    - o >= Is bigger than or equal to the expression specified in the value field

9.4.5.2 Assigning Values to Head Fields

The user can assign values to all non key fields, which are not explicitly set read only. E.g., in the answer table of form group “questionnaire” the user can assign values to the field ***qualitative time*** in the answer table, which displayed at the left side.

9.4.5.3 Saving the table

The table can be saved by choosing the ***save*** action. If there are changes to the displayed table, the user will be asked, whether he wants to store the changes, whenever the window is closed or new data is going to be loaded.

9.4.5.4 Closing the Search/Filter Dialog

The filter dialog is closed, if the user selects the close action. If the purpose was to filter the data (which is true when called from, e.g., the export dialog), all the displayed data are available for the specified action (e.g. exporting). If the filter dialog was used to search for a record, the marked record will be displayed after closing the search/filter dialog. E.g., the selected form will be displayed in the design window or the selected answer set will be displayed in the record data window.

9.4.5.5 The Shortcuts and Icons for the Tasks

Table 9-5 lists the shortcuts and the icons of the search and filter dialog. META is short for the META key used by a specific platform. In the case of Windows XP this would be the CTRL key.

Task	Shortcut	Icon
Add	META-+	⊞
Remove	META--	⊟
Remove All	META-Del	all
Save	META-S	💾
Open	META-ENTER	☑

Table 9-5 Shortcuts of the Filter and Search Dialog. We list the actions, shortcuts, and icons of the search and filter dialog.

10 Future perspectives

10.1 User Management

Future releases could require a user management to be able to define different user types:

- Administrative users for designing forms
- Normal users for recording data

In the case of our study about anorectic girls it might then be possible to let the patient fill in the questionnaire directly instead of needing a paper version and having somebody else recording the parameters for the application.

10.2 Database Management System

In order to improve the performance of the program with huge amounts of data, it is worth considering moving its storage layer to a DBMS instead of a set of XML files in the future.

10.3 Multi User

If the application uses a DBMS instead of a set of XML files, if can be considered to allow multiple user at a time.

10.4 XML Document Validation

If future releases do not come with a DBMS support, it should at least be considered to use one of the various automatic methods of XML Document validation.

10.5 Profile Settings

A future release should consider a menu option for defining the application specific profile. This might ease the set up of the program for a specific application and avoids errors in the profile according to manually editing the profile.

11 Summary

In this thesis a generic editor was developed which is capable of designing forms and recording data for a variety of different applications. The first part of this thesis focuses on problem analysis and concepts. The second part covers design and implementation details. Design trade-offs and implementation issues are explained in details and UML diagrams are used in order to depict the structure and use of the developed Java classes. Finally, the third part demonstrates how to set up the generic editor for a specific application, namely our study of Anorexia nervosa.

The current release is in use at the Department of Child and Adolescent Neuropsychiatry at the Medical University of Vienna. This psychotherapeutic study analyses alternative therapeutic processes of anorectic girls by collecting a huge amount of highly structured time-oriented data through questionnaires. The generic editor could simplify the management of the questionnaires as well as the collection of the data necessary for this study and is able to invoke Gravi++ to make analyzing the data easier for the clinical personal.

IV. Appendix

1 Definitions

1.1 Anorexia nervosa

From [WWW-21] “Anorexia nervosa is a psychiatric diagnosis that describes an eating disorder characterized by low body weight and body image distortion. Individuals with anorexia often control body weight by voluntary starvation, purging, vomiting, excessive exercise, or other weight control measures, such as diet pills or diuretic drugs. It primarily affects young adolescent girls in the Western world and has one of the highest mortality rates of any psychiatric condition, with approximately 10% of people diagnosed with the condition eventually dying due to related factors. Anorexia nervosa is a complex condition, involving psychological, neurobiological, and sociological components.”

1.2 EER

From [WWW-1]: “In computer science, an entity-relationship model (ERM) is a model providing a high-level description of a conceptual data model. Data modeling provides a graphical notation for representing such data models in the form of entity-relationship diagrams (ERD). The first stage of information system design uses these models to describe information needs or the type of information that is to be stored in a database during the requirements analysis. The data modeling technique can be used to describe any ontology (i.e. an overview and classifications of used terms and their relationships) for a certain universe of discourse (i.e. area of interest). In the case of the design of an information system that is based on a database, the conceptual data model is, at a later stage (usually called logical design), mapped to a logical data model, such as the relational model…“

1.3 Parser

From [Harold 2002]: “A parser is a software library that knows how to read XML documents and handles all the markup it finds”.

1.4 Singleton

A singleton is an object oriented design pattern, which ensures that a class has only one instance, and a global point of access is provided for it. Details of the singleton pattern can be found in [Gamma 1995] pp. 127.

2 The File Formats

2.1 Conflation

The conflation file format is described in detail in [Herzog 2004]. It is a format which conflates multiple records to groups in order to be able to evaluate the data with Gravi++. This is an example of a conflation file: the field qualitative timestamp was used as conflation fields. So all records with the same qualitative timestamp and the same parameter source id and the same form id are grouped together:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<linkvis_conflation_data_set>
  <data_record parameter_source_id="0" parameter_source_flag="C"
till_day="01" from_day="01" from_month="01" till_month="01"
from_year="1970" till_year="1970" rowid="0 C kat1" label="kat1">
     <parameter group_id="B-IKS" day="15" month="12" year="2006" id="1"
value="1.5"/>
     </data_record>
  <data_record parameter_source_id="*" parameter_source_flag="C"
till_day="01" from_day="01" from_month="01" till_month="01"
from_year="1990" till_year="1990" rowid="* C kat3" label="kat3">
     <parameter group_id="B-IKS" day="15" month="12" year="2006" id="1"
value="3.0"/>
     </data_record>
</linkvis_conflation_data_set>
```

2.2 CSV (= CSV – multiple row)

The format has a row for each question of an answer set. Furthermore, it has a title line, which depends on the number of additional head fields as defined in the profile. For convenience there is also a question column when importing the data, in order to be able to import the data, which has been exported, without changing anything in the file. But when importing data the question column is ignored.
This is an example:

```
Form-ID;Patient;Day;Month;Year;Beantwortet von:;Qualitativer
Zeitpunkt;Question-ID;Question;Value
"BF-S";"0";"3";"9";"2002";"C";"kat2";"1";"aufgeschlossen";0.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"2";"guter Dinge";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"3";"antriebslos";0.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"4";"anfällig";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"5";"zielstrebig";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"6";"ernst";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"7";"einfallsarm";0.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"8";"empfindlich";3.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"9";"pessimistisch";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"10";"sorglos";3.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"11";"zerschlagen";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"12";"liebesfähig";3.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"13";"schuldig";0.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"14";"erschöpft";0.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"15";"lebensmüde";0.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"16";"gut";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"17";"fröhlich";3.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"18";"geliebt";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"19";"träge";3.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"20";"verschlossen";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"21";"lebendig";0.0
```

```
"BF-S";"0";"3";"9";"2002";"C";"kat2";"22";"temperamentvoll";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"23";"aufmerksam";0.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"24";"verzweifelt";0.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"25";"zufrieden";3.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"26";"ängstlich";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"27";"kraftvoll";6.0
"BF-S";"0";"3";"9";"2002";"C";"kat2";"28";"ausgeglichen";6.0
```

2.3 CSV – single row

This format has a column for each head field of an answer set followed by the answers to the questions of the according questionnaire. So there is only one row per answer set. It does not have a header line, because the number of questions and the question-ids vary, if the form changes.
This is an example:

```
1;ASW;31;10;2006;C;;1;2;3;4;1;2;3;4;1;2
1;ASW;31;10;2006;M;pre;4;2;3;4;1;2;3;4;1;2
1;ASW;30;10;2006;C;;5;2;3;4;1;2;3;4;1;2
1;ASW;29;10;2006;C;;0;2;3;4;1;2;3;4;1;2
1;ASW;28;10;2006;C;;0;;;4;1;2;3;4;1;2
1;ASW;27;10;2006;C;;test;;;4;1;2;3;4;1;2
```

2.4 A Single AnswerFile

This file format is similar to the directory structure format for answers. In contrary to the directory structure format the AnswerFile format keeps all the answers of to question groups in a single file – whilst the Directory structure format has a file for each set of answers. A set of answers are all answers to a form together with all the head values necessary to identify an answer set record non-ambiguously for a certain form group.
In order to keep all answer sets in a single file an additional root tag has been specified. This is an example of an answer file.

```
<genedit_data_records>
<linkvis_data_records parameter_group_id="TEST" parameter_source_id="*"
day="21" month="11" year="2006">
  <parameter id="1" value="2"/>
  <parameter id="2" value="3"/>
</linkvis_data_records>
<linkvis_data_records parameter_group_id="TEST" parameter_source_id="0"
day="21" month="11" year="2006">
  <parameter id="1" value="5"/>
  <parameter id="2" value="6"/>
</linkvis_data_records>
<linkvis_data_records parameter_group_id="TEST" parameter_source_id="11"
day="21" month="11" year="2006">
  <parameter id="1" value="-1"/>
  <parameter id="2" value="30"/>
</linkvis_data_records>
</genedit_data_records>
```

2.5 Multiple AnswerFiles in the AnswerDir subdirecotry

This is an example of a single record in a single file.

```
<linkvis_data_records parameter_group_id="TEST" parameter_source_id="*"
day="21" month="11" year="2006">
  <parameter id="1" value="2"/>
  <parameter id="2" value="3"/>
</linkvis_data_records>
```

2.6 XML File Format for Forms

This is an example of an XML file format for forms:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<linkvis_parameter_master_data>
<parameter_group count="2" description="" id="ASW">
<parameter data_type="integer" display_as="" id="1" max_value="4"
mean_value="" min_value="1" name="Wenn sich Widerstände auftun, finde ich
Mittel und Wege, mich durchzusetzen">
<allowed_values>
<allowed value="1"/>
<allowed value="2"/>
<allowed value="3"/>
<allowed value="4"/>
</allowed_values>
</parameter>
<parameter data_type="integer" display_as="" id="2" max_value="4"
mean_value="" min_value="1" name="Die Lösung schwieriger Probleme gelingt
mir immer, wenn ich mich darum bemühe">
<allowed_values>
<allowed value="1"/>
<allowed value="2"/>
<allowed value="3"/>
<allowed value="4"/>
</allowed_values>
</parameter>
</parameter_group>
<linkvis_parameter_master_data>
```

2.7 XML File Format for Parameter Sources

This is an example of an XML file format for parameter sources:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameter_Source_Master_File Pers_Dir_Source="C:\Dokumente und
Einstellungen\Martina\Eigene
Dateien\Diplom_Miksch\data_fuer_eingabetool2\data_records\">
  <Replier id="10"/>
  <Replier id="12"/>
</Parameter_Source_Master_File>
```

V. Tables of

1 Figures

Figure 5-1 EER of the Data Source 15
Figure 8-1 Package Structure 27
Figure 8-2 UML of the GenericEditor class 28
Figure 8-3 UML of the configuration package’s classes 30
Figure 8-4 UML of TDataAdmin 32
Figure 8-5 UML of TAData 34
Figure 8-6 UML of RecordData 35
Figure 8-7 UML of TableData 37
Figure 8-8 UML TableDataIterator 38
Figure 8-9 UML of FieldComparator 39
Figure 8-10 UML of TAddOnHeadField 40
Figure 8-11 UML of ValueTemplate 41
Figure 8-12 UML of value Package’s Classes 42

Figure 8-13 UML of the Storage Manager Classes 44
Figure 8-14 UML of XMLStorageManagerReplier 45
Figure 8-15 UML of DirectoryStorageManagerReplier 46
Figure 8-16 UML of DirectoryStorageManagerAnswers 47
Figure 8-17 UML of XMLStorageManagerAnswers 48
Figure 8-18 UML of the Importer. 49
Figure 8-19 UML of Exporter. 50
Figure 8-20 UML of GenEditPrintUtility 51
Figure 8-21 UML of XML_Helper 52
Figure 8-22 UML of TMainMenu 54
Figure 8-23 UML of SingleRecordDataWindow 57
Figure 8-24 UML of TDateSpinner 58
Figure 8-25 UML of the CustomListCellRenderer 58
Figure 8-26 UML of TDesignFormWindow 60
Figure 8-27 UML of TReplierAdmin 62
Figure 8-28 UML of FilterDialog 64
Figure 8-29 UML of CommonExportOption 66
Figure 8-30 UML of ExportDialog 67
Figure 8-31 UML of StartExternalDialog 68
Figure 8-32 UML of ExportChooser 69
Figure 8-33 UML of GenEditFileChooser 70
Figure 8-34 UML of ImportDialog 71
Figure 8-35 UML of the custom Event 72
Figure 8-36 UML of utils 73
Figure 9-1 Main Window 81
Figure 9-2 Selection of the Form Group. 82
Figure 9-3 Record Data Window 83
Figure 9-4 Selection of the Form Group for Designing Forms 86
Figure 9-5 Design Form Window 87
Figure 9-6 Parameter Source Maintenance Window 93
Figure 9-7 Selection of the Form Group for Export 95
Figure 9-8 Export Dialog 96
Figure 9-9 Import of Answers 98
Figure 9-10 Export and Start of Gravi++ Dialog 99
Figure 9-11 Search and Filter Dialog 100

2 Tables

Table 5-1 Form Group Entity 15
Table 5-2 Parameter Group Entity 16
Table 5-3 Parameter Entity 16
Table 5-4 Allowed Values Entity 17
Table 5-5 Additional Field Entity 17
Table 5-6 Allowed for Additional Entity 18
Table 5-7 Parameter Source Entity 18
Table 5-8 Measurement Entity 19
Table 7-1 Mutual Exclusiveness 23
Table 8-1 Package Structure 26
Table 8-2 ConfigInfo Methods 29
Table 8-3 Mutual Exclusiveness 33
Table 8-4 TableData Fields 36
Table 8-5 Common Methods of All Objects of the value Package 41

Table 8-6 Public Methods of AbstractStorageManager 43
Table 8-7 Methods for I/O Access 52
Table 8-8 Public Methods of TMainMenu 53
Table 8-9 Public Methods of SingleRecordDataWindow 56
Table 8-10 Public Methods of TDesignFormWindow 59
Table 8-11 Public Methods of TReplierAdmin 61
Table 8-12 Public Methods of the CommonExportOption 65
Table 8-13 Public Methods of the ExportChooser 68
Table 9-1 Tags and Parameters of the Profile 77
Table 9-2 Shortcuts in the Record Data Window 86
Table 9-3 Action of the Design Form Window 92
Table 9-4 Shortcuts of the Actions in the Parameter Source Maintenance Window 94
Table 9-5 Shortcuts of the Filter and Search Dialog 102

3 Codes and Example Files

Code 3-1 Example of Parameter Definition 12
Code 4-1 Example of Measurement Data 14
Code 8-1 Blocking Mouse Events 55
Code 9-1 Default Configuration File 79
Code 9-2 English Profile 80

VI. Bibliography

Literature

[Harold 2002] Elliote, Harold, Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX, Addison-Wesley Professional; 1st edition 2002

[Gamma 1995] Erich, Gamma et. al., Design Patterns: Elements of Reusable Object-Oriented software, Addison-Wesley, 1995, 32nd Printing April 2005

[Darwin 2005] Ian, Darwin, Java Kochbuch, O'Reilly Verlag GmbH & Co KG 2nd Edition 2005, 1st Edition 2002, German translation of Java Cookbook 2nd Edition, O'Reilly Media, Inc.

[Loy 2002] Marc, Loy et. al., Java Swing, O'Reilly Media, Inc. Sebastopol, 2nd Edition 2002, 1st Edition 1998.

[Dirk 2005] Louis, Dirk et. al., Java 5, Praxis der objektorientierten Programmierung (Kompendium), Markt und Technik, München/Germany, 2005

[Zukowski 1997] John, Zukowski, Java AWT Reference, O'Reilly and Associates, Inc., Sebastopol, 1st Edition 1997

[Herzog 2004] Herzog, Herbert, Multiple View Framework for Exploring Highly Structured Data, Master Thesis, Vienna University of Technology, Vienna, October 2004
retrieved at: http://ieg.ifs.tuwien.ac.at/projects/linkvis/linkvis-papers.html on 11.12.2006

[Hinum 2005] Klaus, Hinum et. al. Gravi++: Interactive Information Visualization of Highly Structured Temporal Data; Talk: Workshop IDAMAP 2005 Intelligent Data Analysis in Medicine and Pharmacology (IDAMAP 2005), Aberdeen, Schottland; 07-24-2005; in: Workshop IDAMAP 2005 Intelligent Data Analysis in Medicine and Pharmacology, (2005), 67 - 72
retrieved at: http://publik.tuwien.ac.at/files/pub-inf_2884.pdf on 11.12.2006

[Hinum 2006] Klaus, Hinum, Gravi++ - An Interactive Information Visualization for High Dimensional, Temporal Data., PhD thesis, Institute of Software Technology and Interactive Systems, Vienna University of Technology, Vienna, Austria, 2006

[Lanzenberger 2003 1] Monika, Lanzenberger, The Stardinates - Visualizing Highly Structured Data, in Proceedings of the Seventh International Conference on Information Visualization (IV'03), 2003

[Lanzenberger 2003 2] Monika, Lanzenberger, The Interactive Stardinates - An Information Visualization Technique Applied in a Multiple View System, PhD thesis, Institute of Software Technology and Interactive Systems, Vienna University of Technology, Vienna, Austria, 2003
retrieved at: http://www.ub.tuwien.ac.at/diss/AC03933379.pdf on: 12.12.2006

[Lanzenberger 2003 3] Monika, Lanzenberger, et. al., Exploring Highly Structured Data A Comparative Study of Stardinates and Parallel Coordinates, in Proceedings of the Ninth International Conference on Information Visualization, 2005

Internet Resources

Reference	URL	Last time retrieved: YYYY.MM.DD
[WWW-1]	http://en.wikipedia.org/wiki/Entity_relationship	2006.12.05
[WWW-2]	http://en.wikipedia.org/wiki/Database_system#Features_and_Abilities	2006.12.06
[WWW-3]	http://en.wikipedia.org/wiki/Table_(database)	2006.12.16
[WWW-4]	http://www.w3.org/TR/REC-xml/	2006.12.17
[WWW-5]	http://www.w3schools.com/xml/xml_whatis.asp	2006.12.17
[WWW-6]	http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/Swing-Tutorial-Printing.html	2006.07.16
[WWW-7]	http://java.sun.com/docs/books/tutorial/uiswing/components/example-1dot4/TableSorter.java	2006.05.03
[WWW-8]	http://javaalmanac.com/egs/javax.swing.filechooser/DoneEvent.html?l	2006.07.24
[WWW-9]	http://www.javaworld.com/javaworld/jw-09-1998/jw-09-techniques.html?page=1	2006.12.18
[WWW-10]	http://www.roseindia.net/search/showtutorials.php?id=6488	2006.12.19
[WWW-11]	https://tablelayout.dev.java.net/	2006.12.19
[WWW-12]	http://java.sun.com/docs/books/tutorial/uiswing/components/table.html	2006.12.19
[WWW-13]	http://www.exampledepot.com/egs/javax.swing.table/pkg.html	2006.12.19
[WWW-14]	http://www.esus.com/javaindex/j2se/jdk1.2/javaxswing/editableatomiccontrols/jtable/jtable.html	2006.12.19
[WWW-15]	http://lists.xcf.berkeley.edu/lists/advanced-java/2000-March/027249.html	2006.12.19
[WWW-16]	http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4206341	2006.12.19
[WWW-17]	http://forum.java.sun.com/thread.jspa?threadID=677570&messageID=3953435	2006.12.19
[WWW-18]	http://forum.java.sun.com/thread.jspa?threadID=343633&messageID=1417436	2006.12.19
[WWW-19]	http://www.javaworld.com/javaworld/jw-12-2000/jw-1229-traps.html	2006.12.19
[WWW-20]	http://java.sun.com/j2se/1.3/docs/api/java/lang/Process.html	2006.12.19
[WWW-21]	http://en.wikipedia.org/wiki/Anorexia_nervosa	2006.12.22
[WWW-22]	http://arizonacommunity.com/articles/java_32001.shtml	2007.01.13
[WWW-23]	http://www.deepveininsomnia.com/COMJavaAandD.php	2007.01.13
[WWW-24]	http://www2.gsu.edu/~matknk/java/reg97.htm	2007.01.13
[WWW-25]	http://www.eclipseuml.com/	2007.01.13
[WWW-26]	http://www.toadsoft.com/toaddm/toad_data_modeler.htm	2007.01.13
[WWW-27]	http://www.jguru.com/faq/view.jsp?EID=121069	2007.02.07