FACHHOCHSCHULE KÄRNTEN

Fachhochschulstudiengang
Netzwerk- und Kommunikationstechnik

B A C H E L O R A R B E I T

Java-TemplateGenerator und Jaspersoft aus einer XML-Perspektive

zur Erlangung des akademischen Grades
Bachelor of Science in Engineering (BSc)

Autor: Michael Höflmaier
Matrikelnummer: 1510286006

Erstbetreuer: FH-Prof. Dipl.-Ing. Dr. Helmut Wöllik

Zweitbetreuer: DI Christoph Uran, BSc

Tag der Abgabe:	01.03.2018
Name:	Michael Höflmaier
Matrikelnummer:	1510286006
Geburtsdatum:	20.11.1994
Adresse:	Aiglhofstraße 9, 5020 Salzburg

Eidesstattliche Erklärung

Mit dieser Erklärung versichere ich die erstmalig vorgelegte Bachelorarbeit selbstständig erstellt zu haben.
Es wurden nur die von mir angegebenen Hilfsmittel und Quellen verwendet.

Datum:	01.03.2018
Ort:	Salzburg

Unterschrift: 

Abstrakt

Diese Arbeit behandelt Jaspersoft Studio und das davon verwendete XML-Schema. Verständnis über deren Grundlagen zu erlangen, ist essentiell, um verstehen zu können, wie ein erfolgreiches Design eines JasperReports gelingen kann. Mit Jaspersoft Studio kann man JasperReports erstellen. Diese Berichte liegen in einem JRXML-Format vor und enthält logische und graphische Elemente. Diese Dateien werden durch Jaspersoft Studio interpretiert und können so jederzeit verändert werden. Der Inhalt einer jeden JRXML-Datei wird von einem XML-Schema abgeleitet. Ist man in der Lage dieses Schema korrekt zu lesen, ermöglicht dies, dass man eine JRXML-Datei und somit einen JasperReport in einem einfachen Text-Editor selbst entwerfen kann. Genau an diesem Punkt setzt ein, durch eine Projektgruppe geschriebenes Programm an.

Um die Verarbeitung dieses XML-Schemas durch das selbst erstellte Programm und Jaspersoft Studio zu verstehen muss Verinnerlicht sein, dass XML eine Auszeichnungssprache ist und einige sehr spezielle Eigenschaften besitzt. XML, dessen Grundidee bis in die sechziger Jahre des 20.Jahrhunderts zurückverfolgt werden kann, ist für die digitale Kommunikation im Geschäftsumfeld noch immer von höchster Relevanz. Um im späteren Verlauf ein XML-Schema verstehen zu können, müssen vordefinierte XML-Bausteinen verstanden werden, denn ein XML-Schema besteht ebenfalls aus denselben Bausteinen. Alle in einem XML-Schema definierten Module können dann in Jaspersoft Studio mittels einer graphischen Benutzeroberfläche erzeugt werden.

Zu guter Letzt wird ein JasperReport mit Hilfe von Jaspersoft Studio in einer Schritt für Schritt-Anleitung erklärt.

Abstract

This work discusses Jaspersoft Studio and its utilized XML-Scheme. To grasp its basics is essential to be able to design a working JasperReport. By the use of Jaspersoft Studio one can design JaspersoftReports with ease. Those report are composed of logical and graphic components and are saved within a JRXML-file. Mentioned files can be interpreted and altered by Jaspersoft Studio at all times. The content of a JRXML-file is derived from a XML-Scheme. When capable to apply this scheme, one can write an JRXML-file and therefore a JasperReport within a text-editor. Utilizing this fact, the project-group developed a so called ”TemplateGenerator”.

To comprehend how the generator and Jaspersoft Studio handle the XML-scheme, one must know that XML is a markup language and has multiple special properties. XML origined in the sixties of the past century and still has a big foothold within IT. To understand a XML-scheme in a later stage, basic XML-modules should be understood, because a XML-scheme also consists of this components. All elements defined within an XML-scheme can be used within Jaspersoft Studios graphical user interface.

In the end a step-by-step instruction is given, in which a report is rendered with the use of Jaspersoft Studio.

Suchbegriffe

XML, Auszeichnungssprache, XML-Schema, JasperReport, Jaspersoft Studio, FH Timing

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

FH	Fachhochschule
HTTP	Hypertext Transfer Protokoll
XML	Extensible Markup Language
PDF	Portable Document Format
W3C	World Wide Web Consortium
XSL	Extensible Stylesheet Language
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SVG	Scalable Vector Graphic
XHTML	Extensible Hypertext Markup Language
XSLT	XSL Transformation
UTF	Universal Character Set Transformation Format
DTD	Document Typ Definition
CDATA	Character Data
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
JRXML	Jaspersoft Report Extensible Markup Language
SQL	Structured Query Language

1. Problemstellung

Vorab einige, zum Verständnis nützliche Zusatzinformation an den Leser dieser Bachelorarbeit. Der Projektauftraggeber Herr FH-Prof. Dipl.-Ing. Dr. Helmut Wöllik ist neben seiner Rolle als Lehrkörper an der FH Kärnten auch in der Forschungs- und Entwicklungsarbeit an der Fachhochschule engagiert. Darüber hinaus ist er auch die treibende Kraft hinter der vom Studiengang Netzwerk- und Kommunikationstechnik ins Leben gerufenen Sparte, die sich sehr intensiv mit Zeitnehmungstätigkeiten auf Sportveranstaltungen diverser Größen beschäftigt. Diese Abteilung, in Folge nun „FH Timing“ genannt, wird durch den Projektbetreuer und einem wissenschaftlichen Mitarbeiter geleitet. Der Anspruch von FH Timing ist es Studierende, über ihre Involvierung im Regelunterricht des Studiengangs hinaus, mit Netzwerk- und Kommunikationstechnik in Berührung zu bringen, sowie die Möglichkeit zu geben sich zusätzliches Wissen anzueignen, in Kontakt mit diversen Fachbereichen zu bringen und vieles mehr. Des Weiteren bietet FH Timing der Fachhochschule die Chance eine zusätzliche Präsentationfläche für Werbezwecke zu bieten und damit die Reputation der Fachhochschule zu steigern.

Der erste Kontakt mit der Projektarbeit wurde bereits im Oktober 2017 hergestellt. Zu diesem Zeitpunkt kam der Projektbetreuer auf das Projektteam zu und umriss für uns die Problemstellung die sich wie folgend darstellte. FH Timing erstellt während beziehungsweise nach Veranstaltungen für die Veranstalter und Teilnehmer dieser Events verschiedenste Dokumente. Einige Beispiele für solche Dokumente könnten Teilnahmeurkunden oder Zeitnehmungs- beziehungsweise Rangtabellen sein, die einen Überblick über verschiedene Parameter geben, wie zum Beispiel: Name, Rang, gemessene Zeit, Alter, Geschlecht, Teilnehmerklasse oder Vergleichbares. FH Timing verwendet um solche Dokumente zu erstellen unter anderem die Software Jaspersoft Studio beziehungsweise Jaspersoft Server der Firma TIBCO. Diese Software verknüpft Vorlagen mit Daten, die in einer Veranstaltung erhoben wurden. Diese Dokumente können dynamisch erstellt und in verschiedenen Dateiformaten ausgeben werden. Zuvor genannte Vorlagen werden in weiterer Folge als „Templates“ bezeichnet.

Diese Templates können nun mit Hilfe von Jaspersoft Studio erstellt werden. Da Jaspersoft Studio ein sehr mächtiges Programm ist geht damit auch eine entsprechende Komplexität einher. Diese Komplexität war der grundsätzliche Anstoß für die Entstehung unseres Projekts. Da die Einarbeitungszeit in das Programm einige Zeit in Anspruch nimmt und die Templates nicht jedes Mal von ein und derselben Person erstellt werden, ist damit eine zeitliche Ressourcenbindung verbunden der auch unter wirtschaftlichen Gesichtspunkten nicht vertretbar ist. Daher wurde in der Vergangenheit ein Konzept angewandt bei dem mehrere Grundtemplates erstellt wurden auf denen später aufgebaut wurde. Dies hatte zur Folge, dass bei jeder Erstellung eines neuen Template das Grundtemplate als Ausgangsbasis verwendet wurde. Dadurch wurde bereits bestehender Code aus dem Grundtemplate in die neuen Templates übernommen, wobei dieser in den Neuen nicht erforderlich wäre. Auch die nachträgliche Anpassung ging hierbei nicht immer ohne einen erheblichen Arbeitsaufwand vonstatten. Daher kam der Projektbetreuer mit der Idee eines Template-Generators auf uns zu und stellte uns diesen vor. Dieser sollte dynamisch, den Nutzerwünschen entsprechend, Templates erzeugen können. Darüber hinaus sollte sich das Projektteam in Jaspersoft Studio einarbeiten, um auch die dadurch gewonnenen Erkenntnisse in

unseren Bachelorarbeiten aufzuarbeiten und zu analysieren. Auch sollte hiermit das Verständnis für die Struktur beziehungsweise das Dateiformat, in der Jaspersoft Studio die Templates speichert, gestärkt werden. Dadurch sollte man auch Synergien besser nutzen können und sich gewonnene Erkenntnisse bei der Programmentwicklung zu Nutze machen.

1.1. Ziel und Zweck der Arbeit:

Aus dieser praktischen Problemstellung ergaben sich dadurch auch etliche Bereiche die man näher beleuchten sollte um ein besseres Verständnis für die Materie zu bekommen. Da, wie bereits angedeutet, Jaspersoft Templates in einer eigenen Struktur speichert, liegt in dieser Bachelorarbeit der Fokus auf dem Verständnis dieses Formats. Auch das zugrundeliegende Schema wird genauer beleuchtet und analysiert. Später wird auch noch ein durch den Template-Generator erstelltes Template genauer nach diesen Gesichtspunkten untersucht.

1.2. Erste Lösungsansätze

Noch vor Annahme der Aufgabenstellung wurden in der Projektgruppe mögliche Herangehensweisen diskutiert und auf Vor- beziehungsweise Nachteile hin überprüft. Dabei stellte sich heraus, dass man bei diesem Projekt eine programmiertechnische Herangehensweise wählen sollte, die sich auf inkrementelle Prototypen stützt, da dieses Projekt damit den größtmöglichen Wissenszugewinn generieren würde. Als Programmiersprache wurde Java ausgewählt, da diese sehr vielseitig einsetzbar ist, umfangreiche Bibliotheken bietet und Grundlagen dieser Sprache bereits bekannt waren. Der Nutzer sollte seine Befehle an das Programm entweder mittels einer Konsole oder in einer Konfigurationsdatei niederschreiben und so dem Programm übermitteln. Diese Richtlinien für diese Befehle sollten natürlich während des Projektfortschritts laufend mitdokumentiert, in einer Datei genauestens erklärt und alle möglichen Befehlsparameter aufgezählt werden. In der Schlussphase gab es die Anforderung das Projekt dann auch noch auf mögliche Programmfehler hin getestet werden. Parallel zu Fortschritten der Programmierarbeit sollte auch noch die Funktionalität von Jaspersoft Studio erkundet und dokumentiert werden. Die offizielle Dokumentation sollte als Leitfaden durch die möglichen Programmanwendungszwecke führen und auch somit die Entwicklung des Programms beschleunigen. Das von Jaspersoft Studio verwendete XSD-Schema sollte bei der Programmierung oder bei Fragen zum verwendeten Schema herangezogen werden, um Redundanzen aufzudecken und diese dadurch nicht in den Programmcode zu übernehmen.

2. Motivation und Entscheidung für das Projekt

Die maßgebliche Entscheidung für die Durchführung des Projekts durch die Projektgruppe wurde dadurch angetrieben, dass am Ende seiner Fertigstellung das Projekt auch eine Relevanz für den Projektauftraggeber beziehungsweise das fertiggestellte Produkt sollte. Da dies vor allem durch die vom Projektbetreuer sehr gut ausgedrückte Notwendigkeit dieses Projekts auch für die Projektgruppenmitglieder ersichtlich wurde stellt dies einen großen Motivationsfaktor dar. Auch der Anspruch etwas Neues zu erlernen ist mit diesem Projekt vollends abgedeckt. Wie in Abschnitt 1.2 bereits beschrieben sollte das Projekt auch einige programmiertechnische Herausforderungen in der Programmiersprache Java bereithalten, was schlussendlich auch dazu dient sich noch besser und intensiver mit Programmierung auseinander zu setzen. Zusätzlich hatte ein Projektgruppenmitglied auch schon bereits Erfahrung mit der Verwendung von Java im Zusammenhang mit Projekten. Dies stellte auch noch zusätzlich sicher, dass die Projektgruppe auch einen internen und versierten Ansprechpartner hatte. Die in Aussicht gestellte zusätzlich zu gewinnende Erfahrung im Bereich XML stellte auch noch einen großen Anteil an zusätzlicher Motivation, da diese Materie in den bisherigen Vorlesungen nicht immer in einem für die Projektgruppenmitglieder ausreichenden Umfang behandelt wurde. Da aus dieser Projektarbeit nicht nur ein fertiges Programm entstehen sollte, sondern auch pro Gruppenmitglied eine Bachelorarbeit 1, war dies natürlich auch obendrein noch ein Argument für dieses umfangreiche Projekt. Da diese Anforderung durch das Projekt gestillt werden könnte und Stoff für mehrere Bachelorarbeiten bot, stellte dies auch kein Hindernis für eine Annahme des Projekts dar. Andere Projektangebote wurden, teils auch wegen bereits geschehener Vorbereitungsarbeit, aber vor allem wegen den bereits oben angeführten Argumenten nicht mehr der Vorzug gegenüber diesem Projekt gegeben. Durch die schnelle Entscheidung für dieses bestimmte Projekt konnte die eigentliche praktische Arbeit sehr schnell aufgenommen werden und frühzeitig auch die erste Recherchephase anlaufen.

3. Theoretischer Teil und Grundlagen

In diesem Kapitel werden alle Grundlagen, die man für das Verständnis der in Kapitel 4 behandelten Ausarbeitungen benötigt, behandelt. Dies soll als Wissensgrundstock für ebendiese praktische Bearbeitung der Themen dienen. Das Ziel dieses Teil der Bachelorarbeit ist es, einen einfachen Überblick auf die behandelten Themen zu gewähren, aber auch bei interessanten Detailfragen für den Leser relevante Informationen bereitzuhalten und diese verständlich zu präsentieren.

3.1. XML

Dieses Kapitel widmet sich Voll und Ganz der Auszeichnungssprache XML. Das Ziel in diesem Kapitel ist es, dem Leser möglichst viele der im diesem Zusammenhang oft gestellten Fragen zu beantworten und für den Leser dadurch einen Mehrgewinn zu erzielen. Zusätzlich zu diesen Fragen liegt der Fokus dieser Arbeit Dokumentmodelle betreffend auf XML-Schema. Das Thema DTD (Dokumenttyp-Definition) wird nicht genauer behandelt.

3.1.1. Was ist XML?

Die erste und gleichzeitig offensichtlichste Frage in diesem Zusammenhang lautet, was XML grundsätzlich ist.

Zuerst einmal muss man die Abbreviatur auflösen um auf den wahren Kerninhalt dieser Abkürzung zu stoßen. Dabei folgt die Abkürzung nicht der in der Informationstechnologie üblichen Akronym-Konvention, denn der erste Buchstabe lässt sich zu dem englischen Wort „Extensible“ auflösen. Die weiteren Buchstaben M und L beschreiben die Ausdrücke „Markup“ beziehungsweise „Language“. Aus dieser „Extensible Markup Language“ lässt sich nun auch eine deutsche Übersetzung konstruieren. Daraus ergibt sich der Ausdruck „Erweiterbare Auszeichnungssprache“. [1]

Zu einem ersten Grundverständnis hierzu muss zuerst auf das Wort „Auszeichnungssprache“ eingegangen werden. Dabei lässt sich der Begriff sehr gut beschreiben, indem man ein einfaches Beispiel zur Hand nimmt. Hierzu ist es am besten den berühmtesten Vertreter aller Auszeichnungssprachen HTML anzuführen (Siehe Abbildung 1). [2]

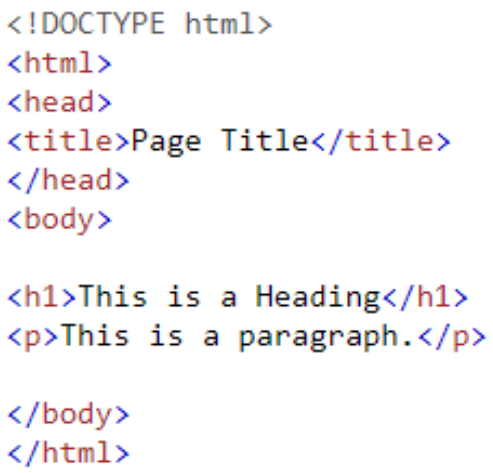


Abbildung 1: Beispiel einer Auszeichnungssprache

Eine Auszeichnungssprache, wie HTML oder XML, drückt sich vor allem dadurch aus, dass ein maschinenlesbares Format ist, welches es einfach macht Anmerkungen beziehungsweise Anweisungen vom textuellen Inhalt zu unterscheiden, da diese in sogenannten „Tags“ formuliert sind. Diese werden später, in Abschnitt 3.1.1, genauer behandelt. Auch der Bedeutungsinhalt des Wortes „Extensible“ wird in Abschnitt 3.1.3 genauer untersucht. [2]

3.1.2. Warum ist XML wichtig?

Nun zu der Frage warum XML wichtig sei. Hierzu sollte man zuerst einigen Anwendungsbereichen Beachtung schenken.

Hier ist zuerst einmal TEI anzuführen, wobei TEI für „Text Encoding Initative“ steht und dazu dient Richtlinien und Standards aufzustellen die helfen Texte in digitaler Form darzustellen. Als zweites Beispiel bietet sich MathML an, dass sich einer standardisierten Darstellung von mathematischen und chemischen Formeln widmet. Auch die Anwendungsbereiche Finanztransaktionen und E-Business gründeten eigene Standards die auf XML fußen und auf Namen wie IFX (Interactive Financial EXchange) oder UBL (Universal Business Language) hören. Auch nach der Veröffentlichung des HTML5-Standards ist die Verbreitung von SVG, welches dazu dient Vektorgrafiken in XML-Code zu transportieren und darzustellen, ungebrochen. Auch zum Datenaustausch zwischen verschiedenen Programmen beziehungsweise Anwendungen ist XML voll in der heutigen Informationstechnologielandschaft integriert. Zum Beispiel stützt sich auch Adobe Acrobat Reader, eine Anwendung zur Bearbeitung und Darstellung von PDF-Dateien, auf XML um auf diese Weise zu ermöglichen, dass die Inhalte der PDF-Datei in anderen Anwendungen auch verwendet werden können. Eine der größten und wichtigsten Sparten von XML ist aber SOAP (Simple Object Notation Protocol), welche den Versuch darstellt einen unkomplizierten Rahmen für den Nachrichtenaustausch zwischen Anwendungen aufzustellen. Die Art des Nachrichtentransports wurde dabei hier extra nicht eng definiert, dennoch wird meist das Standardprotokoll des Internets verwendet. Genauer gesagt HTTP. [1]

All diese Beispiele sind bloß ein Kratzen an der Oberfläche der unterschiedlichen Verwendungsarten von XML, dennoch sollte dies genügen um die Frage zu beantworten warum XML wichtig ist.

3.1.3. Entstehungs- und Entwicklungsgeschichte

In diesem Absatz soll nur kurz auf die Entstehungs- und Entwicklungsgeschichte von XML eingegangen werden. Dies geschieht hier jedoch nur der Vollständigkeit halber und nicht um einen tiefen, detaillierten Einblick in die Geschichte von XML zu geben.

Die Ursprünge von XML liegen sowohl in HTML als auch in SGML. [1] Die Idee für eine Sprache zur Beschreibung von Dokumentenstrukturen wurde bereits in den späten sechziger Jahren des 20.Jahrhunderts geboren, aber erst im Jahre 1986 in einem internationalen Standard aufgenommen. [3]

Dieser Standard, unter dem Namen SGML (Standard Generalized Markup Language) bekannt, war allerdings bereits immer für einen effektiven Einsatz im Alltag ungeeignet. Eine zu SGML verwandte, weil abgeleitete Sprache, HTML (Hypertext Markup Language) genannt, war aber zu inflexibel. Dies kam vor allem davon, da diese Sprache eine sehr eingeschränkte Anzahl von Elementtypen besitzt, welche eine weitere Entwicklung in diese Richtung als sehr schwierig gestaltete. All diese Gründe waren entscheidend für den Beginn der Arbeiten an einer eigenständigen Sprache. [1]

Damals wurden durch die XML-Arbeitsgruppe des W3C verschiedenen Ziele formuliert.

„1. XML soll sich im Internet auf einfache Weise nutzen lassen. 2. XML soll ein breites Spektrum von Anwendungen unterstützen. 3. XML soll zu SGML kompatibel sein. 4. Es soll einfach sein, Programme zu schreiben, die XML-Dokumente verarbeiten. 5. Die Zahl optionaler Merkmale in XML soll minimal sein, idealerweise Null. 6. XML-Dokumente sollten für Menschen lesbar und angemessen verständlich sein. 7. Der XML-Entwurf sollte zügig abgefasst sein.
8. Der Entwurf von XML soll formal und präzise sein. 9. XML-Dokumente sollen leicht zu erstellen sein. 10. Knappheit von XML-Markup ist von minimaler Bedeutung.“ [4]

Diese Entwicklungen gipfelten im Februar 1998 in der Spezifikation von „XML 1.0“ durch das W3C. Ein Jahr später folgte die Erweiterung „XML Namespaces“ und im Mai 2001 eine Sprache zur Definition von Inhaltsmodellen „XML Schema“ genannt. 2004 erschien eine Version 1.1 von XML, jedoch findet diese in der praktischen Alltagsarbeit nicht oft Anwendung, weil sie zur weiter verbreiteten XML-Version 1.0 inkompatibel ist. Die Version 1.1 unterscheidet sich darüber hinaus auch nur geringfügig von der ursprünglichen Version 1.0. Die Version 1.0 wurde seit ihrer Veröffentlichung vor allem aufgrund von Fehlerkorrekturen öfters überarbeitet. Daher liegt XML 1.0 bereits in einer fünften Edition vor. In der Folge wurden dann auch noch einige weitere Standards ergänzt. Darunter XSL zur Formatierung von XML-Dokumenten, XPath, das eine Adressierung von einzelnen Elementen in XML-

Dokumenten ermöglicht und XSLT, eine Sprache die dazu dient XML-Dokumente in ein anderes Format oder ein anderes XML-Dokument umzuwandeln. Über die Zeit entstanden und etablierten sich auch verschiedene Programmierschnittstellen im Zusammenhang mit XML. Einerseits sind DOM (Document Object Model), SAX (Simple API for XML), JAXP (Java API for XML Processing) für die Java-Welt und auch MSXML (Microsoft XML Core Services) für Microsofts .NET-Framework vorhanden. Wie bereits in Abschnitt 3.1.2 ausgeführt, entstanden über die Jahre verschiedenste XML-Anwendungen, welche auf die Namen SOAP, SVG oder XHTML hören. [1]

3.1.1. Was ist ein wohlgeformtes XML-Dokument?

Diese Frage lässt sich grundsätzlich relativ einfach beantworten, jedoch ist es nicht einfach bei dieser Erklärung nicht in die genauere Spezifikation von XML 1.0 abzudriften. Generell gesprochen ist jedes Datenobjekt ein XML-Dokument, wenn es im Sinn der Spezifikation wohlgeformt ist. Darüber hinaus heißt es:

„Ein wohlgeformtes XML-Dokument kann darüber hinaus gültig sein, sofern es bestimmten weiteren Einschränkungen genügt.“ [4]

Diese Erkenntnisse sagen uns zwar was ein wohlgeformtes XML-Dokument ist, aber die Frage wann ein XML-Dokument wohlgeformt ist, wird erst durch ein weiteres einlesen in die Spezifikation beantwortet. Denn die W3C-Spezifiaktion von XML 1.0 umfasst 89 sogenannte Regeln, Produktionen genannt, nach denen ein Datenobjekt geprüft wird. Diese Regeln wurden in einer einfachen Extended-Backus-Naur-Form festgeschrieben und umfassen neben den eben erwähnten Regeln noch eine Vielzahl von Einschränkungen. Es gibt hierbei zwei Arten von Einschränkungen. Einerseits gibt es Einschränkungen die mit dem Ausdruck WFC (well-formedness constraint) auftreten und wie der Name selbst sagt, beachtet werden müssen damit ein XML-Parser das Datenobjekt als wohlgeformtes XML-Dokument akzeptiert. Andererseits gibt es noch Einschränkungen die VC (validity constraint) genannt, die zusätzlich über Gültigkeit des Dokuments entscheiden. [4] [1]

3.1.2. Welche Bausteine besitzt ein XML-Dokument?

In diesem Kapitel wird auf den Erkenntnissen von Abschnitt 3.1.1 aufgebaut. Des Weiteren werden einige wichtige Produktionen dargestellt und erklärt, sowie verschiedene Konventionen, Bausteine und Regeln beleuchtet. All dies soll dem Leser wieder helfen sich in XML zurechtzufinden und verschiedene Strukturen interpretieren zu können.

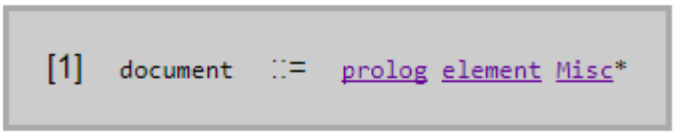


Abbildung 2: Definition eines Dokuments

Diese erste Produktionsregel widmet sich dem möglichen Inhalt eines Dokuments. Es beschreibt das ein Dokument einen Prolog, Elemente und verschiedene Strukturen, wie zum Beispiel Kommentare, Verarbeitungsanweisungen und Leerräume haben darf. Diese Produktion impliziert, dass ein Dokument ein oder mehrere Elemente enthält. Gerade vorgestellte Strukturen werden nun nacheinander vorgestellt. [4]

Ein Beispiel für einen Prolog wäre folgende Zeichenkette:

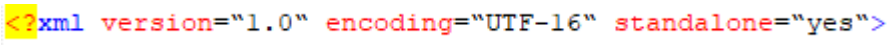


Abbildung 3: Beispiel für einen Prolog

Diese ist nicht zwingend, aber sollte verwendet werden um das Dokument sofort als XML-Dokument identifizieren zu können. Die Schlüsselwörter „Encoding“ und „Standalone“ sind optional und daher nicht der Minimalanforderung entsprechend. Erstes steht für die Art der verwendeten Zeichensatzcodierung. In den meisten Fällen würde dies UTF-16 sein, wobei UTF-8 als Standardwert verwendet wird. Zweites deutet darauf hin, dass keine externen Markup-Deklarationen, wie eine DTD oder ein XML-Schema für die Verarbeitung vonnöten ist. Im nächsten Schritt wird vor allem auf Elemente eingegangen und was bei diesen zu beachten ist. [4]

Abbildung 4: Definition eines Elements

Für die Beschreibung eines Elements ist es vorab wichtig über die Struktur eines XML-Dokuments zu sprechen. Da nach dem Prolog typischerweise meist Elemente und Attribute folgen, die in Form eines Baums dargestellt werden können. Es gibt immer ein Wurzelelement, dass alle anderen Elemente in sich einschließt. Elemente auf der darunterliegenden Schicht werden, wie in der Informatik üblich, als Kindelemente bezeichnet, während die Elemente der darüberliegenden Schicht als Elternelemente beziehungsweise, wenn es sich um Elemente der zweiten Schicht handelt, als Wurzelelemente bezeichnet werden. Diese Hierarchie ermöglicht eine

beliebig tiefe Verschachtelung von Elementen. Aus dieser Baumstruktur lässt sich nun durch die Verwendung von Knoten und Kanten ein Gebilde erzeugen, dass keine Zyklen aufweisen darf. Denn die sequenzielle Abfolge von Elementen in einem XML-Dokument ist durch eine Dokumentreihenfolge implizit vorgegeben. [4]

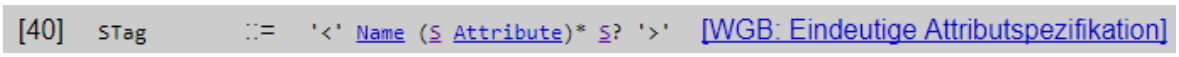


Abbildung 5:Definition des Starttags

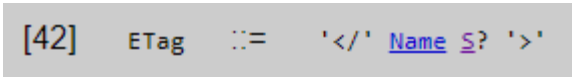


Abbildung 6: Definition des Endtags

Die in Abbildung 5 und Abbildung 6 abgebildeten Regeln für die Erstellung von Start- beziehungsweise Endtags sind bei Elementen von integraler Bedeutung. Allem voran, weil in diesen Tags Metainformation, also Information über den Inhalt des Elements, enthalten ist. Noch wichtiger ist aber, dass die Metainformation auf diese Art und Weise von der Inhaltsinformation getrennt werden kann. Dies hilft Menschen in der Navigation durch die Elementhierarchie eines XML-Dokuments. Bei der Namensgebung von Elementen gibt es verschiedene Dinge zu beachten. Grundsätzlich sollte der Namen eines Elements eindeutig und verständlich sein. Zusätzlich sollte einer Namensgebungskonvention, zum Beispiel camel-casing, gefolgt werden und der VC beachtet werden, der besagt, dass Elementnamen sowohl im öffnenden Tag als auch im schließenden Tag exakt gleich sein müssen. Ein Element übernimmt oftmals auch nur die Funktion eines Containers, innerhalb dessen dann wieder Elemente Platz finden können die mehr Informationen in sich tragen als der sie umgebende Container. Die Kindelemente und deren Reihenfolge, die ein solcher Container beziehungsweise jedes beliebige Element akzeptiert, können über eine DTD oder ein XML-Schema definiert werden. Der nächste Schritt widmet sich vor allem Attributen. [4]

Abbildung 7: Definition eines Attributes

Attribute sollten ausschließlich dazu verwendet werden Zusatzinformation innerhalb eines Elements zu verpacken und mit diesem in einem Zusammenhang zu stehen. Jedoch ist die Entscheidung nicht immer klar, ob eine Information als Attribut oder als in einem Container gepacktes Element geliefert werden sollte. [4]

```
<name>
    <vorname>Max</vorname>
    <nachname>Musterman</nachname>
</name>
```

Abbildung 8: Zusatzinformation in Form von containerisierten Elementen

```
<name vorname="Max" nachname="Mustermann"></name>
```

Abbildung 9: Zusatzinformation in Form von Attributen

Wie in Abbildung 8 und Abbildung 9 ersichtlich sind beide Varianten möglich und auch gültig. Die Attribut-Variante hat jedoch den Nachteil, dass hierbei

Programmierschnittstellen wie Document Object Model es schwieriger haben, auf Attribute zuzugreifen. Die Stärke von Attributen jedoch ist, dass bei diesen die Reihenfolge in der sie in einem Element vorkommen nicht ausschlaggebend und von Relevanz ist. Überdies gibt es auch noch sogenannte „reservierte Attribute“. Diese charakterisieren sich dadurch, dass sie bereits in der XML-Spezifikation vorkommen und deshalb in jedem XML-Dokument standardmäßig verwendet werden können. Beispiele hierfür wären die Attribute zur Sprachidentifikation und für die Leerraumbehandlung. Erstes könnte sich wie folgend in einem Dokument darstellen:

```
<name xml:lang="en">John Doe</name>
<name xml:lang="de">Max Mustermann</name>
```

Abbildung 10: Beispielhafte Verwendung eines Sprachidentifikationsattributs

Diese Sprachidentifikation dient meist dazu, spezifische und durch die Sprache ausgewählte Inhalte mittels eines Stylesheets auszugeben. Sprach- beziehungsweise Ländercodes sind hierbei nach ISO 639 genormt. Zweites, die Leerraumbehandlung, lässt sich durch folgende Abbildung darstellen:

```
<text xml:space="preserve">
In diesem Text bleiben auch diese L e e r z e i c h e n erhalten.
</text>
```

Abbildung 11:Beispielhafte Verwendung eines Leerraumbehandlungsattributs

Das Schlüsselwort „preserve“ in Abbildung 11 stellt beim XML-Prozessor den Wunsch für dieses Element dar, dass alle Leerzeichen erhalten bleiben. Der Standardwert hierbei ist das Schlüsselwort „default“, welches der verarbeiteten Anwendung freistellt wie die Leerzeichen zu verarbeiten sind. Als nächstes sollten CDATA-Abschnitte noch genauer beleuchtet werden, da sich diese oftmals großer Verbreitung erfreuen. Hierzu zuerst einmal die Produktionsregeln für CDATA-Abschnitte aus der XML 1.0-Spezifikation. [4]

```
[18]   CDSect    ::=   CDStart CData CDEnd
[19]   CDStart   ::=   '<![CDATA['
[20]   CData     ::=   (Char* - (Char* ']]>' Char*))
[21]   CDEnd     ::=   ']]>'
```

Abbildung 12: Produktionsregeln für CDATA-Abschnitte

CDATA- Abschnitte werden vor allem dazu benutzt größere Blöcke von reservierten Markup-Zeichen, welche sonst pro Zeichen mittels einer Zeichen- beziehungsweise Entitätsreferenz angegeben werden müssten, effizient zu verwenden. Die Verwendung von CDATA-Abschnitten geht dadurch oft mit einer großen Zeitersparung einher. Oftmals werden innerhalb eines solchen Blocks eigene XML-Dokumente oder HTML-Seiten beschrieben. Aber auch die Platzierung von Programmcodes innerhalb dieser Abschnitte ist erlaubt und dadurch möglich. [4]

```
<![CDATA[<gruss>Hallo Welt!</gruss>]]>
```

Abbildung 13: Beispielhafte Verwendung von CDATA

Bei genauerer Betrachtung der in Abbildung 12 angeführten Produktionsregeln lässt sich nun auch folgendes Beispiel aus der W3C-Spezifikation verstehen. Denn, wie in Abbildung 13 zu sehen, ist es möglich ein mit XML konformes Element innerhalb dieses CDATA-Abschnitte zu platzieren ohne einen Fehler durch einen XML-Parser zu erhalten.

Als vorletzter Punkt werden Kommentare unter XML vorgestellt. Kommentare verhalten sich unter XML sehr ähnlich zu HTML-Kommentaren. Jedoch gibt es die Einschränkung, dass Kommentare nicht innerhalb eines Tags beziehungsweise einer Deklaration verwendet werden dürfen. Außerdem dürfen Kommentare nie vor einer XML-Deklaration verwendet werden, wenn diese in einem Dokument vorhanden ist. Der Vollständigkeit halber folgt nun ein Beispiel für einen Kommentar unter XML. [4]

```
<!-- Dies ist ein Kommentar. Dieser Kommentar darf auch Markup-Zeichen enthalten. <element></element> -->
```

Abbildung 14: Beispielhafter Kommentar innerhalb eines XML-Dokuments

Der letzte Abschnitt dieses Kapitels dreht sich nun um das Thema Namensräume. Auf dieses Thema wird auch noch später in Abschnitt 3.1.3 im Zusammenhang mit XML-Schema eingegangen. Elementnamen unterliegen in XML einer gewissen Freizügigkeit. Diese Freizügigkeit bringt aber auch Probleme mit sich. Diese Probleme treten meist immer da auf, wo Elemente mit exakt gleichen Elementnamen in einem XML-Dokument aufeinandertreffen. Ein kurzes Beispiel wäre, dass das Wort „groesse“ als Elementname verwendet, sowohl für eine Körpergröße, als auch für eine Schriftgröße stehen kann. Grundsätzlich sind Elementnamen in ihren jeweiligen Namensräumen durch den Zusammenhang in dem sie stehen wohl unterscheidbar, jedoch sind sie in einem XML-Dokument das Daten verschiedener Sinnbezirke kapselt nicht mehr zu unterscheiden. Um verschiedene Namensräume zu unterscheiden, kann nun in einem XML-Dokument und innerhalb dessen ein Element mit einem Attribut verwendet werden. In diesem Attribut kann nun ein eindeutiger Pfad zu einem Namensraum angegeben werden. [4]

```
<element xmlns:qname="http://qname.com/qname">
</element>
```

Abbildung 15: Deklarierung eines Namensraums

Wie in Abbildung 15 ersichtlich ist, wird ein XML-Namespace mittels einer URI beziehungsweise einer URL angegeben und mittels eines QNamen/Präfix kann auf diesen dann später verwiesen werden. Daraus ergibt sich dann, dass innerhalb dieses Elements nun alle Kindelemente automatisch auf diesen Namensraum bezogen werden. Wird nun auf der Ebene eines Kindelements ein neuer Namensraum verwendet, ist innerhalb dieses Elements wieder der vorgegebene Namensraum gültig. Attribute dürfen, solange sie dem gleichen Namensraum angehören wie das Element in dem sie verwendet werden, nicht mit einem Präfix versehen werden. [4]

```
<element xmlns="http://qname.com/qname">
</element>
```

Abbildung 16: Deklarierung eines Standard-Namensraums

Es gibt überdies die Möglichkeit einen Namensraum standardmäßig zu verwenden. Hierbei ist zu beachten, dass bei den Kindelementen dann kein Präfix mehr verwendet werden muss. Beispielsweise werden Standardnamensräume daher meist auf dem Wurzelelement eines XML-Dokuments angewendet. Dies führt zu einer erheblichen Verbesserung bezüglich der Lesbarkeit und des Schreibaufwands. Dabei kommt es aber auch zu einer Verkomplizierung bei Attributen, wenn es darum geht Standardnamensräume zu verwenden. [4]

„Werden Attribute ohne Präfix benannt, gehören sie dagegen nicht zum vorgegebenen Namensraum. Sie erben also nicht den vorgegebenen Namensraum des Elements, zu dem sie gehören.“ [1]

Standardnamensräume können auch mehrmals innerhalb eines Dokuments deklariert werden. Die erste Deklaration ist, wie bereits erwähnt, meist im Wurzelelement. Jede weitere Deklaration auf einer tieferen Ebene führt dazu, dass innerhalb dieses Elements der verwendete Namensraum Standard ist. Danach gilt wieder die Deklaration des Wurzelelements. Auf diese Art und Weise lassen sich Standardnamensräume auch ineinander verschachteln. [4]

```
<element xmlns:qname="http://qname.com/qname">

    <qname:elementausnamespace>inhalt</elementausnamespace>

</element>
```

Abbildung 17: Verwendung eines Elements eines bestimmten Namensraums

In Abbildung 17 zeigt sich die Verwendung eines Namensraums mithilfe eines QNamens innerhalb eines Kindelements. Der Elementname „element“ ist hierbei aus einem Standardnamensraum entnommen. [4]

3.1.3. Was ist ein XML-Schema?

In diesem Kapitel wird zunächst die Frage grundsätzlich beantwortet und dann versucht auf Details zum Thema einzugehen und mögliche weitere Fragen zu beantworten.

Inhaltsmodelle, wie zum Beispiel DTD und XML-Schema, widmen sich ganz der Entscheidung ob ein XML-Dokument nach bestimmten, in diesen Inhaltsmodellen aufgestellten Regeln, gültig ist. Jedoch bringen Dokumenttypdefintionen verschiedene Nachteile mit sich die bei Verwendung eines XML-Schemas entfallen. Daher ist bei einer professionellen Verwendung von XML meist immer ein XML-Schema als Inhaltsmodell involviert. Vorab wurden bei der Ausarbeitung des XML-Schema-Standards schon einige Anforderungen getroffen die für die Verwendung von Schemas essentiell geworden sind. Unter anderem wurden Namensräume, Integration von strukturellen Schemas mit primitiven Datentypen, explizite Vererbungsmechanismen und verschiedene gängige Datentypen berücksichtigt. Dabei wurde festgelegt, dass ein XML-Schema auch ein XML-Dokument ist und daher denselben Regeln und Einschränkungen wie ein XML-Dokument einer Instanz entspricht. Über die Zeit haben sich auch einige Konventionen entwickelt, denen nun mehr oder weniger gefolgt wird. Ein Beispiel hierfür wäre die Namensgebung der Schema-Dateien, da diese oftmals auf „.xsd“ enden, wobei dies nicht erforderlich ist und nur dem Verständnis dient. In den folgenden Erklärungen zu den Strukturen eines XML-Schemas wird häufig auf Beispiele aus dem XML-Schema, welches zur Erstellung von XML-Dateien für Jaspersoft-Reports dient, Bezug genommen. Die URL zum Download des XML-Schemas lautet:

http://jasperreports.sourceforge.net/xsd/jasperreport.xsd

Schlussendlich erfordert jedes XML-Dokument, dass auf ein Schema zurückgreift, eine Verknüpfung auf dieses Schema. Diese wird in Form eins Attributs im sogenannten Wurzelelement eines XML-Dokuments festgelegt. Der Wert des Attributs „xsi:schemeLocation“ ist entweder eine URI oder eine URL auf das zu verwendete Schema. Das Präfix „xsi“ wird deshalb verwendet, weil hier auch noch auf den Namespace einer XML-Schema-Instanz zugegriffen werden muss. Diese muss standardmäßig verwendet werden. Diese Deklaration findet sich in Abbildung 18 in Zeile 2. Der Schemazugriff findet sich in Zeile 3. In Zeile 1 ist noch der Default-Namespace für das Dokument vermerkt. Wie bereits erwähnt ist jedes XML-Schema auch ein XML-Dokument. [5]

```
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"></jasperReport>
```

Abbildung 18: Deklaration des zu verwendeten Schemas

Deshalb kommt es auch bei einem XML-Schema zur Verwendung eines Wurzelelements. Ein Beispiel für eine in der Praxis oft verwendete Konstellation findet sich in Abbildung 19. Das Attribut in der zweiten Zeile definiert einen Standardnamensraum aus dem alle Grundbausteine eines Schemas stammen. Dieser wurde vom World Wide Web Consortium standardisiert. Für diesen Namensraum kann auch ein Präfix verwendet werden, wobei dies laut Konvention „xsd“ heißen würde. Dieses Präfix finded aber nur in wenigen Anwendungsbeispielen Verwendung, wie zum Beispiel bei vorgegebenen Datentypen und Namen aus dem XML-Schema Namensraum. In der dritten Zeile

findet sich das Attribut „targetNamespace“, welches ebenfalls einen URI beziehungsweise eine URL enthält. Hinter diesem verbirgt sich ein Zielnamensraum in dem alle in diesem XML-Schema definierten Elemente und Attribute enthalten sind. In der vierten Zeile findet sich noch einmal der Verweis auf denselben Namensraum. Dennoch wurde hierbei ein Präfix gesetzt. Dieses muss immer dann verwendet werden, wenn die Attribute „type“ und „ref“ zum Einsatz kommen und diese sich auf globale Element- beziehungsweise Attributdeklarationen beziehen. Globale Elemente sind Elemente die eine Ebene unter dem Wurzelelement stehen. Ein Beispiel hierfür findet sich in Abbildung 20, denn hier referenziert ein lokales Element ein bereits global definiertes Element. [6]

```
<schema
  xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://jasperreports.sourceforge.net/jasperreports"
  xmlns:jr="http://jasperreports.sourceforge.net/jasperreports"
  elementFormDefault="qualified">
```

Abbildung 19: Wurzelelement des Jaspersoft-Report-Schemas

```
<element ref="jr:property" minOccurs="0" maxOccurs="unbounded"/>
```

Abbildung 20:Deklaration eines Elements

In Abbildung 19 findet sich in Zeile 5 auch ein Attribut mit dem Namen „elementFormDefault“, welches dem Schema-Designer ermöglicht global festzulegen, ob eine Überprüfung vorgenommen werden soll. Diese Überprüfung bezieht sich darauf, ob die Namen der Elemente in einem XML-Dokument, welches das Schema verwendet, mit dem Namensraum des Schemas übereinstimmten. Diese Option gibt es nicht nur für Elemente, sondern ist auch für Attribute verfügbar. [6]

In einem XML-Schema gibt es neben dem Wurzelelement mit dem Namen „schema“ auch noch 35 andere Grundelemente aus denen sich der Inhalt eines Schemas aufbauen und beschreiben lässt. Diese Komponenten lassen sich in drei Klassen aufteilen. In der ersten Gruppe enthalten sind Elementdeklarationen, Attributdeklarationen und einfache beziehungsweise komplexe Typdefintionen. Typdefintionen benötigen im Vergleich zu den restlichen Komponenten dieser Gruppe nicht zwingend einen Namen. Dem gegenüberstehend findet sich die zweite Gruppe, denn deren Komponenten benötigen alle zwingend einen Namen. Die Komponenten lauten: Attributgruppen, Modellgruppendefinitionen, Anmerkungen, Eindeutigkeitsbeschränkungen und Schlüsselreferenzen. Die dritte Gruppe enthält nur Komponenten die immer nur als Teil von anderen Komponenten auftreten. In diese Gruppen fallen Anmerkungen, Modellgruppen, Partikel, Joker und Festlegungen über die Verwendung von Attributen. Nachfolgend werden viele dieser Komponenten vorgestellt und an Beispielen behandelt. [6] [1]

Zu allererst werden nun Datentypen vorgestellt. Allem voran sind komplexe Daten erwähnenswert. Diese stellen sich als sehr mächtige Werkzeige heraus, wenn es darum geht höhere Funktionalität und Tiefe zu erzielen. In Abbildung 21 findet sich eine beispielhafte Verwendung. Hier vorab eine formale Definition komplexer Datentypen:

Ein komplexer Datentyp besteht grundsätzlich aus einem Inhaltsmodell, das die darin vorkommenden Unterelemente auflistet und in einer bestimmten Weise anordnet, gefolgt von einer Liste der Attribute, die zusätzliche Informationen zu den betreffenden Datentypen bereitstellt. [1]

```
<element name="Gemueseladen" type="gemuesesorte">

<complexType name="gemuesesorte" >
    <sequence>
    <element name="gemuesenummer" type="short"/>
    <element name="saison" type="string"/>
    </sequence>
</complexType>

</element>
```

Abbildung 21: Deklaration eines Elements mit komplexer Typdefinition

Das Element „Gemueseladen“ ist vom Datentyp „gemuesesorte“, welcher seinerseits über den Namen des komplexen Typs identifiziert wird. Innerhalb dessen findet sich eine Sequenz von einfachen Elementen die wiederum vorgegebene Datentypen besitzen. Die Namensräume verhalten sich hierbei sehr ähnlich zu denen in Abbildung 19. [7]

Ein komplexer Datentyp setzt sich aus Teilen zusammen die Partikel genannt werden. Beispiele für Partikel wären lokale Elementdeklarationen, welche in Abbildung 21 zu sehen sind, Bezüge auf globale Elementdeklarationen, siehe Abbildung 20 und auch Wildcards. Eine beispielhafte Verwendung von Wildcards findet sich in Abbildung 22. Für spätere Erweiterung erlauben es Wildcards dem Schema-Designer, dass Teile des Datenmodells offenbleiben. Dies wird durch die Platzierung eines Elements mit dem Namen „any“ erreicht. In Verbindung mit dem Attribut „processContent“ und dem Wert „skip“ wird bei „any“ erreicht, dass bei Verwendung in einem XML-Dokument dieser Bereich nicht durch den XML-Prozessor geprüft wird und daher einer Validierung entgeht. Das Element „any“ bezieht sich dabei auf Elemente, während das Element „anyAttribute“ die gleiche Funktionalität für Attribute erfüllt. [7] [1]

```
<any processContent="skip"/>
```

Abbildung 22: Deklaration einer Wildcard

Nach komplexen Typen sind einfache Typen eine Komponente der ersten Gruppe. Einfache Typen sind Elemente die selbst nur einen Wert, zum Beispiel ein Datum oder einen Text, enthalten. Der Wert eines Attributes gehört daher auch immer zum Datentyp „simpleType“ und ist meist innerhalb eines komplexen Typs gekapselt. Ein Beispiel hierfür findet sich in Abbildung 23. Der Übersicht halber wurde bei diesem Beispiel darauf verzichtet das gesamte Element darzustellen. Innerhalb des Attributs mit dem Namen „isStartNewColumn“ findet sich ein einfacher Datentyp, welcher wiederrum nur den Wert des Attributs hält, in diesem Fall entweder „true“ oder „false“. [7]

```
<attribute name="isStartNewColumn" use="optional" default="false">
 <annotation>
  <documentation>Flag that signals if the group header should be printed always on a new column.</documentation>
 </annotation>
 <simpleType>
  <restriction base="string">
   <enumeration value="true">
     <annotation>
      <documentation>Group header section is printed always on a new column.</documentation>
     </annotation>
   </enumeration>
   <enumeration value="false">
     <annotation>
      <documentation>Group header section is printed on the current column, if there is enough space.</documentation>
     </annotation>
   </enumeration>
  </restriction>
 </simpleType>
</attribute>
```

Abbildung 23: Einfache Typdefinition

Bei der Spezifizierung von XML-Schema wurden von Anfang an vorgegebene Datentypen angedacht, die sich nun auch in der Spezifikation wiederfinden. Ein Datentyp besteht laut Spezifikation aus einem Werteraum, einem lexikalischem Raum und aus einer Reihe von Eigenschaften. Diese Eigenschaften werden Facetten genannt. Durch die Modifizierung dieser Facetten lassen sich Datentypen auch ableiten. Eine Möglichkeit dies zu bewerkstelligen ist mittels Einschränkungen eine Veränderung des Datentyps zu erzwingen. Ein Beispiel hierfür ist in Abbildung 24 zu sehen. [7]

```
<simpleType>
    <restriction base="string">
     <enumeration value="false"/>
     <enumeration value="true"/>
    </restriction>
</simpleType>
```

Abbildung 24: Abgeleiteter Datentyp durch Einschränkung

Innerhalb eines simplen Datentyps muss nun eine Einschränkung durch das Grundelement „restriction“ gebildet werden. In diesem Fall wurde eine Einschränkung auf die Wahl zweier Werten erreicht. Es sind keine anderen Wert als „true“ oder „false“ erlaubt. Eine spezielle Art von abgeleiteten Datentypen sind Listen. Ein Beispiel für die Verwendung einer Liste befindet sich in Abbildung 25. Das obere Attribut vom Typ „NMTOKEN“ ist gleichbedeutend mit dem unteren Attribut. Der Datentyp „NMTOKEN“ ist ein Atom der Listenart „NMTOKENS“. Dadurch wurde nur bei dem unteren Attribut eine Liste definiert, welche später ausschließlich Integer-Werte enthalten darf. Bei dem von Jaspersoft verwendetem Schema findet sich jedoch nur die obere Variante. Jaspersoft Studio arbeitet bei dem oberen Attribut allerdings ausschließlich mit Integer-Werte und würde daher den Datentyp „NMTOKEN“ nicht benötigen, da dieser nicht nur Zahlen enthalten kann. Hiernach stellt sich nun die Frage, warum vonseiten des Schema-Designers „NMTOKEN“ und nicht ein anderer Datentyp, wie zum Beispiel „Integer“ verwendet wurde. [7]

```
<attribute name="y" type="NMTOKEN" use="optional"/>

<attribute name="y">
    <simpleType>
        <list itemType="integer" use="optional"/>
    </simpleType>
</attribute>
```

Abbildung 25: Gegenüberstellung „NMTOKEN“ und „list“

Bisher wurden Typdefinitionen angesprochen, die dazu dienen Datentypen zu kreieren. Nun sollen vor allem Deklarationen beleuchtet werden, da diese dazu dienen Elemente und Attribute mit einem Namen zu versehen und mit einem Datentyp zu verbinden.

Diese Datentypen können wie bereits besprochen komplexer oder einfacher Natur sein. Zuerst wird hier die Deklaration von Elementen behandelt. Die Deklaration an sich geschieht mit dem Element „<element>“. Hierzu muss zusätzlich auch noch ein Name vergeben werden. Es können einfache und komplexe Elemente gebildet werden. In Abbildung 21 wurde bereits auf die Deklaration eines komplexen Elements mithilfe eines komplexen Datentyps, eingegangen. Im Gegensatz zu komplexen Elementen finden sich in einfachen Elementen keine Attribute und sie enthalten selbst auch keine Kindelemente. [6] [1]

Nun wird sich der Deklaration von Attributen gewidmet. Attribute werden mit Hilfe des Elements „<attribute>“ deklariert und müssen bei komplexen Datentypen immer hinter der Elementdeklaration erscheinen. Ein Beispiel hierfür lässt sich in Abbildung 25 sehen, wobei diese Attribute auch innerhalb eines komplexen Typs deklariert wurden. Attribute müssen wie Elementdeklarationen einen eindeutigen Namen besitzen und einen einfachen Datentyp verwenden. Dieser Datentyp kann benutzerdefiniert oder ein durch XML-Schema vordefinierter Datentyp sein. Aus dieser Vorgabe folgt, dass ein Attribut keine anderen Elemente enthalten darf. Hingegen sind Referenzen auf andere bereits definierte Attribute erlaubt (Siehe Abbildung 26). Die Reihenfolge der Deklarierung von Attributen innerhalb eines komplexen Typs ist beliebig. [6]

```
<attribute ref="xml:lang"/>
```

Abbildung 26: Referenzierung des "lang"-Attributs

```
<element name="name" type="name"/>

<complexType name="name">

    <attribute name="name" type="string"/>

</complexType>
```

Abbildung 27: Beispiel zu Symbolräumen

Da Elemente und Attribute immer einen Namen fordern, ist es notwendig das Thema Symbolräume anzuführen. Hierbei muss man zunächst in globale und lokale Elemente und Attribute unterscheiden. Im Kontext von verschiedenen komplexen Elementtypen können Kindelemente und Attribute verwendet werden, die denselben Namen tragen. Denn nur weil Elemente und Attribute denselben Namen

tragen sind sie noch lange nicht identisch. Sie werden durch ihren Kontext unterschieden. Jeder komplexe Datentyp eröffnet sozusagen immer einen neuen Symbolraum. Deshalb lässt sich nun auch, wie in Abbildung 27 ersichtlich, ein gleicher Name für ein Attribut und einen Namen vergeben. [1]

```
<element name="telefonnummer" minOccurs="0" maxOccurs="unbounded"/>
```

Abbildung 28: Beispiel einer Häufigkeitsbestimmung

Nunmehr reicht es oft nicht aus in einem XML-Schema nur einzuschränken was eingegeben werden darf. Denn oftmals ist es auch notwendig zu bestimmen wie oft ein Element nun vorkommen darf. Hierzu lassen sich vor allem die Attribute „minOccurs“ und „maxOccurs“ verwenden. Erstes gibt an wie oft das Element mindestens vorkommen muss. Aus dem Beispiel in Abbildung 28 lässt sich nun ablesen, dass das Element nicht verwendet werden muss, da es auch null Mal verwendet werden kann. Wird das Attribut nicht verwendet, gibt XML-Schema einen Standardwert von eins vor. Genauso verhält es ich auch bei der Obergrenze die mit dem Attribut „maxOccurs“ angegeben wird. Hierbei kommt jedoch noch eine zusätzliche Option ins Spiel. „unbounded“ bedeutet hier, dass das Element beliebig oft vorkommen darf. Ähnlich ist dies auch bei Attributen möglich. [1]

```
<attribute name="name" type="string" use="required">
 <annotation>
  <documentation>Name of the parameter.</documentation>
 </annotation>
</attribute>
```

Abbildung 29: Verwendung des Attributs "use"

„use“ ermöglicht es bei Attributen die Auswahl zu treffen, ob die Verwendung des Attributs verpflichtend oder optional ist. Hierzu gelten jeweils die Werte „required“ und „optional“. Hinzu kommt dann noch eine dritte Möglichkeit „prohibited“, welche das Attribut für eine Wertvergabe sperrt. [1]

Bei Elementen und Attributen lassen sich auch noch andere Entscheidungen bezüglich des Designs treffen. Zuallererst gibt es die Möglichkeit eines Standardwertes. Für einen Default-Wert muss angenommen werden, dass der Attribut-Wert im Dokument leer gelassen wurde. Wenn dann, wie in Abbildung 30, das Default-Attribut gesetzt ist wird der eingegebene Standard-String automatisch für das Attribut mit dem Namen „class“ verwendet. [1]

```
<attribute name="class" type="string" use="optional" default="java.lang.String">
 <annotation>
  <documentation>Class of the parameter values.</documentation>
 </annotation>
</attribute>
```

Abbildung 30: Verwendung des Attributs "default"

```
<attribute name="class" type="string" use="optional" fixed="java.lang.String">
 <annotation>
  <documentation>Class of the parameter values.</documentation>
 </annotation>
</attribute>
```

Abbildung 31: Verwendung des Attributs "fixed"

In Abbildung 31 findet sich eine Möglichkeit nur die Eingabe von durch den Schema-Designer vorgegebenen Werten zu erlauben. Dies lässt sich mit dem Attribut „fixed“ bewerkstelligen. Wenn dieses Attribut verwendet wird, muss der Wert genau mit dem in der Attributdeklaration angegebenen „fixed“-Wert übereinstimmen. Diese zwei Optionen stehen, ähnlich wie Attributen, auch Elementen offen. [1]

Bei komplexen Elementtypen, wie in Abbildung 21, werden zur Elementgruppierung meist Sequenzen verwendet. Diese unterliegen jedoch immer bestimmten Regeln. Es gibt aber auch noch andere Elementgruppierungsarten, auch Kompositionen genannt, die jeweils wieder mit eigenen Regeln ausgestattet sind. Sequenzen behalten sich etwa vor die Reihenfolge zu bestimmen in der Elemente vorkommen dürfen. Die Komposition „<all>“ erzwingt im Gegensatz zu „<sequenze>“ etwa weder dass die Elemente vorkommen müssen, noch dass diese in einer exakt vorgegebenen Reihenfolge auftauchen müssen. Die dritte Komposition ist „<choice>“. Ein Beispiel hierzu findet sich in Abbildung 32, in dem innerhalb des Elements „groupHeader“ entweder die Elemente „band“ oder „part“ Verwendung finden dürfen. Überdies ist es natürlich auch möglich diese Kompositionen beliebig zu verschachteln. [6] [1]

```
<element name="groupHeader">
 <annotation>
  <documentation>Contains the definition of the header section for this group.</documentation>
 </annotation>
 <complexType>
  <choice minOccurs="0" maxOccurs="1">
   <element ref="jr:band" minOccurs="0" maxOccurs="unbounded"/>
   <element ref="jr:part" minOccurs="0" maxOccurs="unbounded"/>
  </choice>
 </complexType>
</element>
```

Abbildung 32: "choice"-Komposition

Komplexe Elementtypen enthalten meist immer eine Komposition aus mehreren Elementen. Werden hierbei bestimmte Elemente mehrfach benötigt kann es sinnvoll sein diese in einer Gruppe zusammenzufassen. Eine solche Gruppe bekommt dafür einen im Schema eindeutigen Namen. Dieser kann dadurch später verwendet werden um auf diese Gruppe zu referenzieren. Die Referenzierung geschieht dann meist aus einem komplexen Typ heraus. Eine Gruppe hat daher in gewisser Weise die Funktion eines Containers in dem Elemente abgelegt werden können. Bei Gruppen muss auf zirkuläre Beziehungen unter den Gruppen geachtet werden, da diese nicht entstehen dürfen. Für Attribute gibt es auch diesmal ein ähnliches Element mit dem Namen „attributeGroup“. Die Verwendung erfolgt bei diesem vergleichbar wie zum Element „<group>“. Ein Beispiel für eine Elementgruppe findet sich in Abbildung 33. [6] [1]

```
<group name="kontaktadresse">
    <all>
        <element name="telefonnummer" type="integer" minOccurs="0"/>
        <element name="landesvorwahl" type="integer" minOccurs="0"/>
    </all>
</group>

<complexType name="privatkontakt" >
    <sequence>
        <element name="postadresse" type="string"/>
        <group ref="kontaktadresse"/>
    </sequence>
</complexType>
<complexType name="geschaeftskontakt" >
    <sequence>
        <element name="postadresse" type="string"/>
        <group ref="kontaktadresse"/>
    </sequence>
</complexType>
```

Abbildung 33: Beispiel einer Modellgruppe

Da es in Datensammlungen, die wiederrum oftmals auch in XML abgebildet werden, meist ein Schlüsselelement gibt um auf einen Datensatz gezielt zugreifen zu können, wurde in XML-Schema auch dafür eine Struktur bereitgestellt. In Abbildung 34 findet sich ein Beispiel für eine solche Struktur. Das Element „<key>“ wird innerhalb des Elements „warenliste“ erstellt und ist innerhalb dessen auch gültig. „<key>“ ermöglicht es die Identität eines Elements einzuschränken und auf diese Weise für die Eindeutigkeit des Elements zu sorgen. Innerhalb des „<key>“-Elements gibt es auch noch ein „<selector>“-Element das angibt, dass die Einschränkung für das Element „ware“ gilt. Zusätzlich wird innerhalb von „ware“ noch ein Element angegeben, in diesem Fall „warennummer“, das eindeutig sein sollte. [6] [1]

```
<element name="warenliste">
    <complexType>
        <sequence>
            <element name="ware">
                <complexType>
                    <sequence>
                        <element name="warennummer"/>
                        <element name="warenbezeichnung"/>
                    </sequence>
                </complexType>
            </element>
        </sequence>
    </complexType>
    <key>
        <selector xpath="ware"/>
        <field xpath="warennummer"/>
    </key>
</element>
```

Abbildung 34: Verwendung des "key"-Elements

Außerdem lassen sich auch noch Referenzen auf Schlüsselelemente erstellen, welche eine Überprüfung eines Elements mit dem Schlüsselelement ermöglichen. [6] [1]

```
<complexType name="gefahrgut">
    <complexContent>
        <extention base="gut">
            <sequence>
                <element name="gefahrgutnummer" type="integer"/>
            </sequence>
        </extention>
    </complexContent>
</complexType>
```

Abbildung 35: Erweiterung eines komplexen Elements

Dieser letzte Teil der theoretischen Abhandlung von XML-Schema behandelt nun die Ableitung und Mehrfachverwendung von komplexen Datentypen. Zuerst wird die Erweiterung komplexer Elemente behandelt. In Abbildung 35 findet sich ein Beispiel dazu. Hierzu wird aus dem komplexe Element „gut“ ein neues Gut des Datentyps „gefahrgut“ kreiert. Dieser Datentyp erweitert „gut“ um das Element „gefahrgutnummer“. Diese Ableitungen lassen sich jedoch auch einschränken, um zu verhindern, dass auf diese Weise ein unbrauchbares XML-Schema entsteht. Dies kann zum Beispiel mit dem Attribut „final“ erreicht werden. Eine weitere Art der Mehrfachverwendung stellen Referenzen da. Ein Beispiel hierzu findet sich bereits in Abbildung 20. Elemente die referenziert werden sollten müssen jedoch als globale Elemente deklariert werden. Mittels des Attributs „ref“ kann dann dieses Element aufgerufen werden. Diese Möglichkeit bietet sich nicht nur bei Elementen, sondern ist auch bei Attributen möglich. [1]

3.2. Jaspersoft Studio Community Edition

Dieses Kapitel widmet sich voll und ganz dem Programm Jaspersoft Studio, welches der Projektgruppe für die Projektarbeit in der Community Edition zur freien Verfügung stand. Das Ziel dieses Kapitel ist eine detaillierte Einführung zu geben, allfällige Fragen vorab zu beantworten und dadurch einen Mehrgewinn zu erzielen. Ergänzend sollte dem Leser ein Überblick über die Möglichkeiten und Verwendungszwecke des Programms gegeben werden.

Allgemein gesagt definiert sich ein Jaspersoft-Report immer durch eine XML-Datei. JasperReports spezifiziert hierzu eine JRXML-Datei. Diese Datei ist, wie jede andere XML-Datei auch, in Abschnitte unterteilt. Den verschiedenen Abschnitten werden in einem Jaspersoft-Report allerdings verschiedene Funktionen und Aufgaben zuteil. Während ein Abschnitt, zum Beispiel die physischen Eigenschaften eines Reports beleuchtet und darin etwa die Positionierungen von Feldern oder die Dimensionierung der Seite beschreibt, widmet sich ein Anderer den logischen Strukturen. Ein Beispiel für logische Strukturen in einem Report ist die Deklarierung von Parametern oder Variablen. [8]

In allen Fällen durchläuft ein Report in JasperReport, jedoch 2 grundlegende Phasen. Die erste Phase enthält Teilbereiche wie Design, Planung, Erstellung einer JRXML-Datei und Kompilieren einer Jasper-Datei aus der JRXML-Datei. In der zweiten Phase wird der Report ausgeführt. Die hierbei enthaltenen wichtigsten Schritte sind das Laden der Jasper-Datei, das Füllen des Reports und das Exportieren in ein passendes Ausgabeformat. [8]

In diesem Zusammenhang beschäftigt sich Jaspersoft Studio meist nur mit Phase eins. Jedoch ist es auch möglich Schritte der Phase zwei innerhalb Jaspersoft Studio auszuführen, denn es kann bereits eine Vorschau des designten Reports mit Daten gespeist werden, um die Funktionalität zu gewährleisten. Hierzu muss jedoch abermals erwähnt werden, dass die eigentliche Aufgabe nämlich die Zusammenführung des Templates und der Daten auf Seiten von JasperReports liegt. Denn JasperReport nimmt die Jasper-Datei sowie eine gewünschte Datenquelle und führt diese zusammen. Auf mögliche Datenquellen wird später noch genauer eingegangen. [8]

In den folgenden Absätzen werden nun essentielle Elemente behandelt, die Jaspersoft Studio zum Design eines Reports bereitstellt. Als Erstes werden Streifen thematisiert. [8]

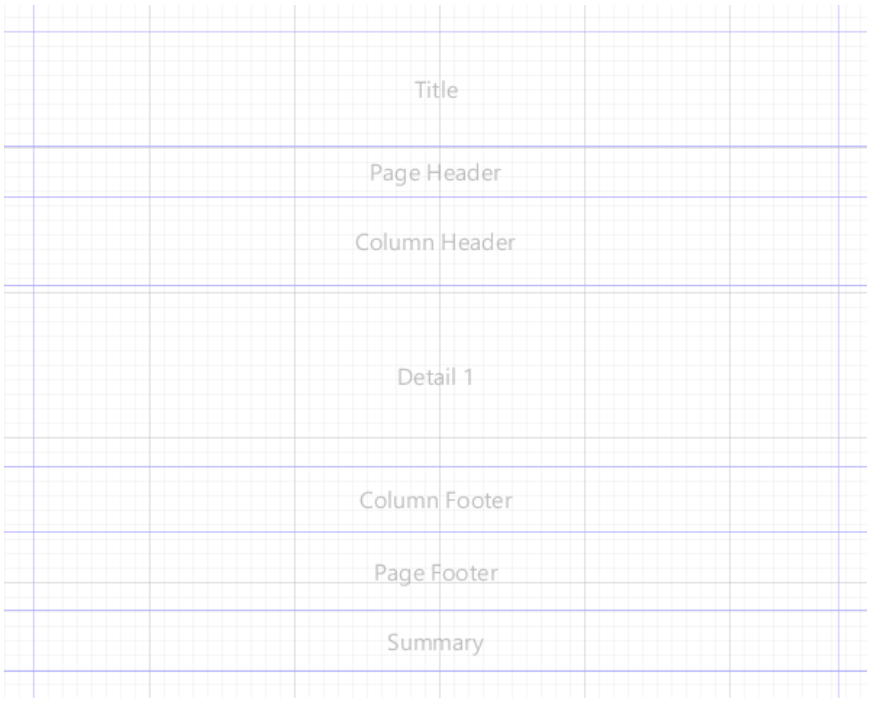


Abbildung 36: Streifen eines leeren JasperReports im Format A4

Innerhalb des Designvorgangs unterteilt Jaspersoft Studio einen Report standardmäßig in neun Streifen. Hierbei den Originalausdruck „bands“ mit dem Wort „Bereichen“ zu übersetzen ist nicht zu 100 Prozent korrekt, denn jeder dieser neun Streifen zieht sich horizontal über den gesamten Report. Die Übersetzung mit dem Wort „Streifen“ ist daher vorzuziehen. Ein jeder Streifen eines Reports ist deshalb auch nur in der Vertikalen variabel. Der Streifen Title erscheint immer nur auf der ersten Seite eines Reports, wenn dieser aus mehreren Seiten besteht. Der Page Header hingegen ist auf allen Seiten zu finden. Der Column Header enthält meist die Spaltennamen eines tabellarischen Reports. Der Detail enthält daher auch meist die eigentliche Tabelle. Der Column Footer tritt meist am Ende einer jeden Column auf. Der Page Footer ist das Äquivalent, zu dem bereits erwähntem Page Header jedoch am Ende einer Seite. Der Summary-Streifen tritt immer auf der letzten Seite eines Reports auf. Überdies gibt es noch einen Background-Streifen der sich über die ganze Seite zieht und somit die Platzierung von Wasserzeichen oder ähnlichen Effekten ermöglicht. Um die horizontale Ausbreitung aller Streifen zu beeinflussen und um etwaige Vorgaben zu erfüllen, ist es jedoch möglich einen Seitenabstand einzufügen oder den gesamten Report ein benutzerdefiniertes Format zu geben. [8]

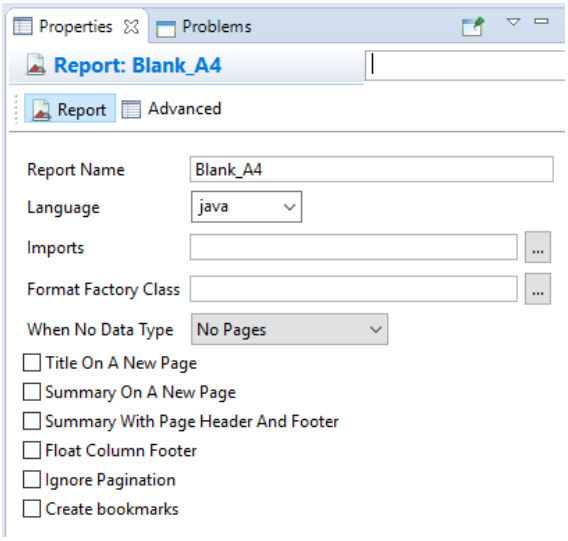


Abbildung 37:Eigenschaften eines Reports

Das Format kann auch nach der Erstellung noch verändert werden, da das Eigenschafts-Tap innerhalb Jaspersoft Studio diese Funktion bietet. Hier lassen sich auch noch andere Funktionalitäten, wie zum Beispiel der Name des Reports ändern. Diese in Jaspersoft Studio visualisierte Option ist in Abbildung 37 ersichtlich. [8]

Nachdem alle Einstellungen vorgenommen wurden und der Report fertiggestellt wurde, kann dieser in eines von vielen verschiedenen Formaten exportiert oder eine Vorschau erstellt werden. Dazu wird die bereits vorgestellte Jasper-Datei mit den gewünschten Daten verschränkt und angezeigt. Von diesem Punkt aus kann dann auch der Bericht exportiert werden. Hierzu stehen unter verschiedenen Formaten auch PDF, XLS, HTML zur Auswahl. [8]

Um aber das gewünschte Ergebnis zu erhalten muss man verschiedene Reportelemente verwenden können. Grundlegende Elemente sind Linien, Rechtecke, Ellipsen, statische Felder, Textfelder, Bilder, Tabellen und etliche andere. All diese Elemente können, wie in Abbildung 38, aus der Palette heraus ausgewählt werden. Beim Einsetzten eines Elements mittels Drag & Drop in die Oberfläche eines Berichts kann dieses Element entweder automatisch oder manuell in der Größe angepasst werden. [8]

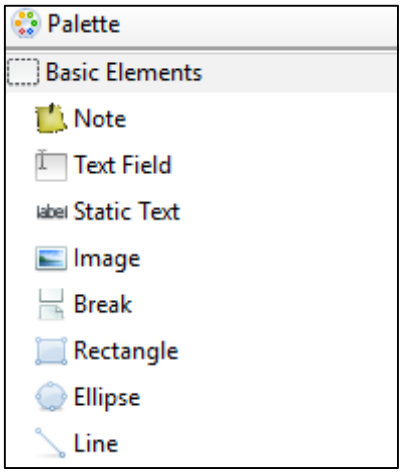


Abbildung 38:Palette zur Elementauswahl

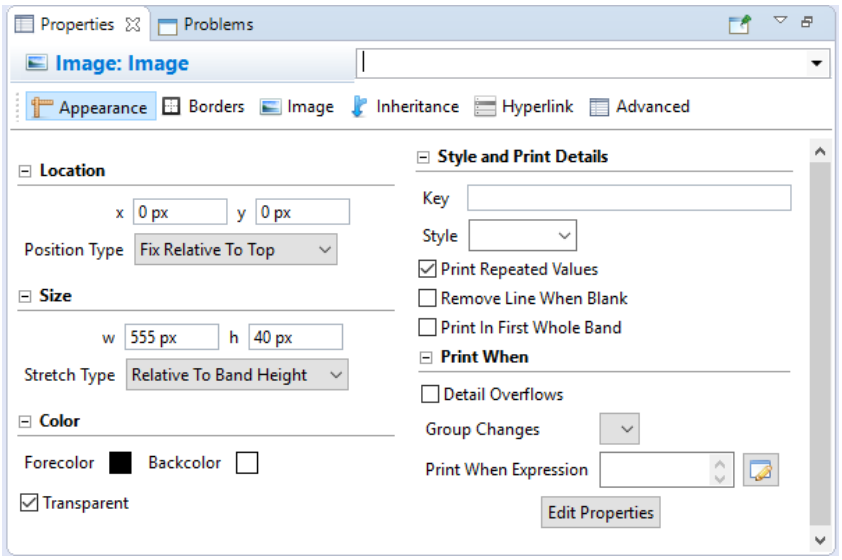


Abbildung 39: Elementeigenschaften

Zu jedem Element können überdies verschiedene Eigenschaften konfiguriert werden (siehe Abbildung 39). Zum Beispiel können im „Appearance“-Tap allgemeine Elementeigenschaften wie die Position des Elements eingestellt und verändert werden. Je nach Elementtyp werden in den Elementeigenschaften auch für den Typ spezifische Optionen angezeigt. [8]

Um eine genauere Positionierung der Elemente zu erreichen stehen dem Benutzer verschiedene Möglichkeiten offen. Die Möglichkeiten umfassen die Platzierung innerhalb eines Gitters, innerhalb eines Streifens oder mittels vom Nutzer eingefügten Führungslinien. Die umfangreichsten Möglichkeiten stellen dabei die Platzierung mittels eines Gitters und die Positionierung in einem Streifen dar. Bei der Positionierung in einem Streifen spricht man eigentlich über eine Positionierung in einem Container, denn Streifen sind eine Unterart eines Containers. Andere Container sind Rahmen und Tabellenzellen, wobei jede Containerart ihre eigenen Eigenheiten in Bezug auf ihre Formatierung aufweist. Die Positionierungsart mittels eines Gitters ist mitunter die flexibelste. Denn dabei kann ein Bereich in eine anpassbare Anzahl von Zeilen und Spalten unterteilt werden. In beziehungsweise über mehrere dadurch entstehende Zellen kann dann ein Element platziert werden. Wird nun der auf diese Weise unterteilte Bereich in seiner Größe verändert, werden automatisch auch die sich darin befindlichen Elemente aufgrund ihrer Größeneigenschaften beeinflusst. Neben diesen Formatierungsarten stehen auch noch allgemeine Formatierungseigenschaften zur Verfügung. [8]

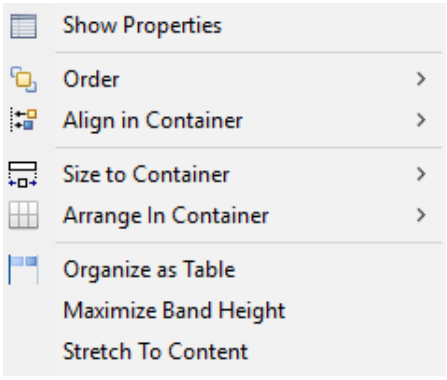


Abbildung 40: Kontextmenü für Formatierung

Diese Formatierungseigenschaften lassen sich über ein mit Rechtsklick aufrufbares Kontextmenü anzeigen (siehe Abbildung 40). Bei diesen Formatierungseigenschaften sind Ebenenoptionen, Ausrichtungsoptionen und auch noch weitere Größenoptionen verfügbar. [8]

Alle verfügbaren Elemente lassen sich in verschiedene Gruppen einteilen. Graphische Elemente enthalten zum Beispiel Bilder, Linien oder Rahmen, während Textelemente die Form von statischem Text oder Textfeldern annehmen können. [8]

Ein wichtiges Element gestalterischer Freiheit, das immer wieder benötigt wird, sind Seitenumbrüche. Diese werden im Designbereich als einfache Linie dargestellt. [8]

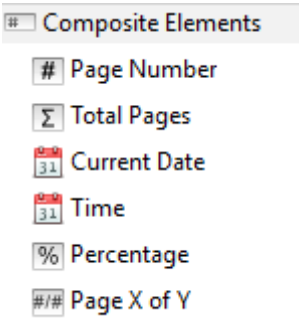


Abbildung 41: Auswahl von zusammengesetzten Elementen

Werden komplexe Strukturen beim Report-Design vermehrt verwendet und benötigt, bietet es sich für manche dieser Strukturen an eine Vorlage für diese zu bilden. Diese Vorlagen können aber keine Tabellen, Diagramme oder Listen enthalten und sind daher nicht für alle verfügbaren Elemente einzusetzen. Sobald zusammengesetzte Elemente erzeugt werden, sind diese in einem Bereich unterhalb der Palette, wie normale Elemente, auszuwählen (siehe Abbildung 41). Wurden vom Benutzer noch keine zusammengesetzten Elemente erstellt, scheinen in dieser Rubrik standardmäßig vorgegebene und bereits definierte Elemente auf , die natürlich auch verwendet werden können. Überdies gibt es in Jaspersoft Studio die Möglichkeit selbst erstellte zusammengesetzte Elemente zu exportieren und zu importieren. [8]

Jaspersoft Studio stellt dem Nutzer auch verschiedene Möglichkeiten zur Verfügung auf interne beziehungsweise externe Elemente zu verweisen. Diese Möglichkeiten lassen sich grob in 3 Kategorien einteilen. [8]

Anker sind verfügbar, um innerhalb eines Elements mittels eines Hyperlinks auf dieses Element zugreifen zu können. Lesezeichen ermöglichen es hingegen beim Exportieren des Reports in das PDF-Format auf diese gesetzten Lesezeichen zuzugreifen. Hyperlinks hingegen können mittels einer URL auf eine externe Ressource verweisen. Bei diesen URLs ist es auch möglich diverse Variablen mit zu übermitteln und auf diese Art und Weise eine dynamische Verlinkung zu ermöglichen. Außerdem ist es möglich auch innerhalb eines Reports auf Elemente zu referenzieren und zum Beispiel damit aus einem Inhaltsverzeichnis heraus auf ein Kapitel zu verweisen. Überdies ist es möglich auf eine bestimmte Seite innerhalb oder auch außerhalb eines Reports zu verweisen. Auch der Verweis auf einen Anker innerhalb eines externen Berichts kann durch Hyperlinks möglich gemacht werden. Mittels Hyperlinks des Typs „ReportExecution“ ist es aber auch möglich die Erzeugung eines neuen Reports durch eine JaspersoftReport Server Instanz zu realisieren. [8]

Neben bereits in Jaspersoft Studio enthaltenen Elementen können durch die Verwendung und Erzeugung von JavaScript-Modulen auch neue Visualisierungsoptionen zu Jaspersoft Studio hinzugefügt werden. [8]

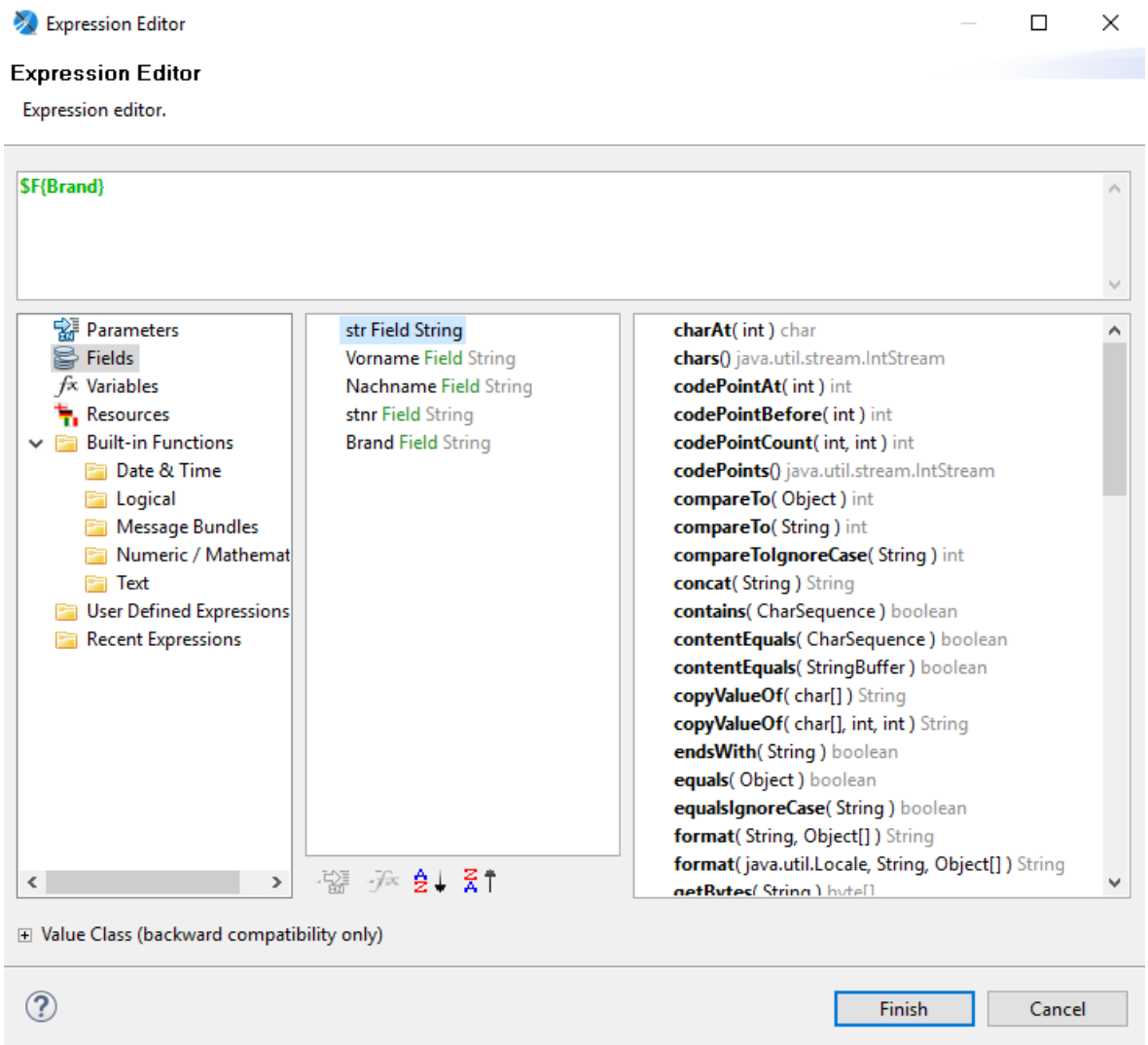


Abbildung 42: Übersicht über alle Felder eines Reports

Felder sind eines der wichtigsten Elemente eines Reports, ermöglichen sie doch das Befüllen des Reports mit Daten. In Abbildung 42 ist zum Beispiel eine Auflistung aller in einem Dokument enthaltenen Felder zu sehen. Daher können Felder auch als Platzhalter für Daten betrachtet werden. Diese Platzhalterrolle beschränkt sich in diesem Fall jedoch nicht nur auf die visuellen Eigenschaften eines Reports. Felder lassen sich mittels eines eindeutigen Namens und eines Datentyps identifizieren. Dieser Name dient dazu ein Schlüssel/Wert-Paar zu bilden. Der Name ist hierbei der Schlüssel und der Wert wird hingegen meist aus einer Datenquelle ausgelesen. Der Name und der Typ der auszulesenden Datenquelle muss dabei mit dem Namen und Datentyp des Feldes übereinstimmen. Die Datenquelle ist dabei meist eine SQL-Datenbank, aber es gibt auch andere Möglichkeiten die genutzt werden können. [8]

Neben Feldern gibt es in Jaspersoft Studio aber auch die Option einer weiteren Struktur, die es ermöglicht Informationen einzubringen. Parameter erfüllen diese Anforderung indem sie Daten einbringen, die sowohl durch einen Zugriff auf die Datenquelle eingebracht werden können als auch durch das Referenzieren auf ein internes Element. Parameter können einfach erstellt werden und dann in SQL-Abfragen verwendet werden. [8]

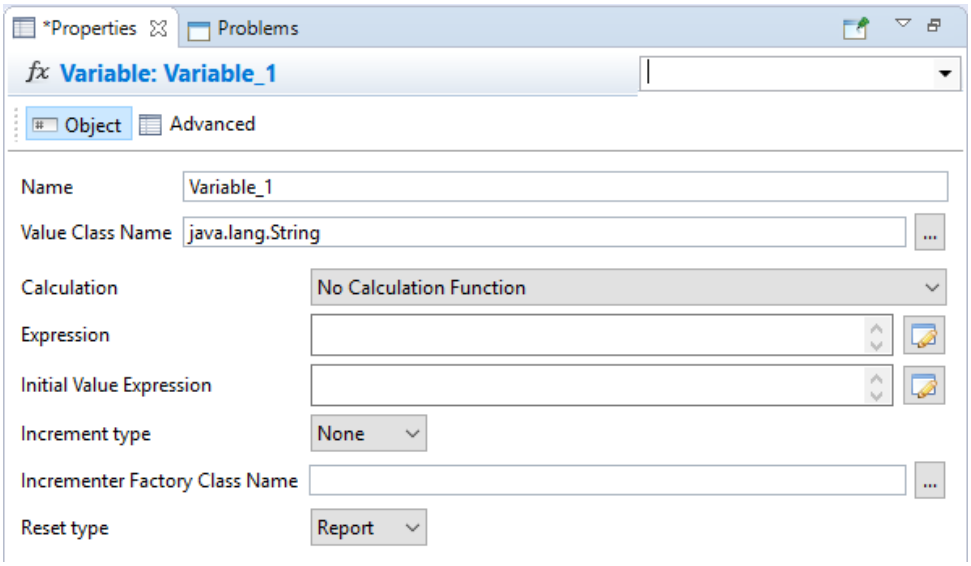


Abbildung 43:Eigenschaften einer Variabel

Variablen können genutzt werden, um Ergebnisse zu speichern und um Berechnungen von Daten zu ermöglichen. Variablen benötigen meist einen Namen, einen Datentyp, einen Ausdruck der einen Wert hält und in dem Berechnungen ausgeführt werden können sowie einen initialen Wert der nicht immer benötigt wird. [8]

Ausdrücke bieten die Option mit Werten zu arbeiten, die mittels Feldern, Parametern oder Variablen zur Verfügung gestellt werden. Ein Ausdruck ist generell gesagt eine Formel die einen Wert verarbeitet und einen anderen zurückliefert. Ausdrücke haben immer einen Typen wobei die Wahl des Typen entscheidend für die Verwendung ist. Denn ein Bild in Base64-Format das verarbeitet werden sollte, bedarf meist eines String-Typs. Da in Jaspersoft Studio meist auf Java-Basis gearbeitet wird, ist der genaue Typ in diesem Fall „java.lang.String“. Diese Basis ermöglicht es überdies verschiedene Operatoren zu verwenden, da diese in allen

Programmiersprachen die Jaspersoft Studio unterstützt grundsätzlich verfügbar sind. Zum Beispiel gibt es den Ungleich-Operator („!=“), den boolschen Verneinungsoperator(„!“) und den Modulooperator („%“) sowohl in Java, als auch in Javascript und Groovey. [8]

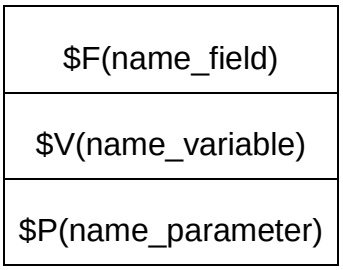


Abbildung 44: Syntax um auf Reportelemente zu referenzieren

```
(($F{name}.length() > 50) ? $F{name}.substring(0,50) : $F{name})
```

Abbildung 45: IF-ELSE Konstrukt

In Abbildung 44 finden sich Beispiele der Syntax verschiedener Reportelemente. Durch diese Syntax wird es möglich auf diesen Elementen Operationen auszuführen. Abbildung 45 zeigt die beispielhafte Verwendung eines IF-ELSE Konstrukts. Wenn die Bedingung, welche sich in der Klammer linksseitig des Fragezeichen befindet, den boolschen Wert „true“ ergibt, wird die erste Operation, welche sich rechts des Fragezeichen, aber links des Doppelpunkts befindet ausgeführt. Wenn die Bedingung den Wert „false“ ergibt, wird das Statement rechts des Doppelpunkts ausgeführt. In diesem Beispiel würde der Inhalt des Felds mit dem Schlüssel „name“ nur ausgeführt werden, wenn die Länge des Inhalts des Feldes einen Wert größer als 50 annimmt. Dieses Konstrukt ermöglicht es dem Nutzer noch mehr dynamische Elemente innerhalb eines Reports zu erzeugen und zu verwenden. Die drei bereits erwähnten und standardmäßig in Jaspersoft Studio unterstützen Sprachen verwenden bei unterschiedlichen Ausdrücken natürlich ihre jeweils eigene native Syntax und besitzen daher auch die für sie charakteristische Schreibweise. [8]

Die Möglichkeiten Jaspersoft Studio an die Bedürfnisse und Wünsche des Nutzers anzupassen drücken sich auch bei den zu verwendenden Schriftarten aus. Es gibt hierbei mehrere Optionen die dem Nutzer einen großen Spielraum bieten und daher auch keine gestalterischen Wünsche offenlassen. So ist zum Beispiel die Einbindung von externen Schriftarten in Jaspersoft Studio kein Problem. Überdies lassen sich Schriftartsätze erstellen, unterschiedliche Schriftarten für verschiedene Länder einstellen und auch die gewählten Optionen auf andere Plattformen exportieren. [8]

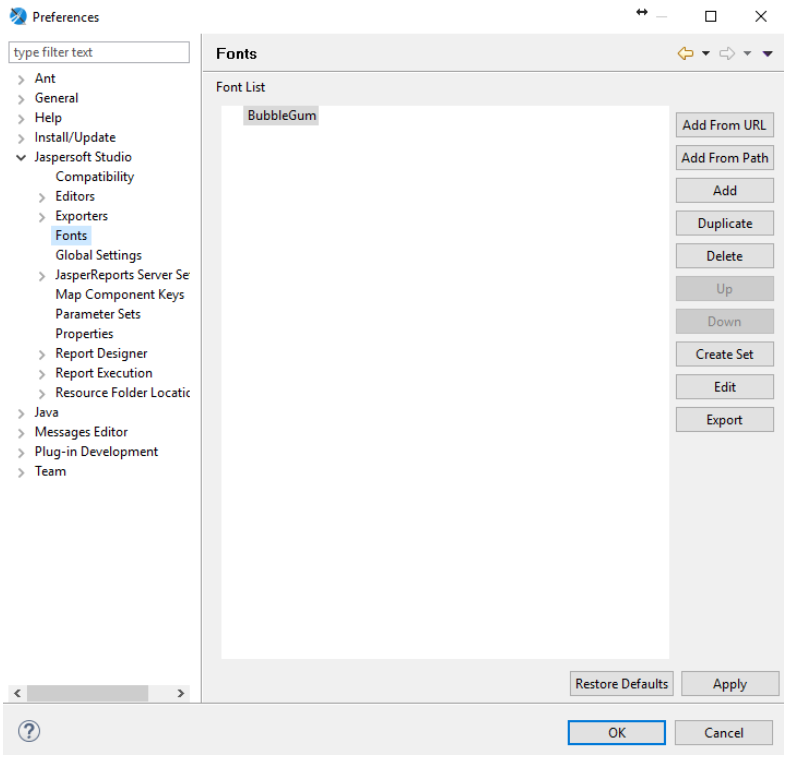


Abbildung 46: Verzeichnis hinzugefügter Schriftarten

Ein Kernelement von Jaspersoft Studio liegt, wie bereits erwähnt, in der Fähigkeit Daten und Design zu einem fertigen Report zusammenzuführen. Dabei können verschiedene Datenquellen gleichzeitig hinzugezogen und in Jaspersoft hinterlegt werden. Die hinterlegten Datenquellen werden dabei in einem XML-Format gespeichert und können dadurch auch zwischen mehreren Jaspersoft-Instanzen getauscht werden. [8]

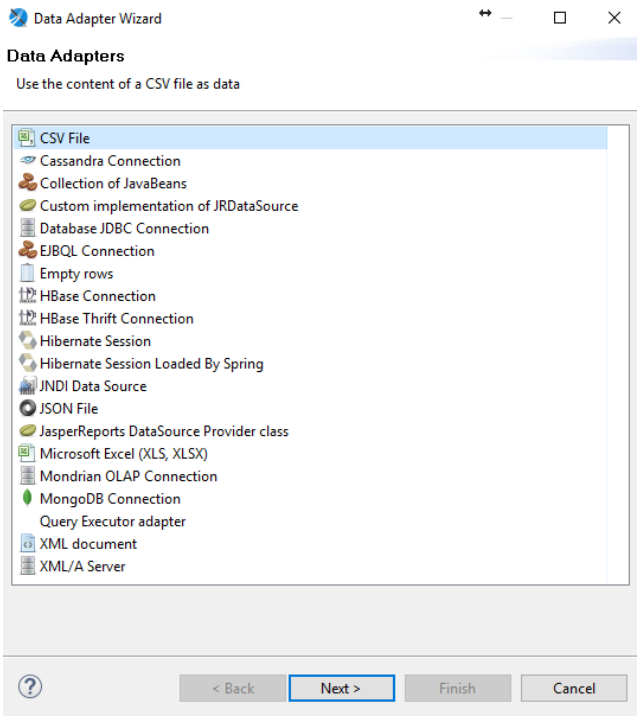


Abbildung 47: Überblick über verschiedene mögliche Datenquellen

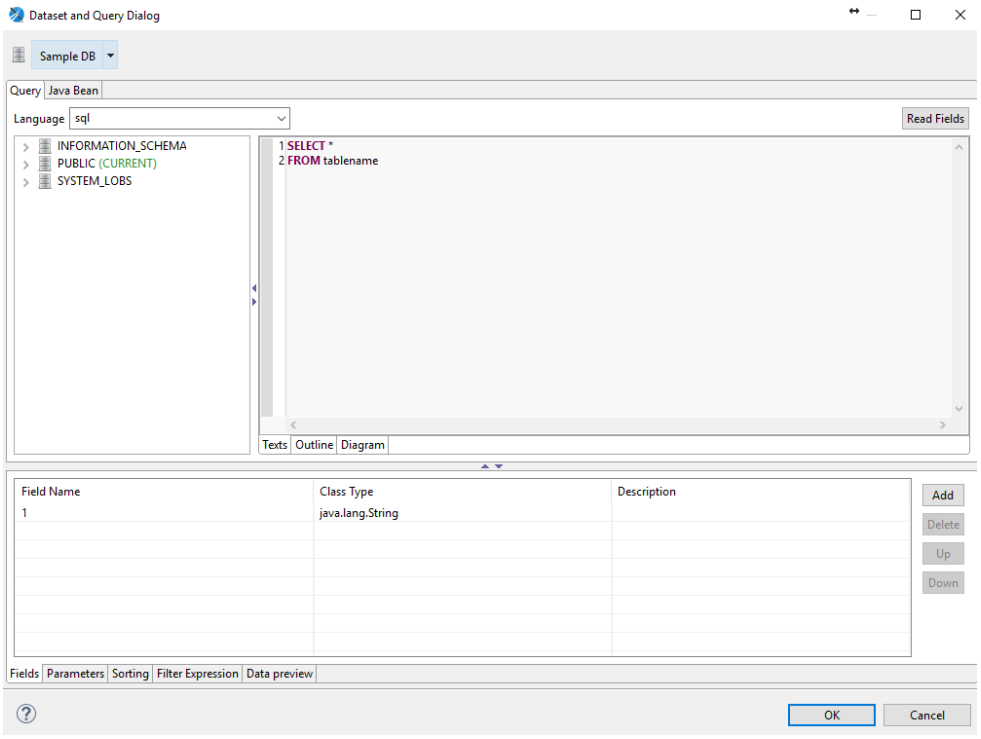


Abbildung 48: Auswahl von Daten aus der Datenbank

Eine Datenquelle lässt sich dabei recht einfach und komfortabel über einen Wizard einfügen (siehe Abbildung 47). Für einen Bericht ist es dabei auch möglich eine Standarddatenquelle zu nutzen. Folgend werden nun einige Datenquellen und ihre Besonderheiten aufgezeigt und vorgestellt. Die wohl einfachste und meist verwendete Art Daten in einen Report zu bringen ist die mittels eines Java Datenbank Verbinders, kurz JBDC. Dabei muss bei der Verwendung von diesem später noch eine Abfrage auf die gewünschten Daten stattfinden. Eine zweite

Möglichkeit eine Datenbankverbindung zu realisieren besteht durch den MongoDB Daten Adapter. MongoDB, eine NoSQL-Datenbank, erfreut sich dabei seit geraumer Zeit einer größer werdenden Nutzerschaft, auch weil hierbei eine eigene Abfragesprache zur Anwendung kommt. Eine weitere Datenbank, die mit speziellen Eigenschaften aufwartet, ist Cassandra von Apache, wobei hier CQL, ein Akronym für Cassandra Query Language, statt SQL für Datenabfragen angewendet wird. Innerhalb von Jaspersoft lässt sich auch noch eine für Java spezifische Art und Weise der Datenimportierung verwenden. Hierbei handelt es sich um JavaBeans, wobei eine JavaBean eine einfache Java-Klasse darstellt, die ihre Attribute nach außen hin bereitstellen kann. Eine weitere Methode umfasst die Verwendung von XML-Dokumenten um Daten auszulesen, wobei auf diese mittels XPath-Ausdrücke zugegriffen werden kann. Eine andere Möglichkeit ist CSV-Dateien, also Werte die durch Beistriche getrennt sind, zu importieren. Überdies besteht die Option eine eigene Art von Datenquelle selbst hinzuzufügen. [8]

Wie bereits in Abbildung 48 zu sehen war, gibt es bei der Verwendung von SQL als Datenbanksprache, eine eigene Oberfläche, die einem die Abfrage von Daten ermöglichen und erleichtern sollte. Dabei lassen sich Abfragen entweder in Textform und mittels SQL im Texts-Tab (siehe Abbildung 49) oder graphisch im Outline- beziehungsweise Diagram-Tab erstellen (siehe Abbildung 51). Hierbei lassen sich über diese Oberfläche alle gängigen SQL-Strukturen für Abfragen verwenden. [8]

```
1 SELECT "PRODUCT"."ID",
2   "PRODUCT"."NAME",
3   "PRODUCT"."COST"
4 FROM "PRODUCT"
```

Abbildung 49:Texts-Tab

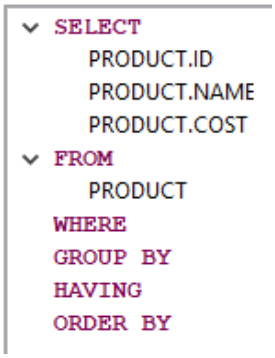


Abbildung 50:Outline-Tab

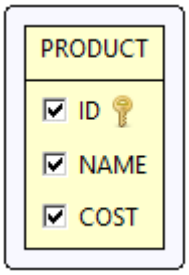


Abbildung 51:Diagram-Tab

In Jaspersoft Studio gibt es die besondere Möglichkeit Jaspersoft Studio und JasperReports Server miteinander zu verbinden, jedoch wird darauf die in dieser Arbeit nicht genau eingegangen, da diese sich hauptsachlich mit Jaspersoft Studio beschäftigt ist. Unter anderem ermöglicht diese Verbindung dem Nutzer auf Ressourcen, in diesem Fall Bilder, Reports und andere Daten des Servers, zuzugreifen. [8]

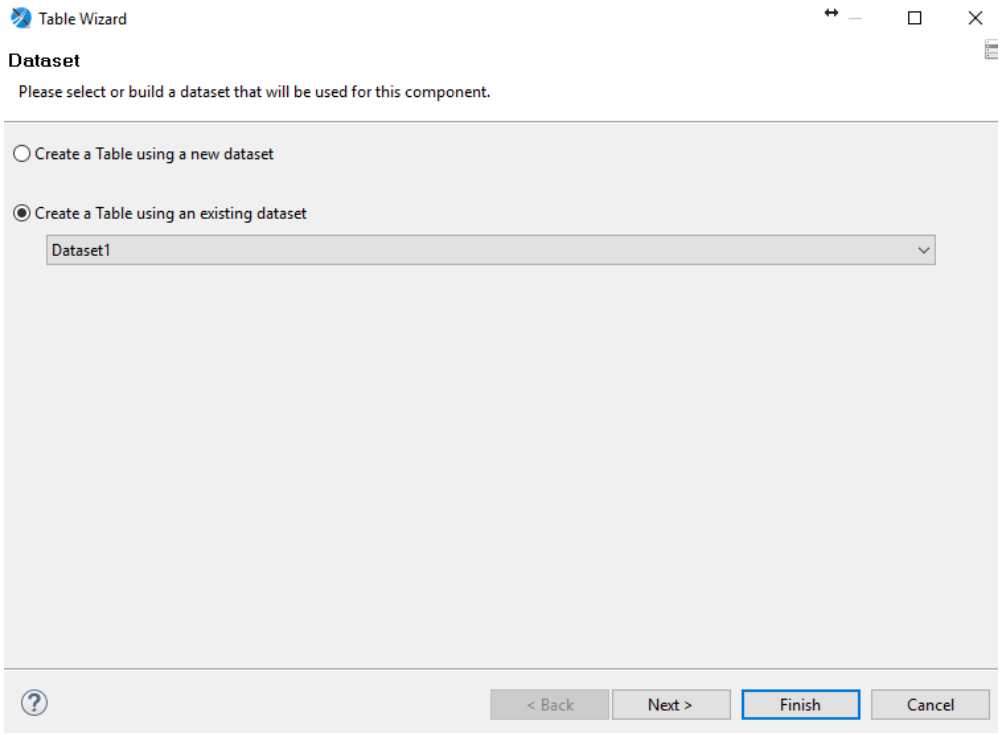


Abbildung 52: Tabellen-Wizard

Um die auf verschiedene Arten zugänglich gemachten Daten auch visuell verfügbar zu machen und in einer strukturierten Weise darzustellen ist es oft vonnöten auf Tabellen zu zugreifen. Tabellen ermöglichen dabei eine wie bei solchen Strukturen übliche Darstellung einer Matrix in Spalten und Zeilen. Um den Entwurf einer Tabelle dabei so einfach wie möglich zu gestalten, gibt es in Jaspersoft Studio die Möglichkeit einen Tabellen-Wizard zu verwenden (siehe Abbildung 52). Dieser wird automatisch, sobald man das Tabellen-Element aus der Palette in den Report zieht, aufgerufen. Im Ablauf des Wizard gibt es auch die Optionen gleich eine Verbindung zwischen einer Tabellen-Zelle und Daten herzustellen oder die Daten später manuell in diese einzufügen. Auf die verschiedenen Elemente einer Tabelle können dabei dann auch wieder individuelle Einstellungen über das jeweilige Einstellungs-Tab getroffen werden. [8]

Analog zu Tabellen gibt es auch noch Querverweistabellen, die auf eine ähnliche Art und Weise verwendet werden können. Querverweistabellen bieten im Gegensatz zu Tabellen jedoch den Vorteil, dass diese nicht nach einer Spalte oder einer Zeile gruppiert werden müssen. Denn Querverweistabellen ermöglichen es Daten nach einer Zeile und einer Spalte zu gruppieren, daher auch deren englische Ausdruck „crosstab“. Ein Anwendungsbeispiel hierzu wäre, dass Zeitnehmungswerte einer Laufveranstaltung mittels zweier Gruppen dargestellt werden, das heißt eine horizontale Zeilen-Gruppe die Zeiten in verschiedenen Sektoren angibt und eine vertikale Zeilengruppe die Übersicht über verschiedene

Teams liefert. Dabei ist es bei Querverweistabellen auch möglich über beide Gruppen, also Zeile und Spalte, eine Berechnung, zum Beispiel eine Summenberechnung zu bilden. Wie bei Tabellen gibt es auch hier viele weitreichende Tabelleneigenschaften. [8]

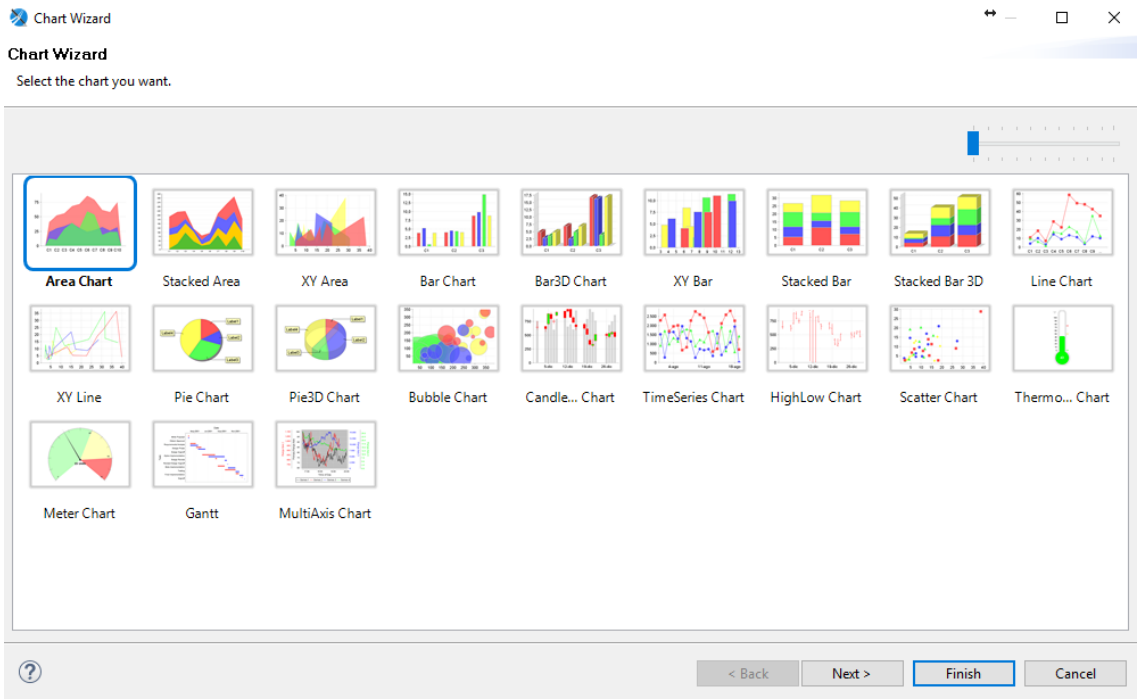


Abbildung 53: Diagram-Wizard

Neben Tabellen sind Diagramme eine beliebte zusätzliche Option Daten darzustellen. Neben den in Abbildung 53 dargestellten Diagrammarten gibt es in der kommerziellen Version von Jaspersoft Studio auch noch zusätzlich die Möglichkeit mit HTML5-Diagrammen zu arbeiten. Mit dieser Option werden eine Vielzahl von zusätzlichen Diagrammen angeboten, die auch mittels eines Diagram-Wizard eingefügt werden können. Dadurch ist es auch wieder relativ einfach möglich verschiedenste externe Daten in das Diagramm einzufügen und somit zu verarbeiten. [8]

Neben allen bereits behandelten Elementen können in Jaspersoft Studio außerdem Landkarten dynamisch mit Positionsdaten versehen und dadurch geographische Informationen einfacher zugänglich gemacht werden. Überdies ist es auch möglich mittels des Subreport-Elements einen Report innerhalb des aktuellen Reports zu platzieren, Barcodes zu erstellen und Listen zu nutzen. [8]

Sobald ein Report entworfen wurde, kann dieser für eine zukünftige Wiederverwendung in Jaspersoft Studio als Vorlage zu hinterlegt werden. Diese Vorlagen lassen sich dann natürlich auch in andere Instanzen von Jaspersoft Studio übertragen. Wenn danach ein neuer Bericht erstellt werden soll, ist es möglich auf diese Vorlage zurückzugreifen und auf diese Weise den Entwurfsprozess zu beschleunigen. In Abbildung 54 sind einige Templates die zur Auswahl stehen ersichtlich. [8]

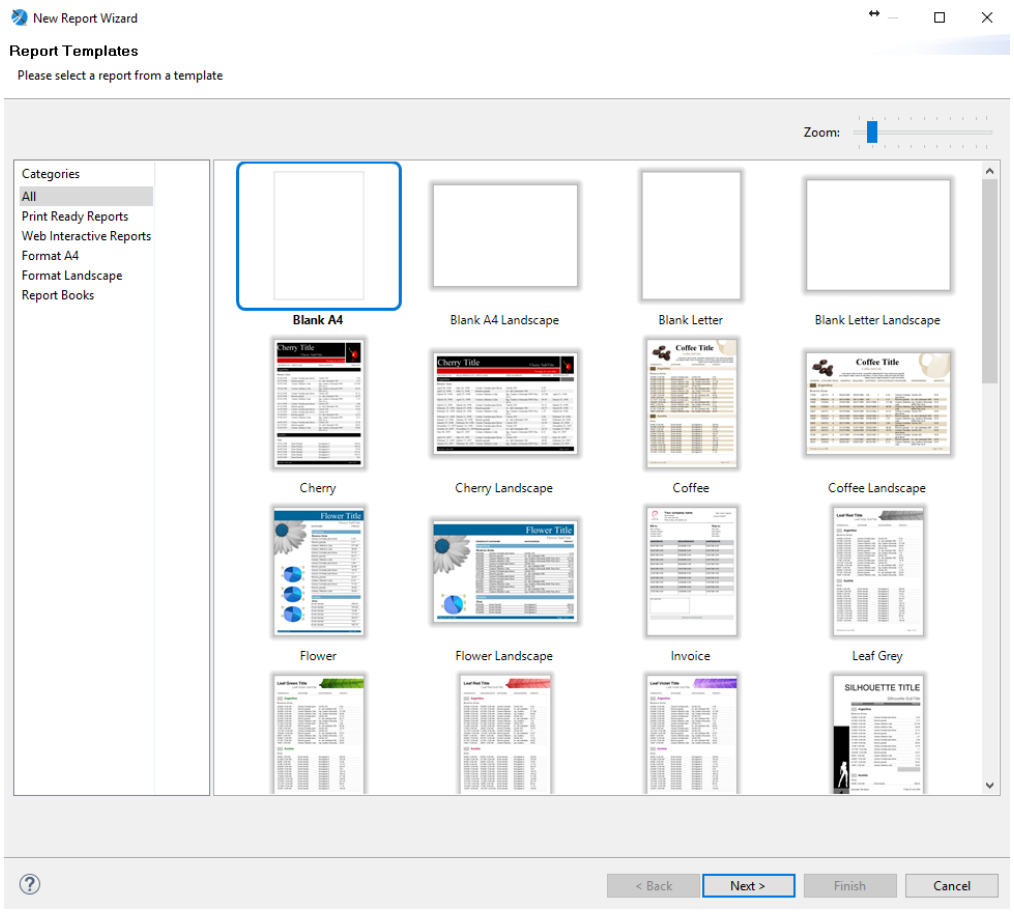


Abbildung 54: Auswahl an vorhandenen Templates

4. Praktischer Teil und Ausarbeitung

Nachdem in Kapitel 3 alle Grundlagen vorgestellt und eine praxisnahe Erklärung aller Dinge stattgefunden hat, ist nun ein Wissensgrundstock für die praktische Bearbeitung der Themen vorhanden. Das Ziel dieses Teils der Bachelorarbeit ist es, einen Blick auf die behandelten Themen des Projekts zu werfen und damit einhergehende Fragen, die praktische Seite des Projekts betreffend, verständlich zu präsentieren und zu beantworten.

4.1. XML

Dieses Kapitel widmet sich der praktischen Verwendung der Auszeichnungssprache XML innerhalb des Projekts. Das Ziel in diesem Kapitel ist es, dem Leser möglichst viele offene Fragen zu beantworten.

4.1.1. Verwendungszweck das FH Timing – Projekt

Da das Projekt die Anforderung hat die dynamische Erstellung von Templates beziehungsweise Berichten zu ermöglichen, ist es vor allem wichtig hervorzuheben, dass diese Anforderung nicht über ein internes Tool innerhalb von Jaspersoft Studio realisiert werden kann. Dazu ist der Entwurf einer Software vonnöten, die diese Aufgabe übernimmt. Daher ist die Verwendung des JasperReport-XML auch die einzig bekannte Möglichkeit dieses Projekt zu realisieren. Dieses bereits vorgestellte TIBCO eigene XML-Schema musste intensiv für diese Verwendung studiert und analysiert werden. Um diesen Prozess zu unterstützen bekam das Projektteam verschiedene Vorlagen. Mit diesen Vorlagen konnte bestimmt werden, welche Funktionen das Programm unterstützen sollte und somit wurde auch ein erster Eindruck über den Umfang des Projekts gewonnen. Zusätzlich hat der Projektauftraggeber weitere Programmfunktionalitäten festgelegt und mit dem Projektteam besprochen. Dabei wurde ersichtlich, dass verschiedenste XML-Elemente verwendet werden müssen und hauptsächlich Tabellen, Variablen, Bilder und Textfelder unterstützt werden sollten. Diese Elemente wurden dann vermehrt studiert und dabei zusätzliche Informationen ausgelesen.

```
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports
        http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
      name="My first report" pageWidth="595" pageHeight="842" columnWidth="535"
      leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20">
  <queryString language="SQL">
    <![CDATA[select * from address order by city]]>
  </queryString>
  <field name="ID" class="java.lang.Integer">
    <fieldDescription><![CDATA[]]></fieldDescription>
  </field>
```

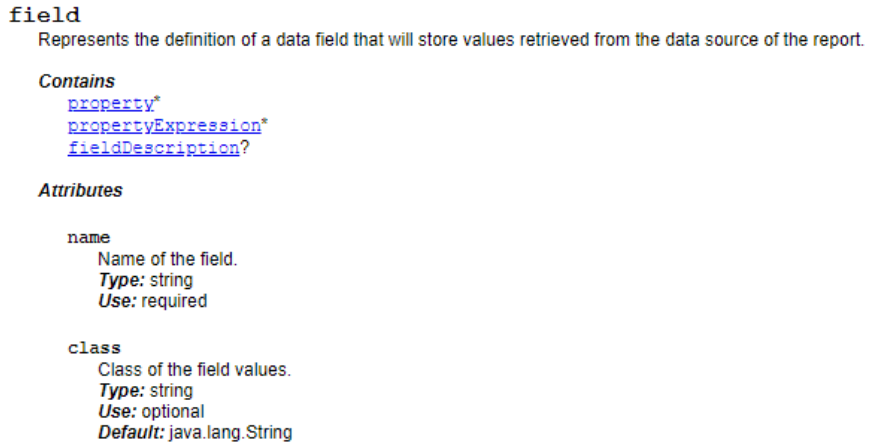
Abbildung 55: Auszug aus einer JRXML-Datei

In Abbildung 55 ist ein Auszug aus einer JRXML-Datei zu sehen, die zu Analysezwecken zu Rate gezogen wurde. Dabei stellte sich heraus, dass sich eine JRXML-Datei, genauso wie ein JasperReport, in logische und physisch vorhandene Teilbereiche aufsplittet. Zusätzliche Informationen durch diverse Beispiele, die sich

auf der der Webseite der Jaspersoft Community befinden, halfen diese Erkenntnisse zu gewinnen. Die Beispiele finden sich unter folgender URL:

https://community.jaspersoft.com/documentation/tibco-jaspersoft-studio-user-guide/v60/jrxml-sources-and-jasper-files

Mithilfe einer von TIBCO veröffentlichten Referenzliste für das verwendete XML-Schema wurde der Prozess auch zusätzlich unterstützt. Ein Auszug für die Verwendung eines Feldes innerhalb dieser Referenz lässt sich in Abbildung 56 einsehen.



field

Represents the definition of a data field that will store values retrieved from the data source of the report.

Contains

property*
propertyExpression*
fieldDescription?

Attributes

name
Name of the field.
Type: string
Use: required

class
Class of the field values.
Type: string
Use: optional
Default: java.lang.String

Abbildung 56: Beschreibung eines Felds

4.1.2. Blick in ein Beispieltemplate

In diesem Abschnitt werden alle Elemente, die in einige JRXML-Datei mittels XML abgebildet sind, erklärt. Somit wird ein praktischer Einblick in ein fertiges Template des TemplateGenerators gewährt. Alle Beispiele werden aus dem gleichen Template entnommen, wobei dieses schon aus der Projektpräsentation bekannt sein sollte. Das Template wird dabei segmentiert vorgestellt. Es werden nur Elemente und Attribute erklärt, die Jaspersoft Studio nicht standardmäßig benötigt.

```
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
              name="template" pageWidth="595" pageHeight="842" orientation="Portrait" whenNoDataType="AllSectionsNoDetail" columnWidth="555"
              leftMargin="20" rightMargin="20" topMargin="20" bottomMargin="20">
```

Abbildung 57: Wurzelelement

In Abbildung 57 ist das Wurzelelement dieser Datei abgebildet. Dieses besitzt Attribute die sowohl auf den Namespace, auf das zugrundeliegende XML-Schema, den Namen des Reports, die Größenangaben, die Reportausrichtung und die Seitenränder Hinweis geben.

```
<property name="ireport.zoom" value="1.0"/>
<property name="ireport.x" value="0"/>
<property name="ireport.y" value="0"/>
<property name="com.jaspersoft.studio.data.defaultdataadapter" value="One Empty Record"/>
<property name="com.jaspersoft.studio.layout" value="com.jaspersoft.studio.editor.layout.HorizontalRowLayout"/>
<property name="com.jaspersoft.studio.unit." value="pixel"/>
```

Abbildung 58: Verschiedene Property-Elemente

Die ersten drei Property-Elemente in Abbildung 58 bewirken, dass kein zusätzlicher Zoom angewendet wird und alle Positionsangaben bei x=0 und y=0 anfangen. Das vierte Element beschreibt, dass kein Standarddatenadapter verwendet wird und daher beim Kompilieren keine falschen Daten in den Report gespielt werden. Das letzte Element gibt an, dass die Standardmaßeinheit des Reports ein Pixel ist.

```
<style name="table">
    <box>
        <pen lineWidth="1.0" lineColor="#000000"/>
        <topPen lineWidth="0.0"/>
        <leftPen lineWidth="0.0"/>
        <bottomPen lineWidth="0.0"/>
        <rightPen lineWidth="0.0"/>
    </box>
</style>
<style name="table_TD" mode="Opaque" backcolor="#FFFFFF">
    <box>
        <topPen lineWidth="0.5" lineColor="#000000"/>
        <bottomPen lineWidth="0.5" lineColor="#000000"/>
    </box>
    <conditionalStyle>
        <conditionExpression><![CDATA[new Boolean($V{REPORT_COUNT}.intValue()%2==0)]]></conditionExpression>
        <style backcolor="#FFF7F7"/>
    </conditionalStyle>
</style>
<style name="table_CH" mode="Opaque" backcolor="rgba(240, 169, 163, 0.09803922)">
    <box>
        <topPen lineWidth="0.5" lineColor="#000000"/>
        <bottomPen lineWidth="0.5" lineColor="#000000"/>
    </box>
</style>
<style name="table_TH" mode="Opaque" forecolor="#FFFFFF" backcolor="rgba(240, 169, 163, 0.28627452)">
    <box>
        <topPen lineWidth="0.5" lineColor="#000000"/>
        <bottomPen lineWidth="0.5" lineColor="#000000"/>
    </box>
</style>
```

Abbildung 59: Style-Elemente

In Abbildung 59 sind alle Style-Elemente des Reports dargestellt. Das erste Style-Element bezieht sich auf die gesamte Tabelle und beschreibt die Farbe und Dicke der Außenlinien. Das zweite Style-Element mit dem name-Attribute-Wert „table_TD“ bezieht sich auf die Hintergrundfarben und Außenlinien. Mittels des genesteten conditionalStyle-Elements wird jede zweite mit Daten gefüllte Zeile mit einer anderen Hintergrundfarbe versehen. Dies wird durch den Java-Ausdruck innerhalb des CDATA-Abschnitts bewerkstelligt.

In Abbildung 60 werden Felder für verschiedene Zwecke angelegt. Das SubDataset-Element dient als Container für die jeweiligen Spaltenelemente und lässt sich über seinen Namen referenzieren. Für jede Spalte werden hierbei die Felder angegeben die verwendet werden sollten. Das Feld mit dem name-Attribut-Wert „tableData“ wird später dazu verwendet eine Datenquelle an ein Subdataset weiterzugeben.

```
<subDataset name="TableData" >
    <field name="str" class="java.lang.String">
        <fieldDescription><![CDATA[]]></fieldDescription>
    </field>
    <field name="Vorname" class="java.lang.String">
        <fieldDescription><![CDATA[]]></fieldDescription>
    </field>
    <field name="Nachname" class="java.lang.String">
        <fieldDescription><![CDATA[]]></fieldDescription>
    </field>
    <field name="stnr" class="java.lang.String">
        <fieldDescription><![CDATA[]]></fieldDescription>
    </field>
    <field name="Brand" class="java.lang.String">
        <fieldDescription><![CDATA[]]></fieldDescription>
    </field>
</subDataset>
<field name="tableData" class="net.sf.jasperreports.engine.JRDataSource">
    <fieldDescription><![CDATA[DataSource for dataset run for the main table]]></fieldDescription>
</field>
```

Abbildung 60: Datenfelder der Tabelle

```
<variable name="top0Image" class="java.lang.String">
    <variableExpression><![CDATA["iVBORw0KGgoAAAANSUhEUgAAD...
</variable>
<variable name="bottom1Image" class="java.lang.String">
    <variableExpression><![CDATA["iVBORw0KGgoAAAANSUhEUgAAD...
</variable>
```

Abbildung 61: Variablen-Definition für Bilder

Um Base64-Format codierte Bilder abzuspeichern werden die Zeichenfolgen innerhalb eines CDATA-Abschnitts gespeichert und die abgelegten Daten über einen eindeutigen Wert des name-Attributs referenziert (siehe Abbildung 61).

```
<pageHeader>
    <band height="66">
        <property name="local_mesure_unitheight" value="pixel"/>
        <property name="com.jaspersoft.studio.unit.height" value="px"/>
        <image scaleImage="FillFrame" hAlign="Center" evaluationTime="Report">
            <reportElement stretchType="RelativeToBandHeight" x="0" y="0" width="555" height="40" uuid="df111196-a65c-4547-bdf6-5e8c979b0fd5">
                <property name="local_mesure_unitheight" value="pixel"/>
                <property name="com.jaspersoft.studio.unit.height" value="px"/>
                <property name="local_mesure_unitx" value="pixel"/>
                <property name="com.jaspersoft.studio.unit.x" value="px"/>
                <property name="local_mesure_unitwidth" value="pixel"/>
                <property name="com.jaspersoft.studio.unit.width" value="px"/>
            </reportElement>
            <imageExpression><![CDATA[new java.io.StringBufferInputStream( new org.w3c.tools.codec.Base64Decoder($V{top0Image}).processString() )]]></imageExpression>
        </image>
        <staticText>
            <reportElement x="430" y="40" width="125" height="26" forecolor="#000000">
                <property name="local_mesure_unity" value="pixel"/>
                <property name="com.jaspersoft.studio.unit.y" value="px"/>
            </reportElement>
            <textElement textAlignment="Right" verticalAlignment="Bottom">
                <font fontName="SansSerif" size="14" isBold="false" isItalic="true"/>
            </textElement>
            <text><![CDATA[31. Jänner 2017]]></text>
        </staticText>
```

Abbildung 62: Ausschnitt des Pageheaders

In Abbildung 62 wird ein Ausschnitt des Pageheaders dargestellt. Innerhalb diesem ist besonders das Image-Element interessant, da dieses mithilfe des imageExpression-Elements die Bilddarstellung ermöglicht. Diese wird durch den Java-Ausdruck innerhalb des CDATA-Abschnitts ermöglicht, weil dieser einen Inputstream bereitstellt der mit einem Base64-Format decodiertem Bild gespeist wird. Die Referenzierung geschieht hierbei wieder über das name-Attribut des Bildes. Andere Attribute und Elemente innerhalb des image-Elements sind unter anderem für die Formatierung zuständig. Unter diesem Element findet sich die erste Deklaration eines statischen Textfeldes.

```
<detail>
    <band height="80">
        <property name="com.jaspersoft.studio.layout" value="com.jaspersoft.studio.editor.layout.FreeLayout"/>
        <property name="com.jaspersoft.studio.unit.height" value="pixel"/>
        <staticText>
            <reportElement mode="Opaque" x="0" y="-21" width="802" height="16" forecolor="#000000"
                           backcolor="#E6E6E6" uuid="5e7bae91-ce1b-40f0-9a4f-0a4a003119f7"/>
            <textElement verticalAlignment="Middle">
                <font fontName="SansSerif" size="12" isBold="true"/>
            </textElement>
            <text><![CDATA[Mixed gender]]></text>
        </staticText>       <componentElement>
```

Abbildung 63: Ausschnitt Nr.1 des Detail

Auf den Pageheader folgend findet sich der detail-Abschnitt, der in diesem Beispiel fast ausschließlich für die Darstellung von Tabellen genutzt wird. Das Tabellen-Element wird in ein componentElement-Element genestet. Bevor jedoch im detail-Abschnitt das Tabellen-Element in Abbildung 64 erstmalig auftritt, ist wiederum ein statischer Text vorangestellt. Dieser ist in Abbildung 63 abgebildet.

```
<jr:table xmlns:jr="http://jasperreports.sourceforge.net/jasperreports/components"
          xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports/components
                              http://jasperreports.sourceforge.net/xsd/components.xsd"
          whenNoDataType="AllSectionsNoDetail">
    <datasetRun subDataset="TableData" uuid="fa979858-ab87-4488-8889-d40ed5a07047">
        <dataSourceExpression><![CDATA[$F{tableData}]]></dataSourceExpression>
    </datasetRun>
```

Abbildung 64: Ausschnitt Nr.2 des Detail

In Abbildung 64 sticht allem voran das datasetRun-Element ins Auge. Dieses übernimmt die Bereitstellung von Instanziierungs-Information des Subdatensatzes, der bereits in Abbildung 60 aufschien, für einen Tabellendatensatz innerhalb von JaspersoftReport.

```
<jr:column width="267">
    <property name="local_mesure_unitwidth" value="pixel"/>
    <property name="com.jaspersoft.studio.components.table.model.column.name" value="Column2"/>
    <jr:columnHeader style="table_TH" height="20" rowSpan="1">
        <property name="com.jaspersoft.studio.unit.width" value="px"/>
        <staticText>
            <reportElement x="0" y="0" width="267" height="20" forecolor="#000000"/>
            <textElement verticalAlignment="Bottom">
                <font size="12" isBold="true"/>
            </textElement>
            <text><![CDATA[Name]]></text>
        </staticText>
    </jr:columnHeader>
    <jr:detailCell style="table_TD" height="20" rowSpan="1">
        <textField isBlankWhenNull="true">
            <reportElement x="0" y="0" width="267" height="20"/>
            <textElement textAlignment="Center" verticalAlignment="Middle">
                <font size="12" isBold="true"/>
            </textElement>
            <textFieldExpression><![CDATA[$F{Vorname}+ " " +$F{Nachname}+ " " +$F{stnr}]]></textFieldExpression>
        </textField>
    </jr:detailCell>
</jr:column>
```

Abbildung 65: Deklaration einer Spalte

Innerhalb des Tabellen-Elements scheinen daraufhin die definierten Spalten auf. In Abbildung 65 ist ein Beispiel dafür zu sehen. Jede Spalte gliedert sich in ein columnHeader- sowie ein deatilCell-Element. Im columnHeader-Element wird ein statischer Text definiert, in dem die Spaltenüberschrift Platz findet und im CDATA-Abschnitt definiert wird. Innerhalb des detailCell-Elements findet sich das textFieldExpression-Elements innerhalb dessen CDATA-Abschnitts die

verschiedenen zu verwendenden Felder Platz finden. Diese Felder waren ebenfalls bereits in Abbildung 60 zu sehen.

```
<pageFooter>
    <band height="70">
        <property name="local_mesure_unitheight" value="pixel"/>
        <textField pattern="dd.MM.yyyy HH:mm:ss">
            <reportElement x="1" y="0" width="190" height="13" uuid="4a5dd045-24e5-4750-9ef4-21db138dab50"/>
            <textFieldExpression><![CDATA[new java.util.Date()]]></textFieldExpression>
        </textField>
        <staticText>
            <reportElement x="331" y="0" width="150" height="13" uuid="20dfebad-caf1-407f-9ab8-c477c7bdfed4"/>
            <textElement textAlignment="Center" verticalAlignment="Bottom">
                <font size="10"/>
            </textElement>
            <text><![CDATA[www.englisch.com]]></text>
        </staticText>
        <textField>
            <reportElement x="650" y="0" width="100" height="13" uuid="5cd3afd5-6849-4b41-bad9-e43fc01102ea"/>
            <textElement textAlignment="Right"/>
            <textFieldExpression><![CDATA["Page " + $V{PAGE_NUMBER}]]></textFieldExpression>
        </textField>
        <textField evaluationTime="Report">
            <reportElement positionType="FixRelativeToBottom" x="750" y="0" width="45" height="13"/>
            <textElement textAlignment="Left"/>
            <textFieldExpression><![CDATA[" of " + $V{PAGE_NUMBER}]]></textFieldExpression>
        </textField>
```

Abbildung 66: Ausschnitt des pageFooters

In Abbildung 66 ist das pageFooter-Element zu sehen, das neben dem pageHeader-Element und dem detail-Element für die physische Unterteilung des Reports in 3 Teile sorgt. In diesem Beispiel sind vor allem die drei Textfelder von besonderer Bedeutung, da diese zusammengesetzte Elemente sind (siehe „zusammengesetzte Elemente“ in Kapitel **Fehler! Verweisquelle konnte nicht g efunden werden.**). Das erste Textfeld in Abbildung 66 führt dazu, dass durch den CDATA-Ausdruck das aktuelle Datum in Textform in den Report eingefügt wird. Das zweite und dritte Textfeld überlagern sich im Design so geschickt, dass somit folgender Ausdruck entstehen kann: „Page 1 of 2“. Wobei das zweite Textfeld immer die aktuelle Seite des Berichts und das dritte die Gesamtseitenanzahl angibt.

All diese Einstellungen führen dazu, dass sobald die mit dem TemplateGenerator erstellte JRXML-Datei mittels Jaspersoft Studio kompiliert wird das Ergebnis als visuell aufbereitete Endversion vorliegt. Die kann in Jaspersoft Studio unter dem Preview-Tab eingesehen und exportiert werden. In Abbildung 67 ist eine graphische Darstellung des Beispielreports zu sehen. Hierbei ist jedoch anzumerken, dass innerhalb der Tabelle in diesem Fall keine Daten enthalten sind, da diese erst durch eine passende Datenquelle hinzugefügt werden müssen. Das Hinzufügen von Daten erfolgt dabei nach Erstellung des Reports nach den Konventionen von FH Timing.

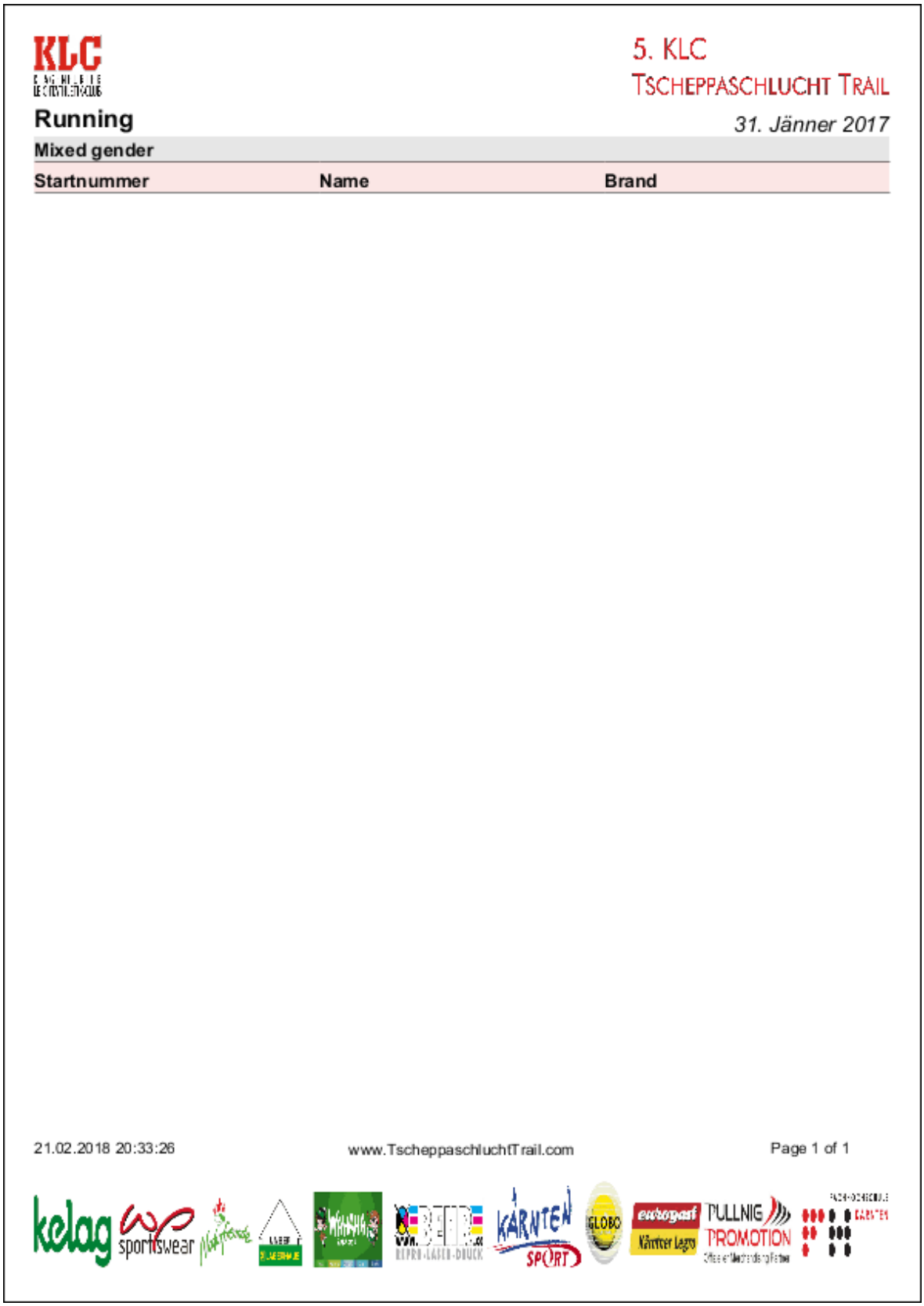


Abbildung 67: Beispielbild des Reports

Leider ist die Qualität der Bilder aus Abbildung 67 und Abbildung 78 nicht besonders hochwertig, jedoch liegt dies an der Funktion von Jaspersoft Studio einen Report in einem Bild zu exportieren.

4.2. Jaspersoft Studio Community Edition

Dieses Kapitel widmet sich der praktischen Verwendung von Jaspersoft Studio in der Community Edition und deren praktischer Verwendung in der Projektarbeit. Das Ziel in diesem Kapitel ist es, den Leser möglichst grundsätzlich über die Chancen und Möglichleiten von Jaspersoft Studio aufzuklären.

4.2.1. Verschiedene Verwendungszwecke

Der Verwendungszweck von Jaspersoft Studio innerhalb von FH Timing steht exemplarisch für die Einfachheit der visuellen Aufbereitung von Daten. Durch Jaspersoft Studio ist nicht mehr die Visualisierung von Daten das Problem und man kann sich in der Entwicklung einer datenverarbeitenden Anwendung auf andere Teilbereiche konzentrieren. Durch Jaspersoft wird jedoch die Struktur und Architektur der Datensammlung sowie die Einbindung dieser Daten wichtiger und gewinnt an enormer Bedeutung. Eine weitere Herausforderung im Zusammenhang mit JasperReports ist auch die automatisierte Erstellung von Bericht-Instanzen.

Die Verwendung von JasperReports und damit auch von Jaspersoft Studio sind keine kreativen Grenzen gesetzt.

Dabei steht vor allem die Integration der Reporttechnologie in Anwendungen im Vordergrund. Mit Hilfe von Jaspersoft Studio kann zum Beispiel eine mobile Applikation eine Datensammlung über die Zeit dynamisch, visuell ansprechend und mit relativ wenig Aufwand dargestellten. Ein Paradebeispiel für einen solchen Verwendungszweck lässt sich in der Android-Applikation der Fachhochschule Kärnten mit dem Namen „studentsLife“ ausmachen. Nach der Benotung einer Klausur beziehungsweise einer Prüfung werden die Notengrade aller Teilnehmer in Form eines Kuchendiagramms dargestellt. Dadurch ist graphisch sofort ersichtlich wie viele „Sehr gut“, „Gut“ et cetera durch alle Examinanden erzielt wurden.

Ein weiterer exemplarischer Anwendungsfall in dem Softwarelösungen der Jaspersoft-Familie häufig zur Anwendung kommen ist innerhalb einer Web-Applikation. Hier wäre ein Anwendungsbeispiel eine Verwaltungsoberfläche innerhalb einer Netzwerkkomponente die diverse Parameter mittels verschiedener Graphiken und Diagramme darstellen kann. Generell könnte jede Benutzeroberfläche, die eine übersichtliche Darstellung von Daten benötigt von den Möglichkeiten die Jaspersoft Studio bietet einfach Gebrauch machen.

Dabei ist es auch mit Hilfe der Performanz von JasperReports Server möglich eine große Anzahl von verschiedenen Berichten innerhalb eines großen Unternehmens in kürzester Zeit zu erstellen. [9]

4.2.2. Anwendung in der Projektarbeit

Innerhalb der Projektarbeit kam Jaspersoft Studio durch das Projektteam nur eingeschränkt und für spezielle Zwecke zur Anwendung, da durch den Lernprozess, welcher vor allem aufgrund der vorhandenen JRXML-Dateien und der XML-Schema-Referenz angestoßen wurde, nur noch bei offenen Fragen auf Jaspersoft Studio zurückgegriffen werden musste. Jaspersoft Studio wurde, wenn benötigt, nur dazu genutzt unbekannte Element-Konstruktionen zu bestimmen und zu verstehen. Dies wird in Jaspersoft Studio vorrangig dadurch ermöglich, dass ein einzelnes benötigtes Element in einen leeren Bericht eingefügt wird und dadurch nur dessen Struktur in Unabhängigkeit von anderen Elementen beleuchtet werden kann. Die größte Verwendung in der Praxis fand anfangs das XML-Schema, welches mit über 8000 Zeilen an Schema-Deklarationen sehr unübersichtlich ist und daher nicht für die komplette Wissensgewinnung praktikabel einzusetzen war. Nach der Erkenntnis der Existenz einer Schema-Referenz wurde diese sehr intensiv genutzt. Diese Referenz-Liste wurde vorrangig dazu verwendet um Attribute und Stil-Optionen besser zu verstehen, weil diese meist eine textuelle Beschreibung aufweisen. Nachdem alle zu verwendenden Elemente in einem für das Projekt erforderlichen Umfang verstanden wurden, lag der Schwerpunkt auf den optischen Optionen die Jaspersoft Studio bietet. Für diesen Zweck bietet es sich wieder an das gewünschte Element in einen leeren Jaspersoft Studio Bericht zu ziehen und dann mittels dem Properties-Tab gewisse Einstellungen vorzunehmen. Diese Verwendung ist in Abbildung 68 zu sehen.

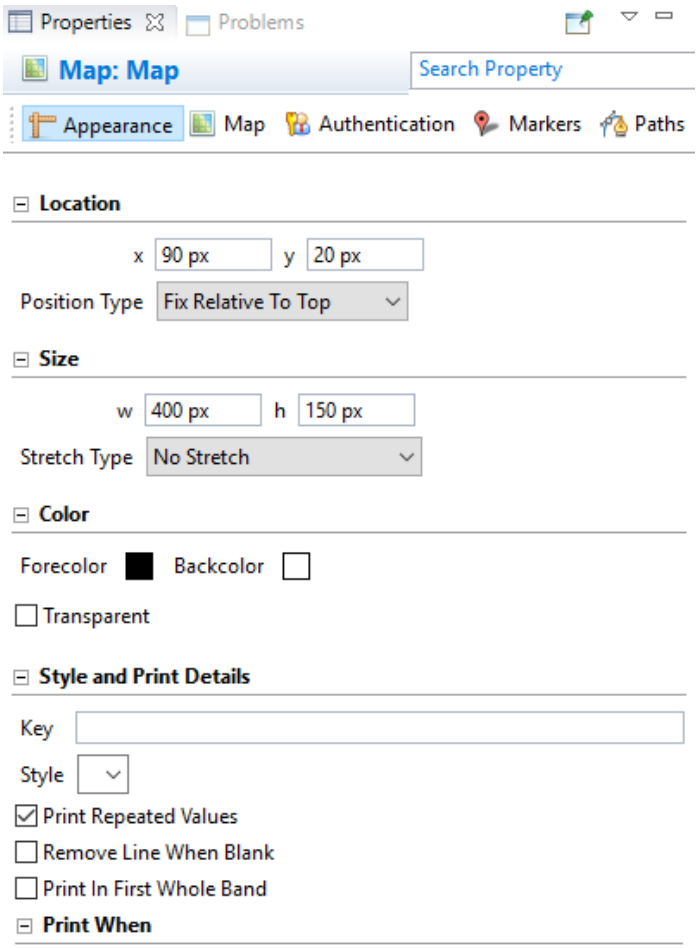


Abbildung 68: Verwendung des Properties-Tab

Hierbei war meist nur der Unterpunkt Appearance für das generelle Erscheinungsbild des Elements wichtig. Bei speziellen Elementen wie bei einer Landkarte oder einem Bild waren überdies meist auch noch die spezifischen Unterpunkte sehr beachtenswert. In Abbildung 69 kann der Image-Unterpunkt eines Bildes mit all seinen wichtigen Bild-Einstellungen eingesehen werden. Bei Bildern sind vorrangig das Fill- und das Scale-Image-Attribut interessant, welches visuelle Bildoptionen betrifft.

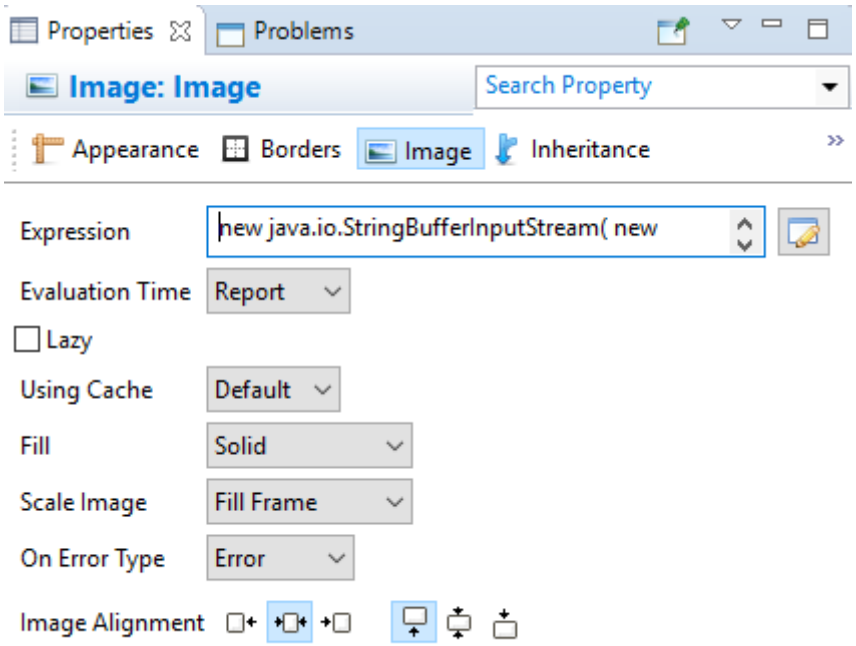


Abbildung 69: Image-Unterpunkt eines Bildes

4.2.3. Erstellung eines JasperReports

In diesem Kapitel wird eine beispielhafte Erstellung eines JasperReports innerhalb von Jaspersoft Studio vorgenommen. Dies sollte vorrangig dazu dienen, den Leser von der Möglichkeit eines leichten Einstiegs in Jaspersoft Studio zu überzeugen und diesen zu unterstützen.

1. Erstellung eines neuen Berichts:
    - „File“ → „New“ → „Jasper Report“
    - Nun öffnet sich dasselbe Fenster wie in Abbildung 54.
    - Ein vorhandenes Template oder einen leeren Report auswählen und klicken Sie dann auf „Next“ klicken.
    - Daraufhin öffnet sich der in ersichtliche Dialog, der mit einem Klick auf „Next“ quittiert wird (siehe Abbildung 70).
    - Danach öffnet sich ein Fenster in dem der Benutzer aufgefordert wird eine Datenquelle anzugeben und diese zu spezifizieren. Da dieser Punkt zu einem späteren Zeitpunkt ausgeführt wird, sollte das Feld „Finish“ gedrückt werden.

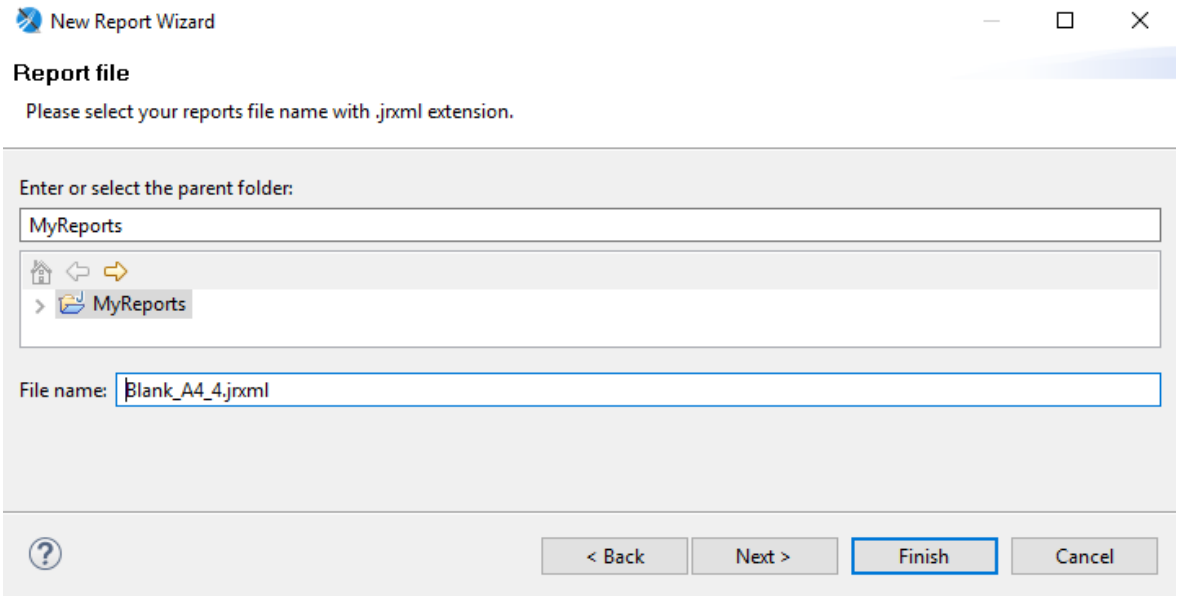


Abbildung 70: Erstellen einer Report-Datei und eines Projektordners

2. Erstellen eines Tabellen-Elements

   - Ziehen eines Tabellen-Elements aus der Palette in die Reportoberfläche.
   - Nun wird man in einem Fenster dazu aufgefordert einen neuen Datensatz anzulegen. Dafür sollte mit „Next“ fortgefahren werden.

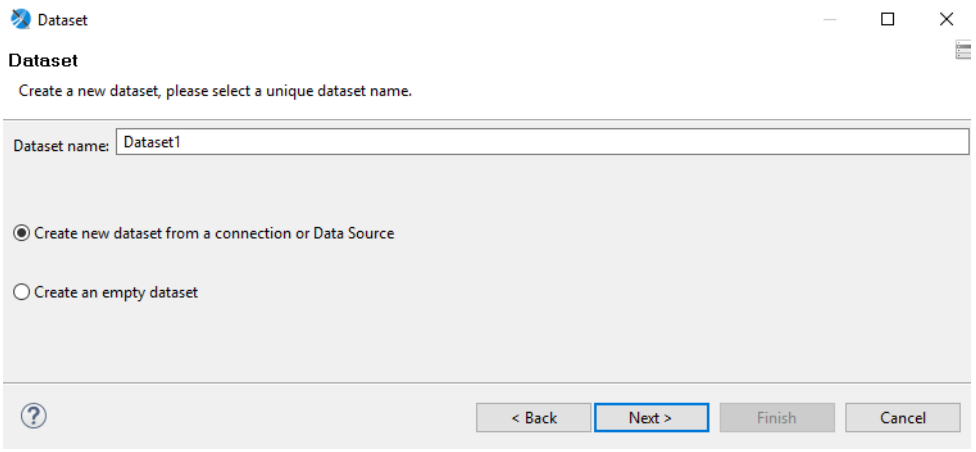


Abbildung 71: Erstellung eines neuen Datensatzes

   - Danach wird man, wie in Abbildung 71 gezeigt, aufgefordert dem Datensatz einen Namen zu geben. Wiederrum sollte mit „Next“ im Wizard vorangeschritten werden.

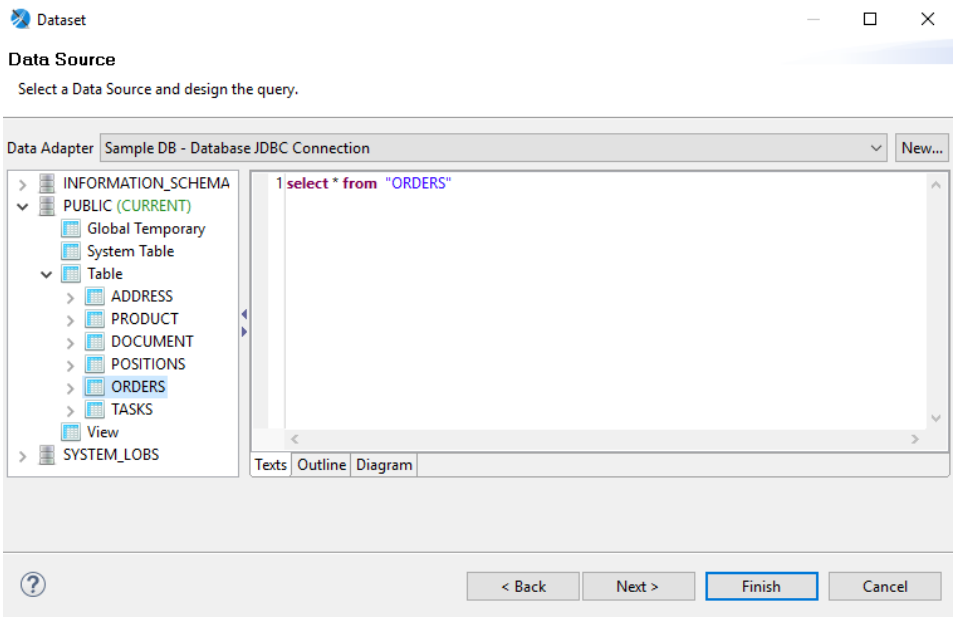


Abbildung 72: Genaue Auswahl der Datenquelle mittels einer Abfrage

- Wie in Abbildung 72 zu sehen, sollte nun auf die Daten der Datenbank eine Abfrage generiert werden. Dabei wird eine SQL-Abfrage verwendet und der Wizard-Verlauf mit „Next“ weitergelaufen.

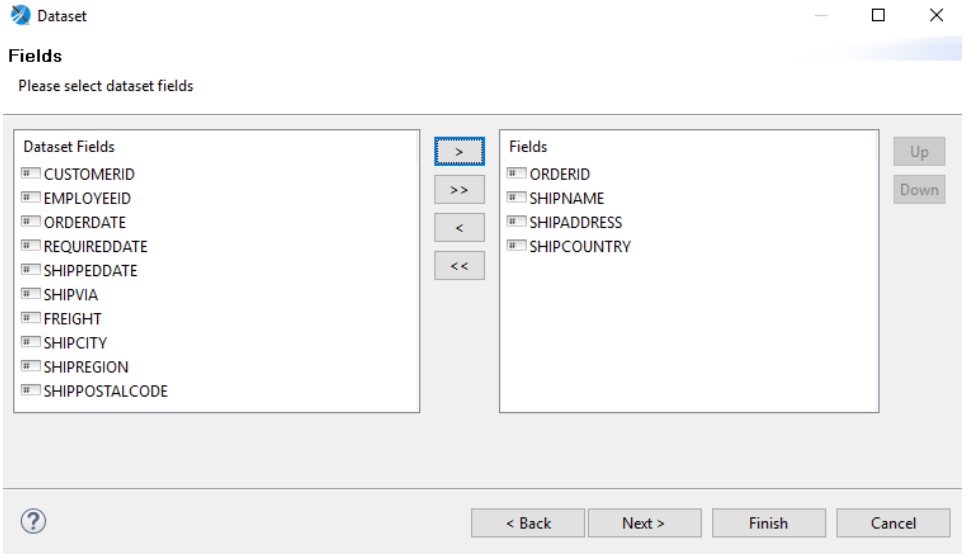


Abbildung 73: Auswahl der zu verwendeten Felder

- Im nächsten Schritt sollten dann die gewünschten Felder der Datenbank-Tabelle in den rechten Feldbereich übertragen werden. Wenn diese Auswahl getroffen wurde sollte mit „Next“ fortgefahren werden.

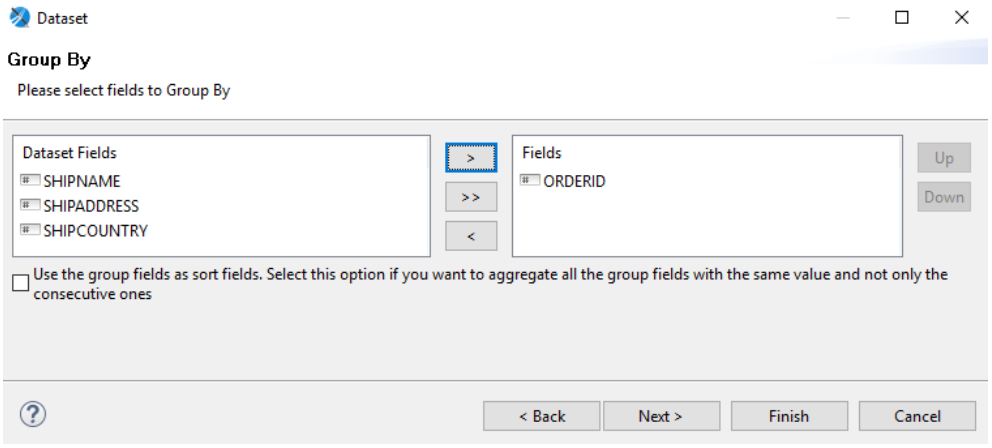


Abbildung 74: Auswahl des Feldes nachdem Gruppiert werden soll

- In dem nun geöffneten Dialog, welcher in Abbildung 74 zu sehen ist, kann ein Feld gewählt werden, nach dessen Wert die Daten-Einträge gruppiert werden sollten.

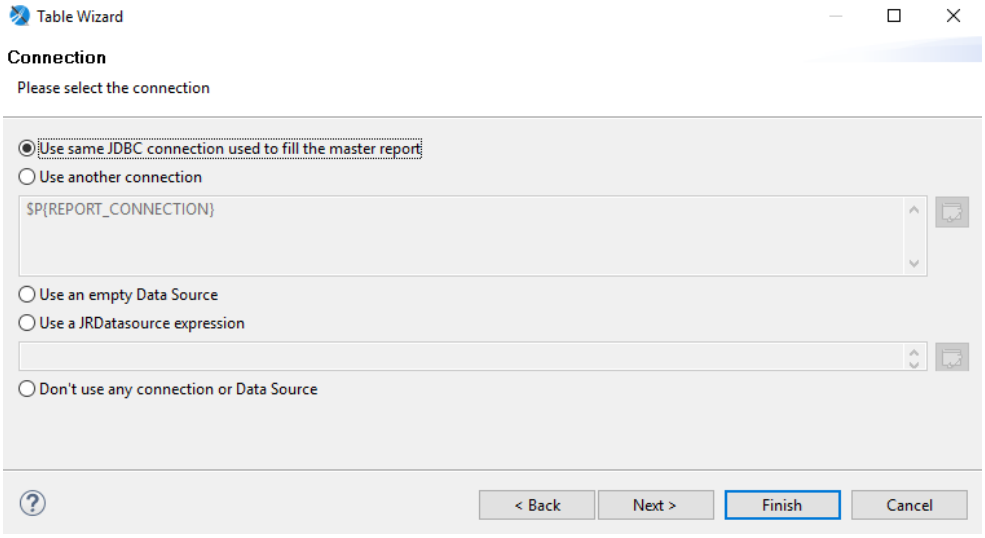


Abbildung 75: Setzen der Verbindung

- Wie in Abbildung 75 zu sehen, muss eine Verbindung ebenfalls angegeben werden. Danach kann der Tabellen-Wizard mit einem Klick auf „Next“ in die nächste Phase übergehen.

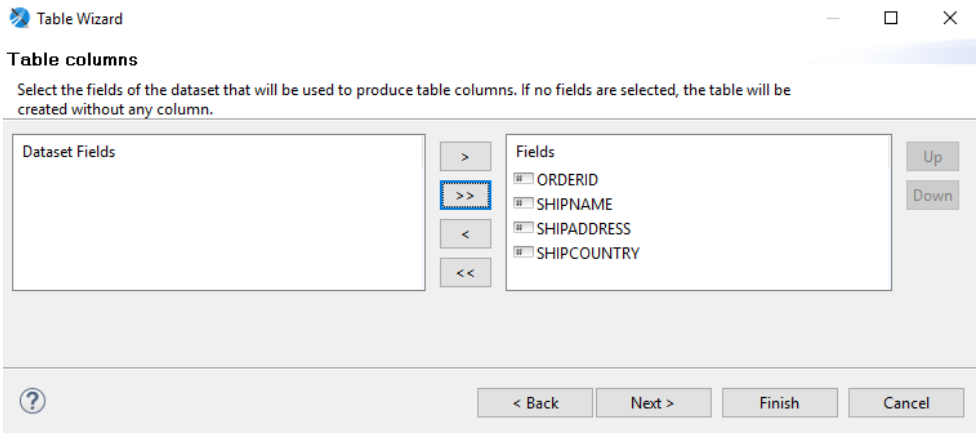


Abbildung 76: Auswahl der verwendeten Datenfelder

- In dem in Abbildung 76 geöffneten Dialogfenster ist es möglich die Felder auszuwählen, die später in der Tabelle aufscheinen sollen. Wird dieser Schritt mit „Next“ abgeschlossen, öffnet sich ein weiteres Fenster.

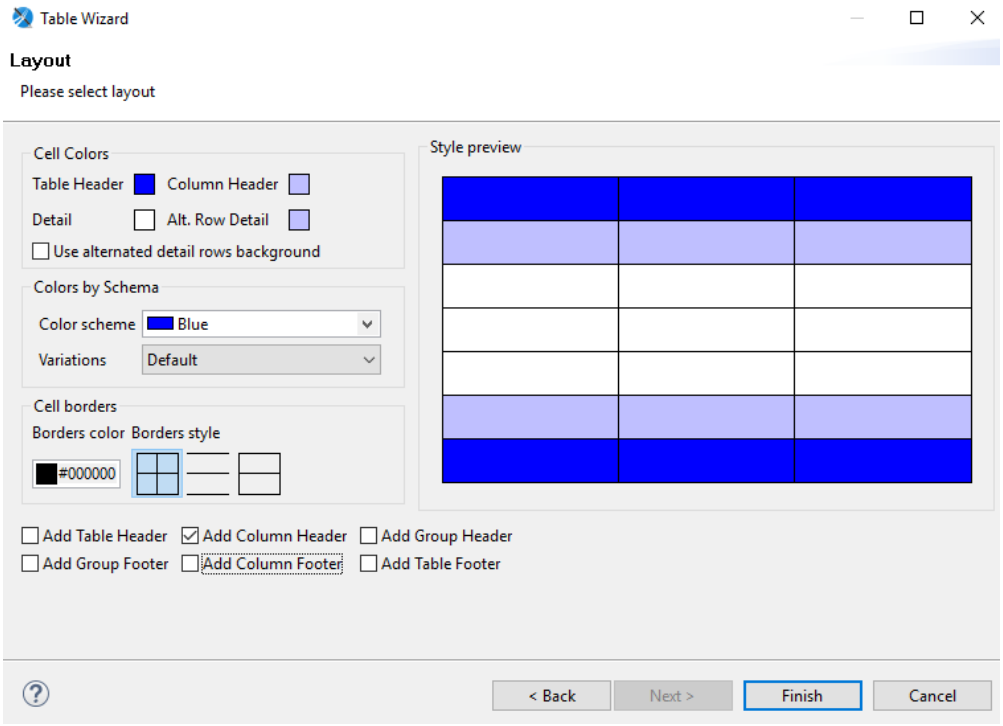


Abbildung 77: Bestimmung des Layouts der Tabelle

- In diesem Fenster geht es darum eine erste Voreinstellung über den Stil der Tabelle zu treffen. Dies ist in Abbildung 77 zu sehen. Hierbei muss bedacht werden, dass die Anzeige im rechten Bildteil nicht immer der aktuellen Auswahl entspricht.
- Mit einem Tastenklick auf „Finish“ wird der Wizard abgeschlossen und die erstellte Tabelle im Bericht angezeigt.
- Jetzt kann die Optik des Berichtes angepasst und weitere Elemente hinzugefügt werden.

Exporte

ID der Bestellung	Schiff	Empfangsort	Empfangsland
10248	Vins et alcools Chevalier	59 rue de l'Abbaye	France
10249	Toms Spezialitäten	Luisenstr. 48	Germany
10250	Hanari Carnes	Rua do Paço, 67	Brazil
10251	Victuailles en stock	2, rue du Commerce	France
10252	Suprêmes délices	Boulevard Tirou, 255	Belgium
10253	Hanari Carnes	Rua do Paço, 67	Brazil
10254	Chop-suey Chinese	Hauptstr. 31	Switzerland
10255	Richter Supermarkt	Starenweg 5	Switzerland
10256	Wellington Importadora	Rua do Mercado, 12	Brazil
10257	HILARION-Abastos	Carrera 22 con Ave. Carlos Soublette #8-	Venezuela
10258	Ernst Handel	Kirchgasse 6	Austria
10259	Centro comercial Moctezuma	Sierras de Granada 9993	Mexico
10260	Ottilies Käseladen	Mehrheimerstr. 369	Germany
10261	Que Delícia	Rua da Panificadora, 12	Brazil
10262	Rattlesnake Canyon Grocery	2817 Milton Dr.	USA
10263	Ernst Handel	Kirchgasse 6	Austria
10264	Folk och fä HB	Åkergatan 24	Sweden
10265	Blondel père et fils	24, place Kléber	France
10266	Wartian Herkku	Torikatu 38	Finland
10267	Frankenversand	Berliner Platz 43	Germany

Februar 22, 2018 **Page 1**

Abbildung 78: Bild des erstellten Reports

- Um den Bericht in Abbildung 78 zu erstellen, müssen noch eine Überschrift und zwei zusammengesetzte Elemente eingefügt werden.

5. Abschluss und erreichte Ziele

In diesem Kapitel wird genauer auf die erreichten Ziele und den Projektabschluss eingegangen.

Das Projekt wurde offiziell am 31.Jänner 2018 mit einer Präsentation abgeschlossen. Diese Präsentation wurde vor Kommilitonen und den Projektbetreuern gehalten. In dieser Präsentation sollte vor allem das Projekt den anwesenden Personen vorgestellt werden und die erreichten beziehungsweise nicht erreichten Meilensteine aufgezeigt werden. Gleichzeitig sollte die Präsentation die Möglichkeit bieten den Zuhörern einen ersten Eindruck über verschiedenste Themenbereiche zu liefern.

In dieser Präsentation wurde auch das Kernziel vorgestellt. Das Hauptziel des Projekts war die Erstellung eines Programms, dass es ermöglicht dynamisch Templates zu erstellen, die als Ausgangsbasis für Jaspersoft-Berichte dienen sollen. Dieses Ziel wurde mit der Fertigstellung des Programms und Dokumentation der Konfigurationsschnittstellen erreicht.

6. Wirtschaftliche Betrachtung

Der Auslöser für dieses Projekt war die Anforderung von FH Timing die Entwicklungszeit eines Berichts durch die Verwendung eines Template-Generators zu reduzieren. Diese Reduktion der Entwurf-Dauer eines Berichts hat natürlich auch wirtschaftliche Auswirkungen. FH Timing bietet auch aus wirtschaftlichen Gründen auf verschiedenen Sportveranstaltungen ihre Services an. Die Vorbereitungszeit für eine solche Veranstaltung ist auch mit einem finanziellen Aufwand verbunden. Dieser Aufwand kann natürlich nicht mit einer genauen Summe beziffert werden, jedoch ist der Vorbereitungsaufwand immer mit hohem Zeitaufwand verbunden. Das Programm könnte als auch dazu führen, dass die Gesamtkosten einer solchen Veranstaltung sich verringern und FH Timing dadurch auch ein verbessertes Jahresergebnis erzielt. Überdies würden dadurch die Lehrbeauftragten der Fachhochschule, die in FH Timing involviert sind, ihre dadurch gewonnene Zeit wieder in neue beziehungsweise andere Projekte investieren können.

7. Anregungen für weiterführende Arbeiten

Der Erfolg des neu erstellten Programms „TemplateGenerator“ wird sich natürlich erst in seiner operativen Anwendung zeigen. Da die Benutzer des Programms jedoch in den Entwicklungsprozess eingebunden wurden ist davon auszugehen, dass der Stand des Programmes den Erwartungen entspricht. In Folge kann die Entscheidung offenstehen, das Programm in eine gewünschte Richtung weiter zu entwickeln und zum Beispiel die Benutzerfreundlichkeit zu erhöhen oder den Funktionsumfang auszubauen. Ein Beispiel für eine bessere Anpassung des Programms hinsichtlich der Benutzerfreundlichkeit wäre der Ausbau beziehungsweise ein Neudesign der graphischen Benutzeroberfläche. Hinsichtlich des Funktionsumfangs könnte man auch noch in Erwägung ziehen weitere JasperReport-Elemente innerhalb des Programmes zu unterstützen und alle bereits unterstützen Elemente genauer anpassen zu können.

8. Literaturverzeichnis

[1] H. Vonhoegen, Einstieg in XML, 8. Hrsg., Bonn: Rheinwerk Verlag, 2015.

[2] Wikipedia: The Free Encyclopedia, „Markup language," Wikipedia Foundation, 3 Januar 2018. [Online]. Available: https://en.wikipedia.org/wiki/Markup_language. [Zugriff am 3 Januar 2018].

[3] Wikipedia: The Free Encyclopedia, „Standard Generalized Markup Language," Wikipedia Foundation, 3 Januar 2018. [Online]. Available: https://de.wikipedia.org/wiki/Standard_Generalized_Markup_Language. [Zugriff am 3 Januar 2018].

[4] T. Bray, J. Paoli, M. Sperberg-McQueen und E. Maler, „Extensible Markup Language (XML) 1.0 (Zweite Auflage) - Deutsche Übersetzung," World Wide Web Consortium, 1 Januar 2018. [Online]. Available: http://www.edition-w3.de/TR/2000/REC-xml-20001006/#sec-intro. [Zugriff am 1 Januar 2018].

[5] D. Fallside, „XML Schema Teil 0: Einführung - Deutsche Übersetzung," World Wide Web Consortium, 1 Januar 2018. [Online]. Available: http://www.edition-w3.de/TR/2001/REC-xmlschema-0-20010502/. [Zugriff am 1 Januar 2018].

[6] H. Thompson, D. Beech, M. Maloney und N. Mendelsohn, „XML Schema Teil 1: Strukturen - Deutsche Übersetzung," World Wide Web Consortium, 1 Januar 2018. [Online]. Available: http://www.edition-w3.de/TR/2001/REC-xmlschema-1-20010502/. [Zugriff am 1 Januar 2018].

[7] P. Biron und A. Malhotra, „XML Schema Teil 2: Datentypen - Deutsche Übersetzung," World Wide Web Consortium, 1 Januar 2018. [Online]. Available: http://www.edition-w3.de/TR/2001/REC-xmlschema-2-20010502/. [Zugriff am 1 Januar 2018].

[8] Inc. TIBCO Software, „Jaspersoft Community User Guide," 28 Dezember 2017. [Online]. Available: https://community.jaspersoft.com/system/files/restricted-docs/jaspersoft-studio-user-guide_5.pdf. [Zugriff am 28 Dezember 2017].

[9] Inc. TIBCO Software, „Jaspersoft Customers," Inc. TIBCO Software, 21 02 2018. [Online]. Available: https://www.jaspersoft.com/customers. [Zugriff am 21 02 2018].