Bachelorthesis

Erstellung einer Schnittstellenbeschreibung für die Integration der Listenerstellung in VOCUS Lohn mit List & Label

Vorgelegt am: 18.08.2014

Von: **Winkel Philipp**
Gutenbergstraße 8
08645 Bad Elster

Studiengang: Wirtschaftsinformatik
Studienrichtung: Wirtschaftsinformatik

Seminargruppe: WI11

Matrikelnummer: 4000275

Praxispartner: Vocus Computer- und Softwaresysteme GmbH
Wiesenstraße 14
08258 Markneukirchen

Gutachter: Dipl. Ing. Holger Kremp
(Vocus Computer- und Softwaresysteme GmbH)

Prof. Dr.-Ing. Wolfgang Oehme
(BA Gutachter)

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
GLAUCHAU
UNIVERSITY OF COOPERATIVE EDUCATION

Themenblatt Bachelorthesis

Studiengang Wirtschaftsinformatik

Student:	**Philipp Winkel**
Matrikelnummer:	**4000275**
Seminargruppe:	**4WI11-1**

Thema der Bachelorthesis

Erstellung einer Schnittstellenbeschreibung für die Integration der Listenerstellung in VOCUS Lohn mit List & Label

Gutachter/ Betreuer:	Dipl.-Ing. Holger Kremp
Gutachter (Studienakademie):	Prof. Dr. Wolfgang Oehme

Ausgabe des Themas:	**20.05.2014**
Abgabe der Arbeit an den SG am:	**18.08.2014, 14:00:00**

Flehmig

Dr. Katja Flehmig
Vorsitzende des Prüfungsausschusses
Wirtschaft

www.ba-glauchau.de

Freistaat Sachsen
Berufsakademie Sachsen, Staatliche Studienakademie Glauchau **Dienstsitz** Kopernikusstraße 51, 08371 Glauchau
Telefon +49 (0)3763/173-0, **Telefax** +49 (0)3763/173-180, **URL** www.ba-glauchau.de, **Mail** info@ba-glauchau.de

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

DLL	Dynamic Link Library (Dynamische Programmbibliothek)
HTML	Hypertext Markup Language
PAP	Programm Ablauf Plan
UML	Unified Modeling Language
XML	Extensible Markup Language

1 Einleitung

Das Thema Druck und Reporting ist fester Bestandteil im Vocus Lohn und Gehalt. Der Druck diverser Listen muss problemlos im Programm ablaufen und ist eine essentielle Komponente. Dazu wird seit 15 Jahren QuickReport eingesetzt. Aus Kostengründen und der Möglichkeit zur Gestaltung flexibler Layouts konnte es sich lange Zeit im Programm durchsetzen.
Die aktuellen Anforderungen erfüllt die Reportingkomponente jedoch nur mangelhaft. Aufgrund dessen ist sie durch eine neuere, mit mehr Funktionalitäten und einfacherer Handhabung, zu ersetzen. Auch die Komplexität der Listen wächst stetig. Diese ist durch QuickReport nur noch bedingt realisierbar.
Die bisher verwendete Reportingkomponente QuickReport soll durch die modernere List & Label-Komponente ersetzt werden, da diese den wachsenden Anforderungen nach schneller Druckaufbereitung, Ansteuerung aller neueren Druckertreiber und Export in gängige Dateiformate gerecht wird. Dabei unterscheidet sie sich grundlegend von QuickReport, vor allem bei der Datenverarbeitung und dem Erstellen von Layouts.
List & Label bietet neben den bekannten Funktionen aus QuickReport auch neue, wie Suchfunktion in Dokumenten, Export in die gängigen Dateiformate sowie das einfache Erstellen von Listenlayouts mithilfe eines integrierten Designers.

Dies bedingt eine neue Schnittstelle zu den Druckfunktionen des Programms. Es muss dem Programmierer der Firma Vocus möglich sein, der Schnittstelle eine eigenständige Datenmenge zu übergeben und auf Basis dieser eine gewählte Liste zu drucken. Dabei soll das bisherige Prinzip der Parametrierung der Schnittstelle kaum geändert werden. Für die interne Druckaufbereitung sind demnach neue Routinen, speziell für List & Label, zu schreiben, damit die Komponente die Datenmenge verarbeiten und drucken kann.
Im Zuge der neuen Schnittstelle bedarf es auch einer Neuimplementierung des Druckdialogs, in welchem der Anwender die neuen Funktionalitäten, wie Export und Bearbeiten der Einstellungen für den jeweiligen Drucker, zur Verfügung gestellt bekommt.

In dieser Arbeit soll eine Schnittstellenintegration zu List & Label konzipiert und anschließend in das Vocus Lohn- und Gehaltsprogramm implementiert werden. Ziel hierbei ist es eine fehlerfreie Druckansteuerung zu schreiben, welche sich flexibel im Programm einsetzen lässt.

2 Konzeption

2.1 Ausgangslage

Eine reibungslos funktionierende Druckerschnittstelle ist in einem Gehaltsabrechnungsprogramm ist von essentieller Bedeutung. Dem Nutzer müssen schnell und unkompliziert große Datenmengen zur Auswertung aufbereitet zur Verfügung stehen. Dabei darf die Übersichtlichkeit eines Reports nicht verloren gehen.
Das Lohn- und Gehaltsabrechnungsprogramm der Vocus GmbH existiert bereits seit dem Jahr 1990. Die Druckschnittstelle ist für die Verwendung von Nadeldruckern ausgelegt und wurde seitdem nur angepasst, um auch neuere Drucker wie normale Tintenstrahl- und Laserdrucker, verwenden zu können. Dabei hat sich an der Datenverarbeitung jedoch nichts geändert. Hinter der Programmkomponente für den Druck – aktuell noch „QuickReport“ – steht ein einheitliches System zur Datendefinition auf einem Report, welches wie folgt beschrieben werden kann:

In einer Textdatei befinden sich definierte Felder, welche dann während des Drucks gefüllt werden, um die Daten im Layout anzuordnen. Für die Programmierer der Software Vocus Lohn- und Gehalt bedeutet das: Für jedes neue Dokument müssen sie von Hand eine neue Textdatei erzeugen, in der sie die Felder für die Position ihrer auszudruckenden Daten auf dem Dokument definieren. Dabei geschieht das Erzeugen der Datei meist durch „Try-and-Error“, da für das Anlegen der Felder in der Datei keine geeignete, einfach zu bedienende Software existiert.
In der Textdatei halten sogenannte Zeilendefinitionen die einzelnen Felder. So gibt es beispielsweise eine Kopf- und Fußdefinition, um Informationen zu drucken, welche auf jeder Seite erscheinen sollen, und mehrere Zeilendefinitionen, welche die Daten enthalten. Nach dem Erzeugen der Textdatei, wird sie in einem zentralen Listenverzeichnis abgelegt. Der Programmierer muss nun die zu druckenden Daten in einer StringList[1] bereitstellen und diese in der korrekten Reihenfolge hinzufügen. Die Reihenfolge richtet sich nach den zuvor angeordneten Feldern. Anschließend werden der Druckroutine beide Informationen (Name der Liste und Daten in Form der StringList) und weitere Parameter zum gewünschten Druckverhalten übergeben. Diese erzeugt nun aus den Daten und Definitionen den Report. Dabei kann das Erzeugen des Dokumentes mit den übergebenen Informationen bei großen Datenmengen eine gewisse Zeit in Anspruch nehmen.
Die weitere Schrittfolge für das Erzeugen und Drucken des Reports wird mit den weiteren übergebenen Paramatern festgelegt. So kann zum Beispiel definiert werden, dass der Nutzer zwar drucken, jedoch nicht exportieren darf oder das Dokument wird

[1] StringList: Ein Objekt, welches mehrere Zeichenketten in einer Liste verwalten kann.

gar nicht angezeigt, sondern nur zu Nachverfolgungszwecken in dem Archiv abgelegt. Der Ablauf ist jedoch immer der gleiche:

1. Felder in eine Definitionsdatei erzeugen und im Listenverzeichnis abspeichern
2. dazugehörige Daten in der korrekten Reihenfolge in einer StringList übergeben
3. beide Informationen und weitere optionale Parameter dem Druckaufruf mitteilen
4. Druckroutine erstellt anhand der Informationen den Report und lädt diese in das QuickReport-Modul für den Druck

2.2 Zielsituation

Um auch in Zukunft den wachsenden Anforderungen nach gut strukturierten und schnell einsehbaren Reports gerecht zu werden, ist es nötig eine neue Druckerkomponente und so auch eine neue Druckschnittstelle im Vocus Lohn und Gehalt zu implementieren. Die derzeitige Situation muss dabei überbrückt werden, so dass auch ältere Reports weiterhin gedruckt werden können. Von Vorteil wäre dabei, dass ältere Reports konvertiert und mittels der neuen Komponente druckbar sind.
Die Erstellung und Nutzung neuer Reports sollen folgende Eigenschaften aufweisen:

- Einfachheit: Dem Nutzer sollen möglichst einfach neue Reports zur Verfügung gestellt werden können.
- Schnelligkeit: Auswertungen sollen selbst bei großen Datenmengen in einem akzeptablen Zeitraum zur Verfügung stehen.
- Nutzerkomfort: Die Einführung einer neuen Druckerkomponente wird automatisch auch einige Änderungen im Verhalten und Aussehen des Programms mit sich bringen. Der Anwender soll sich einfach an die neue Bedienung gewöhnen und eventuell auch eigene Auswertungen erstellen können.
- Minimaler Programmieraufwand: Um einen neuen Report zu erzeugen, sollen Programmierer nicht mehr eine Textdatei mit Definitionen erstellen müssen, sondern einen Editor nutzen können, über welchen das Layout zu definieren ist.
- Einheitliches Datenkonzept: Zur Übergabe der Daten an den Report soll eine einheitliche Datenphilosophie gelten. Datentypen sollen genau definiert werden und auch so im Archiv speicherbar sein.

Die Firma Vocus entschied sich bei der neuen Druckkomponente für das leistungsstarke Reporting-Tool „List & Label“ der Firma „combit“. Dieses bietet alle geforderten Funktionalitäten. Weiterhin stellt es keine Anforderungen wie die Daten dem Druck zu übergeben sind und erfordert so die Anforderung nach einem flexiblen Sys-

tem. Es lässt sich sowohl mit einer Datenbank im Hintergrund verwenden, als auch mit einer selbst definierten Datenmenge im Programm. Für dieses soll nun eine geeignete und einfach zu bedienende Schnittstelle geschrieben werden, um alle Funktionen des Tools nutzen zu können.
Um einen Überblick zu bekommen wie sich die Informationsverarbeitung im Hinblick auf die Erstellung eines Reports zusammensetzt und welche Faktoren zu beachten sind, wird im folgenden Kapitel die Prozesskette der Informationsversorgung beschrieben.

2.3 Prozesskette der Informationsversorgung

Hinter jedem Report, der erzeugt werden soll, stehen Informationen, die einen gewissen Zweck erfüllen. Dabei durchlaufen sie einen Prozess von der Entstehung eines Informationsbedarfes bis hin zur Nutzung einer Information. In Anlehnung an den Informationsfluss eines Managements ist im Folgenden beschrieben, wie ein solcher Informationsfluss vom Bedarf bis hin zum Report und dessen Nutzung im Vocus Lohn- und Gehaltsprogramm abläuft. Dabei ist eine klare Differenzierung in Gegenüberstellung zu der Lohnabrechnung wichtig.

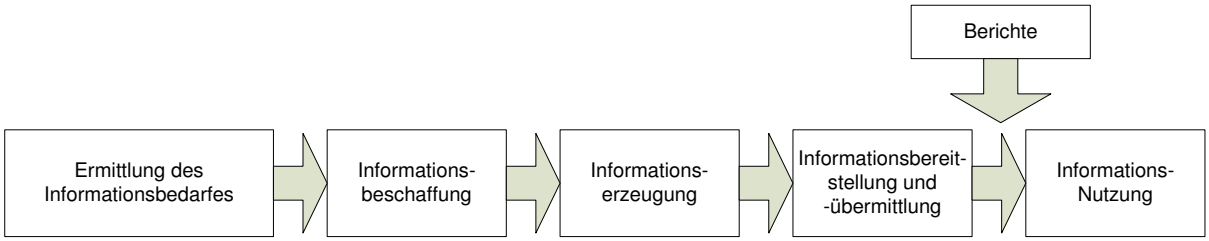


Abbildung 1 Prozesskette der Informationsversorgung

(eigene Darstellung in Anlehnung an GLEICH; HORVÁTH; MICHEL, S. 18)

Der Programmierer ermittelt an einer bestimmten Stelle im Programm einen Informationsbedarf für den Anwender, so dass dieser über eine bestimmte Gegebenheit detailliert informiert wird. Die Auswahl der gewünschten Daten übernimmt der Entwickler und stellt sie in einer Datenmenge bereit. Weiterhin ergänzt er die Daten um bestimmte Kennzahlen wie Summen, Durchschnittswerte oder andere für die Auswertung wichtige Informationen. Nun stellt der Entwickler dem Nutzer die Informationen bereit. Im speziellen Fall der Reporting Problematik erzeugt er eine Auswertung. Am Ende des Prozesses verfügt der Anwender über die Daten in aufbereiteter Form zur Anzeige und zum Druck.
In diesem Prozess entscheidet der Programmierer vorerst, welche Daten verfügbar gemacht werden sollen. Im Anschluss daran kann er dem Anwender ermöglichen die Daten nach gewissen Kriterien zu sortieren oder zu filtern. Weiterhin stellt der Programmierer dem Nutzer des Programms einen Dialog und / oder Report mit den In-

formationen bereit. An dieser Stelle im Prozess werden die Daten zu Informationen und auf der Oberfläche dargestellt. So kann der Anwender sie nutzten.
HORVÁTH ist der Ansicht, dass Management Reporting ein Teil des betrieblichen Berichtswesens ist und durch die Phasen der Informationsbereitstellung und Informationsnutzung definiert wird. Berichte haben dabei die Aufgabe eine Planung und Kontrolle zu ermöglichen.[2] Für die Nutzung eines Report-Tools in der Vocus Lohn- und Gehaltssoftware soll der Begriff „Management Reporting“ abgewandelt werden. Reporting im Vocus Lohn und Gehalt schließt alle Phasen der Prozesskette der Informationsversorgung ein und ist u.a. für die Planung und Kontrolle der Abrechnung zuständig. Weiterhin werden in einem Report wichtige, verbindliche und beweisbare Aussagen generiert, so dass das Berichtswesen hier eine besondere Stellung einnimmt.
Zusammenfassend lässt sich sagen, dass Reporting in Vocus Lohn und Gehalt eine besondere Stellung, im Vergleich zum Management Reporting einnimmt. Zudem wird dabei auch ein anderes Aufgabenfeld bedient. Die Gemeinsamkeiten sind die Informationsaufbereitung und Informationsnutzung. Der Programmierer übernimmt in diesem Prozess die Erzeugung und die Bereitstellung der Information. Für den Anwender ist die Informationsnutzung von essentieller Bedeutung, vor allem in Hinblick auf die durchzuführenden Aufgaben wie die Lohnabrechnung. Über Auswertungen kann er Unstimmigkeiten in den Abrechnungen finden und diese korrigieren.

In Ausblick auf die weitere Verfahrensweise soll dieser Prozess für die Firma Vocus bis zur Informationsbereitstellung in Form von Berichten durch eine neue Druckschnittstelle vereinfacht und optimiert werden.

2.4 Schnittstellenentwurf

2.4.1 Bisherige Druckschnittstelle

Wie in den vorherigen Kapiteln beschrieben, nutzt die Firma Vocus für ihre Listenerstellung eine einheitliche Schnittstelle, die auf Textdateien mit Felddefinitionen und StringLists zur Datenhaltung basiert. Beides wird an das Drucktool QuickReport übergeben, welches daraus einen Report erstellt. Änderungen im Layout der Listen sind nur schwer möglich und benötigen eine gewisse Zeit. Auch das Erlernen für die Syntax der Felder stellt eine Schwierigkeit dar. In der momentanen Situation bedarf es einiges an Zeit, wenn ein Report geändert oder ein neuer erstellt werden muss.
Ein weiteres Problem ist das Erzeugen des Reports. Bei vielen Daten und vielen Feldanordnungen kann es unter Umständen eine Weile dauern, bis aus den Felddefinitionen und den Daten ein fertiger Report geparst wurde. Das folgende Flussdiagramm soll die Problematik graphisch verdeutlichen:

[2] vgl. GLEICH; HORVÁTH; MICHEL, S. 20

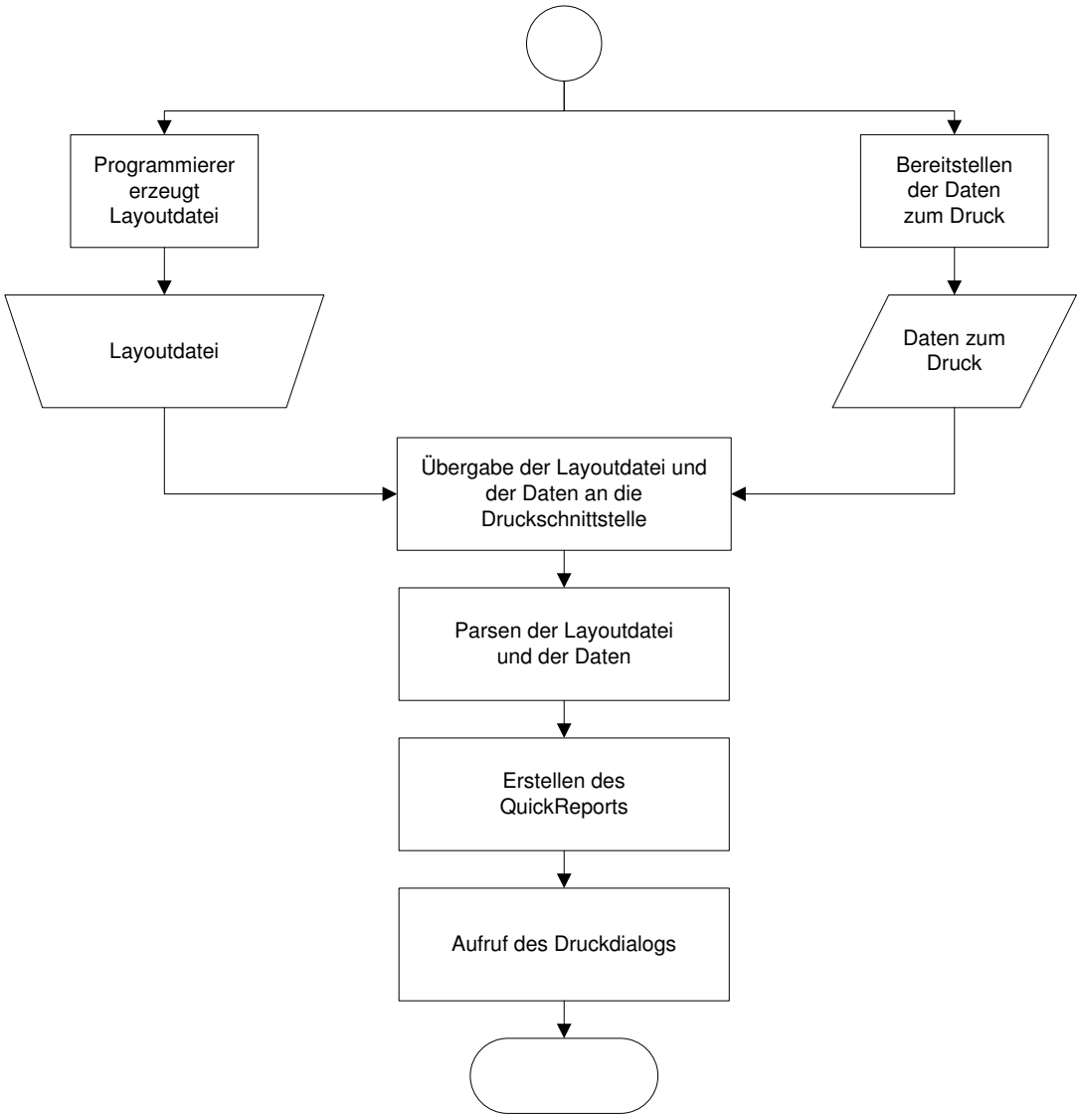


Abbildung 2 Ansteuerung bisherige Druckerschnittstelle[3]

Im Moment ist es nicht möglich, die einzelnen Prozessschritte auch parallel durchzuführen, so dass dieser Prozess strikt vorwärts gerichtet ist. Dies führt dazu, dass der Nutzer keine Möglichkeit des Abbruches hat, bis der Druckdialog angezeigt wird. In der obigen Abbildung wurden die Reporterstellung durch den Programmierer und der Druck durch den Anwender kombiniert. Der Programmierer erstellt die Layoutdatei und stellt die Daten bereit. Beides wird an die Druckschnittstelle übergeben. Ab diesem Punkt verlaufen alle Prozesse (Parsen der Layoutdatei und der Daten, Erstellen des QuickReports bis zum Aufruf des Druckdialogs) bedingt streng nacheinander ab und der Anwender hat keine Möglichkeit diese Vorgänge abzubrechen.

[3] Legende: Rechteck: Prozess; Raute: Daten; Trapez: Datei

Um eine neue Schnittstelle planen zu können, ist eine Analyse der bisherigen Abläufe zwingend notwendig. Zum einen können anhand dieser derzeitige Schwachpunkte analysiert werden. Zum anderen ist es möglich herauszufinden, an welchen Abläufen sich orientiert werden kann, um die Umgewöhnung auf das neue System sowohl bei dem Anwender als auch beim Programmierer zu beschleunigen.

2.4.2 Planung einer neuen Druckschnittstelle

Zur Planungsphase gehören auch diverse Dokumente, um die Ideen zu visualisieren und festzuhalten. Dies soll hier anhand von Texten und Grafiken geschehen. George THALLER ist der Ansicht, dass überladene Bilder in Softwaredokumentationen genauso wenig Nutzen haben wie unübersichtliche und lange Texte. Der Zusammenhang zwischen Text und Bild darf nicht verloren gehen. Gerade in der Anfangsphase einer Softwarerealisierung, der kreativsten Phase, kommt es auf die Dokumentation an.[4] In diesem konkreten Fall sollen erst die Anforderungen und der gedankliche Ablauf zur Ansteuerung der Schnittstelle schriftlich festgehalten und dann in einem UML-Diagramm (Unified Modeling Language) graphisch verdeutlicht werden.
Für die Planung einer neuen Schnittstelle zum Druckmodul „List & Label“ spielt die Anforderungsanalyse eine wichtige Rolle. Dazu gehören der bisherige Schnittstellengebrauch sowie die neuen Möglichkeiten des Tools. Anhand des Pflichtenheftes (Anhang 1) lassen sich die Anforderungen ermitteln und zusammenfassend Folgendes formulieren:

- Die Schnittstelle soll sich einfach ansteuern lassen, dabei ist sich an der bisherigen Verfahrensweise zu orientieren.
- Vorhandene Reportausgaben im Quelltext sollen umstellbar sein.
- Dem Nutzer soll ein verbesserter, an gängiger Software-Ergonomie orientierter Druckdialog geboten werden.
- Entwickler sollen in der Lage sein, eine Datenmenge zu übergeben und zu dieser einen Report designen zu können. Dabei sollen auch komplexe und große Reports kein Problem darstellen.
- Beim Erstellen des Layouts für den Report sollte man die Möglichkeit der Live-Vorschau haben, um seine Daten im Layout anzusehen (Echtdatenvorschau).
- Um eine bessere Zeitperformance zu erhalten, soll es möglich sein, den Report erst aufzubereiten, wenn er tatsächlich gebraucht wird (zum Beispiel beim Klicken auf „Drucken“ oder „Vorschau“). Hierbei soll die Möglichkeit bestehen, sich bereits aufbereitete Seiten anzusehen oder drucken zu können.
- Für den Anwender sollte es möglich sein, Dokumente in gängige Dateiformate wie Word, Excel, PDF, etc. zu exportieren.

[4] vgl. THALLER, S. 102

List & Label ist ein Report-Tool, welches datenbankgestützt arbeitet. Das Vocus Lohn- und Gehaltsprogramm verwendet zwar intern eine Firebird-Datenbank, diese wird aber nur für bestimmte Programmfunktionen genutzt. Die meisten Programmdaten liegen in einem Lohnamtsverzeichnis und werden mit einer bestimmten Struktur in Dateien abgelegt.
Für das programminterne Verarbeiten von Daten steht ein Listenobjekt bereit, das sämtliche Datentypen aufnehmen kann (ähnlich einem Datensatz in einer Datenbanktabelle). Die sogenannte „DataList“ bietet auch Methoden um Daten in DataSets zu übertragen, welche wie eine temporäre Datenbank im Hauptspeicher arbeiten. Um alle Funktionalitäten von List & Label nutzen zu können, ist dies eine Möglichkeit, die übergebene Datenmenge in eine lokale Datenbank zu übertragen und diese anschließend List & Label zur Verfügung zu stellen.
Der Quelltext zur Aufbereitung vorhandener Druckdaten des Vocus Lohn- und Gehaltsprogrammes muss daher für die neue Druckschnittstelle angepasst werden. Dabei müssen die Daten, welche vorher per Programmlogik in StringLists geschrieben wurden, in DataLists gespeichert werden. Ein vorhandener Datensatz (eine Zeile in der bisherigen StringList) muss jetzt in ein Data-Objekt geschrieben werden. Werte, welche bisher einzeln in der StringList aneinandergereiht gespeichert wurden (getrennt mit einem vertikalen Strich („|“)), sind jetzt in einzelne Felder eines Data-Objektes zu hinterlegen.

Für die Erstellung von Layouts der Reports bietet List & Label einen Designer an.
Der Ablauf beim Drucken oder Öffnen eines Dokuments zur Bearbeitung unterscheidet sich nur minimal. Für die Programmierer wäre es außerdem von Vorteil, wenn sie eine Möglichkeit zum Öffnen des Designers im Druckdialog hätten. Das erstellte Layout kann dann (wie bisher) im Listenverzeichnis unter einem Dateinamen abgelegt werden. Der Aufbau dieser Datei unterscheidet sich vollständig von den bisherigen Definitionsdateien, so dass beide gefahrlos in einem Verzeichnis untergebracht werden können. Des Weiteren erhalten sie eine neue Dateierweiterung um Verwechslungen auszuschließen.
So würde sich bei der Ansteuerung der Druckschnittstelle eine veränderte Parametrierung ergeben. Anstatt der StringList wird nun eine DataList übergeben und alle anderen Parameter bleiben erhalten. Damit bietet sich auch der Vorteil für die Entwickler, dass sie beim Drucken einer Datenmenge keine neuen Parameter beachten müssen. Die Daten sind lediglich in Form der DataList zu übergeben.
Für die Weiterführung der Konzeption soll nun der Sachverhalt in einem UML-Diagramm modelliert werden.

2.4.3 Software-Engineering mit UML-Modellierung

2.4.3.1 Grundlagen

Der systematische Aufbau eines Software-Projektes ist entscheidend für die gute Qualität des zu entwickelnden Programms. Die komplexen Ansätze der objektorientierten Programmierung und das Arbeiten in Teams an einem Software-Projekt bedingen einen notwendigen Überblick über das gesamte Großprojekt und eine einheitliche Verfahrensweise bei der Programmentwicklung.[5] Schon in der Phase der Weiterentwicklung der Programmiersprache wurde dies erkannt und durch Vorgehensmodelle gelöst. Diese beinhalten folgende Phasen: Analyse, Planung, Realisierung und Inbetriebnahme des Software-Projektes.[6] Die Ergebnisse der Phasen sind so zu dokumentieren, damit jeder Beteiligte sie nachvollziehen kann.
Besondere Beachtung der Bedürfnisse gilt hierbei den Stakeholdern des Programms. In der Vocus Lohn- und Gehaltssoftware sind die Interessenten ausschließlich die Programmierer und Anwender. Sie bedürfen einer genauen Information, wie sich das Programm in diversen Situationen verhält. Für die Nutzer der Software gibt es eine programminterne Hilfe und es werden zusätzlich regelmäßige Schulungen angeboten, um die Anwender mit der Software vertraut zu machen.
Um das Projekt zu veranschaulichen wurde die UML-Modellierung gewählt. Die UML ist zu einem wichtigen Werkzeug in der Softwareentwicklung geworden und kann in allen Entwicklungsphasen eingesetzt werden.[7]
Die systematische Programmierung beruht auf dem Prinzip der Erfahrung der Programmierer. Dabei leitet sich die weitere Vorgehensweise häufig aus dem Erkennen der Standardaufgaben und dem erneuten Anwenden bereits ermittelter Lösungen aus den bisherigen Arbeiten ab. Neben den Standardaufgaben gibt es noch den kreativen Teil der Softwareentwicklung, bei dem Lösungen zu nicht standardisierten Problemen gesucht werden.[8] Die Entwicklung einer neuen Druckschnittstelle zählt zu einer kreativen Aufgabe, da hierfür noch keine anwendbaren Lösungen existieren.

Abschließend, zur Erläuterung der Grundlagen des „Software-Engineering“, kann der Begriff wie folgt definiert werden:
Die Wissenschaft der systematischen Entwicklung von Programmen wird als Software-Engineering bezeichnet. Das Stellen der Anforderungen, Auslieferung und Abnahme bis hin zur Wartung definieren die Phasen des Software-Engineerings. Um die Entwicklung erfolgreich und zeitgemäß abzuschließen, werden für Teilaufgaben bereits vorhandene Lösungsansätze überprüft und mit neuen Technologien kombi-

[5] vgl. KLEUKER, S. 2 f.
[6] vgl. KLEUKER, S. 3
[7] vgl. KLEUKER, S. 3
[8] vgl. KLEUKER, S. 4

niert. Die Umsetzung ihrer Anwendbarkeit bedarf einer vorherigen Überprüfung. Die Dokumentation von Software-Engineering findet mittels UML-Diagrammen statt.[9]

2.4.3.2 Aktivitätsdiagramm

In folgendem Aktivitätsdiagramm soll der Prozess der Reporterstellung der Firma Vocus im Programm Lohn und Gehalt dargestellt werden. Ergänzt wird das Aktivitätsdiagramm durch eine (textuelle) Beschreibung. Hierbei werden die einzelnen Prozessschritte, in diesem Diagramm Aktionen genannt, graphisch veranschaulicht.[10] Der Prozess der Reporterstellung und des Druckens mit einer vom Entwickler definierten Datenmenge kann wie folgt veranschaulicht werden:

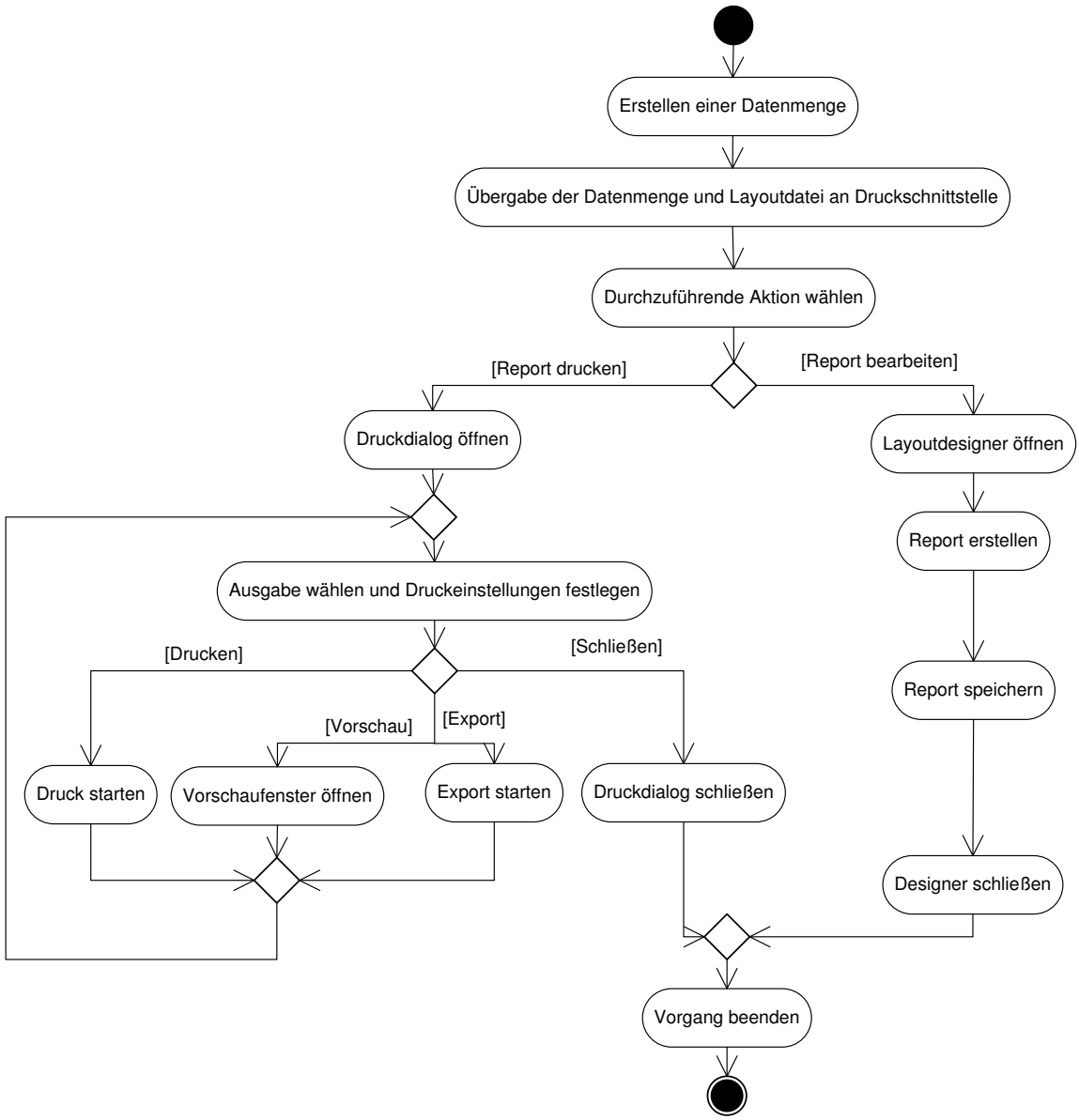


Abbildung 3 Aktivitätsdiagramm – Übergabe der Daten an Druckschnittstelle

[9] vgl. KLEUKER, S. 4 f.
[10] vgl. KLEUKER, S. 11

Der Programmierer erstellt eine Datenmenge in Form einer DataList. Diese Datenmenge, der Name der Liste aus dem Listenverzeichnis und einige Parameter die das Druckverhalten bestimmen, übergibt er der Druckschnittstelle.

Die Schnittstelle muss anschließend die DataList in Datenbankobjekte (ClientDataSets) umwandeln und ggf., wenn Daten (Data-Objekte) in der DataList untereinander verknüpft sind, diese auswerten und die Struktur innerhalb der ClientDataSets in Master-Detail Beziehungen widerspiegeln. Daraus ergibt sich eine lokale Datenbankstruktur, welche dem Report übergeben werden kann. Hierbei genügt es, dem Report alle verfügbaren Tabellen und deren Beziehungen untereinander bekannt zu machen und die einzelnen Felder anzumelden. Für den Druck und die Aufbereitung zur Vorschau genügt eine einheitliche Routine, welche den Report mit Daten aus den Tabellen füllt und diverse Nachrichten der List & Label-Routine verarbeitet.

Nun kann der Entwickler bestimmen, welche Aktion er durchführen möchte: Den Report mit seinen Daten drucken oder einen neuen Report dazu erstellen bzw. den bereits existierenden im Layoutdesigner bearbeiten. Je nach Auswahl erscheint nun ein neues Dialogfenster.
Bei der Wahl von „Drucken“ wird der aktuelle Report auf dem im Dialog gewählten Drucker ausgegeben. „Vorschau“ bewirkt das Anzeigen des gesamten Reports in einem separaten Vorschaufenster. Für den Fall, dass der Programmierer (oder später der Anwender) sein vollständiges Dokument in ein bestimmtes Dateiformat (PDF, Microsoft Word oder Excel) exportieren möchte, kann er „Export“ auswählen. Der Druckdialog wird solange offen gehalten, bis die Aktion „Schließen“ gewählt wird. Hintergrund von diesem Ablauf ist, dass evtl. mehrere Aktionen durchzuführen sind, wie zum Beispiel das Drucken und Exportieren eines Dokumentes.
Wählt man im ersten Schritt „Report bearbeiten“ aus, wird der Report im List & Label Designer geöffnet, wo er anschließend bearbeitet werden kann. Sollte noch kein Report verfügbar sein, kann ein neuer erstellt werden. Abschließend wird der Report gespeichert und der Designer wieder geschlossen.

2.4.3.3 Anwendungsfalldiagramm

Zur Erläuterung der Hauptfunktionalität der Druckschnittstelle hilft ein Use Case (Anwendungsfall-) Diagramm. Darin wird die Kernfunktionalität eines neuen Systems festgehalten.[11] In diesem UML-Diagramm werden die Aufgaben der Druckschnittstelle modelliert. Hinzu kommen die Akteure (Stakeholder) und mit welchem Bezug zum System sie in Erscheinung treten.[12] Dabei können Akteure nicht nur Menschen sein, sondern auch diverse Programme wie Datenbanken oder im speziellen Fall hier die

[11] vgl. KLEUKER, S. 58
[12] vgl. KLEUKER, S. 59 f.

Vocus Lohn- und Gehaltsabrechnungssoftware. Da die UML-Modellierung hier nur für die Druckschnittstelle gelten soll, wurde der Bereich des Softwaresystems auf die Schnittstelle minimiert.

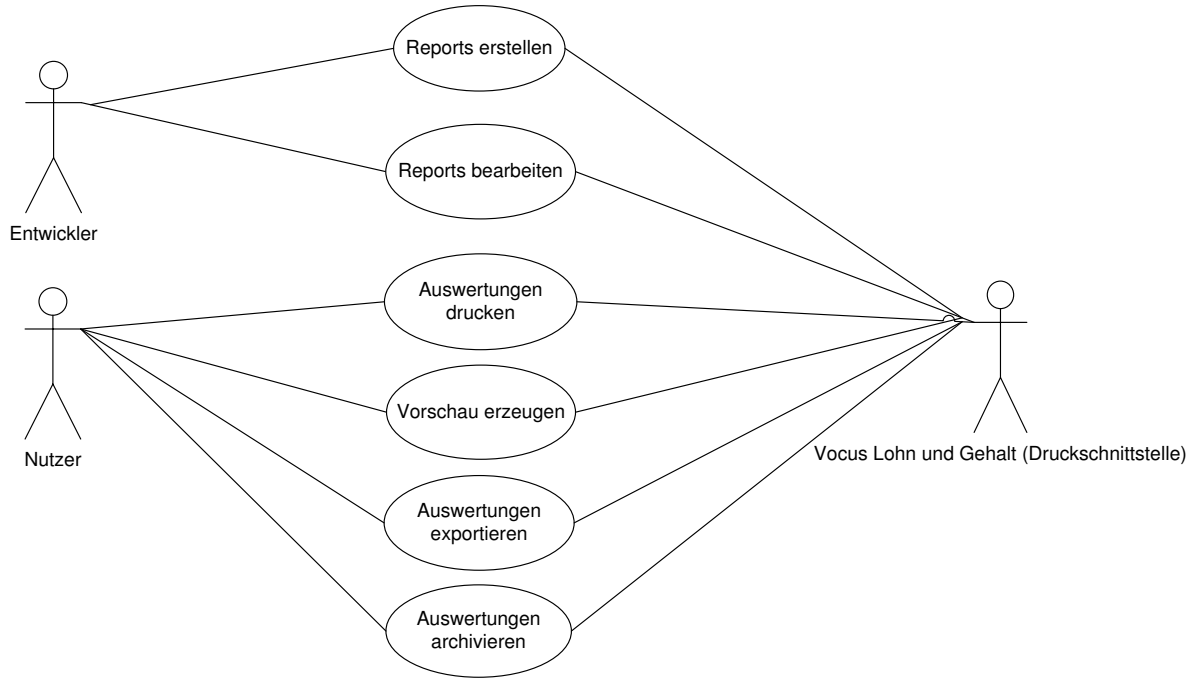


Abbildung 4 Anwendungsfalldiagramm der Druckschnittstelle

Die Stakeholder der Druckschnittstelle sind Entwickler und Nutzer des Vocus Lohn- und Gehaltsprogrammes. Sie haben unterschiedliche Anforderungen an die Schnittstelle, die aber im Grunde eine Basis haben: die Reporterstellung und Auswertung ihrer Daten. Dabei müssen unterschiedliche Aspekte betrachtet werden. Der Entwickler hat ein Interesse daran, Reports zu erstellen und zu bearbeiten. Fertige Reports stellt er im Programm zur Verfügung, so dass der Nutzer diese für seine Daten und Auswertungen verwenden kann. Daraus resultieren folgende Anforderungen für Entwickler und Nutzer:

Akteur	Anforderung	Erläuterung
Entwickler	Reports erstellen	Die Schnittstelle muss in der Lage sein, neue Reports anhand von Datenmengen generieren lassen zu können.
	Reports bearbeiten	Bereits vorhandene Reports müssen bearbeitet werden können.
Nutzer	Auswertungen drucken	Reports müssen über die Schnittstelle anhand einer Datenmenge und einer Layoutdatei gedruckt werden können.
	Vorschau erzeugen	Bevor Reports gedruckt werden, muss es möglich sein eine Vorschau derer anzuzeigen.

Akteur	Anforderung	Erläuterung
	Auswertungen exportieren	Auswertungen müssen in diverse gängige Dateiformate konvertiert werden können, um sie mit anderen Programmen weiterzubearbeiten.
	Auswertungen archivieren	Gedruckte Reports müssen sich abspeichern und im programminternen Listenarchiv hinterlegen lassen, um sie zur späteren Betrachtung wieder aufbereiten zu können.

Tabelle 1 Anforderungen der Stakeholder an die Druckschnittstelle

Durch diese funktionalen Anforderungen wird gewährleistet, dass das Softwareendprodukt wie gewünscht entsteht. Es gibt jedoch noch weitere Anforderungen, die sich direkt auf das zu entwickelnde System auswirken.[13]
Diese Anforderungen werden unter dem Begriff der nicht-funktionalen Anforderungen gefasst. KLEUKER nennt hier als Beispiele das verzögerungsfreie Arbeiten, damit der Nutzer möglichst effizient agieren kann, und eine ordentliche Nutzungsdokumentation zur Erklärung der Funktionen.[14]
Alle Anforderungen werden schließlich in das Pflichtenheft übertragen.

2.4.3.4 Lasten- und Pflichtenheft

Im Lastenheft werden alle Forderungen an die Software niedergeschrieben, die der Kunde an das Programm hat. Der Auftragnehmer überträgt die Anforderungen aus dem Lastenheft in das Pflichtenheft und spezifiziert sie ggf.[15]
Da im Falle der Entwicklung einer neuen Druckerschnittstelle für das Lohn- und Gehaltsprogramm der Auftragnehmer und der Auftraggeber die Firma Vocus ist, kann das Lastenheft direkt als Pflichtenheft verwendet werden. Das vollständige Pflichtenheft ist unter Anhang 1 hinterlegt.
Sind alle Anforderungen schriftlich fixiert, kann mit dem Grobdesign der Schnittstelle begonnen werden.

2.4.3.5 Grobdesign: Klassendiagramm

Um das vorläufige Klassendiagramm zu entwickeln, sollten zuerst ein paar Vorüberlegungen getroffen werden.
Die Firma Vocus möchte die neue Schnittstelle mittels des Reporttools List & Label (Version 18) der Firma combit realisieren. Die Komponente besitzt eine Basisklasse „TDBL18_“, welche für die Ansteuerung der List & Label internen Aufrufe an eine DLL (Dynamic Link Library) genutzt werden kann. Diese Klasse stellt auch alle grundlegenden Methoden bereit, um eine Datenübergabe an List & Label zu realisie-

[13] vgl. KLEUKER, S. 77
[14] vgl. KLEUKER, S. 77
[15] vgl. KLEUKER, S. 81

ren. Für die programminterne Schnittstelle kann diese Klasse vererbt und um weitere Methoden ergänzt werden, wie die Verwaltung der Kundenlizenz für List & Label.
Die Verwendung einer Klasse, welche das Laden und Entladen der List & Label DLLs übernimmt, ist ebenfalls sinnvoll: Die Klasse wird beim Programmstart instanziiert, prüft ob die DLLs verfügbar sind und lädt sie in den Hauptspeicher. So kann vor jeder Ansteuerung der Schnittstelle gewährleistet werden, dass es zu keinen Fehlern aufgrund fehlender DLLs kommt. Die Programmbibliotheken erfolgreich laden zu können, ist essentiell wichtig für die Schnittstelle.

List & Label bietet beim Entwerfen von Reports an, diese gleich in einer Echtdatenvorschau mit Live-Daten anzusehen oder auch zu drucken. Der Entwickler erhält so die Möglichkeit, seinen Report direkt an das Layout anzupassen und die Auswirkungen ansehen zu können. Voraussetzung hierfür ist, dass mehrere Threads verarbeitet werden können. Delphi stellt für Threads eine eigene Basisklasse bereit, welche zu vererben ist.
An dieser Stelle soll nicht weiter auf die Klassen eingegangen werden, da dieses Thema ausführlich im Hauptabschnitt Implementierung unter dem Kapitel 3.2 erläutert wird. Im Folgenden wird der theoretische Ansatz zur Verwendung eines Data Dictionary angewandt, um eine korrekte Parametrierung der Druckschnittstelle zu gewährleisten.

2.5 Aufstellen eines Data Dictionary für die Datenübergabe mittels DataList

Da es zwischen den Modulen eines Programmes immer wieder zu Fehlern in der Datenübergabe kommt, die auf falsche Datentypen zurückzuführen sind, empfiehlt sich die Einführung eines Data Dictionary.[16] Mit Hilfe dessen sind zwar nicht alle Fehler auffindbar, doch die Arbeit beim Überprüfen der Schnittstellen wird erleichtert.[17]
Die Druckschnittstelle wird hauptsächlich durch die übergebenen Daten der DataList bestimmt. In der nachfolgenden Tabelle wird der Aufbau der Parameterliste für die Druckroutine dargestellt:

Parameter	Erklärung	Typ	Wertebereich
APrintDataList	beinhaltet die zu druckenden Daten, incl. ihrer Struktur	TDBDataList	-

[16] vgl. THALLER, S. 113
[17] vgl. THALLER, S. 114

Parameter	Erklärung	Typ	Wertebereich
ListFileName	Pfadangabe zur Liste (liegt die Liste im Listen-verzeichnis, kann der Pfad selbstständig ermit-telt werden)	AnsiString	-
ListName	Anzeigename der Liste im Druckdialog	AnsiString	-
ReportOutput	Ausgabemöglichkeiten	TLLOutput	llOutput_Print, llOutput_Preview, llOutput_TTY, llOutput_Fax, llOutput_Export_PDF, llOutput_Export_HTML, llOutput_Export_DOC, llOutput_Export_RTF, llOutput_Export_XLS, llOutput_Export_TXT_Layout, llOutput_Export_TXT, llOutput_Export_Graphic, llOutput_Export_XPS, llOutput_Export_XML
ArchivDate	wenn Report archiviert werden soll, gültiges Archivierungsdatum ein-tragen	TDateTime	-
ArchivMode	Drucken eines Reports aus dem Archiv	Boolean	False, True
PreviewOnly	nur Vorschau, kein Druck möglich	Boolean	False, True
ArchivOnly	Report nur im Archiv speichern	Boolean	False, True
NoPrint	kein Druck möglich	Boolean	False, True

Tabelle 2 Data Dictionary der Druckschnittstelle

Durch die Aufstellung des Data Dictionary für die neue Schnittstelle soll verhindert werden, dass Programmierer falsche Parameter setzen und dies zu unerwarteten Programmreaktionen führt.

Der Entwurf und die Konzeption der neuen Druckerschnittstelle sind an dieser Stelle mithilfe der UML-Diagramme, Grobdesigns der Klassen und des Data Dictionary ab-geschlossen, nun folgt die Implementierung.

3 Implementierung

3.1 Grundlagen

Die Implementierung der List & Label Druckschnittstelle in das Vocus Lohn- und Gehaltsprogramm wird, wie auch die gesamte Software, in Delphi realisiert. Delphi ist eine objektorientierte Programmiersprache basierend auf Object Pascal.[18] Derzeit entwickelt die Vocus GmbH mit der Version Delphi XE2 von Embarcadero. Der Erwerb der Lizenz zur Verwendung der List & Label Komponente (Version 18) war eine einmalige Ausgabe der Firma Vocus und birgt keine weiteren laufenden Kosten. Mit dem Abschluss der vollwertigen Implementierung der Komponente werden auch wieder Ressourcen für andere Programmierarbeiten in der Softwareentwicklung frei.

In Delphi werden Quelltexte auf mehrere Units aufgeteilt. Jede Unit soll dabei eine logische, von anderen Modulen abgetrennte Einheit darstellen.[19] Im dargestellten Fall bildet diese logische Einheit die Schnittstelle zu allen Druckfunktionalitäten von List & Label. Jeder Entwickler, der die Druckschnittstelle in seinen Klassen und Prozeduren verwenden möchte, bindet diese Unit in seinem Quellcode ein.
List & Label liefert selbst eine Grundklasse zur Ansteuerung seiner Druckkomponenten mit. Allerdings benötigt List & Label zum einen eine Entwicklerlizenz, welche auf jedem Entwicklungsrechner installiert sein muss, und eine im Quellcode verankerte Kundenlizenz, welche der List & Label Basisklasse übergeben werden muss. Die Basisklasse für die Ansteuerung von List & Label mittels Datenbank lautet „TDBL18_“. Um die Echtdatenvorschau im Designer nutzen zu können, benötigt List & Label Zugriff auf Threads, weswegen die interne Klasse „TThread“ benötigt wird. List & Label hat alle Programmfunktionen in DLLs ausgelagert. Um das Laden und Entladen dieser steuern zu können, wurde eine Klasse namens „TL18_Libraries“ generiert. Diese tritt allerdings nur beim Programmstart und beim Programmende in Erscheinung, um zu prüfen ob alle benötigten DLLs verfügbar sind und geladen werden konnten.
Weiterhin gelten in der Vocus GmbH Regeln, um den Quellcode leserlich zu halten (Namenskonventionen). Diese orientieren sich am gängigen Standard von Delphi.

[18] vgl. NIEMANN, S. 71
[19] vgl. NIEMANN, S. 97

Folgende Beispiele sollen den Sachverhalt näher darstellen:

- alle Dateinamen für Units beginnen mit „f“, um sie von Delphi-Internen zu unterscheiden
- für Klassen-, Datei- und Variablennamen gilt die „Kamelhöckerschreibweise“[20]
- Konstanten werden in Großbuchstaben benannt
- Klassen beginnen mit einem „T“, zum Beispiel „TL18_Report“
- klasseninterne Felder beginnen mit einem „F“
- die interne Stringverarbeitung findet mit Ansi Zeichen statt

Für die Implementierung der Schnittstelle wurde eine eigenständige Unit „fReportListLabel“ erzeugt, welche alle Klassen und Funktionen zur Ansteuerung von List & Label enthält.

3.2 Klassen

3.2.1 Überblick

Die objektorientierte Programmierung hat sich aus der Generation der Maschinen- und Assemblersprachen heraus entwickelt. Sie zählt zu den höheren Programmiersprachen. Der Code solcher Sprachen muss zunächst über Compiler in einen für Maschinen lesbaren Code umgewandelt werden, bevor er auf einem Rechner ausgeführt werden kann.[21]
Der Sinn der objektorientierten Programmiersprachen liegt in der Aufhebung, Daten und Prozeduren voneinander zu trennen. Stattdessen existieren Objekte, die Daten und Methoden kapseln. Mittels Nachrichten können sie untereinander kommunizieren.[22] In Objekten werden Informationen zu ihren Zuständen hinterlegt und Methoden beschreiben ihr Verhalten.[23] Die verschiedenen Objekte haben keinen Einfluss auf ein anderes, hierfür bedarf es der Schnittstellen zwischen den Objekten.[24]
Objekte entstehen durch Instanziierung einer Klasse, dabei bildet eine Klasse den „Bauplan“ für das Objekt. In der Laufzeit des Programms werden aus Klassen Objekte instanziiert, dabei werden die Eigenschaften der Klasse mit Werten gefüllt.[25] Durch Vererbung können Unterklassen von Oberklassen bestimmte Eigenschaften und Methoden übernehmen, ohne sie selbst noch einmal neu zu implementieren. Daraus geht der Polymorphismus hervor. Das bedeutet: Geerbte Methoden und Eigenschaften können überschrieben und mit demselben Bezeichner neu definiert werden.[26]

[20] Mehrere Wörter werden zusammenhängend jeweils mit einem Großbuchstaben begonnen, zum Beispiel TBasisKlasse
[21] vgl. BRAUN; ESSWEIN; GREIFFENBERG, S. 7
[22] vgl. BRAUN; ESSWEIN; GREIFFENBERG, S. 9
[23] vgl. BRAUN; ESSWEIN; GREIFFENBERG, S. 17
[24] vgl. BRAUN; ESSWEIN; GREIFFENBERG, S. 18
[25] vgl. BRAUN; ESSWEIN; GREIFFENBERG, S. 19
[26] vgl. BRAUN; ESSWEIN; GREIFFENBERG, S. 20 ff.

Damit Objekte untereinander kommunizieren können bedarf es diverser Schnittstellen zwischen ihnen. Interfaces werden von Klassen implementiert und gewährleisten, dass das Objekt die erforderlichen Methoden zum Empfangen von Nachrichten enthält.[27]

Für die programmatische Umsetzung der Druckschnittstelle in die objektorientierte Programmiersprache Delphi bedarf es der theoretischen Grundlagen für die Implementierung der Klassen. Die Anforderungen zum Ansteuern der Druckkomponente und zur programmierlogischen Umsetzung wurden in den vorherigen Kapiteln beschrieben.
Dabei sind ausgehend von dem Konzept und der Analyse der List & Label Report-Komponente folgende Klassen entstanden.

3.2.2 DLL Hilfsklasse „TL18_Libraries“

Die DLL Hilfsklasse ist für das ordnungsgemäße Laden und Entladen der List & Label Programmbibliotheken zuständig. Weiterhin sorgt sie während des Programmablaufs dafür, dass die DLLs im Hauptspeicher stehen und das Vocus Lohn- und Gehaltsprogramm darauf zugreifen kann. Die Klasse wird beim Projektstart instanziiert und beim Beenden des Programms wieder freigegeben. So steht die Klasse in einer globalen Instanz programmweit zur Verfügung. Ein Vorteil bietet sich dadurch für die Programmierer: Sie können, wenn es die Situation erfordert, selbstständig prüfen, ob die DLLs geladen und die Reporting-Komponente zur Verfügung steht. Wenn nicht, können sie dementsprechend reagieren, Fehlermeldungen ausgeben und weitere Ausführungen des Programms verhindern, welche zwangsläufig auf die Druckfunktionen zugreifen müssen. Die DLLs müssen für einen einwandfreien Programmablauf in einem Unterverzeichnis des Lohn- und Gehaltsprogramm-Systemverzeichnisses liegen. Dies ist im Regelfall auch durch diverse Installations- und Updateroutinen gewährleistet. Sollten die DLLs nicht auffindbar sein, deutet dies auf eine fehlerhafte Installation oder ein fehlgeschlagenes Update hin.
List & Label sieht standardmäßig vor, dass alle zugehörigen DLLs im gleichen Verzeichnis wie das Programm liegen.[28] Um jedoch weiterhin an der Struktur des Verzeichnisaufbaus des Gehaltsprogrammes festhalten zu können, mussten die DLLs in einem Unterverzeichnis untergebracht und die von List & Label bereitgestellten Laderoutinen der DLLs dahingehend verändert werden. Die Klasse greift direkt auf die Laderoutinen zu und überprüft diese auf korrekte Ausführung.
Zur Überprüfung der Einsatzbereitschaft von List & Label stellt die Klasse eine öffentliche Eigenschaft bereit, welche „True“ meldet, wenn alle DLLs erfolgreich geladen

[27] vgl. BRAUN; ESSWEIN; GREIFFENBERG, S. 22
[28] online: vgl. COMBIT GMBH, 2013, S. 85 (17.7.2013)

werden konnten. Im Fehlerfall gibt sie „False“ zurück. Nachfolgendes Klassendiagramm der DLL Hilfsklasse soll den Sachverhalt verdeutlichen:

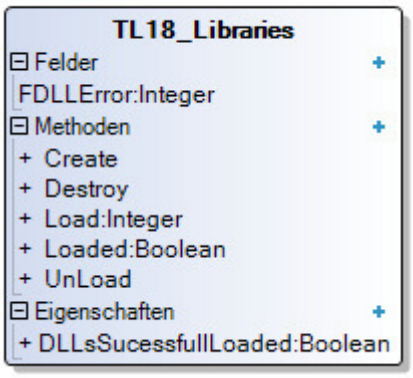


Abbildung 5 Klassendiagramm TL18_Libraries

Die Klasse implementiert die Methoden „Load“, „Loaded“ und „UnLoad“. „Create“ und „Destroy“ stellen jeweils den Konstruktor und Destrukor dar. Sie wird im „initialisation“-Abschnitt der Delphi-Unit in ein globales Objekt „L18_Libraries“ instanziiert, so dass jede weitere Unit, welche auf diese verweist, Zugriff auf das Objekt bekommt. Beim Programmstart wird in der Unit „fProject“ die grundlegende Initialisierung für Vocus Lohn und Gehalt durchgeführt. Hierbei wurde an die Stelle der alten Schnittstelle angeknüpft und das Laden der DLLs, sowie die Schnittstelleninitialisierung, implementiert. Der Quellcode hierfür liegt unter Anhang 2.
Können die DLLs aus dem angegeben Verzeichnis (hier der Pfad unter der der Lohn und Gehalt-Installation im Ordner namens „cmbtll“) erfolgreich geladen werden, erfolgt die Initialisierung der systemweiten Druckschnittstelle mit allen ihren Parametern. Die Beschreibung hierfür folgt in Kapitel 3.2.4 (Reportklasse für Druckschnittstelle „TL18_Report“). Wird ein Fehlercode ungleich „0“ zurückgegeben, erfolgt eine Dialogmeldung an den Nutzer, dass die Reportmodule nicht gefunden wurden.
Nach erfolgreichem Programmstart kann die Druckschnittstelle benutzt werden. Hierfür wird die List & Label-Basisklasse benötigt.

3.2.3 List & Label Klasse „TDBL18_“

Die von List & Label bereitgestellte Klasse „TDBL18_“ erbt von der Basisklasse „TL18_“ und erweitert diese um die Fähigkeit der Datenanbindung mittels eines „TDataSource“-Objektes.[29] So bietet sie alle Methoden, um List & Label so zu konfigurieren, dass eine Datenverknüpfung hergestellt und der Druckablauf mit den Daten aus der Datenbank gesteuert werden kann.
List & Label benötigt, um auch beim Kunden korrekt zu funktionieren, einen Lizenzstring, welcher diesem Objekt zu Beginn übergeben werden muss. Weiterhin bietet

[29] online: vgl. COMBIT GMBH, 2013, S. 76 (17.7.2013)

List & Label an, diverse Exportmodule, die dem Kunden nicht zur Verfügung stehen sollen, programmseitig zu deaktivieren.
Um alle diese Funktionen zu nutzen und an die spezifischen Bedürfnisse des Vocus Lohn und Gehalt anzupassen, wurde diese Klasse vererbt und eine Unterklasse erstellt, welche den Typenbezeichner „TL18_Lic“ trägt. Diese erhält durch das Prinzip der Vererbung der objektorientierten Programmiersprache alle Methoden ihrer Oberklasse. So ergibt sich folgendes Klassendiagramm:

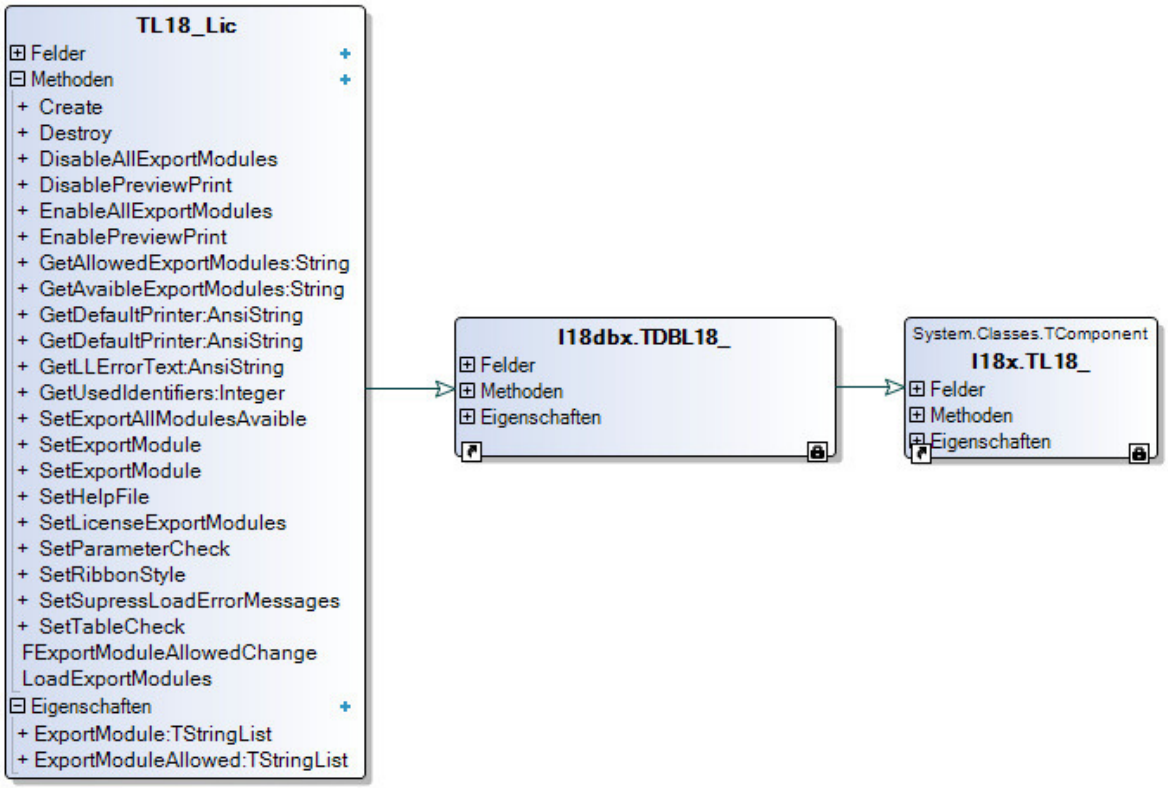


Abbildung 6 Klassendiagramm TL18_Lic

Aus Gründen der Übersichtlichkeit wurde auf die detaillierte Ausführung und Kennzeichnung aller Methoden und Eigenschaften der Basisklassen von „TL18_Lic“ verzichtet. Es soll lediglich veranschaulicht werden, in welcher Reihenfolge die Klassen vererbt werden und wie die, für die Druckschnittstelle geschaffene, Unterklasse aufgebaut ist. Diese speichert alle von List & Label zur Verfügung gestellten Exportmodule in einer Liste und hält in einer anderen, die für den Endanwender (per Lizenzfreischaltung) verfügbar sein sollen. Mit den im Klassendiagramm dargestellten Methoden lässt sich die List & Label Installation weiterhin konfigurieren: Es ist möglich eine spezifische Hilfedatei zu setzen, die beim Anfordern der Hilfe in einem List & Label Dialog geöffnet wird. Auch Fehlertexte und Hinweise über den Druckverlauf können ausgegeben oder unterdrückt werden. Weiterhin ist es möglich, das Aussehen der Druck- und Designerdialoge festzulegen.

Diese Klasse bildet den Kern der Druckschnittstelle. In ihr sind alle essentiellen Funktionen für den Druckablauf und für den Zugriff von List & Label verankert. In einer weiteren Klasse wird nun der Report im Gesamten gefasst und für den Druck vorbereitet. Diese Klasse wird die „TL18_Lic“-Klasse in einem Objekt instanziieren, um auf sie zuzugreifen.

3.2.4 Reportklasse für Druckschnittstelle „TL18_Report“

3.2.4.1 Grundlagen

List & Label arbeitet datenbankunabhängig und benötigt daher keinen eigenen Datenbanktreiber. Dafür muss der gesamte Druckablauf und die Datenversorgung selbst programmiert werden. Dies bringt aber enorme Vorteile wie:

- die vollständige Kontrolle über den Druckablauf
- eine höhere Geschwindigkeit beim Druck
- das Verwenden von speziellen Systemen wie im Falle der Vocus Lohn- und Gehaltssoftware, das Verwenden einer DataList.[30]

Für List & Label ist es nur wichtig, dass es Daten aus einer Datenquelle beziehen kann.[31] Die Daten für die Reports in Vocus Lohn und Gehalt werden mit DataLists übergeben. DataLists können mehrere Data-Objekte aufnehmen, welche wie ein Datensatz in einer Tabelle aufgebaut sind. Dabei kann jeder Satz (ein Data-Objekt) eine eigene Struktur haben. Wichtig ist, dass Datensätze, die im Report in einer Tabelle erscheinen sollen, die gleiche Struktur haben. Dass bedeutet wiederrum: Sie müssen die gleiche Anzahl an Feldern haben und in ihren Feldnamen übereinstimmen. Sollen Data-Objekte untereinander in Master-Detail Beziehungen gebracht werden, muss die Parent-Eigenschaft auf das übergeordnete Data-Objekt zeigen. Auf diesem Wege lassen sich hierarchische Reports erstellen.
Dies führt zu folgender Problematik: Nach dem Übergeben der DataList müssen die Daten in Datenbankobjekte „umgewandelt“ werden, damit sie List & Label zur Verfügung stehen. Um List & Label lokal und temporär für jeden Druckauftrag eine Datenbank bereitzustellen, werden die Daten in ein Objekt der Klasse „TClientDataSet“ kopiert. Diese Komponente gestattet das Halten von Daten wie in einer Tabelle. Um Master-Detail Datensätze zu realisieren, wird zu jeder Struktur eines Data-Objektes ein ClientDataSet instanziiert und in einer Liste gehalten. Die Abbildung der Tabellenbeziehungen untereinander geschieht dabei durch die Überprüfung der Parent-Eigenschaft der Data-Objekte. Verweist ein Data-Objekt auf ein anderes, wird eine

[30] online: vgl. COMBIT GMBH, 2013, S. 87 f. (17.7.2013)
[31] Anm.: Auch Daten aus einer anderen Quelle als eine Datenbank können mit List & Label gedruckt werden, allerdings ist hierfür eine andere Komponente zuständig und einige Funktionen, wie Drill-Down in Berichten, stehen nicht zur Verfügung.

Verknüpfung zu dieser Tabelle hergestellt. Bedingung hierfür ist, dass sich beide Data-Objekte in der Struktur unterscheiden.
Parallel dazu erzeugt eine Methode der Klasse für jedes ClientDataSet DataSources, welche die Daten dann an List & Label übergeben. Für das Abbilden und Übergeben der Datenstruktur an List & Label, bedarf es einer vorherigen Analyse der Tabellen und den beinhaltenden Feldern. Hierbei werden die einzelnen ClientDataSets auf ihre Felder überprüft und in eine Liste geschrieben.
Mit Master-Detail Beziehungen ist es möglich, Informationen in einem Report detailliert darzustellen. Hierbei bildet der Masterdatensatz die Grundlage und wird durch die Detailinformationen näher beleuchtet. In modernen Reports bilden diese Verknüpfungen die Grundlage für strukturierte und übersichtliche Informationen. Für die Verwendung von Drill-Down Berichten bedarf es der Master-Detail Beziehungen. Eine Methode analysiert diese und speichert sie in einer Liste ab.
Durch die Übergabe der Informationen an den Report kennt List & Label nun die Tabellen und kann daraufhin die enthaltenen Daten verarbeiten. Dieser Vorgang wird auch als das „Mappen der Tabellen“ bezeichnet.
Aus den Grundlagen und Überlegungen zu den Abläufen lassen sich die gesuchten Methoden und Eigenschaften für die Klasse „TL18_Report“ ableiten.

3.2.4.2 Aufbau der Klasse

Für die Verwendung von List & Label innerhalb der „TL18_Report“-Klasse ist eine Instanziierung von „TL18_Lic“ erforderlich. Diese stellt alle Funktionen bereit, um List & Label anzusteuern. Die Komponente bedarf einer grundlegenden Initialisierung bei jedem Start des Programms. Die Initialisierungsroutine setzt diverse Pfade wie zum Listenverzeichnis und temporären, lokalen Verzeichnis des Nutzers. Außerdem gilt es hier die Einstellungen für das Verhalten von List & Label festzulegen.
Essentiell für die Klasse ist die Methode, welche den Druckablauf einheitlich steuert. Dabei spielt es keine Rolle, welchen Vorgang der Nutzer gestartet hat (Druck, Vorschau oder Export). Daran anschließend soll der Anwender auch in der Lage sein seinen Drucker zu konfigurieren um beispielsweise die Anzahl der gewünschten Kopien, schwarz-weiß Druck und andere vom Druckertreiber zur Verfügung gestellten Konfigurationsmöglichkeiten zu nutzen.
Der List & Label Designer ist für das Erstellen der Layouts der Reports zuständig. Für dessen Verwendung ist die Implementierung einer Routine erforderlich, welche den List & Label Designer öffnet und ihm die Daten übergibt.

Das Aufbereiten der Daten für den Druck oder das Designen von Reports unterscheidet sich nicht, so dass hier eine einheitliche, nicht öffentliche Prozedur diese Aufgabe übernehmen kann. Der Prozedur wird die aktuelle Datenmenge übergeben und diese überträgt sie in die ClientDataSets.

Für die Archivierung der Listen muss es möglich sein die Layoutdatei und die Daten zusammen abzuspeichern, um sie bei Bedarf wieder auszulesen und den Report neu zu erzeugen. Die DataList verfügt über die Möglichkeit, ihre Daten in das XML-Format (Extensible Markup Language) zu übertragen. Die Reportlayouts werden bereits in einer Textdatei abgelegt. So ist es möglich, beide Dateien zu einer zusammenzufassen und diese im Archiv komprimiert und verschlüsselt abzulegen. Im Vocus Lohn und Gehalt Programm gibt es bereits eine einheitliche Komprimierungs- und Verschlüsselungsfunktion, welche hierfür geeignet ist.

Nach erfolgreichem Druck und ggf. der Ablage im Archiv muss das Objekt wieder ordnungsgemäß bereinigt, Einstellungen auf Standard zurückgesetzt und nicht mehr benötigte Ressourcen wieder freigegeben werden.
Es spielen noch viele weitere Methoden und Funktionen eine Rolle. Letztendlich wird der Report jedoch über die „Execute“-Prozedur aufgerufen. Ihre Parametrierung wurde bereits in einem Data Dictionary (Tabelle 2 Data Dictionary der Druckschnittstelle) erläutert. Die Prozedur entscheidet anhand der Parameter wie mit dem Report zu verfahren ist. Wird über den Parameter beispielsweise festgelegt, dass ein Druckdialog angezeigt werden soll, blendet sie den Dialog ein und entscheidet anhand der Auswahl des Nutzers über den weiteren Ablauf der Prozedur. Dabei gibt sie den eigentlichen Ablauf des Druckes und auch analog den des Exports und der Vorschau an die „Print“-Methode ab und startet diese mit den jeweiligen Parametern. Auf die genaue Methodik dieser Prozeduren (und auch einige anderer) wird im Kapitel 3.3 eingegangen.
Um die Druckschnittstelle von allen Programmpunkten aus zu nutzen, wird die Klasse in einer programmweiten, globalen Instanz („L18_Report“) bereitgestellt. Programmierer, die die Schnittstelle verwenden möchten, können über die globale Instanz auf sie zugreifen. Lediglich auf die „fReportListLabel“-Unit muss in ihrem Code verwiesen werden. Das vollständige Klassendiagramm hierzu ist unter Anhang 3 zu finden.

3.2.5 Threadklasse für Echtdatenvorschau im Designer „TLLPrintThread“

Im Designer von List & Label ist es möglich, neue Reports zu erstellen oder bereits existierende zu bearbeiten. Dabei besteht die Möglichkeit sich im Designer den Report mit Live-Daten anzuschauen. Für die Entwickler des Vocus Lohn- und Gehaltsprogramm bietet diese Funktionalität einen enormen Vorteil, um Zeit bei der Reporterstellung zu sparen.

Verglichen mit der bisherigen Druckschnittstelle bedurfte dies einer Anpassung der Textdateien und des dazugehörigen Quellcodes. Dies bedeutet automatisch, dass

nach der Reportbearbeitung eine Überprüfung der Darstellung nur nach einem Programmneustart möglich war. Durch die Möglichkeit der Echtdatenvorschau direkt im Designer entfällt dieses Problem. Weiterhin bietet List & Label an, Reports direkt aus dem Designer zu drucken oder zu exportieren. Da hierfür die Abläufe dieselben sind, können alle Fälle einheitlich unter der Problematik der Echtdatenvorschau beschrieben werden.
Um die Echtdatenvorschau zu nutzen, benötigt List & Label Zugriff auf Threads. In der objektorientierten Programmiersprache Delphi steht hierfür eine Klasse „TThread“ zur Verfügung. Die Klasse implementiert unter anderem folgende Methoden und Eigenschaften zur Behandlung von Threads:

Methode	Beschreibung
Execute	Die abstrakte Methode muss überschrieben werden, in ihr wird festgelegt, was in dem Thread zu tun ist. Sollte die Eigenschaft „Terminated“ auf „True“ stehen, muss Execute verlassen werden.
Resume	Mit Resume wird ein angehaltener Thread wieder fortgesetzt.
Suspend	Hält einen Thread an.
Synchronize	Zugriffe auf Objekte werden aus Threads heraus synchronisiert.
Terminate	Die „Terminated“-Eigenschaft wird auf True gesetzt, Execute sollte verlassen werden, um den Thread zu beenden.

Tabelle 3 Methoden von der Klasse TThread

(eigene Darstellung in Anlehnung an KOSCH, S. 322)

Die Kernmethode dieser Klasse ist die „DesignerPrintPreview“-Prozedur. Der Aufruf erfolgt sobald die Echtdatenvorschau mittels Execute gestartet und im weiteren Verlauf synchronisiert wird. Zuerst sendet sie eine Botschaft an das Designerfenster, welches die Toolbar daraufhin aktualisiert. Die Methode erstellt eine temporäre Projektdatei und übergibt diese List & Label. Weiterhin wird List & Label über den Start der Vorschau benachrichtigt, so dass der Designer mit dem Erzeugen und Anzeigen der Vorschau beginnt. Das Vorschau-Control erhält nun die Kontrolle über die Vorschaudatei. Darauf folgt eine einfache Druckroutine, welche die Daten in der Vorschau anzeigt. Zum Abschluss der Routine wird die „Terminate“-Methode aufgerufen und der Thread beendet.
Das in „TL18_Report“ implementierte „TL18_Lic“-Objekt enthält ein Event, welches „gefeuert“ wird, sobald der Entwickler die Echtdatenvorschau im Designer aufruft.

„TL18_Report“ bildet mit der Methode „DesignPrintJob“ den Eventhandler des Events, empfängt die Nachricht und initiiert den Start des Threads. Entsprechend des Inhaltes der empfangenen Nachricht startet die Prozedur unterschiedliche Aktionen: Export, Vorschau oder Druck, Vorgang abbrechen oder beenden.

Mit Hilfe der Thread-Klasse und deren Implementierung durch „TLLPrintThread“ wird eine Echtdatenvorschau zur Designzeit realisiert und der Entwickler kann schnell seinen Report auf das richtige Layout bringen.

3.3 Erläuterung ausgewählter Methoden

3.3.1 Laden der Druckschnittstelle

Im Folgenden sollen einige Kernmethoden der Klasse „TL18_Report“ erläutert werden, welche einen wesentlichen Beitrag zum Druck leisten.
Damit die Druckfunktionen auf verschiedenen Kundeninstallationen korrekt arbeiten, bedarf es der Initialisierung der Pfadangaben zu diversen Verzeichnissen (wie Listen-, Exportverzeichnis) und Druckeinstellungen beim Programmstart. Die globale Startroutine des Programms in der Unit „fProject“ ermittelt diese Pfadangaben und Einstellungen des Benutzers und übergibt sie der Druckschnittstelle. Dabei sind Folgende drei Pfadangaben für die Schnittstelle von Bedeutung:

1. Pfad zum Listenverzeichnis des Lohnamtes
2. Temporäres Verzeichnis des jeweiligen Nutzers
3. Exportverzeichnis des Lohnamtes

Der Pfad zum Listenverzeichnis wird zum Laden der richtigen Layoutdateien benötigt, welche sich in einem Verzeichnis befinden müssen. Beim Ansteuern der Schnittstelle ist diese dann in der Lage anhand des Dateinamens die korrekte Liste aus dem Listenverzeichnis zu laden. Für die Entwickler der Vocus GmbH bedeutet das, dass sie beim Ansteuern der Schnittstelle nicht den kompletten Pfad zur Liste übergeben müssen, sondern nur den Dateinamen. Dies mindert die Gefahr, dass Layouts aufgrund falscher Pfandangaben nicht auffindbar sind. Das temporäre Verzeichnis des Nutzers wird von List & Label zur Generierung von Vorschaudateien benötigt. Beim Schließen des Vorschaudialoges löscht eine Funktion alle, während der Vorschau erzeugten, im temporären Verzeichnis gespeicherten, Dateien.
Sollte der Entwickler einen stillen Export durchführen, legt die Druckschnittstelle standardmäßig alle erstellten Dateien in das Exportverzeichnis des Lohnamtes. Stiller Export bedeutet, dass der Entwickler während des Exports keine Benutzerinteraktion zulässt.
Ein weiterer Parameter übergibt die Lizenzinformationen von der jeweiligen Vocus Lohn und Gehalt-Installation. Über diese schaltet die Laderoutine diverse Exportoptionen, abhängig vom Lizenzstatus, frei.
Zuletzt übergibt die Routine Programminformationen wie Version- und Stationsnummer, um diese auf den Reports andrucken zu können. Die Prozedur verarbeitet die übergebenen Parameter und anschließend lädt List & Label die Reportkomponenten

mit den gesetzten Einstellungen. Dies beinhaltet auch das Instanziieren der Klasse, die die Echtdatenvorschau mittels Thread aus dem Designer realisiert, instanziiert wird. Der Quellcode für diese Methode ist unter Anhang 4 hinterlegt.
Wurde die List & Label Komponente erfolgreich geladen, kann die Druckschnittstelle im gesamten Programm verwendet werden.[32]

3.3.2 Umwandlung der DataList in ClientDataSets

Die Methode „DataListToClientSets“ ist für die Umwandlung der DataList in ClientDataSets verantwortlich. Das Umwandeln in Datenbank-Objekte ist für das Drucken mittels List & Label grundlegend. ClientDataSets verhalten sich wie Tabellen einer Datenbank. Ihre Besonderheit besteht darin, dass sie temporär im Hauptspeicher liegen und an keine Datenbank gebunden sind.[33] List & Label kann Datensätze aus diesen Datenbank-Objekten laden und im Report anzeigen. Das vollständige PAP der Methode ist unter Anhang 5 und der Quellcode unter Anhang 6 hinterlegt.
Die Klasse TL18_Report enthält folgende Felder, die für diese Methode von Bedeutung sind:

Feld	Beschreibung
FClientDataSets	Enthält die Liste aller aus der DataList erzeugten ClientDataSets für den Druck.
FDataSources	Für jedes ClientDataSet steht ein DataSource-Objekt zu Verfügung um die Daten des DataSets zu übergeben, diese werden in der FDataSources Liste verwaltet.
FDataStructureList	Speichert alle erkannten Strukturen der DataList.
FDataList	Enthält die zu konvertierende DataList, welche der Druckschnittstelle übergeben wurde.

Tabelle 4 Verwendete Felder der DataListToClientDataSets-Methode

Die Prozedur initialisiert zu Beginn alle Felder. D.h., es werden die objektinternen Listen geleert und die lokalen Variablen auf die Ausgangswerte gesetzt. Dann prüft der Algorithmus, wie viele Einträge sich in der DataList befinden. Sollten keine Einträge vorhanden sein, wird die Prozedur abgebrochen, da keine Daten zum Drucken verfügbar sind.
Sind Daten vorhanden, nummeriert der Algorithmus im ersten Schritt alle Datensätze, um sie später eindeutig zuordnen zu können. Der Eintrag der laufenden Nummer erfolgt im Datenfeld „DATA_ID“. Weiterhin wird geprüft, ob dieses Datenfeld bereits

[32] Anm.: Derzeit befindet sich die Schnittstelle noch im internen Test, weswegen im Quellcode Abfragen zur Verwendung von List & Label hinterlegt sind.
[33] online: vgl. EMBARCADERO TECHNOLOGIES, 2012 (11.8.2014)

verfügbar ist und die Daten bereits eine eindeutige Nummer haben. Diese wäre beispielsweise der beim Laden eines Reports aus dem Archiv der Fall.
Mittels einer while-Schleife wird nun die DataList durchlaufen und jedes enthaltene Data-Objekt ausgewertet. Zeigt die „Parent“-Eigenschaft des Data-Objektes nicht auf die DataList und ist er auch nicht „nil“, muss ein Algorithmus den zugehörigen, verknüpften Masterdatensatz finden. Dies übernimmt die Subroutine „GetMasterData“. Sie sucht nach dem verknüpften Data-Objekt und liefert den gefundenen Masterdatensatz an die lokale Variable „MasterData“ zurück. Die laufende Nummer des Masterdatensatzes wird im aktuellen Data-Objekt in dem Feld „DATA_PARENT_ID“ gespeichert.
Im nächsten Schritt wird geprüft, ob die Struktur des Data-Objektes bekannt ist und hierfür bereits ein ClientDataSet erzeugt wurde. Ist die Struktur noch nicht bekannt, wird das ClientDataSet in der Unterroutine „CreateClientDataSet“ erzeugt und die aktuelle ID dessen zurückgegeben.
Die Daten des Data-Objekts speichert die Routine nun im ClientDataSet. Hierfür wird es mit diversen Routinen auf die Aufnahme eines Datensatzes vorbereitet. Mittels einer for-Schleife fügt die Routine alle bekannten Felder dem ClientDataSet hinzu. Jedes Feld besitzt einen bestimmten Datentyp. Danach entscheidet sich wie der Wert im ClientDataSet abgelegt wird. Nach der Abarbeitung aller Felder folgt die Übermittlung des Datensatzes an das ClientDataSet mittels „Post“ und die Freigabe für einen weiteren Datensatz.
Abschließend prüft der Algorithmus den Wert der lokalen Variablen „i“ und vergleicht ihn mit der Anzahl der Datensätze in der DataList. Entspricht dieser dem letzten Satz wird „EndOfDataList“ auf „True“ gesetzt und somit die while-Schleife verlassen. Um den nächsten Datensatz auslesen zu können, wird „i“ um eins inkrementiert.
Die „DataListToClientDataSet“-Methode bietet grundlegende Operationen, um die variable Datenmenge einer DataList für List & Label, in Form von Datenbankobjekten, bereitzustellen. Die lokale Datenbank wird in der Druckroutine durchlaufen, um die Daten dem Report zu übergeben und zu drucken.

3.3.3 Zentrale Druckfunktion

Die zentrale Druckfunktion wird innerhalb der Execute-Methode aufgerufen und bildet den Kern aller Druckabläufe. Je nach Parametern wird ein Druck ausgeführt, die Vorschau eingeblendet oder der Report in ein bestimmtes Dateiformat exportiert.
Folgende Parameter benötigt die Funktion zur Steuerung des Druckablaufes:

Parameter	Beschreibung
aDataList	Übergabe der zu druckenden Datenmenge
ListFileName	Listenname im Listenverzeichnis
ListName	Anzeigename der Liste
PrintOption	Druckoptionen für List & Label
ExportOutput	Zielformat für eventuellen Export
ExportFileName	Zieldateiname für eventuellen Export
ExportOptions	Einstellungen für das gewählte Exportformat

Tabelle 5 Parameter der internen Druckfunktion

Der Quelltext ist unter Anhang 7 zu finden und läuft wie folgt ab:
Zuerst wird das Feld „FReportIsClean“ der Klasse „TL18_Report“ geprüft. In ihm ist hinterlegt, ob die Eigenschaften des Report-Objekts auf Standardeinstellungen gesetzt sind. Ist dies nicht der Fall bereinigt eine Methode den Report und setzt alle Einstellungen für das Erstellen eines neuen Reports. Um zu überprüfen, ob die im Parameter „aDatalist“ übergebene Variable gesetzt ist, wird sie auf „nil“ getestet. Enthält die Variable einen gültigen Pointer, speichert die Prozedur sie im internen Feld „FDataList“ des Report-Objektes ab.
Die Funktion „GetListFileName“ ermittelt anhand des übergeben Listennamens den vollständigen Pfad zur Listendatei, sofern sie im Listenverzeichnis abgelegt ist. „ListName“ wird mit einer Methode dem ganzen Objekt mittgeteilt und die in der „Load“-Funktion übergebenen Programminformationen dem aktuellen Report hinzugefügt. Abhängig vom Parameter „PrintOption“ setzt die Funktion abschließend noch diverse Einstellungen. Die Druckvorbereitungen sind an dieser Stelle abgeschlossen. Im weiteren Verlauf realisiert der Algorithmus die Datenübergabe zum Druck. Um die Datenmenge umzuwandeln wird die im vorherigen Kapitel beschriebene Funktion „DataListToClientDataSets“ aufgerufen.
Die dabei entstandenen Tabellen und ihre Beziehungen zu einander analysiert der Aufruf von „MapTables“ und „PassDataStructure“ übergibt sie an List & Label. Wurde als Ausgabeoption Export festgelegt, bestimmen die gesetzten Parameter das Exportformat und dessen Konfiguration.

Es folgt der eigentliche Start der Druckroutine:
Der Aufruf von „LLPrintWithBoxStart“, welcher von der Klasse TL18_Lic bereitgestellt wird, informiert List & Label über den Druckstart. Diese liefert bei einem erfolgreichen Start des Druckjobs den Code „0“ zurück. Tritt ein Fehler auf, liefert sie einen Fehlercode zurück und gibt dessen Fehlernachricht aus.

Das interne Fehlerprotokoll des Vocus Lohn- und Gehaltsprogramms nimmt diesen anschließend auf. Der Kunde kann sich dann an den Support der Firma Vocus wenden, welcher sich umgehend um eine Fehlerbehebung kümmert.
Sollte der Anwender einen Exportauftrag an die Druckroutine gesendet haben, informiert sie List & Label über das Zieldateiformat. Ist ein Drucker Ziel des Auftrags, übernimmt der Algorithmus die eingestellte Anzahl von Kopien aus dem Druckdialog.
Es folgt eine Überprüfung, ob der Benutzer den Druckauftrag vorzeitig abgebrochen hat, wenn ja wird auch der Druckprozess beendet und die Druckfunktion sofort verlassen. Wenn nicht, beginnt der Algorithmus mit dem Auslesen der Daten und dem Hinzufügen zum Report. Es folgt eine Überprüfung der Mastertabellen mit dem Reportlayout. Ist eine Anzahl ungleich „0“ vorhanden, beginnt eine repeat-until-Schleife mit dem Auslesen der Tabellen bis alle Untertabellen der Mastertabelle angedruckt wurden. Die Funktion „PrintDataView“ wird dabei rekursiv für jede Untertabelle aufgerufen. Ihr Quellcode ist unter Anhang 8 dokumentiert.
Sie erhält als Parameter das zu bearbeitende ClientDataSet, welche Tiefe die Tabelle in der Struktur besitzt (Level) und die eben gedruckte Tabellennummer.
Folgender Algorithmus liegt ihr zugrunde:
Es erfolgt zunächst eine Prüfung auf den Tabellennamen. Ist es eine statische Tabelle ohne Inhalt aus einem ClientDataSet, wird ihr Inhalt ausgegeben und ggf. Seitenumbrüche ausgelöst. Handelt es sich um eine Tabelle mit dynamischen Daten aus einem ClientDataSet, wird der Zeiger auf den ersten Datensatz positioniert. Für das Anzeigen einer Fortschrittskontrolle werden die Hauptdatentabellen gezählt und deren Anzahl als maximaler Wert für den Fortschrittsbalken festgelegt. In der Layoutdatei ist festgelegt wie die Daten zu sortieren sind. Die Art der Sortierung wird ausgelesen und in der Variable „SortOrder“ gespeichert.
Anschließend wird das ClientDataSet in einer for-Schleife durchlaufen und der aktuelle Record gedruckt. Sollte der letzte übergebene Datensatz nicht mehr auf die Seite gepasst haben, meldet List & Label „LL_WRN_REPEAT_DATA“ und löst einen Seitenumbruch aus. Der Algorithmus übergibt daraufhin den Datensatz erneut zum Druck.
Besitzt der letzte gedruckte Datensatz einen zugehörigen Detail-Datensatz, wechselt „PrintDataView“ zu der entsprechenden Tabelle und ruft sich selbst rekursiv auf.
Die zu druckende Untertabelle wird mittels der gemappten Tabellenstruktur gefunden und dem Algorithmus rekursiv übergeben. Der Druck eines Datensatzes aus einer Haupttabelle bewirkt schließlich, dass die Fortschrittsanzeige aktualisiert wird. So kann der Nutzer erkennen wie weit sein Druckauftrag fortgeschritten ist. Am Ende der for-Schleife wird der Zeiger des ClientDataSets auf den nächsten Datensatz verschoben. Hat der Algorithmus alle Daten des ClientDataSets rekursiv durchlaufen, ruft dieser „LLPrintFieldsEnd“ auf und benachrichtigt List & Label so über das Ende einer Mastertabelle. Danach findet ggf. ein Wechsel zur nächsten Haupttabelle statt.

Nachdem alle Tabellen durchlaufen sind, erhält List & Label eine Information („LLPrintEnd“), dass der Druckauftrag abgeschlossen ist.
Die Routine gewährleistet die schnelle Bearbeitung und Durchführung des Druckauftrags. Alle relevanten Operationen finden mit Daten aus dem Hauptspeicher statt.
Standardmäßige Überprüfungsroutinen sind deaktiviert worden, da diese nur zusätzliche Rechenzeit in Anspruch nehmen und keine sinnvollen Informationen für den Benutzer liefern. Der integrierte Debugmodus in List & Label kann hierbei zur Fehlersuche verwendet werden.

3.4 Dialoggestaltung

3.4.1 Definition „Dialog“

Ein Druckaufruf wird unter anderem dadurch gekennzeichnet, welche Möglichkeit der Nutzer in Interaktion mit dem Programm hat. Um ihm die Interaktion so benutzerfreundlich wie möglich zu gestalten, wurde ein neuer Druckdialog entwickelt.
HERCEG beschreibt den Dialog als „ein Kommunikationsmedium, das Präsentations-, Verhaltens-, Zustands- und Unterstützungseigenschaften besitzt.“[34] Er benennt diese Eigenschaften auch als „Look and Feel“ einer Benutzerschnittstelle.[35] Über die Ein- und Ausgabeschnittstelle sind Regeln definiert, wie der Benutzer und das System kommunizieren können. Unter Eingaberegeln sind die verfügbaren Mittel zu verstehen, mit denen ein Benutzer eine oder mehrere Aktionen im System auslösen kann. Hierzu zählen die Eingabegeräte wie Maus und Tastatur. In den Ausgaberegeln ist festgelegt, wie Objekte (zum Beispiel Eingabemasken, Dialogfenster) und Operationen (zum Beispiel Ergebnisse einer Abarbeitung, Statusinformationen) der Software, dem Nutzer zu präsentieren sind.[36] Folgende Grafik stellt den Dialog als Schnittstelle zwischen Benutzer und Software dar:

Abbildung 7 Dialog als Schnittstelle zwischen Benutzer und Software

(eigene Darstellung in Anlehnung an HERCZEG, S. 104)

Über den Dialog interagieren Benutzer und Software miteinander. Die Software kann Informationen über ihren Zustand mittels Dialog an den Nutzer übermitteln und den Nutzer zur Eingabe von Informationen auffordern. Der Dialog ist demnach für den

[34] HERCZEG, S. 103
[35] vgl. HERCZEG, S. 103
[36] vgl. HERCZEG, S. 104

Nutzer ein Werkzeug, um mit dem Software-System zu kommunizieren. Die DIN-Norm 66234 Teil 8 beschreibt 5 Gestaltungsgrundsätze[37] zur Dialoggestaltung:

1. Aufgabenangemessenheit
2. Selbstbeschreibungsfähigkeit
3. Steuerbarkeit
4. Erwartungskonformität
5. Fehlerrobustheit

Die Grundsätze sollten unter der Berücksichtigung der jeweiligen Nutzer angewandt werden.[38] Zusammenfassend können die Grundsätze wie folgt beschrieben werden:

Grundsatz	Erläuterung
Aufgaben-angemessenheit	Der Dialog soll den Nutzer bei seinen Aufgaben unterstützen, ohne ihn zu überlasten.
Selbst-beschreibungs-fähigkeit	Der Benutzer muss diverse Möglichkeiten haben, den Dialog zu verstehen. Dabei kann der Dialog selbsterklärend sein oder durch Unterstützung eines Hilfesystems erläutert werden.
Steuerbarkeit	Die Steuerbarkeit der Kommunikation obliegt entweder dem Benutzer oder dem System. Wird der Dialog als Gesamtes betrachtet, sind drei Dialogtypen zu unterschieden: • Systemgesteuerter Dialog: Das System fragt Eingaben vom Benutzer ab, der Benutzer hat keinen Einfluss auf die nächste Aktion. • Benutzergesteuerter Dialog: Der Benutzer startet Aktionen und das System meldet den Status der Abarbeitung, das System hat keinen Zugriff auf die nächste Aktion. • Gemischter Dialog: Die Kontrolle wechselt innerhalb des Dialoges zwischen Benutzer und System.
Erwartungs-konformität	Benutzer entwickeln mit dem Verwenden des Systems Erfahrungen über die Steuerung der Software und erwarten beim Interagieren mit ihr ein bestimmtes Verhalten.
Fehlerrobustheit	Der Benutzer kann während der Eingabe Fehler machen, welche auf verschiedene Ursachen zurückzuführen sind. Es muss gewährleistet sein, dass diese Fehleingaben mit minimalem Aufwand korrigiert werden können. Dabei darf es zu keinen fehlerhaften Systemzuständen kommen.

Tabelle 6 Grundsätze der Dialoggestaltung

(eigene Darstellung in Anlehnung an HERCZEG, S. 106 ff.)

[37] vgl. DEUTSCHES INSTITUT FÜR NORMUNG E.V., DIN 66234, Teil 8
[38] vgl. HERCZEG, S. 105 f.

HERCEG formuliert weiter die „Acht goldenen Regeln des Dialogdesigns“ in Anlehnung an SHNEIDERMAN[39], welche er wie folgt übersetzt und zusammenfasst:

1. Konsistenz erreichen: Die Dialoggestaltung innerhalb des Programms sollte einheitlich steuerbar sein. Für gleiche Aktionen sind gleiche Namen zu verwenden.
2. Abkürzungen anbieten: Durch Anbieten diverser Abkürzungen, beispielsweise durch Tastenkombinationen, erhält der Nutzer die Möglichkeit einen Dialog schnell abzuarbeiten.
3. Informatives Feedback geben: Wenn der Nutzer eine Aktion startet, sollte der Dialog entsprechend reagieren. Möchte der Nutzer zum Beispiel ein Kommando ausführen wollen, welches eine gewisse Zeit zur Bearbeitung benötigt, sollte der Dialog mit einem Ladebalken reagieren.
4. Dialoge abschließen: Dialogfolgen sollten Anfang, Mitte und Ende besitzen. Der Benutzer soll die Abfolge abschließen und sich dann seiner nächsten Aufgabe widmen können.
5. Einfache Fehlerbehandlung anbieten: Sollte der Nutzer eine Fehleingabe getätigt haben, darf das System nicht unerwartet reagieren oder abstürzen. Der Fehler sollte durch eine Fehlerbehandlung korrigierbar sein.
6. Rücksetzungsmöglichkeiten anbieten: Dem Benutzer sollte die Sicherheit gegeben werden, dass er jede Aktion wieder erfolgreich zurücknehmen kann.
7. Dialog benutzergesteuert unterstützen: Der Dialog sollte durch den Nutzer kontrollierbar sein, so dass er das Gefühl bekommt, diesen im Griff zu haben.
8. Kurzzeitgedächtnis entlasten: Der Dialog sollte durch den Anwender schnell erfassbar sein.[40]

Unter Zuhilfenahme der Dialoggestaltungsgrundsätze und Anwendung der „Acht goldenen Regeln des Dialogdesigns“ kann ein neuer, verbesserter Druckdialog entworfen werden.
Dabei soll der Druckdialog als modales Fenster erscheinen. Dies bleibt solang geöffnet, bis der Anwender eine Auswahl getroffen hat. Sobald der Nutzer eine Aktion startet, wird der Dialog ausgeblendet und entsprechend der Auswahl erscheint das nächste Fenster. Ein modales Fenster ist deshalb zu favorisieren, da es keine weiteren Interaktionen mit dem Programm zulässt. Damit wird verhindert, dass der Nutzer im Hintergrund keine weiteren Programmfunktionen aufrufen kann bis eine Auswahl im Dialog getroffen wurde.

[39] vgl. SHNEIDERMAN, S. 88 f.
[40] vgl. HERCZEG, S. 114

Im Folgenden wird anhand der Grundsätze und Überlegungen ein überarbeiteter Druckdialog entworfen.

3.4.2 Umsetzung eines Druckdialogs

3.4.2.1 Analyse des derzeitigen Druckdialogs

Für eine Neugestaltung des Druckdialogs bedarf es einer vorherigen Analyse des aktuellen. Ziel ist es, nur notwendige Änderungen vorzunehmen, um den Benutzer nicht zu irritieren, jedoch ihn mit mehr Funktionen und besserer Bedienbarkeit auszustatten. So sollen Positionen der Schaltflächen gleich bleiben und der Informationsgehalt des Dialogs nicht abnehmen. Um den Nutzer bei Fragen zu unterstützten, soll eine Hilfedatei aufrufbar sein, welche mit Druck auf „F1“ erscheinen soll. Der derzeitige Druckdialog sieht wie folgt aus:

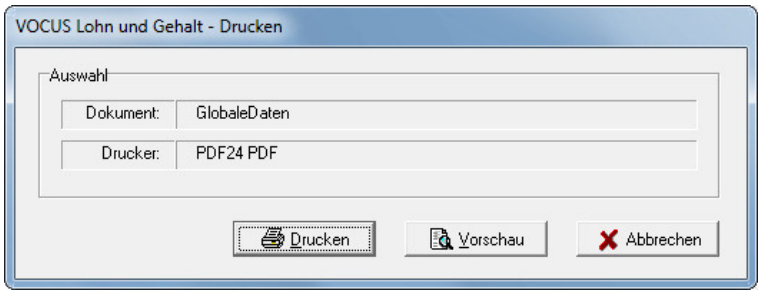


Abbildung 8 Bisheriger Druckdialog

Der Druckdialog zeigt, welches Dokument eben gedruckt wird und welcher Drucker gewählt ist. Verfügbare Interaktionen mit dem Benutzer sind die Auswahl von „Drucken“, um das Dokument auf dem angezeigten Drucker auszugeben, „Vorschau“ für die Voransicht des Reports und die Schaltfläche „Schließen“, um den Dialog ohne Ausgabe zu beenden. Wie zu erkennen ist, gibt es hier keine Möglichkeit, um den Drucker direkt zu ändern, erst beim Klicken auf „Drucken“ kann der Benutzer die Auswahl des Druckers vornehmen. Weiterhin bietet der Dialog keine Auswahl diverser Exportverfahren an, diese sind erst im Vorschaudialog wählbar.

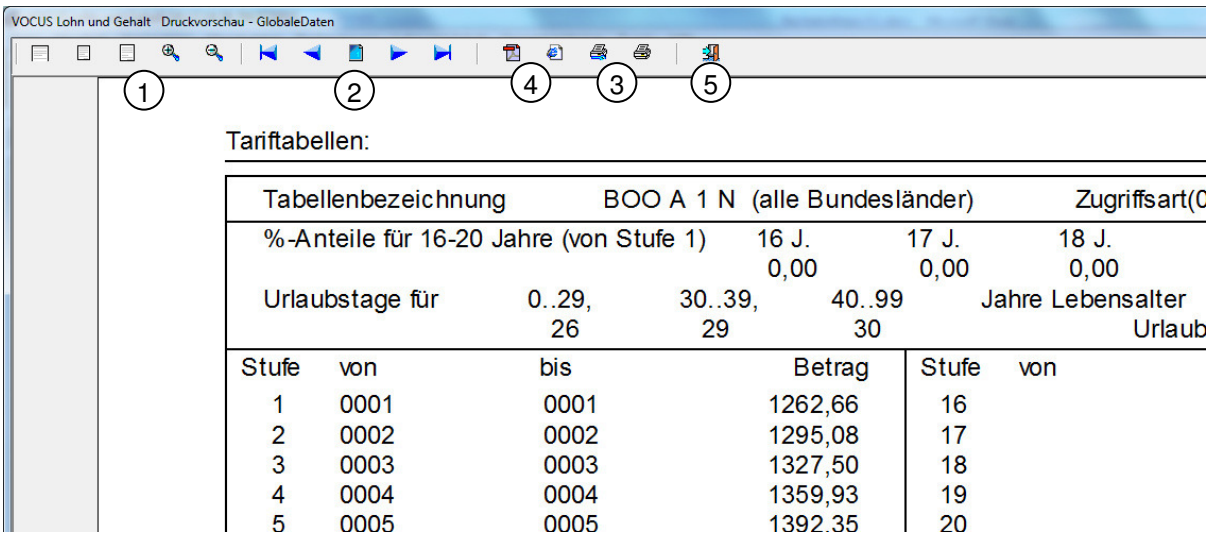


Abbildung 9 Bisherige Druckvorschau

Oben in der Symbolleiste finden sich Optionen, wie Zoomen (1), Seitennavigation (2), Druckerauswahl, Druck (3), Export in die Formate PDF und HTML (4) und das Beenden der Vorschau (5).
Durch die neue Druckschnittstelle mit dem Reporttool List & Label ist es möglich, weitere Optionen und Verarbeitungsmethoden im Druck- und Vorschaudialog anzubieten.

3.4.2.2 Entwurf eines neuen Druckdialogs

Durch Anwenden der Gestaltungsgrundsätze soll ein neuer Dialog entworfen werden. Die Aufgabe des Druckdialogs ist es, dem Nutzer Möglichkeiten anzubieten, ein Dokument auf dem Ausgabemedium seiner Wahl auszugeben. Dabei soll er, wenn nötig, in der Lage sein, die jeweiligen Ausgabeeinstellungen anzupassen. Der Dialog wird durch geeignete Beschreibungen und bekannte Icons selbsterklärend. Zur ausführlichen Erläuterung ist eine Hilfe eingebunden. Dabei ist der Dialog benutzergesteuert, d.h. der Benutzer hat zu jeder Zeit die Kontrolle und das System reagiert auf seine Eingaben. Die Dialogsteuerung orientiert sich an der bisherigen Vorgehensweise. Grundlegende Abläufe bleiben gleich, jedoch erhält der Benutzer nach Wahl eines Ausgabemediums die Möglichkeit einer weiteren Ausgabe auf ein anderes Medium, in dem der Dialog im Hintergrund geöffnet bleibt. Dabei darf der Nutzer erwarten, dass bei der Auswahl von Druck oder Vorschau die entsprechenden Aktionen starten und bei der Wahl von Export diverse Exportmöglichkeiten zur Verfügung stehen. Sollte der Nutzer eine Fehleingabe tätigen, wie zum Beispiel die Auswahl einer negativen Anzahl von Kopien, korrigiert das System die Fehleingabe.
Ausgehend von den „Acht goldenen Regeln des Dialogdesigns“ wurden diese wie folgt angewandt:

1. Konsistenz erreichen: Der Druckdialog ist in allen Stellen des Programms einheitlich. Es wurde sich an der Darstellung des bisherigen Druckdialogs orientiert.
2. Abkürzungen anbieten: Der Nutzer kann mithilfe der „Alt“-Taste und mit dem unterstrichenen Buchstaben der Schaltfläche die jeweilige Funktion schnell aufrufen.
3. Informatives Feedback geben: Der Dialog reagiert entsprechend der gewählten Funktion. Je nach Auswahl wird ein Ladefenster, der Vorschau- oder Entwurfsdialog angezeigt bzw. der Dialog geschlossen.
4. Dialoge abschließen: Eine Druckdialogfolge wird mit dem Schließen des Druckdialogs beendet. Bei Bedarf kann der Anwender mehrere Ausgabeaktionen starten.
5. Einfache Fehlerbehandlung anbieten: Der Dialog korrigiert Fehleingaben automatisch, zum Beispiel eine negative Anzahl an Kopien wird nicht zugelassen.
6. Rücksetzungsmöglichkeiten anbieten: Ein gestarteter Vorgang kann ggf. abgebrochen werden.
7. Dialog benutzergesteuert unterstützen: Durch einfache Auswahlmöglichkeiten wie Drucker und deren Einstellungen erhält der Benutzer die Kontrolle über den Dialog.
8. Kurzeitgedächtnis entlasten: Alle wichtigen Informationen sind auf einen Blick erfassbar wie das zu druckende Dokument, Zieldrucker und Anzahl der Kopien.

Aus dem Anwenden der Grundsätze wurde folgendes Dialogdesign gewählt:

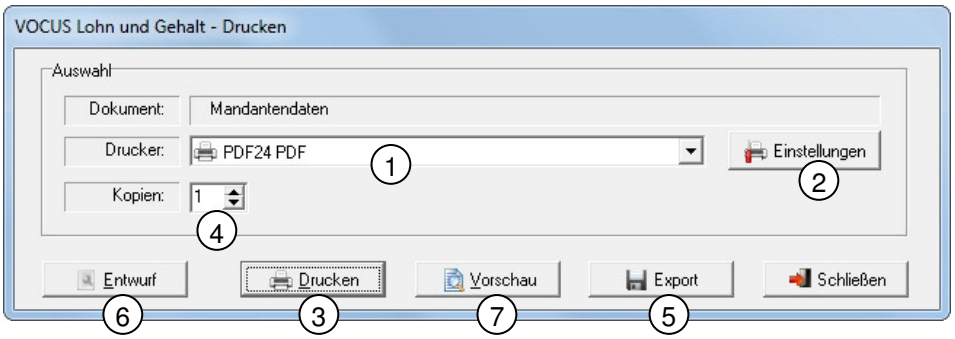


Abbildung 10 Neues Druckdialogdesign

Die Auswahl des Druckers kann jetzt direkt im Dialog erfolgen (1). Sollte der Nutzer Druckereinstellungen ändern wollen, kann er dies nun über die Schaltfläche „Einstellungen“ (2) bewerkstelligen. Das hat zur Folge, dass beim Klicken auf „Drucken“ (3) keine weitere Abfrage nach dem Zieldrucker mehr erfolgen muss, sondern das Do-

kument direkt an den gewählten Drucker gesandt wird. Weiterhin lässt sich jetzt direkt im Dialog die Anzahl der Kopien festlegen (4).
Neu im Dialog ist die Exportoption (5), welche es ermöglicht, dass Dokument in diverse Dateiformate zu exportieren. Dabei öffnet sich ein Kontextmenü, in welchem das Zielformat zu bestimmen ist. Anhand der Lizenzinformationen werden nur die freigeschalteten Exportarten angezeigt.
Für die Entwickler der Vocus Lohn- und Gehaltssoftware lässt sich über die Tastenkombination „Strg + Alt + P“ die Schaltfläche „Entwurf“ (6) einblenden. Für den normalen Benutzer ist diese Schaltfläche per Tageskennwort[41] gesperrt. Sie ermöglicht den Aufruf des List & Label Designers, um Reports zu bearbeiten. So erhalten die Entwickler eine einfache Option um ihre Layouts ohne größeren Programmieraufwand zu bearbeiten. Trifft der Benutzer die Auswahl „Vorschau“ (7), öffnet sich der List & Label Vorschaudialog, der deutlich mehr Funktionalität bietet als der bisherige:

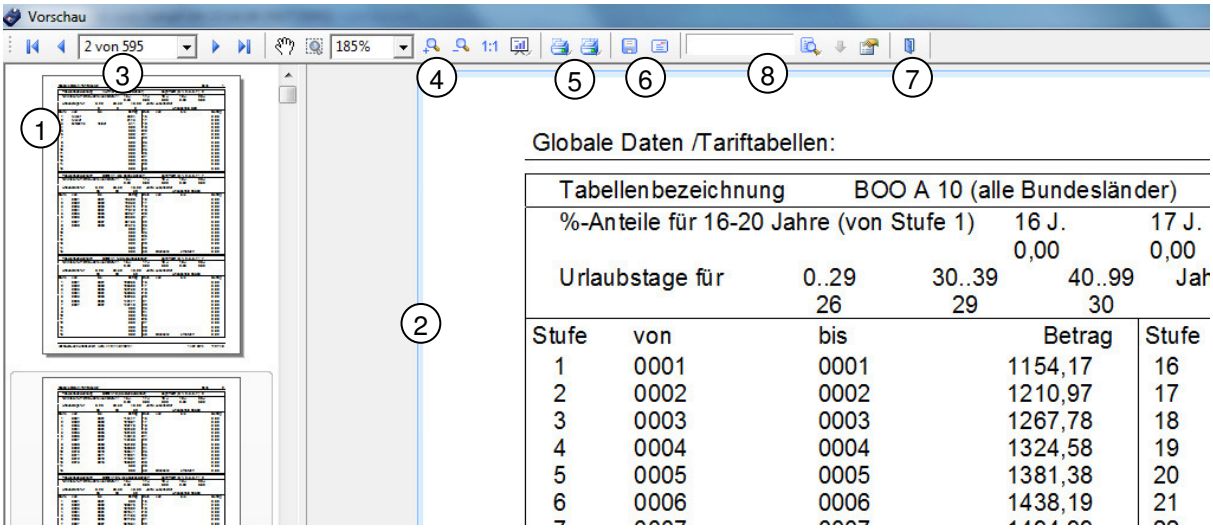


Abbildung 11 List & Label Druckvorschau

Die Druckvorschau wird bereits nach dem Vorbereiten der ersten Seite angezeigt. Alle folgenden Seiten werden im Nachhinein erzeugt. Das beschleunigt die Anzeige des Dialogs und der Nutzer kann bereits die ersten Seiten betrachten. Weiterhin bietet der neue Dialog die Möglichkeit alle Seiten in Miniaturansicht auf der linken Seite (1) darzustellen, um so einfacher im Dokument navigieren zu können. In der Hauptansicht (2) ist es jetzt möglich mittels der Kombination von „Strg + Mausrad“ stufenlos zu zoomen. Die Symbolleiste bietet, ähnlich wie im alten Vorschaudialog, die Möglichkeit durch die Seiten zu navigieren (3), zu zoomen (4), einen Druck zu starten (5), das Dokument zu exportieren und per E-Mail zu versenden[42] (6), oder die Vorschau zu beenden (7). Neu ist hier die Funktion, das Dokument nach Stichwörtern zu

[41] Anm.: Das Tageskennwort für die Software ändert sich täglich. Dies ermöglicht es bestimmte, gesperrte Aktionen freizuschalten und ist dem Anwender i.d.R. nicht zugänglich.
[42] Hierfür muss ein E-Mail Client, wie Microsoft Outlook, konfiguriert sein.

durchsuchen (8). Das bietet vor allem bei großen Auswertungen, auf der Suche nach einem bestimmten Wert, enorme Vorteile.
Für die Entwickler des Vocus Lohn- und Gehaltsprogramms steht im Druckdialog die Option „Entwurf“ bereit, um Reports zu entwerfen.

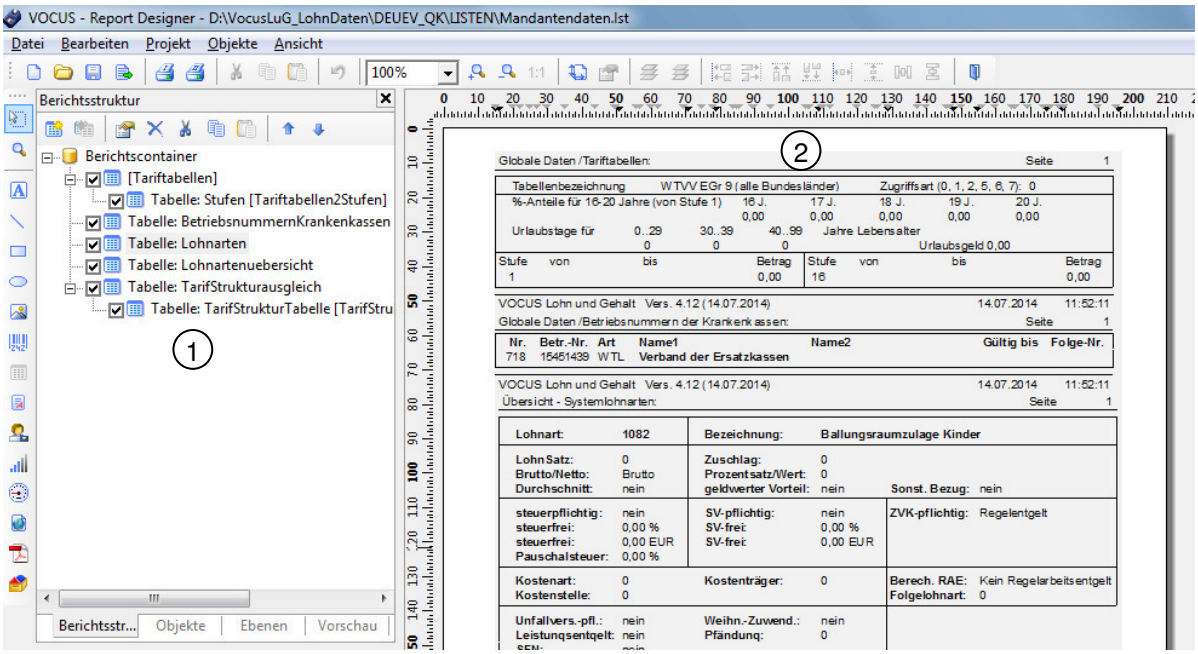


Abbildung 12 List & Label Designer

Im linken Bereich stehen die erkannten Tabellen in Master-Detail Form bereit (1). Der Designer ermöglicht diese Tabellen per „Drag-and-Drop“ auf dem Report (2) zu platzieren. Für jede (Unter-) Tabelle ist das Verhalten der Kopf-, Daten- und Fußbereiche bestimmbar. Der Programmierer kann damit beispielsweise festlegen, dass vor Ausgabe einer Tabelle oder einem bestimmten Datensatz ein Seitenumbruch erfolgen muss. Die Möglichkeiten einen Report mit dem Designer zu Gestalten sind sehr vielfältig, so dass sie den Rahmen dieser Arbeit übersteigen würden.

3.5 Datenverarbeitung

3.5.1 Interne Datenverarbeitung

Die interne Datenverarbeitung stützt sich vor allem auf die DataList eines Reports. In ihr sind alle betreffenden Daten zur Auswertung zusammengestellt. Das DataList-Objekt bietet einen effizienten Zugriff auf ihre beinhaltenden Daten und ist variabel einsetzbar. Der Programmierer kann in den einzelnen Data-Objekten bestimmen welchen Datentyp die gespeicherten Werte haben. Dies macht sie flexibel einsetzbar und stellt die Grundlage für die Datenhaltung der Reports. Zur Verwendung der DataList in List & Label Reports bedurfte es zur Verwendung der Master-Detail Beziehungen geringfügiger Anpassungen.

Um die Beziehungen in ihrer Struktur abzubilden, wurden die in ihr enthaltenen Data-Objekte um die Eigenschaft „Parent“ erweitert. Die Eigenschaft wird beim Erstellen eines Data-Objektes in einer DataList standardmäßig mit dem Pointer auf die DataList initialisiert. D.h. der hinzugefügte Datensatz ist ein Masterdatensatz. Für die Verwendung von Detaildatensätzen muss die Parent-Eigenschaft des Data-Objektes auf das übergeordnete Master-Data-Objekt zeigen.
Die Data-Objekte können mehrere, verschiedene Datentypen aufnehmen. Diese kann der Report im Entwurf berücksichtigen. Per Code ist es damit möglich dem Data-Objekt beispielsweise ein Integer-Feld zu übergeben, welches dann der Report entsprechend seiner Einstellung formatiert. Weiterhin kann der Report dann Berechnungen, wie zum Beispiel das Bilden einer Summe, durchführen.
Die DataList ist ein einfaches und wirkungsvolles Tool, wenn es um die Datenhaltung und Weitergabe im Vocus Lohn- und Gehaltsprogramm geht.
Für die erfolgreiche Verarbeitung der Daten durch List & Label und um eine gute Performance der Druckschnittstelle zu erzielen, ist es nötig die Daten in eine lokale Hauptspeicherdatenbank (ClientDataSets) zu laden. Jedes DataSet-Objekt repräsentiert eine Tabelle, die Beziehungen zu anderen Tabellen aufbauen kann. Anhand der übergebenen Struktur an die DataList ist es möglich die Beziehungen unter den Tabellen aufzubauen.
In einem ersten Schritt der Verarbeitung analysiert eine Prozedur die Struktur eines Data-Objektes. Anschließend wird die Struktur mit den anderen Datensätzen der DataList verglichen. Daten, die zu einer Tabelle gehören, müssen die gleiche Feldstruktur aufweisen. Dann werden diese in ihre zugehörige Tabelle geschrieben und anhand der Parent-Eigenschaft wird ermittelt, welche Tabellen miteinander zu verknüpfen sind. Der Report erhält die erkannten Tabellen und Relationen und die Druckroutine liest diese danach aus. Beim Auslesen der Daten werden untereinander verknüpfte Datensätze beachtet. Wenn beim Druck einer Mastertabelle ein Datensatz auf einen oder mehrere Detailsätze zeigen sollte, wird zu der Detailtabelle gewechselt und die Datensätze angedruckt.
Dem Report wird beim Aufruf der Druckschnittstelle das zu verwendende Layout aus dem Listenverzeichnis mitgegeben. Dabei findet eine Überprüfung statt, ob die Liste zu der übergebenen Datenmenge passt. Die Namen der Tabellen und ihre Beziehungen sind in der Layoutdatei hinterlegt, damit es zu keinen Verwechslungen kommen kann. Sollte eine Liste nicht mit den übergebenen Daten zusammenpassen, wird eine Fehlermeldung durch List & Label ausgegeben.
Um die Daten mit ihrem Layout zu archivieren, bedarf es einer konsistenten Lösung, die beide Elemente zusammen in einer Datei ablegt.

3.5.2 Speicherung der Reports im internen Archiv

Um eventuelle Änderungen rückverfolgen zu können, müssen alle Reports, die im Programm gedruckt werden, im Archiv ablegbar sein. Analog dazu müssen sich alle erzeugten Dokumente wieder öffnen lassen. Dies betrifft auch alle Dokumente, welche mit der bisherigen Druckschnittstelle gedruckt wurden. Diese Reports sind zunächst auf ihren Inhalt und ihr Layout zu analysieren. Aus den ermittelten Informationen heraus entscheidet sich, ob diese umgewandelt und mit der neuen Schnittstelle gedruckt werden können. Ist dies nicht möglich wird die derzeitige Druckschnittstelle verwendet um Darstellungsfehler im Report zu vermeiden.

Für die Speicherung der Reports über die neue Druckschnittstelle gilt es eine Routine zu entwickeln, welche die Layoutdatei und die Daten der DataList extrahiert und im Archiv ablegt. Die DataList bietet eine Methode an, all ihre Daten als XML-Datei zu speichern. Damit die DataList ihre hierarchischen Strukturen jetzt auch in eine XML-Datei exportieren kann, bedurfte es einiger Anpassungen der zugehörigen Speicher- und Lademethode. Die Anpassungen für eine strukturierte XML-Verarbeitung der DataList anhand der „Parent“-Eigenschaft wurden in die Klasse einprogrammiert. Damit erhalten alle Programmierer der Vocus GmbH Zugriff auf die neuen Funktionen. Der Quellcode zum Ein- und Auslesen von DataLists mit Hierarchien ist im Anhang 9 und 10 zu finden. Die Archivdatei wurde aus drei Teilen konstruiert:

1. Versionsinformationen, mit denen die Datei erstellt wurde
2. Layoutdatei von List & Label, um den Report darzustellen
3. XML-Daten der DataList

Um die Archivdatei so zu konstruieren, dass sie auch bei Änderungen im Programmcode (zum Beispiel neue List & Label Version, Änderungen im Aufbau der DataList) noch geladen werden kann, wurde ihr ein Header hinzugefügt. In diesem befinden sich die Versionsinformationen von List & Label, die Version der DataList, der Listenname und die Standarddateiendungen. Gefolgt vom Header wird das Layout aus dem aktiven Report und die exportierten XML-Daten der DataList hinzugefügt.
Der Algorithmus für das Erstellen einer Archivdatei aus einem geladenen Report ist wie folgt beschrieben:
Erzeuge für jede Sektion eine StringList und speichere die DataList als temporäre XML-Datei ab. Erstelle für den Report einen Header, lade die Textdatei mit den Layoutinformationen aus der List & Label Layoutdatei und die XML-Daten aus der temporären Datei. Füge alle drei Sektionen zusammen, so dass die endgültige Datei die oben genannte interne Struktur besitzt. Das PAP für diesen Algorithmus ist unter Anhang 11 und der Quelltext unter Anhang 12 beigelegt.

Übergibt der Programmierer im Aufruf der Druckschnittstelle einen gültigen Archivierungsparameter, wird die Archivdatei in der Feldvariable „FPrintList“ der Klasse „TL18_Report“ gespeichert und anschließend mit der Klassenmethode „SaveArchivFile“ der Klasse „TPrintArchiv“ in der Archivdatei abgelegt. Diese Datei wird in jedem Monatsverzeichnis pro Mandant angelegt und enthält alle zugehörigen archivierten Dokumente.
Die folgende Abbildung zeigt das Archivmodul von Vocus Lohn und Gehalt mit der Implementierung der neuen Druckschnittstelle mit dem überarbeiteten Druckdialog.

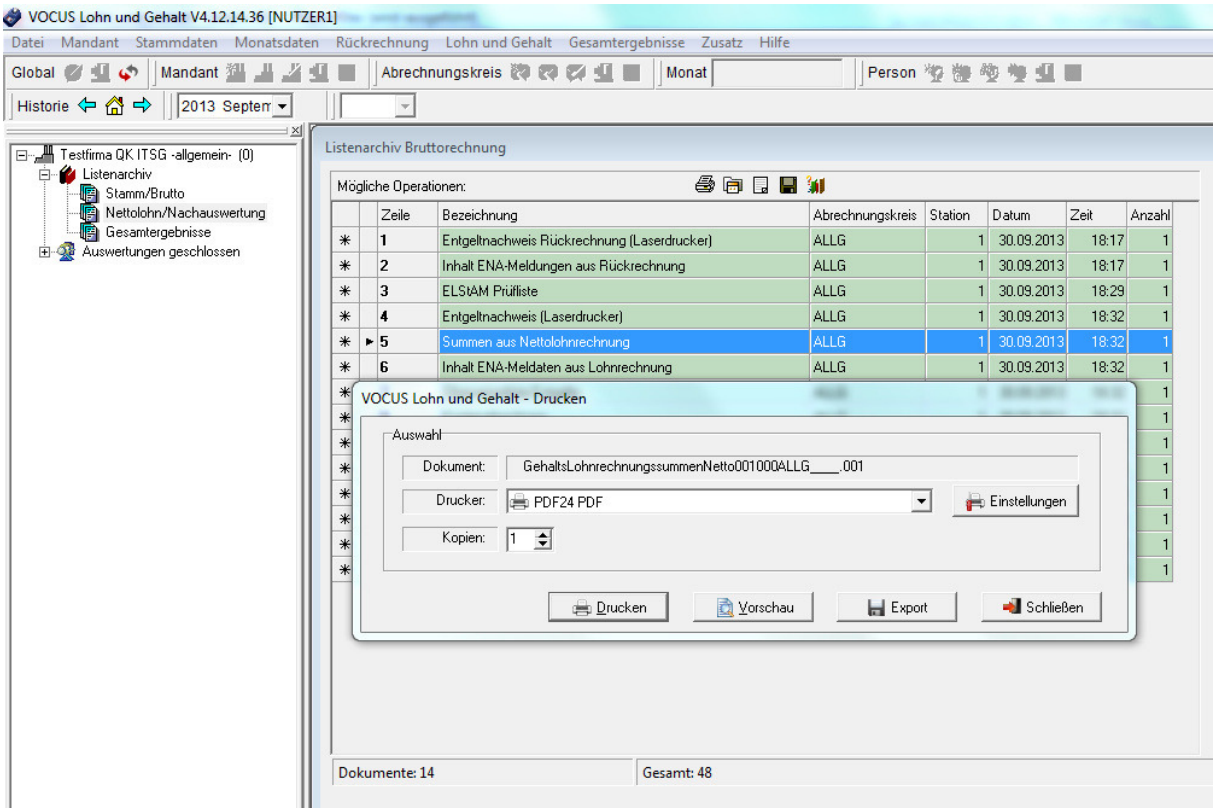


Abbildung 13 Archivmodul

Um die verschiedenen Listenarten im Archiv voneinander zu unterscheiden, bedurfte es einiger Anpassungen an die Ladefunktion von Reports aus dem Archiv. Zur Überprüfung, ob es sich um eine Datei handelt, die mit der neuen Druckschnittstelle erstellt wurde, wird die erste Zeile des gewählten Reports analysiert. Enthält diese die vom Header implementierte Zeile „[ListLabelArchivFile]“, so wird die Datei wieder in ihre ursprünglichen Bestandteile aufgeteilt und der neuen Druckschnittstelle übergeben. Die Dokumentation des Quellcodes der Prozedur „PrintArchivFile“ erfolgt unter Anhang 13.

Die Routine parst die Inhalte des Reports, so dass die XML-Laderoutine sie wieder in die DataList einlesen kann. Die Layoutsektion wird extrahiert und als temporäre

Datei gespeichert, um sie beim Druckaufruf wieder laden zu können. Um die Archivdatei auf Konsistenz und Kompatibilität mit der aktuellen Programmversion zu prüfen, wird ein „ArchivFileVersionCheck“ durchgeführt, welcher hauptsächlich die Versionsinformationen von List & Label und der DataList überprüft. Sollten Änderungen aufgetreten sein, wird die Datei ggf. angepasst und neu abgelegt.
Somit ist gewährleistet, dass die Archivierung über mehrere Versionen hin funktioniert und immer kompatibel ist. Hiermit ist die Anforderung an eine funktionierende Archivierung über die neue Druckschnittstelle erfüllt.

3.5.3 Datensicherheit und Benutzerkontrolle für das Archivsystem

3.5.3.1 Grundlagen

Datensicherheit und Datenschutz (hier durch eine Benutzerkontrolle realisiert) bedarf einer näheren Betrachtung in der Software Vocus Lohn und Gehalt, da besonders im Lohnbereich der Datenschutz eine wichtige Rolle spielt. Genauer soll hierbei das Archivsystem betrachtet werden, welches alle erzeugten Dokumente zur Nachverfolgung ablegt. Zur Anwendung der Datensicherheit und des Datenschutzes auf das Programm müssen zuvor die jeweiligen Definitionen betrachtet werden.

Die Definition der Datensicherheit nach WITT lautet:
„Schutz der gespeicherten Daten vor Beeinträchtigung durch höhere Gewalt, menschliche oder technische Fehler und Missbrauch.“[43]

Da im Archivsystem des Vocus Lohn und Gehalt personenbezogene Daten gespeichert werden, muss auch die Definition des Datenschutzes erläutert werden:
„Schutz des Einzelnen vor Beeinträchtigung seines Persönlichkeitsrechts bei Umgang mit seinen personenbezogenen Daten.“[44]

Das Lohn- und Gehaltsprogramm der Firma Vocus arbeitet mit streng vertraulichen Lohndaten, welche unter besonderem Schutz stehen müssen. Jeder Kunde hat selbstständig dafür zu sorgen, dass seine Datenbestände vor fremden Zugriffen geschützt sind. Auf das umsichtige Verhalten der Kunden hat die Firma Vocus keinen Einfluss. In den Zuständigkeitsbereich der jeweiligen Kunden fällt der Punkt Schutz der Daten „vor Beeinträchtigung durch höhere Gewalt, menschliche oder technische Fehler und Missbrauch“[45].
Jedoch muss vom Programm her gewährleistet sein, dass nur befugte Benutzer Einsicht in bestimmte Daten erhalten.[46] Dies bedarf einer Rechtesteuerung und Benut-

[43] WITT, S. 3
[44] WITT, S. 4
[45] WITT, S. 3
[46] vgl. WITT, S. 24

zerkontrolle im Programm. In dem Softwaresystem werden alle Änderungen der Nutzer an Mandanten und Personen aufgezeichnet und protokolliert. Für die Verwaltung der Nutzer gibt es programmintern eine Benutzersteuerung zum Anlegen und Löschen von Benutzern. Diese gestattet es, nur bestimmte Personen für den Zugriff auf bestimmte Mandanten freizuschalten. Diese Funktion ist mit einem Administratorenpasswort geschützt, so dass diese Änderungen nur von vertraulich erklärten Personen durchgeführt werden dürfen. Die Benutzersteuerung ist in folgende Hierarchiestufen aufgegliedert:

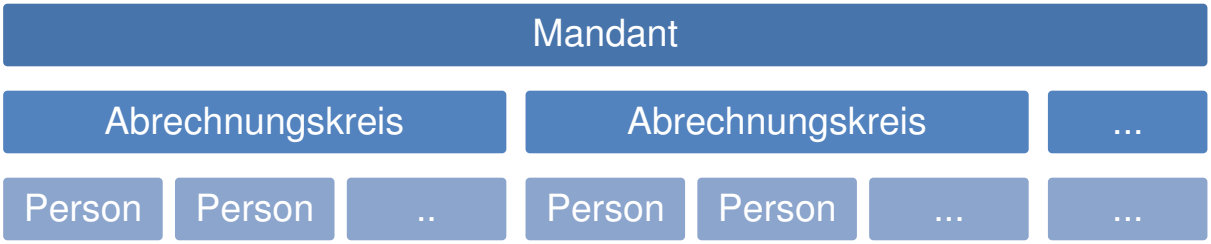


Abbildung 14 Hierarchie der Benutzersteuerung

Für einen Benutzer werden zuerst die Berechtigungen für die jeweiligen Mandanten, die er ansehen und bearbeiten darf, vergeben. Anschließend erhält er für jeden Mandanten den Zugriff über deren Abrechnungskreise. Dabei ist zu beachten, dass der Zugriff auf einen Mandanten keinen Zugriff auf alle seine Abrechnungskreise impliziert, dies bedarf einer expliziten Rechtevergabe. Im Gegensatz dazu werden bei Vergabe einer Berechtigung an einen Abrechnungskreis diese automatisch an alle Personen des Abrechnungskreises vererbt. Die Benutzersteuerung findet über das folgende abgebildete Dialogfenster statt:

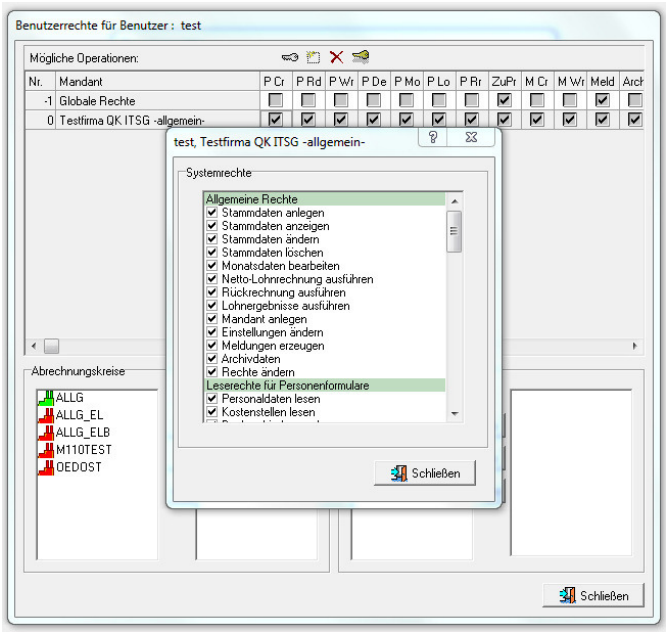


Abbildung 15 Benutzersteuerung im Vocus Lohn und Gehalt

In diesem Dialog lassen sich die gewünschten Rechte der jeweiligen Benutzer administrieren. So können für Mandanten und Abrechnungskreise dem Nutzer Lese- und / oder Schreibrechte gegeben werden.
Um dem Datenmissbrauch außerhalb des Programmes vorzubeugen, werden alle personenrelevanten Daten innerhalb des Lohnamtsverzeichnisses verschlüsselt abgelegt. Somit ist es nicht möglich, die Lohndaten ohne das Lohn- und Gehaltsprogramm einzusehen oder gar zu ändern. Für die Kontrolle und Verwaltung der Datensicherheit außerhalb des Programmes sind die Kunden selbst verantwortlich.

3.5.3.2 Konzept für eine Benutzerkontrolle des Archivsystems

Die Problematik des Datenschutzes und daraus resultierend einer Benutzerkontrolle soll nun auf das bestehende Archivsystem übertragen und bewertet werden. Nur so ist festzustellen, ob evtl. Änderungen und Verbesserungen vorzunehmen sind. Dabei erfolgt die Realisierung der Benutzersteuerung für das Archiv unabhängig von der neu implementierten Druckschnittstelle.
Um eine Benutzersteuerung für das Archiv zu konzipieren, wird ein Soll- / Ist-Vergleich vorgenommen. Folgende Soll-Funktionalität wurde durch das Pflichtenheft (Anhang 1) für die Archivberechtigungssteuerung formuliert:

- Unterstützung einer Berechtigungssteuerung
- Hierarchisierung nach Mandant, Abrechnungskreis, Person, Dokument

Der Ist-Stand wurde durch den folgenden Test ermittelt:
Es wurde ein neuer Benutzer angelegt, der Zugriff auf einen Mandanten und dessen Archivdokumente bekam. Des Weiteren erhielt der Benutzer keine Rechte für die vom Mandanten zugeordneten Abrechnungskreise. Daraufhin wurde folgendes Ergebnis erwartet: Der Benutzer sollte Zugriff auf den Mandanten haben und sein Archiv öffnen können, jedoch sollten in diesem keine Dokumente zu sehen sein.

Folgendes Ergebnis wurde formuliert:
Trotz der vergebenen Berechtigungen war es möglich, alle Dokumente des Mandanten und deren Abrechnungskreise einzusehen.

Schlussfolgerung:
Die Berechtigungssteuerung wurde zwar im Programm und in die Lohnabrechnung implementiert, jedoch nicht bis zum Archiv konsistent weitergegeben. So ist es für alle Benutzer, die Zugriff zu dem Mandanten haben, möglich, die personenbezogenen Daten der nicht für den Benutzer freigeschalteten Abrechnungskreise einzusehen.

Für die Erreichung des Sollzustandes muss eine konsistente Benutzersteuerung für das Archiv entworfen werden, welche sich in der globalen Benutzeradministration regeln lässt.
Die Implementierung sollte direkt an der Schnittstelle zwischen Benutzer und Archiv stattfinden. Hierbei muss ermittelt werden, zu welchen Abrechnungskreisen des Mandanten der Benutzer Zugriff hat und welche Dokumente dies betrifft. Die Realisierung einer Filterung der Dokumente nach Personen ist dahingehend nicht sinnvoll, weil der Nutzer Zugriff auf alle Personen eines Abrechnungskreises besitzen muss um eine Lohnabrechnung durchzuführen. Das bedeutet, dass er mindestens Leserechte auf Personen erhält.
Das Archiv des Vocus Lohn- und Gehaltsprogrammes teilt sich in drei Bereiche auf: Stamm/Brutto, Nettolohn/Nachauswertung und Gesamtergebnisse. In den ersten beiden Bereichen werden die Dokumente abrechnungskreisweise abgelegt, d.h. es sind personenbezogene Daten enthalten. In den Gesamtergebnissen werden die Ergebnisse aller Abrechnungskreise des Mandanten gespeichert. Diese enthalten i.d.R. keine personenbezogenen Informationen. So kann das Verhalten der zukünftigen Benutzersteuerung bestimmt werden. In den Archivbereichen Stamm/Brutto und Nettolohn/Nachauswertung sollen nur die Dokumente der freigeschalteten Abrechnungskreise und Personen zu sehen sein, wofür der Nutzer die Rechte besitzt. Die Gesamtergebnisse dürfen komplett eingesehen werden, sobald der Benutzer Zugriff auf einen Abrechnungskreis des Mandanten erhält.

Für die Realisierung der Datensicherheit und Benutzerkontrolle im Archiv des Vocus Lohn- und Gehaltsprogrammes gibt es ein einheitliches Interface. Dieses steuert die Ablage eines zu archivierenden Dokumentes. Diesem werden die Daten als verschlüsselte StringList übergeben und die Schnittstelle zum Archiv entscheidet dann, wohin die Daten abgelegt werden müssen. Das Dokument wird anschließend durch die StringList verschlüsselt in einer großen Archivdatei abgelegt. In einer Indexdatei werden die beinhaltenden Dokumente verwaltet. Die Indexdatei speichert alle nötigen Informationen zu einem Dokument wie Erstellungsdatum, Mandanten-Nummer, Benutzer-Nummer und Version. In der Reportklasse „TL18_Report“ werden diese Informationen bereits mit abgelegt, so dass sie nur noch eine Übergabe an die Archivschnittstelle erfolgen muss.
Somit bedarf es einer Erweiterung der Benutzersteuerung, damit im Archiv liegende Dokumente nicht von Benutzern ohne Zugriffsrechte eingesehen werden können.

4 Test (Validierung)

4.1 Testansteuerung der Schnittstelle

Zum Abschluss der Integration der neuen Druckschnittstelle ist diese noch zu validieren. Hierfür müssen Tests durchgeführt werden, um die vom Pflichtenheft vorgeschriebenen Funktionalitäten zu überprüfen. Die wichtigste Funktion einer Druckschnittstelle ist der Druck und alle Aufgaben die damit inhärent sind wie Vorschau und Druckerkonfiguration. Für den Test wurde eine einfache und effiziente Methode gewählt, um alle implementierten Komponenten zu überprüfen. Dazu wurde eine schon im Lohnprogramm existierende Liste gewählt und diese mittels der neuen Druckschnittstelle ausgegeben. Das Ergebnis des Druckes wird schließlich mit dem derzeitigen Dokument verglichen. Anhand des Tests wird weiterhin veranschaulicht wie die neue Datenzusammenführung in einer DataList im Vergleich zu einer StringList funktioniert.

Ein Beispiel, wie der Quellcode für den Druck mit einer DataList angepasst wurde, ist unter Anhang 14 zu sehen. Zum direkten Vergleich wurde die Routine der ursprünglichen Datenaufbereitung mit beigefügt (Anhang 15). Alle Felder, die sonst in einen String geschrieben wurden, werden nun in ein Feld eines Data-Objektes abgelegt. Dabei entfallen auch die Formatierungen mittels Quellcode. Durch die Übergabe des Datentypen im jeweiligen Feld ist es nun möglich, dass dies der Report übernimmt. Der Aufruf der Druckfunktion erfolgt analog zu der bisherigen.

Die Schnittstelle erkennt beim ersten Aufruf der Liste, dass unter diesem Dateinamen noch keine Listenvorlage im Listenverzeichnis existiert. Der Entwickler hat nun im Druckdialog die Möglichkeit, eine neue Liste zu entwerfen. Ist die Liste erstellt, kann sie im Entwurfsmodus in der Echtdatenvorschau mit den übergebenen Daten angesehen werden.

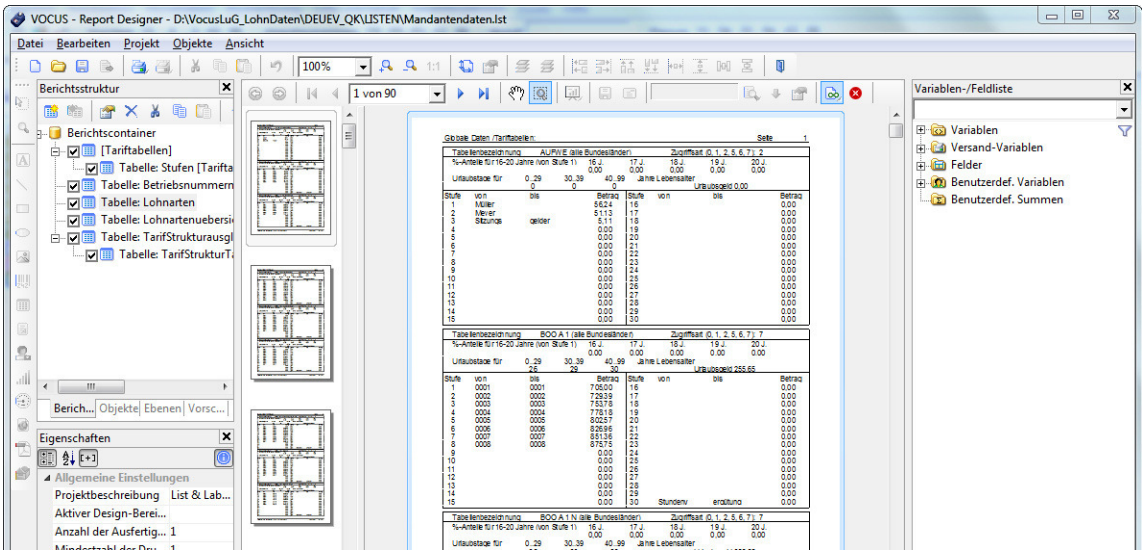


Abbildung 16 Echtdatenvorschau im Designer

Die Liste wird im Listenverzeichnis abgelegt und dem Kunden per Update eingespielt. Sie liegt nun bereit, um ihr Daten zu übergeben und über die Druckroutine ausgegeben zu werden. Ein vollständiger Test der essentiellen Druckkomponenten wurde durchgeführt und protokolliert. Das Protokoll ist unter Anhang 16 zu finden. Zum Abschluss der Tests wird ein Vergleich der Druckschnittstelle mit QuickReport und der von List & Label durchgeführt.

4.2 Vergleich mit der vorherigen Druckerschnittstelle

Damit die neue Druckschnittstelle sich im Lohnprogramm etablieren kann und dem Nutzer Vorteile bringt, ist es wichtig die einzelnen Komponenten miteinander zu vergleichen. Dabei spielen hauptsächlich die Anforderungspunkte des Pflichtenheftes, aber auch der Aspekte aus der Entwicklung der Schnittstelle eine Rolle. Die folgende Tabelle beinhaltet in der linken Spalte die Bewertungskriterien für den Vergleich. Mit den Tabellenüberschriften „QuickReport“ und „List & Label“ sind nicht nur die Komponenten an sich gemeint, sondern auch ihre implementierten Druckschnittstellen.

Kriterium	QuickReport	List & Label
Druckaufbereitung	Die Druckaufbereitung erfolgt anhand von Textdateien und StringLists. Der Druck bzw. die Vorschau startet erst, wenn das Dokument vollständig aufbereitet wurde	Die Druckdaten werden in einer DataList geliefert. Diese wird ausgewertet und in Tabellenstrukturen überführt. Ist die Überführung abgeschlossen, kann List & Label den Druck oder die Vorschau bereits starten, sobald die ersten Seiten aufbereitet wurden.
Druckdialog	Der implementierte Druckdialog bietet grundlegende Druckfunktionen. Die Druckereinstellungen müssen für jedes Dokument bestätigt werden.	Der neue Druckdialog besitzt ebenfalls alle grundlegenden Druckfunktionalitäten. Ergänzt werden diese durch Exportoptionen in alle gängigen Dateiformate. Die Druckeinstellungen können direkt bearbeitet werden und werden für jedes Dokument spezifisch gespeichert. Für Entwickler besteht die versteckte Option zur Bearbeitung des Layouts.

Kriterium	QuickReport	List & Label
Vorschaudialog	Der QuickReport Vorschaudialog ermöglicht das Betrachten einzelner Seiten und Exportieren in diverse Dateiformate. Grundfunktionen einer Vorschau wie Drucken, Zoomen und Blättern sind vorhanden.	Die Druckvorschau ist in zwei Bereiche unterteilt: Miniaturvorschau und Hauptbereich. In der Toolbar stehen alle gewohnten Funktionen zur Verfügung. Diese werden durch eine Suchfunktion und direkter E-Mail unterstützt. Es ist nun auch möglich, stufenlos eine Seite zu zoomen.
Erstellung neuer Listen	Das Erstellen neuer Listen erfolgt durch Bearbeiten der Definitionsdateien. Kenntnisse über deren Aufbau sind zur Erstellung und Bearbeitung notwendig.	Wird mit dem durch List & Label mitgelieferten Designer-Tool realisiert. Über den Aufbau des zugrundeliegenden Dateiformats bedarf es keiner Kenntnis.
Handhabung der Daten zur Übergabe an die Schnittstelle	Die Felddefinitionen müssen in der richtigen Reihenfolge angeordnet werden. Formatierungen sind im Quelltext vorzunehmen. Änderungen am Aussehen einer Liste bedürfen fast immer einer Änderung des Quelltextes.	Die Daten werden in einer DataList gehalten. Formatierungen der Werte müssen nicht explizit vorgenommen werden, dies kann im Designer bestimmt werden. Müssen Änderungen am Layout vorgenommen werden, bedingt dies nicht zwangsläufig eine Änderung des Quellcode.
Archivierung	Felddefinitionen werden zusammen mit den gedruckten Daten abgelegt. Die Daten lassen sich, wenn es nötig wird, nicht mehr per Programmcode rekonstruieren und einlesen.	Das Archivformat wurde in drei Sektionen aufgeteilt. Im Header werden Versionsinformationen gespeichert, darauffolgend das Layout der List & Label Datei und als dritten Teil alle Daten im XML-Format. Diese können bei Bedarf wieder in eine DataList eingelesen und vom Programm verarbeitet werden.
Ressourcenverbrauch[47] **Prozessor:** **RAM:**	 ø 25 % 170.750 KB	 ø 28 % 218.500 KB

Tabelle 7 Vergleich der Integration von QuickReport mit List & Label

[47] Erzeugen einer Vorschau auf einem Rechner mit folgenden Spezifikationen: 8 GB RAM, Intel i5 Prozessor; Testdokument Seitenumfang: ca. 700 Seiten; Angaben incl. Ressourcenverbrauch des Vocus Lohn- und Gehaltsprogrammes

Durch den Vergleich lässt sich Folgendes formulieren:
Die neue Druckschnittstelle mit ihrer implementierten List & Label-Komponente bietet deutlich mehr Funktionalität als bisher. Sie ist leichter zu handhaben, jedoch ist der Aufbau der DataList komplexer gegenüber der StringList. Änderungen im Layout der Reports sind nun einfacher durchführbar und bedürfen nicht in jedem Fall einer Änderung des Quellcodes. Durch die neue Druckaufbereitung werden Dokumente schneller angezeigt. Der Nutzer kann bereits die ersten Seiten betrachten, während im Hintergrund die Folgeseiten erzeugt werden. Aufgrund dessen steigt die Auslastung der Ressourcen des Rechners.
Sowohl für den Programmierer als auch für den Anwender des Vocus Lohn und Gehalt entsteht ein deutlicher Mehrwert durch die neue Druckschnittstelle. Druckaufträge können schneller durchgeführt und Reports durch den Designer schneller und einfacher erstellt werden. Außerdem stehen mehr Exportmöglichkeiten zur Verfügung.

5 Zusammenfassung und Ausblick

Das Ergebnis dieser Arbeit ist eine neue Druckschnittstelle zur Übergabe von datenbankähnlichen Objekten, welche in der Druckvorbereitung in eine Datenbankstruktur umgewandelt werden. Durch dieses Verfahren ist die Schnittstelle leicht zu handhaben, steuerbar und flexibel einsetzbar. Die Verwendung von Datenbanken mit List & Label bringt enorme Geschwindigkeitsvorteile im Gegensatz zur Aufbereitung der Daten mit der Schnittstelle zu QuickReport. Im neuen Druckdialog können die Benutzer leicht Anpassungen an den Ausgabeeinstellungen (egal ob Drucker oder Export) vornehmen und ggf. mehrere Dokumente unterschiedlichen Typs erzeugen. Das Erstellen der Dokumente funktioniert dabei schnell und effizient: Sobald eine Seite erzeugt wurde, wird sie ausgegeben. Alle folgenden Seiten werden im Hintergrund nachgeladen. So ist es dem Benutzer möglich das Dokument bereits einzusehen, bevor es vollständig erzeugt wurde.

Die neue Druckschnittstelle mit List & Label befindet sich derzeit bei der Firma Vocus im Lohn- und Gehaltsprogramm in der internen Testphase, parallel zu der Implementierung mit QuickReport. In der kommenden Zeit soll die Testphase auf Pilotkunden ausgeweitet werden.
Als Übergangslösung wurde eine Konvertierungsroutine programmiert, welche die bisherigen Reports aufarbeitet und der neuen Druckschnittstelle übergibt. In Zukunft werden nach und nach alle Daten für Reports auf das Konzept der DataList umgestellt, entsprechende neue Layouts mit dem Designer entworfen und im Listenverzeichnis abgelegt. Um die Abwärtskompatibilität zu den Dokumenten im Archiv zu gewährleisten, überprüft eine Routine beim Druckaufruf, ob das Dokument sich aufbereiten lässt, um den Druck mit der neuen Druckschnittstelle durchzuführen, und übergibt es dann der neuen Schnittstelle. Für nicht kompatible Reports wird die bisherige Druckschnittstelle unverändert im Programm belassen.
In der kommenden Zeit werden dann die Listen im neuen Archivformat abgelegt werden, was die Prüfung hinsichtlich der Kompatibilität überflüssig macht. Dadurch können sie direkt über die neue Druckschnittstelle ausgegeben werden.

Einige der Kunden des Vocus- Lohn und Gehaltsprogrammes verwenden ein Dokumentenmanagementsystem. Für diese wird es in Zukunft eine Schnittstelle geben, über welche sie direkt Dokumente des Programmes in ihr System einpflegen können. Dabei wird parallel zum Druck eine Datei erzeugt, welche dem Dokumentenmanagementsystem übergeben wird. Bis zum jetzigen Zeitpunkt lagen noch keine Informationen über die Ansteuerung der Schnittstelle vor. In der kommenden Zeit muss die Druckschnittstelle um diese Funktion erweitert und den jeweiligen Kunden zur Verfügung gestellt werden.

In Zukunft soll es möglich sein, dass der Anwender des Vocus Lohn und Gehalt selbst Auswertungen erstellen und drucken kann. So kann er die Daten seinen Bedürfnissen anpassen und Listen über die von ihm auszuwertenden Daten drucken. Weiterhin wird derzeit das Konzept der Datenhaltung in einem Verzeichnis auf eine Firebird-Datenbank umgestellt. Im Ausblick auf die kommenden Entwicklungen wird es nötig sein, die Schnittstelle an die Verwendung einer Firebird-Datenbank anzupassen.

In dieser Arbeit wurde gezeigt, dass es möglich ist, eine variable Datenmenge so aufzubereiten, dass aus ihr ein komplexer Report erstellt werden kann. In Kombination mit dem Reporttool List & Label ist es der Firma Vocus in Zukunft möglich, selbst anspruchsvolle und komplexe Reports mit weniger Aufwand als bisher zu erstellen. Zudem wird die Möglichkeit geboten, fertige Dokumente in verschiedene Dateiformate zu exportieren und zu versenden. Der neue Vorschaudialog bietet hierfür entsprechende Funktionalitäten. In Zukunft werden Druckaufträge schneller aufbereitet und verarbeitet, was dem Nutzer wichtige Arbeitszeit spart. Dem Entwickler wird eine einfache Möglichkeit geboten, seine Daten an einen Report zu übergeben und so schnell dem Nutzer wichtige Informationen bereitzustellen.

Mit der neu implementierten Druckschnittstelle erhält sowohl der Entwickler von Vocus als auch der Kunde einen Mehrwert durch mehr Funktionalitäten, Geschwindigkeit und höhere Performance.

Quellenverzeichnis

Braun, Robert; Esswein, Werner; Greiffenberg, Steffen: Einführung in die Programmierung Berlin, Heidelberg, New York: 2006

combit GmbH: List & Label 18 Programmier Referenz. 2013, In: http://www.combit.net/de-de/reporting-tool/handbuch-report-generator-List-Label.pdf

DIN 66234, Teil 8: 1998
Bildschirmarbeitsplätze. Deutsches Institut für Normung e.V. Berlin: 1998

Embarcadero Technologies: A ClientDataSet in Every Database Application. 2012, In: http://edn.embarcadero.com/article/28876

Gleich, Ronald [Hrsg.]; Horváth, Péter [Hrsg.]; Michel, Uwe [Hrsg.]: Management Reporting. 1. Aufl. München: 2008

Herczeg, Michael: Software-Ergonomie. 1. Aufl. Bonn, Paris [u.a.]: 1994

Kleuker, Stephan: Grundkurs Software-Engineering mit UML. 2., korrigierte und erweiterte Aufl. Wiesbaden: 2011

Kosch, Andreas: Delphi Win 32 Lösungen Frankfurt: 2000

Niemann, Alexander: Das Einsteigerseminar Delphi 6. 1. Aufl. Kaarst: 2001

Shneiderman, Ben: Designing the user interface. 3rd ed Reading, Mass: 1998

Thaller, Georg Erwin: Software-Dokumente Hannover: 1995

Witt, Bernhard C.: Datenschutz kompakt und verständlich. 2., aktualisierte und ergänzte Aufl. Wiesbaden: 2010

Anhangsverzeichnis

Pflichtenblatt zum Einsatz einer neuen Druckerschnittstelle

Ausgangssituation und Zielsetzung

Seit 15 Jahren wird im Lohnprogramm VOCUS Lohn und Gehalt in der Entwicklungsumgebung die Drucksoftware Quick Report eingesetzt. Die Wahl fiel aus Kostengründen und aufgrund der flexiblen Layoutsteuerung auf diese Software. Bei der damaligen Umstellung des Lohnprogramms von DOS auf Windows konnten mit relativ geringem Aufwand die im Quelltext aufwändig programmierten Drucklisten umgestellt werden.

Quick Report verwendet ein eigenes Druckerobjekt für die Druckeransteuerung. Die Parametrierung der Drucker erfolgt ausschließlich über dieses Objekt. Hier wurde in den 15 Jahren wenig weiterentwickelt, so dass die funktionalen Anforderungen in Verbindung mit den modernen Druckern nicht mehr erfüllt werden. Außerdem fehlen moderne Funktionen, wie Datenexport in verschiedene Formate, Suchfunktion, komfortabler Layoutentwurf mit Test etc.

Systemeinsatz und Umgebungsbedingungen

Die Druckerschnittstelle soll in erster Linie in unserem Lohnprogramm „VOCUS Lohn und Gehalt“ mit hohen Sicherheitsanforderungen (Verschlüsselungsmöglichkeit von Datenausgaben) und komplexen Anforderungen an das Layout der Drucklisten (komplexe Zeilen- und Tabellenstrukturen) eingesetzt werden.
Weitere Einsatzgebiete sind diverse kleinere Programmtools zur Lohndatenverwaltung.

Anwenderschnittstelle

Die Bedienoberfläche für den Benutzer soll einfach und intuitiv sein, dass sie vor jeder Listenausgabe dem Anwender vorgeblendet wird. Durch den Anwender soll ein Standard vorbelegbar, und eine Parametrierung hinsichtlich Druckerauswahl und Druckereigenschaften möglich sein.

Die Entwickleroberfläche zur Layoutentwicklung soll übersichtlich und funktional eine schnelle Erfassung und sofortigen Test zulassen.

Funktionale Anforderungen

Die im Lohnprogramm vorhandenen Quellen zur Verwaltung und Ausgabe der Drucklisten müssen weiterverwendbar sein, evtl. ist für bestimmte Ausdrucke eine Erweiterung der vorhandenen Parameter mit geringem Aufwand vorzusehen.
Weiterhin muss das vorhandene Listenarchiv über die vorhandene 10-Jahreshistorie weiterhin verwendet werden können.
Für beide o.g. Anforderungen ist eine Konvertierung entweder temporär (für den aktuellen Ausdruck) oder dauerhaft (Erzeugung neuer Druckvorlagen aus den Vorhandenen und Ablage der Daten in einem neuen Format) erforderlich. Die konvertierten Listen müssen ohne Einschränkung lesbar sein.
Erfüllung neuer Funktionen hinsichtlich erweiterter und zukünftiger Druckerfunktionalitäten durch Erweiterung der Parameter in den Druckvorlagen.
Für die Unterstützung zukünftiger Betriebssysteme und Druckerhardware permanente Updatefähigkeit der Software.
Erschließung neuer Funktionen (z. B. Dokumentensuche), Exportmöglichkeit in verschiedene Formate (PDF, HTML, DOC, ODF, Entgeltnachweis Online über Intranet etc.)
Unterstützung einer Berechtigungssteuerung hierarchisch nach Mandant und Abrechnungkreis.
Die Unterstützung fremder Archivierungssoftware soll möglich sein.

Qualitätsanforderungen

Wichtig für den Anwender ist eine einfache intuitive Bedienbarkeit, um Anwendungsfehler gering zu halten.
Im Fehlerfall ist ein aussagekräftiges Fehlerprotokoll für den schnellen Support notwendig.

Technologische Rahmenbedingungen

Möglichst einfache Integration in die vorhandene Programmstruktur.
Die vorhandenen Formate der Druckvorlagen, Dateninhalte und das Listenarchiv sollen weiterverwendet werden können. Die Daten müssen ohne größeren manuellen Eingriff konvertiert werden können, um das alte Druckprogramm abzulösen.

Organisatorische Rahmenbedingungen

Die neue Druckschnittstelle wird schrittweise im Parallelbetrieb über Pilotkunden eingeführt.
In einem zweiten Schritt erfolgt dann ein Rollout an alle Kunden und im 3. Schritt, nach erfolgreicher Einführung, werden die alten Daten und Vorlagen dauerhaft konvertiert.
Damit wir die endgültige Ablösung der alten Druckschnittstelle Quick-Report vollzogen.

Fehlertoleranzmaßnahmen

Durch die vollkommene Neugestaltung der Druckvorlagen und fehlende Eigenschaften in den alten Druckvorlagen sind Abweichungen bei der Darstellung von Listen aus dem Archiv möglich. Hier muss zumindest die Lesbarkeit gewährleistet sein. Eventuell müssen in den Archivdokumenten bei der Konvertierung Parametererweiterungen vorgenommen werden.

Anforderungen an die Dokumentation

Erstellung einer Dokumentation für die Struktur der Vorlagen, das Konvertierungsprogramm und eine Onlinehilfe für die Bedienung durch den Anwender.

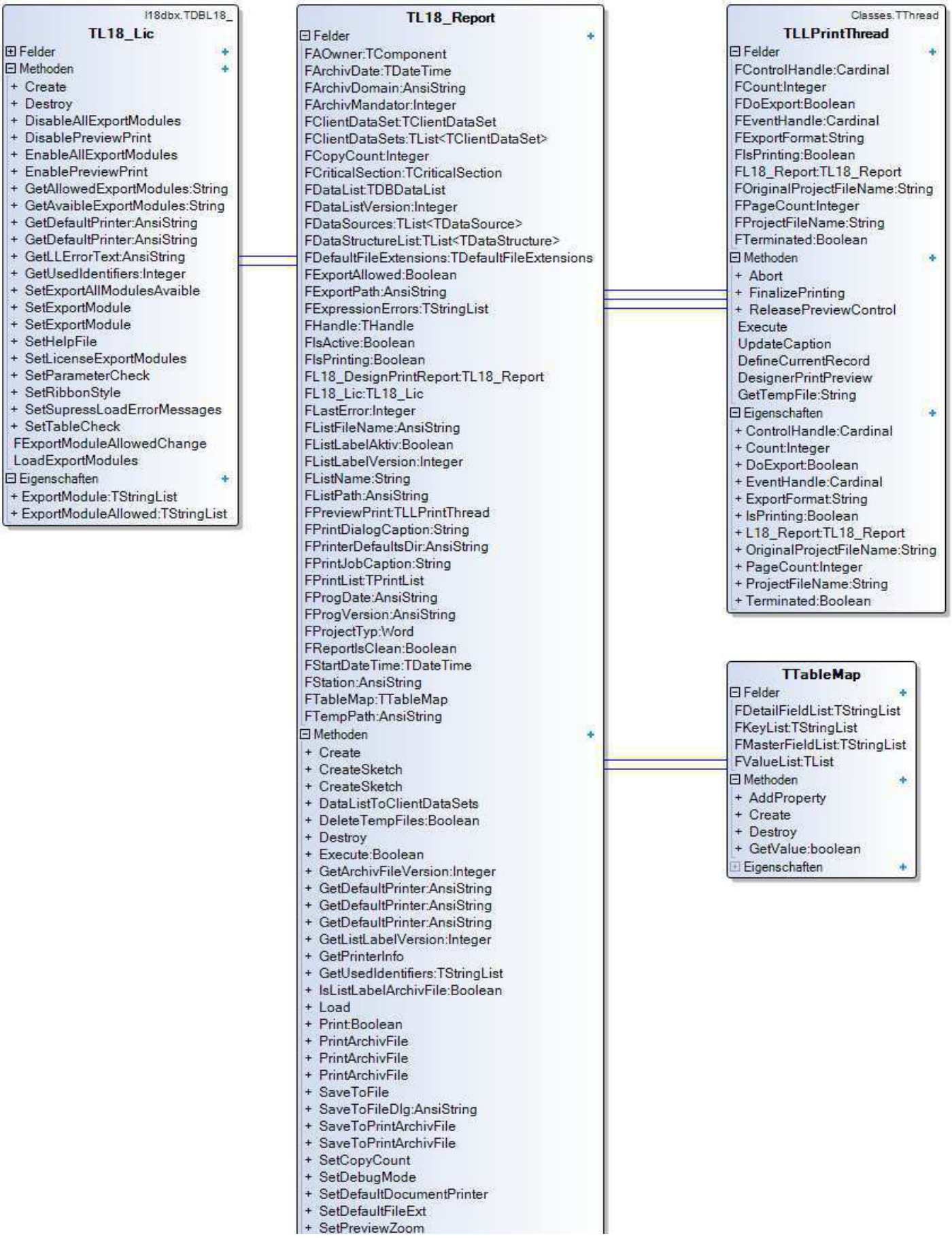
Abnahmebedingungen

Die Abnahme erfolgt über die Pilotphase bei ausgewählten Kunden bzw. im eigenen Haus.

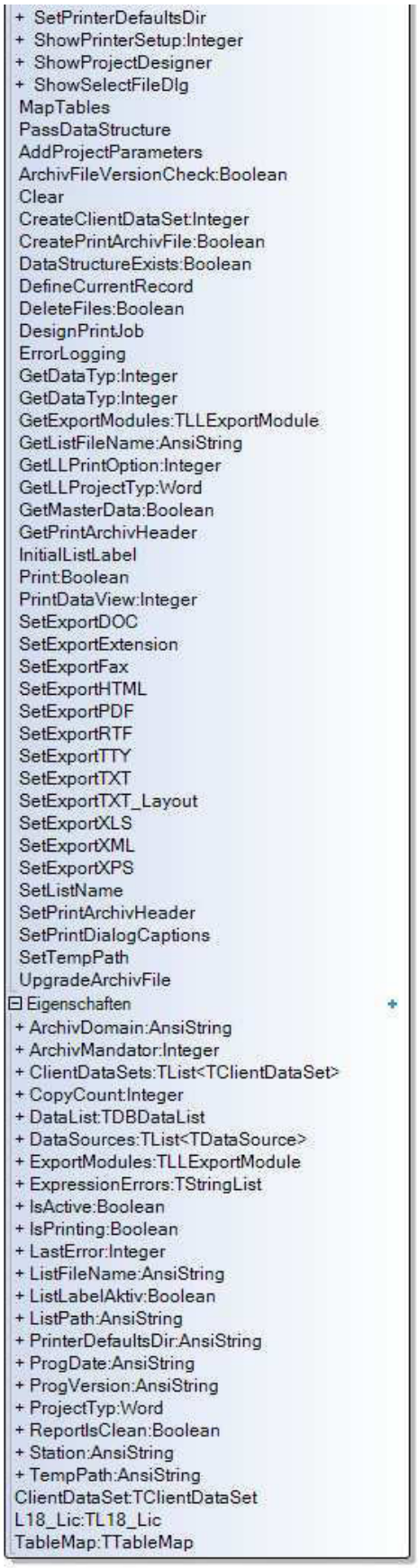
_____
Ort, Datum, Unterschrift des Auftraggebers

_____
Ort, Datum, Unterschrift des Auftragnehmers

```
685:        try
686:          if L18_Libraries.Load(Project.Dirs.getDir('SystemDir') + '\cmbtll') = 0 then
687:          begin
688:            SysUtils.ForceDirectories(Project.Dirs.SystemTemp + '\VocusReportTemp');
689:            L18_Report.Load(Project.Dirs.getDir('ReportDir'), Project.Dirs.SystemTemp + '\VocusReportTemp',
690:              StripR(Project.Dirs.Root, '\') + '\EXPORT',
691:              GlobalData.ValueExists('SysTab' + Project.Defs.Kuerzel),
692:              FormatDateTime('dd.mm.yyyy', Project.FSystemInformation.ProgDate),
693:              Format('%s   %s', [SFirmaProgHeader, GlobalData.GetString('DeuevProgVersion')]),
694:              Project.SystemInformation.StationName);
695:          end
696:          else
697:            DialogsData.FmtDialog(SFirmaProgHeader + ' - Startprüfung',
698:              'Report Module wurden nicht gefunden!', [mbOK], mbOK, True);
699:        except
700:          on E: Exception do
701:            ShowException(Application, @InitProject);
702:        end;
```

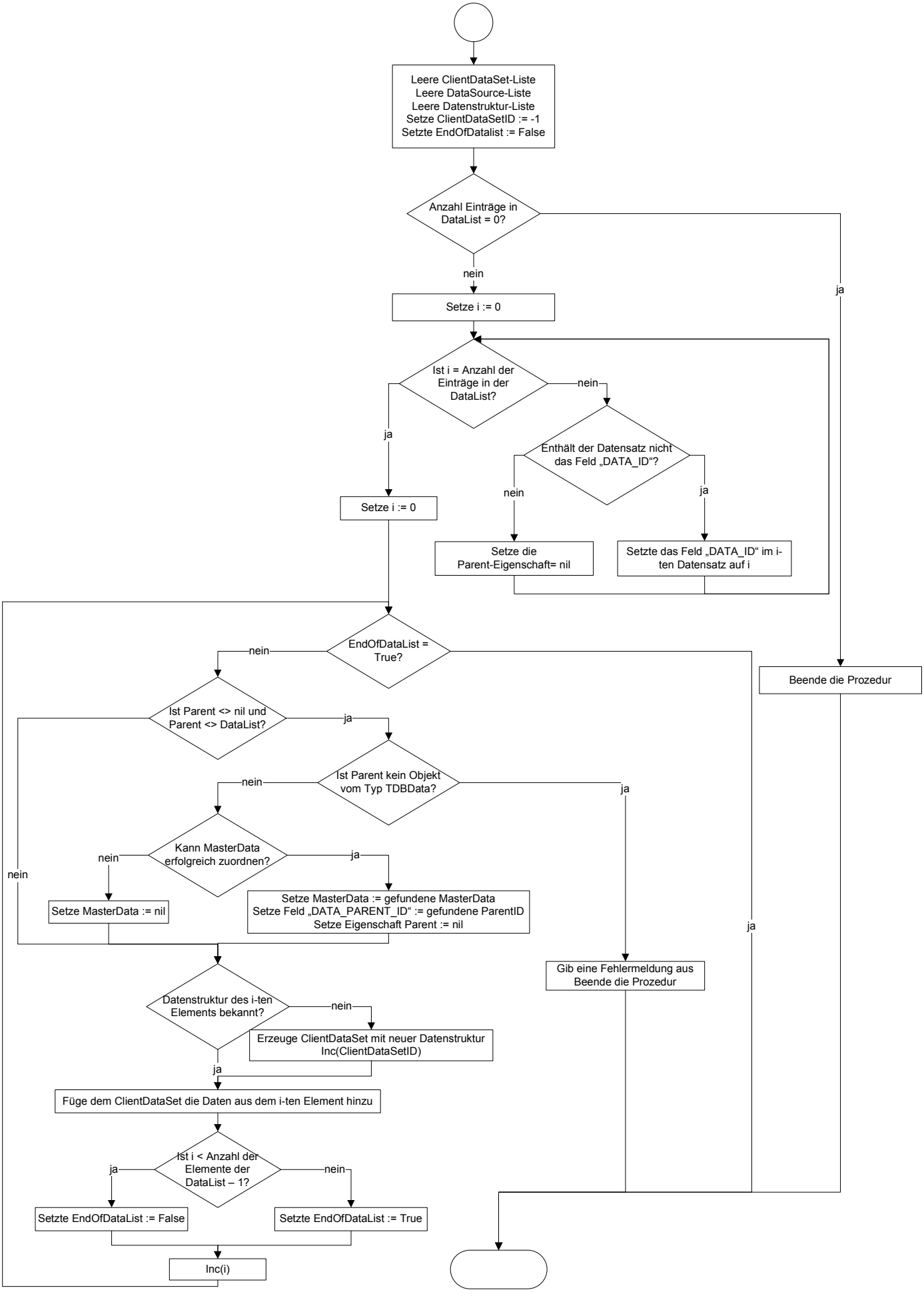


Fortsetzung der TL18_Report-Klasse auf nächster Seite


+ SetPrinterDefaultsDir
+ ShowPrinterSetup:Integer
+ ShowProjectDesigner
+ ShowSelectFileDlg
MapTables
PassDataStructure
AddProjectParameters
ArchivFileVersionCheck:Boolean
Clear
CreateClientDataSet:Integer
CreatePrintArchivFile:Boolean
DataStructureExists:Boolean
DefineCurrentRecord
DeleteFiles:Boolean
DesignPrintJob
ErrorLogging
GetDataTyp:Integer
GetDataTyp:Integer
GetExportModules:TLLExportModule
GetListFileName:AnsiString
GetLLPrintOption:Integer
GetLLProjectTyp:Word
GetMasterData:Boolean
GetPrintArchivHeader
InitialListLabel
Print:Boolean
PrintDataView:Integer
SetExportDOC
SetExportExtension
SetExportFax
SetExportHTML
SetExportPDF
SetExportRTF
SetExportTTY
SetExportTXT
SetExportTXT_Layout
SetExportXLS
SetExportXML
SetExportXPS
SetListName
SetPrintArchivHeader
SetPrintDialogCaptions
SetTempPath
UpgradeArchivFile
Eigenschaften
+ ArchivDomain:AnsiString
+ ArchivMandator:Integer
+ ClientDataSets:TList<TClientDataSet>
+ CopyCount:Integer
+ DataList:TDBDataList
+ DataSources:TList<TDataSource>
+ ExportModules:TLLExportModule
+ ExpressionErrors:TStringList
+ IsActive:Boolean
+ IsPrinting:Boolean
+ LastError:Integer
+ ListFileName:AnsiString
+ ListLabelAktiv:Boolean
+ ListPath:AnsiString
+ PrinterDefaultsDir:AnsiString
+ ProgDate:AnsiString
+ ProgVersion:AnsiString
+ ProjectTyp:Word
+ ReportIsClean:Boolean
+ Station:AnsiString
+ TempPath:AnsiString
ClientDataSet:TClientDataSet
L18_Lic:TL18_Lic
TableMap:TTableMap

```
1809: procedure TL18_Report.Load(const ListDir, TempPath, ExportPath: AnsiString;
1810:   ExportAllowed: Boolean; ProgDate, ProgVersion, Station: AnsiString);
1811: begin
1812:   try
1813:     if L18_Libraries.Loaded then
1814:     begin
1815:       FL18_Lic := TL18_Lic.Create(FAOwner);
1816:       FL18_DesignPrintReport := TL18_Report.Create(FAOwner, FHandle);
1817:       SetTempPath(InPathD(TempPath));
1818:       InitialListLabel;
1819:     end;
1820:     FListPath := InPathD(ListDir);
1821:     FExportAllowed := ExportAllowed;
1822:     FExportPath := ExportPath;
1823:     if FExportAllowed then
1824:       FL18_Lic.EnableAllExportModules
1825:     else
1826:     begin
1827:       FL18_Lic.DisableAllExportModules;
1828:       FL18_Lic.SetExportModule(llOutput_Export_PDF, True);
1829:       FL18_Lic.SetExportModule(llOutput_Export_HTML, True);
1830:     end;
1831:     FProgDate := ProgDate;
1832:     FProgVersion := ProgVersion;
1833:     FStation := Station;
1834:     // Entscheidung ob List Label genutzt werden darf!!
1835: {$IFDEF VOCUSLUG}
1836:     FListLabelAktiv := (Project.Defs.SysTab or
1837:       (Project.Defs.Kuerzel = 'SDG') or // Dargun
1838:       (Project.Defs.Kuerzel = 'GRS') or // Röslau
1839:       (Project.Defs.Kuerzel = 'VEL') or // Velden
1840:       (Project.Defs.Kuerzel = 'GSH') or // Stamhamm
1841:       (Project.Defs.Kuerzel = 'GEF') or // Edelsfeld
1842:       (Project.Defs.Kuerzel = 'GSB') or // Schmiedeberg
1843:       (Project.Defs.Kuerzel = 'GGR') or // Radibor
1844:       (Project.Defs.Kuerzel = 'SSB') or // Selbitz
1845:       (Project.Defs.Kuerzel = 'UNI') or // Werkzeug- und Gerätebau
1846:       (Project.Defs.Kuerzel = 'ABC') or // Britz Chorin
1847:       (Project.Defs.Kuerzel = 'KOK'));
1848: {$ENDIF}
1849: {$IFDEF PW_ListLabel}
1850:     FListLabelAktiv := True;
1851: {$ENDIF}
1852:   except
1853:     on E: Exception do
1854:       Exception.CreateFmt('List and Label Komponente konnte nicht geladen werden!'
1855:         + #13#10 + '[%s]', [E.ToString]);
1856:   end;
1857: end;
```

Anhang 5 PAP: DataListToClientDataSets (Umwandlung der DataList in DataSets)



```
1044: procedure TL18_Report.DataListToClientDataSets;
1045: var
1046:   ClientDataSetID: Integer;
1047:   i: Integer;
1048:   j: Integer;
1049:   MasterData: TDBData;
1050:   EndOfDataList: Boolean;
1051:   FieldName: AnsiString;
1052: begin
1053:   FClientDataSets.Clear;
1054:   FDataSources.Clear;
1055:   FDataStructureList.Clear;
1056:   ClientDataSetID := -1;
1057:   EndOfDataList := False;
1058:   try
1059:     if FDataList.CardEntries = 0 then
1060:     begin
1061:       { Keine Daten zum Drucken vorhanden }
1062:       Exit;
1063:     end;
1064:     for i := 0 to FDataList.CardEntries - 1 do
1065:     begin
1066:       if not FDataList[i].ValueExists(DATA_ID) then
1067:         FDataList[i].setInteger(DATA_ID, i)    // alle Daten bekommen eine eindeutige Nummer
1068:       else
1069:         FDataList[i].Parent := nil;   // Daten sind bereits nummeriert, Parent zurücksetzen
1070:     end;
1071:     { Ladebalken für Fortschritt }
1072:     //DialogsInfo.FmtShowMeter(METER_DESCRIPTION_LOAD_DATA, 0, smOpen);
1073:     i := 0;
1074:     while not EndOfDataList do
1075:     begin
1076:       if (FDataList[i].Parent <> nil)
1077:         and (FDataList[i].Parent <> FDataList) then
1078:       begin
1079:         if not (FDataList[i].Parent is TDBData) then
1080:           raise Exception.CreateFmt(EXCEPTION_PRINT_ALLG, ['Parent ist kein DBData-Objekt']);
1081:         if GetMasterData(i, TDBData(FDataList[i].Parent), MasterData) then
1082:         begin
1083:           FDataList[i].setInteger(DATA_PARENT_ID, MasterData.getInteger(DATA_ID));
1084:           FDataList[i].Parent := nil;
1085:         end
1086:         else
1087:           MasterData := nil;
1088:       end;
1089:
1090:       if not DataStructureExists(FDataList[i], ClientDataSetID) then
1091:         ClientDataSetID := CreateClientDataSet(FDataList[i]);
1092:       if not FClientDataSets[ClientDataSetID].Active then
1093:         FClientDataSets[ClientDataSetID].Open;
1094:       FClientDataSets[ClientDataSetID].LogChanges := False;
1095:       FClientDataSets[ClientDataSetID].DisableControls;
```

```
1096:       {
1097:       !!! Ab hier kein Debuggen durchführen !!!
1098:       Die Client DataSets reagieren sehr sensibel auf Zeitdifferenzen beim Hinzufügen von Daten!!!
1099:       Ansonsten kommt es zum Fehler, dass die Tabelle weder im Entwurfs- noch Einfügemodus geöffnet ist.
1100:       }
1101:       FClientDataSets[ClientDataSetID].Append;
1102:       for j := 0 to FClientDataSets[ClientDataSetID].Fields.Count - 1 do
1103:       begin
1104:         FieldName := AnsiupperCase(FClientDataSets[ClientDataSetID].Fields[j].FieldName);
1105:         case FClientDataSets[ClientDataSetID].Fields[j].DataType of
1106:           ftString:
1107:             begin
1108:               if length(FDataList[i].getString(FieldName)) > FClientDataSets[ClientDataSetID].Fields[j].DataSize then
1109:                 FDataList[i].setString(FieldName,
1110:                   Copy(FDataList[i].getString(FieldName), 1, FClientDataSets[ClientDataSetID].Fields[j].DataSize - 1));
1111:               FClientDataSets[ClientDataSetID].Fields[j].AsString := FDataList[i].getString(FieldName);
1112:             end;
1113:           ftInteger:
1114:             FClientDataSets[ClientDataSetID].Fields[j].AsInteger := FDataList[i].getInteger(FieldName);
1115:           ftWord:
1116:             FClientDataSets[ClientDataSetID].Fields[j].AsInteger := FDataList[i].getWord(FieldName);
1117:           ftFloat:
1118:             FClientDataSets[ClientDataSetID].Fields[j].AsFloat := FDataList[i].getExtended(FieldName);
1119:           ftCurrency, ftBCD:
1120:             FClientDataSets[ClientDataSetID].Fields[j].AsCurrency := FDataList[i].getCurrency(FieldName);
1121:           ftDate:
1122:             FClientDataSets[ClientDataSetID].Fields[j].AsDateTime := FDataList[i].getDate(FieldName);
1123:           ftTime:
1124:             FClientDataSets[ClientDataSetID].Fields[j].AsDateTime := FDataList[i].getTime(FieldName);
1125:           ftBoolean:
1126:             FClientDataSets[ClientDataSetID].Fields[j].AsBoolean := FDataList[i].getBoolean(FieldName);
1127:         else
1128:           raise EInternalError.CreateFmt(SInternalError, ['InvalidFieldType', FDataList[i].ClassName, 1]);
1129:         end;
1130:       end;
1131:       FClientDataSets[ClientDataSetID].Post;
1132:       { Debuggen ab hier wieder möglich }
1133:       FClientDataSets[ClientDataSetID].EnableControls;
1134:       if i < FDataList.CardEntries - 1 then
1135:         EndOfDataList := False
1136:       else
1137:         EndOfDataList := True;
1138:       //DialogsInfo.FmtShowMeter(METER_DESCRIPTION_LOAD_DATA, Round(i/(FDataList.CardEntries)*100), smRefresh);
1139:       Inc(i);
1140:     end;
1141:   finally
1142:     //DialogsInfo.FmtShowMeter(METER_DESCRIPTION_LOAD_DATA, 100, smClose);
1143:   end;
1144: end;
```

```
2015: function TL18_Report.Print(aDataList: TDBDataList;
2016:   const ListFileName: AnsiString;
2017:   const ListName: AnsiString;
2018:   const PrintOption: TLLPrintOption;
2019:   ExportOutput: TLLOutput;
2020:   ExportFileName: AnsiString;
2021:   ExportOptions: TLLExportOptions): Boolean;
2022: var
2023:   ProjectFileName: String;
2024:   nRet: Integer;
2025:   TableName,
2026:   SortOrder,
2027:   FMasterField,
2028:   FDetailField: String;
2029:   TableIndex: Integer;
2030: begin
2031:   if not FReportIsClean then
2032:     Clear;
2033:   FReportIsClean := True;
2034:   if aDataList <> nil then
2035:     FDataList := aDataList
2036:   else
2037:     raise Exception.CreateFmt(EXCEPTION_PRINT_ALLG,
2038:       ['DataList nicht zugewiesen']);
2039:   FListFileName := GetListFileName(ListFileName);
2040:   SetListName(ListName);
2041:   AddProjectParameters;
2042:   ProjectFileName := String(FListFileName);
2043:   SetPrintDialogCaptions(PrintOption);
2044:   DataListToClientDataSets;
2045:   TableIndex := 0;
2046:   MapTables;
2047:   PassDataStructure;
2048:
2049:   { Exportoptionen definieren }
2050:   if PrintOption = llPrint_Option_EXPORT then
2051:   begin
2052:     case ExportOutput of
2053:       llOutput_Export_PDF:
2054:         SetExportPDF(ExportFileName, ExportOptions);
2055:       llOutput_Export_HTML:
2056:         SetExportHTML(ExportFileName, ExportOptions);
2057:       llOutput_Export_RTF:
2058:         SetExportRTF(ExportFileName, ExportOptions);
2059:       llOutput_Export_XLS:
2060:         SetExportXLS(ExportFileName, ExportOptions);
2061:       llOutput_Export_XPS:
2062:         SetExportXPS(ExportFileName, ExportOptions);
2063:       llOutput_Export_TXT:
2064:         SetExportTXT(ExportFileName, ';', ExportOptions);
2065:       llOutput_Export_TXT_Layout:
2066:         SetExportTXT_Layout(ExportFileName, ExportOptions);
```

```
2067:       llOutput_Export_XML:
2068:         SetExportXML(ExportFileName, ExportOptions);
2069:       llOutput_Export_Graphic:;
2070:
2071:       llOutput_Export_DOC:
2072:         SetExportDOC(ExportFileName, ExportOptions);
2073:     end;
2074:   end;
2075:
2076:   {D:  Druckjob starten. Als Druckziel alle Exportformate erlauben. Fortschrittsbox mit Abbruchbutton.}
2077:   {nRet := FL18_Lic.LlPrintWithBoxStart(FProjectTyp, ProjectFileName, LL_PRINT_EXPORT, LL_BOXTYPE_STDABORT,
2078:     FHandle, Title);}
2079:   nRet := FL18_Lic.LlPrintWithBoxStart(FProjectTyp, ProjectFileName, GetLLPrintOption(PrintOption),
2080:     LL_BOXTYPE_STDABORT or LL_PRINT_REMOVE_UNUSED_VARS, FHandle, FPrintJobCaption);
2081:   {D:  Häufigste Ursache für Fehlercode: -23 (Syntax Error).}
2082:   if nRet <> 0 then
2083:   begin
2084:     if nRet = LL_ERR_ALREADY_PRINTING then { Versuche Druckjob zu beenden und neuzustarten }
2085:       FL18_Lic.LlPrintEnd(0);
2086:     raise Exception.CreateFmt(EXCEPTION_LIST_LABEL,
2087:       [nRet, FL18_Lic.GetLLErrorText(nRet)
2088:       + #13#10 + FExpressionErrors.Text]);
2089:   end;
2090:   {D:  Druckoptionsdialog. Aufruf ist optional, es können sonst Ausgabeziel und
2091:       Exportdateiname über LlXSetParameter() gesetzt werden bzw. der Drucker und
2092:       die Druckoptionen über LlSetPrinterInPrinterFile() vorgegeben werden.}
2093:   if PrintOption = llPrint_Option_EXPORT then
2094:   begin
2095:     case ExportOutput of
2096:       llOutput_Export_PDF:
2097:         FL18_Lic.LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT, 'PDF');
2098:       llOutput_Export_HTML:
2099:         FL18_Lic.LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT, 'HTML');
2100:       llOutput_Export_RTF:
2101:         FL18_Lic.LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT, 'RTF');
2102:       llOutput_Export_XLS:
2103:         FL18_Lic.LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT, 'XLS');
2104:       llOutput_Export_XPS:
2105:         FL18_Lic.LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT, 'XPS');
2106:       llOutput_Export_TXT:
2107:         FL18_Lic.LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT, 'TXT');
2108:       llOutput_Export_TXT_Layout:
2109:         FL18_Lic.LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT, 'TXT_LAYOUT');
2110:       llOutput_Export_XML:
2111:         FL18_Lic.LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT, 'XML');
2112:       llOutput_Export_Graphic:;
2113:         //FL18_Lic.LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT, 'PDF');
2114:       llOutput_Export_DOC:
2115:         FL18_Lic.LlPrintSetOptionString(LL_PRNOPTSTR_EXPORT, 'DOCX');
2116:     end;
2117:   end
2118:   else
```

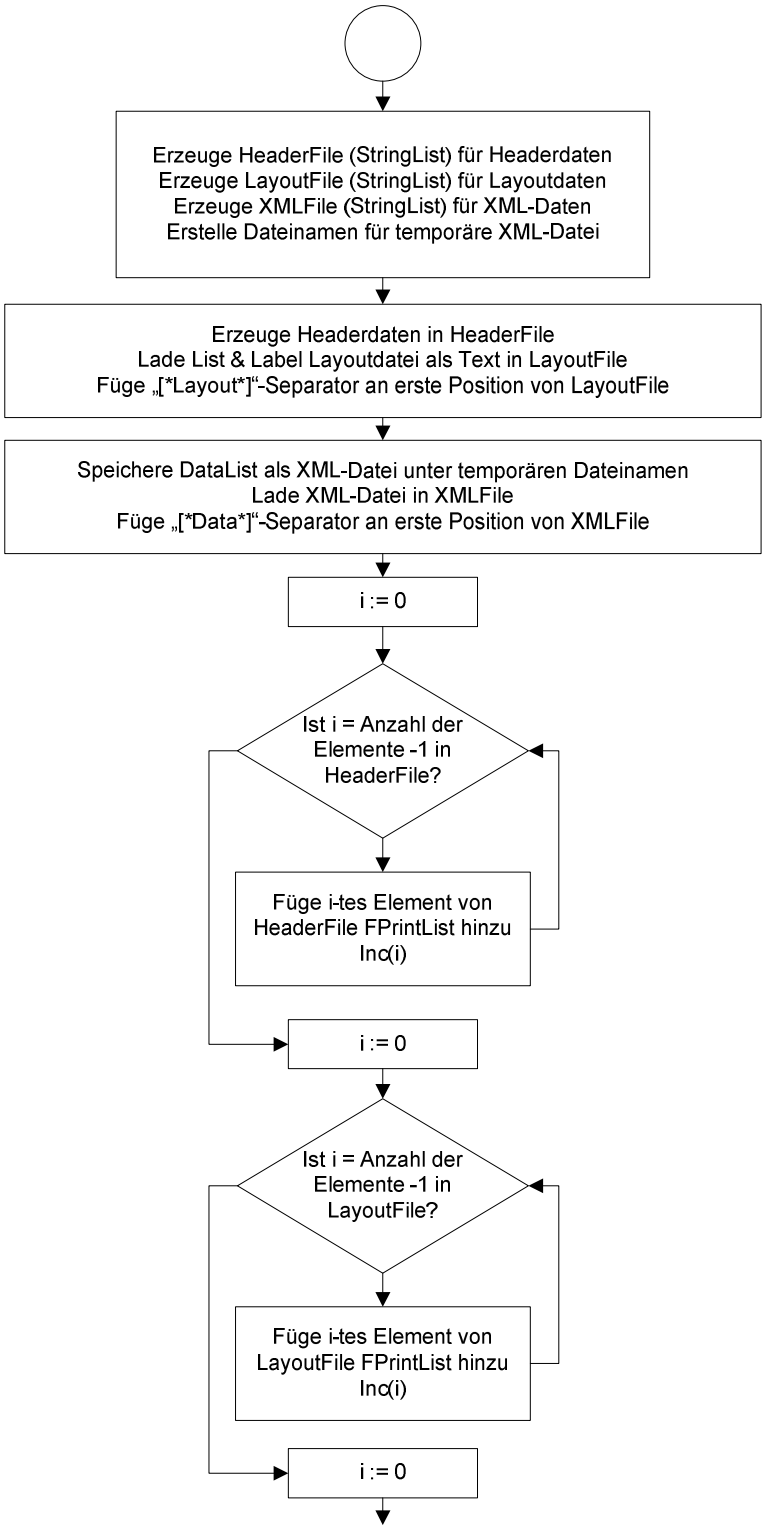
```
2119:     SetCopyCount(FCopyCount);
2120:
2121:   if nRet = LL_ERR_USER_ABORTED then
2122:   begin
2123:     FL18_Lic.LlPrintEnd(0);
2124:     Result := False;
2125:     Exit;
2126:   end;
2127:   {D:  Erste Seite initialisieren; auch hier kann schon durch Objekte vor der Tabelle
2128:        ein Seitenumbruch ausgelöst werden}
2129:   while FL18_Lic.LlPrint = LL_WRN_REPEAT_DATA do;
2130:   {D:  Root-Tabelle bestimmen und solange drucken bis keine mehr vorhanden}
2131:   FIsPrinting := True;
2132:   if (FL18_Lic.LlPrintDbGetRootTableCount <> 0) then
2133:   begin
2134:     repeat
2135:       FL18_Lic.LlPrintDbGetCurrentTable(TableName, false);
2136:       if TableName = EmptyStr then
2137:         FL18_Lic.LlDebugOutput(0, 'WRN: TableName is empty!!!');
2138:       if (TableName <> 'LLStaticTable')
2139:         or (TableName <> EmptyStr) then
2140:       begin
2141:         FTableMap.GetValue(TableName, FClientDataSet, FMasterField, FDetailField);
2142:         FL18_Lic.LlPrintDbGetCurrentTableSortOrder(SortOrder);
2143:         Inc(TableIndex);
2144:       end;
2145:     until (PrintDataView(FClientDataSet, 0, TableIndex) <> LL_WRN_TABLECHANGE);
2146:   end;
2147:   {D:  Druck beenden}
2148:   FLastError := nRet;
2149:   if FL18_Lic.LlPrintEnd(0) = 0 then
2150:     Result := True
2151:   else
2152:     Result := False;
2153:   FIsPrinting := False;
2154:   FReportIsClean := False;
2155: end;
```

```
2294: function TL18_Report.PrintDataView(ClientDataSet: TClientDataSet; Level,
2295:   CurrentTableNo: Integer): Integer;
2296: var
2297:   SortOrder,
2298:   TableName: String;
2299:   nRet: Integer;
2300:   Percentage,
2301:   MaxPercentage: Currency;
2302:   i: Integer;
2303:   FMasterField,
2304:   FDetailField: String;
2305: begin
2306:   FL18_Lic.LlPrintDbGetCurrentTable(TableName, False);
2307:   if (TableName = 'LLStaticTable')
2308:     or (TableName = EmptyStr) then
2309:   begin
2310:     nRet := FL18_Lic.LlPrintFields();
2311:     while nRet = LL_WRN_REPEAT_DATA do
2312:     begin
2313:       FL18_Lic.LlPrint();
2314:       nRet := FL18_Lic.LlPrintFields();
2315:       while nRet = LL_WRN_REPEAT_DATA do
2316:         nRet := FL18_Lic.LlPrintFieldsEnd;
2317:     end;
2318:   end
2319:   else
2320:   begin
2321:     ClientDataSet.First;
2322:     MaxPercentage := 100/FL18_Lic.LlPrintDbGetRootTableCount();
2323:     {D:  Sortierung der aktuellen Tabelle bestimmen}
2324:     FL18_Lic.LlPrintDbGetCurrentTableSortOrder(SortOrder);
2325:     {D:  Seitenumbruch auslösen, bis Datensatz vollständig gedruckt wurde}
2326:     for i := 0 to ClientDataSet.RecordCount - 1 do
2327:     begin
2328:       DefineCurrentRecord('', ClientDataSet, true);
2329:       nRet := FL18_Lic.LlPrintFields();
2330:       {D:  Seitenumbruch auslösen, bis Datensatz vollständig gedruckt wurde}
2331:       while nRet = LL_WRN_REPEAT_DATA do
2332:       begin
2333:         FL18_Lic .LlPrint();
2334:         nRet := FL18_Lic.LlPrintFields();
2335:       end;
2336:       {D:  Tabellenwechsel und Rekursion beginnen}
2337:       while nRet = LL_WRN_TABLECHANGE do
2338:       begin
2339:         FL18_Lic.LlPrintDbGetCurrentTable(TableName, false);
2340:         FTableMap.GetValue(TableName, FClientDataSet, FMasterField, FDetailField);
2341:         nRet := PrintDataView(FClientDataSet, Level + 1, 0);
2342:       end;
2343:       {D: Aktualisierung der Fortschrittsanzeige wenn Root-Datensatz gedruckt wird}
2344:       if Level = 0 then
2345:       begin
```

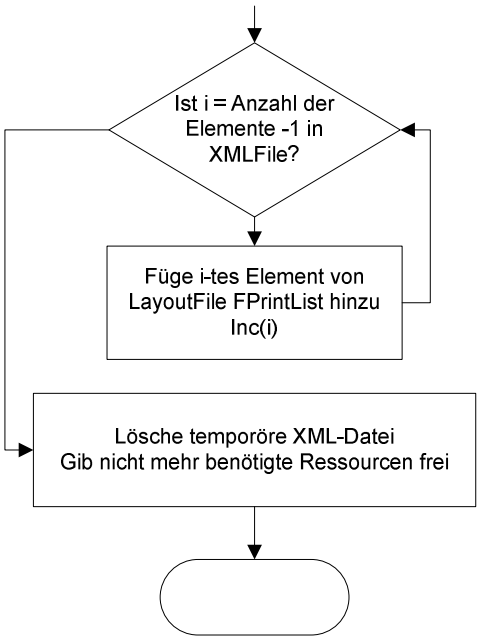
```
2346:         Percentage := MaxPercentage*(CurrentTableNo-1) + i/ClientDataSet.RecordCount*MaxPercentage;
2347:         FL18_Lic.LlPrintSetBoxText(FPrintDialogCaption, Round(Percentage));
2348:       end;
2349:       ClientDataSet.Next;
2350:     end;  // for i
2351:   end;
2352:   nRet := FL18_Lic.LlPrintFieldsEnd();
2353:   {D:  Seitenumbruch auslösen, bis Datensatz vollständig gedruckt wurde}
2354:   while nRet = LL_WRN_REPEAT_DATA do
2355:     nRet := FL18_Lic.LlPrintFieldsEnd();
2356:   Result := nRet;
2357: end;
```

```
675: procedure TDBDataList.LoadFromXMLFileWithStructure(
676:   const pFilePath: AnsiString; Encrypted: Boolean);
677: var
678:   XMLParser: TXmlParser;
679:   rootTag: TXMLTag;
680:   iRootData: Integer;
681:   parentData: TObject;
682:
683:   procedure SetXMLTagToData(aXMLTag: TXMLTag);
684:   var
685:     iTag: Integer;
686:     aData: TDBData;
687:     iData: Integer;
688:   begin
689:     if (aXMLTag.Attributes.Count = 0)
690:       and (aXMLTag.Value = EmptyAnsiStr) then
691:     begin
692:       aData := Self.AddData;
693:       aData.GetFromXML(aXMLTag);
694:       aData.Parent := parentData;
695:       parentData := aData;
696:     end;
697:     { Suche in den Subtags rekursiv nach TDBData Objekten }
698:     if aXMLTag.Items.TagCount > 0 then
699:     begin
700:       for iTag := 0 to aXMLTag.Items.Count - 1 do
701:         SetXMLTagToData(aXMLTag.Items.TagsByIdx[iTag]);
702:       { Subtags verlassen und Parent zurücksetzen }
703:       if TDBData(parentData).Parent is TDBData then
704:         parentData := TDBData(parentData).Parent
705:       else
706:         parentData := Self;
707:     end;
708:
709:   end;
710: begin
711:   // --> PW 25.03.2014: Laden einer strukturierten XML-Datei
712:   XMLParser := TXmlParser.Create;
713:   rootTag := nil;
714:   try
715:     if Encrypted then
716:       rootTag := XMLParser.ParseCryptedXMLFile(pFilePath)
717:     else
718:       rootTag := XMLParser.ParseXMLFile(pFilePath);
719:     if Assigned(rootTag) then
720:     begin
721:       Self.ListName := rootTag.Caption;
722:       parentData := Self;
723:       for iRootData := 0 to rootTag.Items.Count - 1 do
724:         SetXMLTagToData(rootTag.Items.TagsByIdx[iRootData]);
725:     end;
726:   finally
727:     XMLParser.Free;
728:     rootTag.Free;
729:   end;
730: end;
```

```
527: procedure TDBDataList.SaveToXMLFileWithStructure(
528:   const pFilePath: AnsiString; Encrypted: Boolean);
529: const
530:   XML_TAG = 'XML_TAG';
531: var
532:   XMLParser: TXmlParser;
533:   rootTag: TXMLTag;
534:   i: Integer;
535:   procedure CreateXMLTag(aDBData: TDBData; aRootXMLTag: TXMLTag);
536:   var
537:     aTag: TXMLTag;
538:   begin
539:     aTag := TXMLTag.create(aRootXMLTag, aDBData.DataName);
540:     aDBData.setPointer(XML_TAG, aTag);
541:     aDBData.SetToXML(aTag);
542:     if aTag.Items.ExistsTag(XML_TAG) then
543:       aTag.Items.Delete(aTag.Items.IndexOfTag(XML_TAG));
544:   end;
545: begin
546:   //--> PW 25.03.2014: Speichert die DataList mit Wurzel und Kind Tags
547:   // Erstellt eine XML-Datei mit Struktur
548:   XMLParser := TXmlParser.Create;
549:   rootTag := TXMLTag.create(nil, Self.ListName);
550:   try
551:     for i := 0 to Self.CardEntries - 1 do
552:     begin
553:       { Für jedes DBData wird ein XML-Tag erzeugt und im DBData hinterlegt }
554:       CreateXMLTag(Self[i], rootTag);
555:       if (Self[i].Parent <> nil)
556:         and (Self[i].Parent <> Self) then
557:         if Self[i].Parent is TDBData then
558:         begin
559:           { Sollte ein TDBData Parent existieren, wird der XML-Tag aus der Root gelöscht
560:             und im entsprechenden Parent neu erzeugt }
561:           rootTag.Items.Delete(rootTag.Items.IndexOf(Self[i].getPointer(XML_TAG)));
562:           CreateXMLTag(Self[i], TXMLTag(TDBData(Self[i].Parent).getPointer(XML_TAG)));
563:         end;
564:     end;
565:     if Encrypted then
566:       XMLParser.SaveToCryptedXML(rootTag, pFilePath)
567:     else
568:       XMLParser.SaveToXML(rootTag, pFilePath, True, True, ISO_8859_15, True);
569:   finally
570:     XMLParser.Free;
571:   end;
572: end;
```



Fortsetzung auf der nächsten Seite


Ist i = Anzahl der Elemente -1 in XMLFile?
Füge i-tes Element von LayoutFile FPrintList hinzu
Inc(i)
Lösche temporöre XML-Datei
Gib nicht mehr benötigte Ressourcen frei

```
 986: function TL18_Report.CreatePrintArchivFile: Boolean;
 987: var
 988:   i: Integer;
 989:   aDomain: AnsiString;
 990:   aMandatorNr: AnsiString;
 991:   DataListXMLFileName: AnsiString;
 992:   HeaderFile,
 993:   XMLFile,
 994:   LayoutFile: TStringList;
 995: begin
 996:   Result := True;
 997:   { Speichert einen List & Label Report im Archiv
 998:     [*Header*]
 999:     Informationen zur Datei
1000:     [*Layout*]
1001:     LST-Datei von List & Label
1002:     [*DataList*]
1003:     XML-Struktur }
1004:   HeaderFile := TStringList.Create;
1005:   LayoutFile := TStringList.Create;
1006:   DataListXMLFileName := ReplaceStr(FListFileName, '.'
1007:     + FDefaultFileExtensions.ProjectExt, '.xml');
1008:   XMLFile := TStringList.Create;
1009:   try
1010:     try
1011:       GetPrintArchivHeader(HeaderFile);   // Header erzeugen
1012:       LayoutFile.LoadFromFile(FListFileName);   // Layout übernhemen
1013:       LayoutFile.Insert(0, PRINTARCHIVFILE_LAYOUT);
1014:       FDataList.SaveToXMLFile(DataListXMLFileName, False);
1015:       XMLFile.LoadFromFile(DataListXMLFileName);    // Datalist XML übernehmen
1016:       XMLFile.Insert(0, PRINTARCHIVFILE_DATA);
1017:       for i := 0 to HeaderFile.Count - 1 do
1018:         FPrintList.Add(HeaderFile[i]);
1019:       for i := 0 to LayoutFile.Count - 1 do
1020:         FPrintList.Add(LayoutFile[i]);
1021:       for i := 0 to XMLFile.Count - 1 do
1022:         FPrintList.Add(XMLFile[i]);
1023:       DeleteFile(DataListXMLFileName);
1024:     except
1025:       Result := False;
1026:     end;
1027:   finally
1028:     HeaderFile.Free;
1029:     LayoutFile.Free;
1030:     XMLFile.Free;
1031:   end;
1032: end;
```

```
2236: procedure TL18_Report.PrintArchivFile(aArchivFile: TPrintList;
2237:   ListName: AnsiString; PreviewOnly, NoPrint: Boolean; Output: TLLOutput);
2238: var
2239:   IndexHeader,
2240:   IndexLayout,
2241:   IndexData: Integer;
2242:   Header,
2243:   Layout,
2244:   Data: TStringList;
2245:   DataList: TDBDataList;
2246:   LayoutFileName,
2247:   DataListXMLFileName: AnsiString;
2248:   i: Integer;
2249: begin
2250:   { Druck einen List & Label Report aus dem Archiv
2251:     Auslesen der Archivdatei um einen druckbaren Report zu erzeugen
2252:     Daten sind im XML-Format für eine TDBDataList }
2253:
2254:   { Validierung der Archivdatei }
2255:   if not IsListLabelArchivFile(aArchivFile) then
2256:     raise Exception.Create('Keine gültige Archivdatei!');     // Debug
2257:     //Exit;
2258:   { Sektionen wieder aufsplitten }
2259:   IndexHeader := aArchivFile.IndexOf(PRINTARCHIVFILE_HEADER) + 1;
2260:   IndexLayout := aArchivFile.IndexOf(PRINTARCHIVFILE_LAYOUT) + 1;
2261:   IndexData := aArchivFile.IndexOf(PRINTARCHIVFILE_DATA) + 1;
2262:
2263:   Header := TStringList.Create;
2264:   Layout := TStringList.Create;
2265:   Data := TStringList.Create;
2266:   DataList := TDBDataList.Create(TDBData);
2267:   { Layoutdatei erzeugen, TDBDataList aus XML einlesen }
2268:   try
2269:     for i := IndexHeader to IndexLayout - 2 do
2270:       Header.Add(aArchivFile[i]);
2271:     SetPrintArchivHeader(Header);
2272:     { Versionsvergleich und ggf. Anpassungen in der Verarbeitung }
2273:     if not ArchivFileVersionCheck then    // Versionsprüfung der Archivdatei
2274:       UpgradeArchivFile(aArchivFile);    // evt. Anpassung innerhalb der Datei
2275:     for i := IndexLayout to IndexData - 2 do
2276:       Layout.Add(aArchivFile[i]);
2277:     LayoutFileName := FTempPath + '\' + FListName + '.'
2278:       + FDefaultFileExtensions.ProjectExt;
2279:     Layout.SaveToFile(LayoutFileName);
2280:     for i := IndexData to aArchivFile.Count - 1 do
2281:       Data.Add(aArchivFile[i]);
2282:     DataListXMLFileName := FTempPath + '\' + FListName + '.xml';
2283:     Data.SaveToFile(DataListXMLFileName);
2284:     DataList.LoadFromXMLFileFast(DataListXMLFileName, False);
2285:     DeleteFile(DataListXMLFileName);
2286:     Execute(DataList, LayoutFileName, ListName);
2287:     DeleteFile(LayoutFileName);
2288:   finally
2289:     Header.Free;
2290:     Layout.Free;
2291:     Data.Free;
2292:     DataList.Free;
2293:   end;
2294: end;
```

```
527: procedure TTarifTabelle.Print(const ABundesland: AnsiString; aPrintDataList: TDBDataList);
528: const
529:   ItemsPerCollumn = 15;
530: var
531:   ix: integer;
532:   TabellenKopf,
533:   Stufen: TDBData;
534:   EoList: Boolean;
535: begin
536:   { TData 1 }
537:   TabellenKopf := TDBData(aPrintDataList.AddData);
538:   TabellenKopf.DataName := 'Tariftabellen';
539:   TabellenKopf.setString('Tabellenbezeichnung', Self.Name);
540:   TabellenKopf.setString('Bundesland', ABundesland);
541:   TabellenKopf.setInteger('Zugriffsart', Self.Zugriffsart);
542:   EoList := Self.FList.Count - ItemsPerCollumn <= 0;
543:   ix := 0;
544:   while not EoList do
545:   begin
546:     Stufen := TDBData(aPrintDataList.AddData);
547:     Stufen.DataName := 'Stufen';
548:     Stufen.Parent := TabellenKopf;
549:     Stufen.setString('von1', Self.Stufe[ix].von);
550:     Stufen.setString('bis1', Self.Stufe[ix].bis);
551:     Stufen.setExtended('Betrag1', Self.Stufe[ix].Betrag);
552:     Stufen.setString('von2', Self.Stufe[ix+ItemsPerCollumn].von);
553:     Stufen.setString('bis2', Self.Stufe[ix+ItemsPerCollumn].bis);
554:     Stufen.setExtended('Betrag2', Self.Stufe[ix+ItemsPerCollumn].Betrag);
555:     if Self.FList.Count-ItemsPerCollumn-1 > ix then
556:     begin
557:       EoList := False;
558:       Inc(ix);
559:     end
560:     else
561:       EoList := True;
562:   end;
563:   with Self do
564:   begin
565:     TabellenKopf.setExtended('ProzAntStufe1_16J', ProzAntStufe1_16J);
566:     TabellenKopf.setExtended('ProzAntStufe1_17J', ProzAntStufe1_17J);
567:     TabellenKopf.setExtended('ProzAntStufe1_18J', ProzAntStufe1_18J);
568:     TabellenKopf.setExtended('ProzAntStufe1_19J', ProzAntStufe1_19J);
569:     TabellenKopf.setExtended('ProzAntStufe1_20J', ProzAntStufe1_20J);
570:     TabellenKopf.setInteger('AnzUrlaub_0_29J', AnzUrlaub_0_29J);
571:     TabellenKopf.setInteger('AnzUrlaub_30_39J', AnzUrlaub_30_39J);
572:     TabellenKopf.setInteger('AnzUrlaub_40_99J', AnzUrlaub_40_99J);
573:     TabellenKopf.setExtended('Urlaubsgeld', Urlaubsgeld);
574:   end;
575: end;
```

```
577: procedure TTarifTabelle.Print(const ABundesland: AnsiString; APrintFile: TPrintList);
578: var
579:   ix: integer;
580:   s: AnsiString;
581: begin
582:   APrintFile.Add('<DA> 9|Tabellenbezeichnung|' + Self.Name + '  (' + ABundesland + ')|Zugriffsart(0,1,2,5,6,7):|' +
583:     Format('%d', [Self.Zugriffsart]) +
584:     '|%-Anteile für 16-20 Jahre (von Stufe 1)|16 J.|17 J.|18 J.|19 J.|20 J.|Urlaubstage für|' +
585:     '0..29,|30..39,|40..99|Jahre Lebensalter|Urlaubsgeld :|Stufe|von|bis|Betrag|Stufe' +
586:     '|von|bis|Betrag in EUR|1|16|2|17|3|18|4|19|5|20|6|21|7|22|8|23|9|24|10|25|11|26|12|27|13|28|' +
587:     '14|29|15|30|&');
588:   s := EmptyAnsiStr;
589:   for ix := 0 to Self.FList.Count - 1 do
590:     with Self.Stufe[ix] do
591:       s := s + von + '|' + bis + '|' + Format('%8.2f', [Betrag]) + '|';
592:   with Self do
593:     APrintFile.Add(s + Format('%5.2f', [ProzAntStufe1_16J]) + '|' +
594:       Format('%5.2f', [ProzAntStufe1_17J]) + '|' + Format('%5.2f', [ProzAntStufe1_18J]) + '|' +
595:       Format('%5.2f', [ProzAntStufe1_19J]) + '|' + Format('%5.2f', [ProzAntStufe1_20J]) + '|' +
596:       Format('%d', [AnzUrlaub_0_29J]) + '|' + Format('%d', [AnzUrlaub_30_39J]) + '|' +
597:       Format('%d', [AnzUrlaub_40_99J]) + '|' + Format('%8.2f', [Urlaubsgeld]));
598: end;
```

Testplan: Druckschnittstelle mit List & Label

Einleitung

Das vorliegende Dokument dient zu Ermittlung der Schnelligkeit und Funktionalitäten der neuen Druckschnittstelle mit List & Label im Vocus Lohn und Gehalt. Grundlage für den Test bildeen das Pflichtenheft und die allgemeinen qualitativen Anforderungen an die Lohn- und Gehaltssoftware.

Zu testende Objekte

Getestet werden soll die Schnittstelle für die neue Druckanbindung, hauptsächlich die Klasse „TL18_Report“, welche die Schlüsselkomponente der Schnittstelle bildet. Zwangsläufig muss hierbei auch das Laden der DLLs beim Starten des Programmes getestet werden.

Weiterhin soll die Handhabung und Steuerung des neuen Druckdialogs und des neuen Vorschaudialogs überprüft werden.

Zu testende Leistungsmerkmale

Leistungsmerkmal	Spezifikationen
Schnelligkeit der Druckaufbereitung	Die Aufbereitung der Daten soll in einer angemessenen Zeit stattfinden, der Nutzer soll über den Fortschritt informiert werden.
Handhabung des Druckdialogs	Der Druckdialog (Druckauswahl und Vorschau) sollen einfach und verständlich durch den Nutzer handhabbar sein.
Einstellungen des Druckertreibers	Die Einstellungen, welche im Drucker gesetzt werden, müssen von der Schnittstelle beachtet werden, z.B. Anzahl Kopien, Duplexdruck.
Archivierung des gedruckten Reports	Der ausgegebene Report muss im Archiv erfolgreich ablegbar und wieder einsehbar sein. Das Wiederraufbereiten darf nicht länger als das ursprüngliche Erzeugen des Dokumentes dauern. Die Darstellung darf nicht abweichen.
Exportfunktionen	Der Export in die angegebenen Dateiformate muss in akzeptabler Zeit ablaufen. Der Export in bestimmte Dateiformate (wie Excel, Word) soll nur für lizensierte Kunden möglich sein.
Laden der DLLs beim Programmstart	Es soll überprüft werden, wie das Programm reagiert, wenn die DLLs im Lohnamt fehlen oder nur teilweise verfügbar sind.

Nicht zu testende Leistungsmerkmale

Leistungsmerkmal	Begründung
Schnelligkeit der Zusammenstellung der Daten für die Schnittstelle	Dieses Merkmal ist nicht im direkten Zusammenhang mit der Druckerschnittstelle und dessen Geschwindigkeit.
Dauer und Qualität des Exports	Dies hängt von der List & Label Komponente ab, die Schnittstelle hat darauf keinen Einfluss.
Suchfunktion in der Vorschau	Kein direkter Zusammenhang mit der Schnittstelle, diese Funktionaltät wird von der List & Label Komponente bereitgestellt

Teststrategie

Testansatz

Das Testen der Schnittstell erfolgt über eine große Datenmenge, innerhalb des Vocus Lohn und Gehalt. Dabei wird eine bisherige Druckliste mittels DataList erzeugt und der Schnittstelle übergeben.

Hierfür werden Anpassungen im Quellcode vorgenommen, um den bisherigen Programmablauf nicht zu gefährden, werden alle Tests in einer isolierten Lohnumgebung vorgenommen.

Testmethoden

Es werden diverse Funktionstests durchgeführt, die sich an den zu testenden Leistungsmerkmale orientieren. Weiterhin soll der Zeitfaktor überprüft werden.

Es wird der gesamte Ablauf einer Ansteuerung der Druckschnittstelle getestet. Von der Übergabe der Daten, zur Aufbereitung für den Druck bis hin zur Ausgabe im Vorschaudialog und auf dem Drucker.

Ziel ist es die Schnittstelle auf ihre volle Funktionalität und Einsetzbarkeit zu überprüfen.

Pass / Fail – Kriterien

Der Test ist erfolgreich verlaufen, wenn das Dokument wie bisher dargestellt wird und in einer kürzeren Zeitdauer aufbereitet wird. K.o.-Kriterium ist außerdem die fehlerfreie Darstellung der übergebenen Daten.

Analog dazu ergeben sich die Kriterien für den Fehlschlag des Tests.

Durchführung

Zeittest

Es wird die Liste mit den „Globalen Daten“ der Schnittstelle übergeben. Es wird im ersten Test gemessen, welche Zeit notwendig ist, um die Daten für den Druck vorzubereiten, und mit der bisherigen verglichen. Gemessen wird die Zeit vom Start der Vorschau bis der Vorschaudialog erscheint.

Ergebnis: 3 Sekunden (Vergleichswert: 10 Sekunden)

Handhabung des Druckdialogs

Die Handhabung des Druckdialogs wurde verändert, die Funktionen sind aber gleich geblieben bzw. wurden ergänzt. Der Nutzer erreicht Funktionen jetzt schneller, muss sich jedoch an ein abgeändertes Dialogdesign und Verhalten gewöhnen. Der Dialog schließt nun nicht mehr automatisch nach dem Start einer Aktion, sondern bleibt im Hintergrund geöffnet, um weitere Aufgaben auszuführen, wie z.B. erst ein Dokument drucken und anschließend als PDF-Datei exportieren. Im Test wurde das neue Verhalten als positiv bewertet, es ist jedoch zu abzuwarten, in wie weit sich das neue Verhalten bei den Kunden etabliert.

Einstellungen des Druckertreibers

Für diesen Test wurde wieder die Liste mit den Globalen Daten gewählt und davon sechs Tariftabellen für den Druck ausgewählt, so dass genau zwei Seiten voll bedruckt werden. Als erstes wird getestet, ob der Druckertreiber korrekt angesprochen wird, indem ein Duplexdruck gestartet wird. Hierfür wurde das Multifunktionsgerät „Kyocera FS-6530MFP KX“ gewählt.

Der Test war erfolgreich, das Dokument wurde beidseitig bedruckt.

Zum Testen der direkten Einstellung der Anzahl von Kopien wird ein weiterer Druckauftrag durchgeführt. Getestet wird die direkte Auswahl von 2 Kopien im Druckdialog.

Auch dieser Test war erfolgreich.

Archivierung gedruckter Reports

Per Parameter wurde ein Dokument dem Testarchiv hinzugefügt. Dies wurde korrekt eingeordnet und wird als Dokument aufgeführt. Das Anzeigen des Dokumentes aus dem Archiv ist ohne erwähnenswerte Verzögerungen geschehen und wurde im abgelegten Erscheinungsbild wieder angezeigt.

Der Test war erfolgreich, Dokumente können archiviert und wieder aufbereitet werden.

Export in diverse Dateiformate

Da die Routine zum Drucken und zum erstellen von Exportdateien grundlegend gleich sind, wird erwartet, dass auch dieser Test erfolgreich verläuft. Ausgewählt wird wieder die Liste der Globalen Daten, welche in das Word-, HTML- und PDF-Format exportiert werden soll.

Exportmodul	Auswertung
Word	Erfolgreich, jedoch benötigt diese Exportvariante eine gewisse Zeit. Leider kann man daran nichts ändern, da der Export von den List & Label Komponenten durchgeführt wird. Desweiteren ist die Zeit kein Leistungskriterium für diesen Test.
PDF	Erfolgreich, ohne nennenswerte Verzögerungen.
HTML	Erfolgreich, ohne nennenswerte Verzögerungen.

Alle Export-Tests wurden erfolgreich abgeschlossen.

DLL-Ladetest

Für diesen Test werden alle DLLs für List & Label aus der Installation entfernt. Dabei ist zu erwarten, dass das Programm nicht abstürzt, sondern lediglich beim Starten eine Fehlermeldung ausgibt, dass die Report-Module nicht gefunden wurden. Innerhalb des Programms, sollten die Druckfunktionen deaktiviert sein, stattdessen soll derzeit noch auf die alte Schnittstelle zurückgegriffen werden.

Starttest: Erfolgreich, Programm startet, trotz des Fehlens der DLLs

Drucktest: Erfolgreich, das Programm verwendet stattdessen die alte Schnittstelle (Abwärtskompatibilität)

Ergebnis

Alle durchgeführten Tests wurden erfolgreich abgeschlossen. Die Druckschnittstelle ist bereit für einen Piloteinsatz. Durch das Feedback der Pilotkunden können Verbesserungen durchgeführt werden. Die geforderten Merkmale des Pflichtenheftes wurden erfüllt.

Testumgebung

Als Testumgebung dienten ein Entwicklungsrechner mit Windows 7, 64bit und ein weiterer Testrechner mit Windows 7, 32bit. Der Druck wurde u. a. auf einem Kyocera, sowie einem Brother-Drucker durchgeführt. Derzeit laufen noch längerfristige Tests auf anderen Systemen.

Sonstige Bemerkungen

Die Testreihe verlief sehr zufriedenstellend. Die essentiellen Funktionen, die von der Schnittstelle erwartet werden, sind einsatzbereit und voll funktionstüchtig. Weitere Tests werden durchgeführt, speziell was den Einsatz während der Lohnrechnung und den Druck auf Nadeldruckern betrifft.

Thesenblatt

- Eine funktionierende Druckschnittstelle ist für eine reibungslose Lohnabrechnung unabdingbar.

- Die derzeitige QuickReport-Komponente in Vocus Lohn und Gehalt bietet nicht die geforderten Funktionalitäten, welche von einer modernen Reportingkomponente zu erwarten sind.

- List & Label bietet alle geforderten Funktionalitäten. Ihre Implementierung in das Vocus Lohn- und Gehaltsprogramm bedarf einer neuen Druckschnittstelle.

- Durch die Implementierung der neuen Druckschnittstelle ist es möglich einfach und schnell komplexe Reports zu erstellen.

- Die neue Druckschnittstelle besteht aus mehreren Klassen. Diese übernehmen unterschiedliche Aufgaben. Für das Ausführen eines Druckauftrages ist die „TL18_Report“-Klasse zuständig.

- Die Daten werden in einer DataList dem Report zur Verfügung gestellt. Damit List & Label sie verarbeiten kann, bedarf es deren Umwandlung in Datenbankobjekte.

- Ein Neuentwurf des aktuellen Druckdialogs bringt dem Anwender mehr Funktionen und einen komfortablen Zugriff auf die Druckereinstellungen.

- Der Druck von großen Listen wird schneller durchgeführt als bisher.

- Die neue Druckschnittstelle wurde an das bestehende Archivierungssystem angepasst.

- Die Benutzerkontrolle muss auf das Archivsystem ausgeweitet werden, damit die personenrelevanten Daten nur für zugriffsberechtigte Personen einsehbar sind.

- Eine Anbindung der Druckschnittstelle an externe Dokumentenmanagementsysteme ist in Planung.

Ehrenwörtliche Erklärung

"Ich erkläre hiermit ehrenwörtlich",

1. dass ich meine Bachelorthesis mit dem Thema

Erstellung einer Schnittstellenbeschreibung für die Integration der Listenerstellung in VOCUS Lohn mit List & Label

ohne fremde Hilfe angefertigt habe,

2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und

3. dass ich meine Bachelorthesis bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Ort, Datum

Unterschrift