Einleitung

Extension Objects und Document Extensions sind ein hervorragendes Mittel, um die Business Intelligence Software QlikView und Qlik Sense an die individuellen Anforderungen des Anwenders anzupassen. Die Entwicklung der Erweiterungen erfolgt ausschließlich über Webtechnologien wie HTML, CSS und JavaScript. Die Begrenzung auf eine einzige Programmiersprache bedeutet jedoch nicht, dass ausschließlich JavaScript-Entwickler für die Erstellung der Erweiterungen infrage kommen.

Es existiert eine Reihe von Compilern, welche Quellcode anderer Hochsprachen in JavaScript umwandeln. Einer davon ist der Compiler dart2js der Programmiersprache Dart. Der Autor der vorliegenden Arbeit sammelte in vergangenen Projekten bereits einige Erfahrungen mit dieser neuen Programmiersprache. Sie soll daher als Beispiel für die Analyse der Anwendungsmöglichkeiten der JavaScript generierenden Hochsprachen für die Erweiterungsentwicklung für QlikView und Qlik Sense verwendet werden.

Die Entwicklung der Erweiterungen mit Dart soll nicht weniger komfortabel vonstattengehen, als es mit JavaScript der Fall ist. Durch eventuelle Vorgaben der Plattformen QlikView und Qlik Sense könnte es jedoch Einschränkungen geben. Welche das sind und wie auf diese reagiert werden kann, soll in dieser Arbeit herausgefunden werden.

Ziel der vorliegenden Arbeit ist es darüber hinaus, die strukturgebenden Konzepte der Programmiersprache Dart zu verwenden, um die Erweiterungsentwicklung effektiver und effizienter zu gestalten. Der Einarbeitungsaufwand soll verringert, die Entwicklung beschleunigt und die Portierbarkeit maximiert werden.

Die Arbeit ist folgendermaßen strukturiert:

Das Kapitel 1 fasst die notwendigen Grundlagen der Business Intelligence Plattformen QlikView und Qlik Sense, der dazugehörigen Erweiterungen sowie der Programmiersprache Dart zusammen.

Im Kapitel 2 werden ein Extension Object und eine Document Extension für QlikView sowie ein Extension Object für Qlik Sense mit JavaScript entwickelt. An den Beispielen wird der Umgang mit den APIs beschrieben.

Anschließend wird im Kapitel 3 für jede erstellte Erweiterung eine entsprechende Variante in Dart entwickelt. Dafür wird zunächst eine Klassenbibliothek erstellt, welche die Entwicklung der Erweiterungen vereinfachen soll. Die durch den zusätzlichen Schritt der Kompilierung entstehenden Nachteile sollen mit der Entwicklung von zwei Pub Transformern kompensiert werden.

Im Kapitel 4 werden die Vor- und Nachteile der Entwicklung von Erweiterungen mit Dart gegenübergestellt und abschließend eine Empfehlung ausgesprochen.

Der Ausblick in Kapitel 5 beschreibt weitere Anwendungsgebiete, welche mit der Programmiersprache Dart für die Entwicklung von Erweiterungen möglich wären.

1 Grundlagen

1.1 Die Business Intelligence Plattformen QlikView und Qlik Sense

QlikView QlikView ist ein durch QlikTech entwickeltes Werkzeug zur Analyse von Unternehmensdaten. Es erlaubt dem Nutzer, durch einen Klick mehr Informationen zu den gewünschten Datensätzen zu erhalten. Innerhalb von wenigen Sekunden wird eine neue Analyse für die ausgewählte Selektion durchgeführt. Um diese Geschwindigkeit zu ermöglichen, werden die Daten in den Arbeitsspeicher geladen.[1]

In Abbildung **??** ist ein Sheet im sogenannten WebView von QlikView 11 zu sehen. Auf der linken Seite sind Objekte in einem Fenster aufgelistet, die dem Sheet hinzugefügt werden können. Die Objekte haben in QlikView den Namen Sheet Objects. Es handelt sich dabei um Textfelder, Listen, Tabellen und diverse Diagrammtypen. Per Drag-and-Drop können die Sheet Objects auf der Arbeitsfläche positioniert werden. Auf der rechten Seite der Abbildung sind eine Listbox und einige der Diagramme abgebildet. Im unteren Bereich des Fensters ist der Reiter *Erweiterungsobjekte* zu sehen. Er öffnet eine Auflistung der sogenannten Extension Objects, die in der vorliegenden Arbeit näher thematisiert werden.

Ändert sich die Selektion der Daten, werden die Diagramme neu generiert, wie in Abbildung **??** auf Seite **??** zu sehen ist. Nachdem zwei Datensätze ausgewählt wurden, zeigen die Diagramme ausschließlich die Daten der ausgewählten Selektion an.

In QlikView erstellte Dokumente können über die Anzeige in einem Webbrowser auf unterschiedlichen Geräten dargestellt werden. Die Sheet Objects sind allerdings absolut positioniert und passen sich daher nicht der Größe des Displays an.

Qlik Sense Qlik Sense ist eine weitere Business Intelligence Software der Firma, die sich mittlerweile in Qlik umbenannt hat.[2] Die Software ähnelt QlikView in vielerlei Hinsicht, ist jedoch leichter zu bedienen und wurde für die Anzeige auf Geräten verschiedenster Größe vorbereitet. In Qlik Sense werden die Sheet Objects in einem Raster eingepasst. Das Raster füllt immer den zum Zeichnen verfügbaren Bereich aus.[3]

Abbildung **??** zeigt ein Arbeitsblatt in Qlik Sense, in der ähnliche Sheet Objects zum Einsatz kommen wie bereits in Abbildung **??**. Einige der in QlikView verfügbaren Sheet Objects existierten jedoch zum Zeitpunkt der Erstellung der vorliegenden Arbeit nicht in Qlik Sense.

1. Vgl. Stasieńko, »BI IN-MEMORY–THE NEW QUALITY OF BUSINESS INTELLIGENCE SYSTEMS«, S. 90
2. Vgl. QlikTech, *Business Intelligence and Data Visualization Software Company | Qlik*
3. Vgl. O'Donovan, »Qlik Sense For Beginners«, S. 11

In der Liste im linken Bereich der Abbildung erscheinen Sheet Objects sowie Extension Objects gleichermaßen und werden nicht wie in QlikView kategorisiert.

1.2 Extension Objects und Document Extensions in QlikView

QlikView verfügt bereits über eine Reihe von Diagrammtypen. Spezielle Anforderungen an die Anwendung erfordern jedoch auch speziellere Diagramme. Daher gibt es Bedarf nach neuen Diagrammtypen, die für einen individuellen Anwendungsfall benötigt werden, jedoch aufgrund ihrer Individualität nicht in QlikView integriert werden können. Zu diesem Zweck wurden in der Version 10 von QlikView die Extension Objects eingeführt.[4]

Diese Erweiterungen können genau wie die herkömmlichen Sheet Objects im Dokument frei positioniert und skaliert werden. Sie können Daten vom Server anfordern und sie mithilfe von Webtechnologien wie HTML, SVG, CSS und JavaScript beliebig visualisieren. Eine weitere Möglichkeit QlikView zu erweitern stellen die Document Extensions dar, die in der Version 11 von QlikView hinzugefügt wurden.[5] Anders als die Extension Objects handelt es sich hier nicht um positionierbare Objekte im Dokument, sondern um Erweiterungen des Dokumentes an sich. So kann beispielsweise mit den Document Extensions die Darstellung des Dokumentes angepasst werden.

In den weiteren Ausführungen dieses Kapitels wird beschrieben, wie Extension Objects und Document Extensions aufgebaut sind, wie sie erstellt und installiert werden, welche Nachteile Extension Objects in QlikView gegenüber den Sheet Objects haben und wie sich Extension Objects und Document Extensions auf die Performance auswirken.

1.2.1 Struktur der Extension Objects und Document Extensions

Extension Objects und Document Extensions in QlikView sind sich in ihrer Struktur sehr ähnlich. Sie haben beide eine *Definition.xml*-Datei, in der beispielsweise Name und Typ definiert werden, und eine *Skript.js*-Datei, in der das Verhalten der Erweiterung programmiert werden kann. Extension Objects haben darüber hinaus noch eine *Properties.qvpp*-Datei, in welcher der Konfigurationsdialog beschrieben wird, und eine *Icon.png*-Datei, die als Vorschaubild in der Liste der verfügbaren Extension Objects angezeigt wird. Beide Dateien werden bei der Erstellung von Document Extensions ignoriert, da es keinen Konfigurationsdialog und auch keine Liste verfügbarer Document Extensions gibt, in der ein Vorschaubild auftauchen könnte.

Im Nachfolgenden wird beschrieben, welche Funktion die einzelnen Dateien haben.

Definition.xml Die *Definition.xml*-Datei bestimmt die grundlegende Konfiguration des Extension Objects bzw. der Document Extension. Der Name, der Dateipfad und der Typ (Extension Object oder Document Extension) werden hier definiert. In der *Definition.xml*-Datei wird die Anzahl der von dem Extension Object verwendeten Dimensionen und Formeln

4. Vgl. Redmond, *QlikView for Developers Cookbook*, S. 163
5. Vgl. Redmond, *QlikView for Developers Cookbook*, S. 164

festgelegt. Außerdem müssen benutzerdefinierte Parameter in dieser Datei beschrieben werden. Zu guter Letzt können noch Startwerte für die Dimensionen, Formeln und benutzerdefinierten Parameter festgelegt werden.

In Abbildung **??** auf Seite **??** ist unter anderem der Inhalt der Definition.xml-Datei zu sehen.

Der Wurzelknoten ist bei Document Extensions und Extension Objects gleich und lautet *ExtensionObject*. Um festzulegen, dass es sich um eine Document Extension handelt, wird dem Knoten das Attribut *Type* mit dem Wert *document* zugewiesen. Für Extension Objects ist das Attribut optional, kann jedoch mit dem Wert *object* gesetzt werden.[6]

Das Attribut *Label* speichert die Bezeichnung von dem Extension Object. Diese taucht in der Liste der Extension Objects auf.

Für Document Extensions hat das Vergeben des Attributes *Label* jedoch keinen Zweck. In der Dokumentation gibt es keinen Hinweis darauf, für welchen Typ von Erweiterungen das Attribut *Label* verwendet werden kann und somit auch nicht darauf, dass es für Document Extensions keinen Effekt hat. Viele der im Internet öffentlich verfügbaren Document Extensions verwenden möglicherweise aus diesem Grund das Attribut dennoch, darunter die *qvAutoRefresh* Document Extension[7]. In der Liste der Document Extensions tauchen die Erweiterungen lediglich über den Dateipfad auf. In der Abbildung **??** im Anhang A ist dies zu sehen. Statt dem Wert *Inhalt Label* des Attributes *Label* wird hier der Name des Ordners *DocumentExtensionExample* angezeigt. In der Erweiterung *DocumentExtensionWithoutLabel* wurde das Attribut *Label* nicht gesetzt und taucht dennoch in der Liste auf.

Es ist auch möglich das Attribut *Label* für Extension Objects zu entfernen, ohne eine Fehlermeldung erwarten zu müssen. Ist das Attribut nicht auffindbar, so wird bei der Bezeichnung statt des Inhalts dieses Attributs wiederum der Ordnername angezeigt.

In dem Attribut *Description* kann ein Beschreibungstext hinterlegt werden. Er taucht für eine Document Extension in dem Dialog für die Eigenschaften des Dokuments im Reiter *Erweiterungen* in dem Textfeld *Beschreibung der Erweiterung:* auf, wenn die Erweiterung in der Liste angewählt wurde. Für ein Extension Object steht der Beschreibungstext in der Kurzinfo. Die Kurzinfo wird angezeigt, wenn der Mauszeiger in der Liste der verfügbaren Extension Objects über die gewünschte Erweiterung bewegt wird.

Alternativ kann auch das Attribut *Path* zugewiesen werden. Der dort eingetragene Wert wird in der Kurzinfo für die Extension unter *Name* angezeigt. Das Setzen des Attributes ist jedoch nicht notwendig, da hier der Name des Ordners, in dem die Extension gespeichert ist, eingetragen werden muss. Stimmt der Wert des Attributes nicht exakt mit dem Ordnernamen überein, so wird die Extension nicht in der Liste der verfügbaren Extensions auftauchen. Ist das Attribut einfach nicht vorhanden, so taucht der Ordnername ganz automatisch in dem Feld *Name* der Kurzinfo auf. Die weniger redundante und sichere Methode ist es also, das Attribut bei der Erstellung der Datei einfach auszulassen.

6. Vgl. QlikTech, *QlikView Extension Defintion file*
7. Vgl. Walther, *qvAutoRefresh/Definition.xml at master · QlikDev/qvAutoRefresh · GitHub*

Properties.qvpp Ab der Version QlikView 11 ist es möglich, das Erscheinungsbild des Dialoges für die Konfiguration des Extension Objects nach den Wünschen des Anwenders anzupassen. Zu diesem Zweck wird die automatische Generierung des Dialoges deaktiviert, wenn eine *Properties.qvpp*-Datei im Ordner des Extension Objects vorhanden ist. In der Datei wird mittels HTML und CSS die visuelle Darstellung des Dialoges und seiner Eingabefelder beschrieben.[8]

Für Document Extensions gibt es keinen Dialog für die Konfiguration, daher kann die *Properties.qvpp*-Datei für diesen Anwendungsfall ignoriert werden.

Script.js In der *Script.js*-Datei befindet sich der JavaScript-Code, der das Verhalten des Extension Objects bzw. der Document Extension steuert. Dies erfolgt durch Aufrufen von Funktionen der QlikView Ajax API. Sie ermöglichen die Interaktion der Erweiterung mit der QlikView Engine. So wird beispielsweise die Funktion *AddExtension* verwendet, um ein Extension Object zu registrieren. Ihr werden der Name der Erweiterung sowie eine Funktion übergeben.[9] Nach der Registrierung wird diese Funktion unter anderem immer dann aufgerufen, wenn die Erweiterung auf dem Sheet positioniert wird oder die Engine eine Veränderung der Selektion durch den Benutzer erkennt.

Icon.png Die Icon.png-Bilddatei repräsentiert das Vorschaubild der Extension. Es ist nur für Object Extensions verfügbar, da es für Document Extensions keinen Auswahldialog gibt, in dem Vorschaubilder angezeigt werden. Das Bild kann eine Auflösung von 24×24 Pixeln haben. Es wird in der Liste der verfügbaren Extension Objects angezeigt. Sollte die Datei nicht in demselben Ordner zu finden sein, in dem sich auch die Extension befindet, so wird statt des Vorschaubildes ein Fehlerhinweis angezeigt. Dies ist in der Abbildung **??** zu sehen.

1.2.2 Installation von Extension Objects und Document Extensions

Extension Objects und Document Extensions lassen sich auf die gleiche Art und Weise installieren. Für den QlikView Desktop Client gibt es dafür zwei Möglichkeiten, für die Installation auf dem QlikView Server gibt es eine. Die Installationsmethode, die für beide Technologien anwendbar ist, funktioniert folgendermaßen: Der Ordner, welcher den Namen des Extension Objects bzw. der Document Extension trägt und die benötigten Dateien enthält, wird dazu in ein spezielles Verzeichnis kopiert. Es befindet sich für den QlikView Desktop Client und für den QlikView Server an jeweils unterschiedlichen Orten. Für den QlikView Desktop Client ist der Pfad *%UserProfile%\AppData\Local\QlikTech\QlikView\Extensions\Objects*.[10]

Sollte der Benutzername der Windows-Anmeldung beispielsweise *user1* und der Datenträger, auf dem die Daten des Benutzers gespeichert sind, das Laufwerk *C* sein, so wäre der Pfad *C:\Users\user1\AppData\Local\QlikTech\QlikView\Extensions\Objects*.

8. Vgl. QlikTech, *QlikView Properties Pages*
9. Vgl. QlikTech, *QlikView Ajax JavaScript Library - AddExtension Function*
10. Vgl. Redmond, *QlikView for Developers Cookbook*, S. 164

Für den QlikView Server ist der Pfad *%ProgramData%\QlikTech\QlikViewServer\Extensions\Objects*.[11]

Sollte sich der Ordner *ProgramData* auf dem Laufwerk *C* befinden, so ist der absolute Pfad *C:\ProgramData\QlikTech\QlikViewServer\Extensions\Objects*.

Nach dem Kopieren des Ordners ist die Installation bereits vollzogen.

Für den QlikView Desktop Client gibt es eine weitere Möglichkeit, in der die Lokalisation des Ordners für die Erweiterungen erspart bleibt. Diese Option ist besonders für die Bereitstellung auf dem Zielrechner geeignet, da die Installation sehr einfach auch durch einen Laien mit einem Doppelklick erfolgen kann. Dazu wird der Ordner, der die Daten der Erweiterung enthält, in ein ZIP-Archiv gespeichert. Anschließend wird die Dateiendung *.zip* durch *.qar* ersetzt. Sollte der QlikView Desktop auf dem Zielrechner verfügbar sein, muss die Datei auf dem Rechner lediglich ausgeführt werden, beispielsweise durch einen Doppelklick. Dabei werden die Ordner der Erweiterungen völlig automatisch in den korrekten Pfad kopiert.

1.2.3 Nachteile von Extension Objects gegenüber Sheet Objects

Einige der Funktionen, die in QlikView für die Sheet Objects zur Verfügung stehen, sind nicht für Extension Objects verfügbar. Eine sehr nützliche Funktion in QlikView sind die Dimension Limits. Sie sind in den Eigenschaften des Diagramms zu finden, welche sich über einen Rechtsklick auf das Diagramm und der Auswahl des Eintrages *Eigenschaften...* aufrufen lassen. In der deutschen Version sind die Dimension Limits unter dem Reiter *Beschränkung* zu finden. Die Dimension Limits erlauben die anzuzeigenden Daten zu beschränken und bieten sich vor allem dann an, wenn die Anzahl der Daten das Diagramm unübersichtlich werden lässt. Der Benutzer kann die anzuzeigenden Werte auf eine fixe Anzahl beschränken oder alternativ nur Werte anzeigen lassen, die eine vom Benutzer definierte Bedingung erfüllen. Alle sonstigen Daten können auf Wunsch unter einer beliebigen Bezeichnung zusammengefasst werden. Für Extension Objects in QlikView gibt es eine solche Funktion zur Begrenzung von Werten nicht. Lediglich durch eine eigene Implementation im Skript des Extension Objects kann eine solche Funktionalität hinzugefügt werden.

1.2.4 Erschwerte Fehlersuche durch das Fehlen von Entwicklerwerkzeugen

Der QlikView Desktop Client hat ausschließlich die Möglichkeit über den sogenannten *WebView* den Sheet in einer webbasierten Ansicht anzuzeigen. In der herkömmlichen Anzeige werden Extension Objects nicht dargestellt.[12]

Der integrierte *WebView* hat keine Entwicklerwerkzeuge und nicht einmal eine JavaScript-Konsole ist vorhanden. Das Einsehen von Log- und Fehler-Ausgaben des Skriptes ist daher nicht möglich. Lediglich ein Extension Object mit dem Namen *QvConsole* kann verwendet werden, um eine JavaScript-Konsole im Sheet darzustellen.[13] Doch dies ersetzt nicht die

11. Vgl. Redmond, *QlikView for Developers Cookbook*, S. 164
12. Vgl. Redmond, *QlikView for Developers Cookbook*, S. 164
13. Vgl. Walther, »A Comprehensive List of QlikView Object Extensions (01/2013)«

Entwicklerwerkzeuge eines Browsers, denn das Erstellen von Haltepunkten und Einsehen von Werten der Variablen ist damit nicht möglich. Dies erschwert die Fehlersuche enorm.

Die einzige Möglichkeit ist, die Anwendung auf einem QlikView Server auszuführen. Dann lässt sich das Dokument auch im Webbrowser öffnen und somit ist auch eine Fehlersuche mithilfe der Entwicklerwerkzeuge des Browsers einfach möglich. Doch der QlikView Server ist nicht kostenfrei.[14] Außerdem erfordert es deutlich mehr Aufwand ein QlikView-Dokument auf dem QlikView Server zu öffnen als im Desktop Client. Für das Dokument muss mithilfe der *QlikView Management Console* zunächst ein sogenannter *Task* erstellt werden, um es bereitzustellen.[15]

1.2.5 Auswirkungen auf die Performance

Wird eine QlikView-Anwendung mit dem Browser geöffnet, werden die anzuzeigenden Diagramme von dem QlikView Server berechnet und als PNG-Datei an den Client übertragen. Damit liegt nicht nur der Großteil der Rechenlast auf dem Server, sondern auch die Auslastung des Netzwerkes ist erhöht. Zudem muss jede Interaktion des Nutzers an den Server kommuniziert werden. Dazu gehört auch das Bewegen des Mauszeigers über einen Bereich im Diagramm. Dabei wird die Koordinate der Maus relativ zum Diagramm an den Server geschickt. Der Server berechnet, in welchem Bereich des Diagramms die Maus positioniert wurde und sendet die dazugehörigen Informationen an den Client zurück. Auf diese Weise werden in QlikView die Kurzinfos realisiert.

Extension Objects sowie Document Extensions verhalten sich anders. Sie werden mithilfe von Webtechnologien erstellt, die auf dem Webbrowser des Clients zur Ausführung gebracht werden. Die Rechenleistung wird damit auf dem Rechner des Clients beansprucht. Die Interaktion des Nutzers mit einem Extension Object wird nicht mehr zum Server übertragen, sondern kann direkt durch das dem Extension Object beigelegten Skript verarbeitet werden. Die anzuzeigenden Visualisierungen können mithilfe von dafür konzipierten Bibliotheken wie beispielsweise D3 erstellt und müssen so nicht über das Netzwerk übertragen werden. Ausschließlich die anzuzeigenden Daten, die durch das Extension Object dargestellt werden sollen, müssen an den Client gesendet werden.

Fazit Extension Objects und Document Extensions haben keinen negativen Einfluss auf die Ausführungsgeschwindigkeit des QlikView Servers, da lediglich die Rechenleistung des Clientrechners beansprucht wird. Darüber hinaus kann davon ausgegangen werden, dass Extension Objects die Ausführungsgeschwindigkeit sogar positiv beeinflussen können, wenn sie das Zeichnen der Diagramme auf dem Client Rechner ausführen, somit den Server entlasten und die Menge der über das Netzwerk zu übertragenden Daten verringern.

14. Vgl. Redmond, *QlikView Server and Publisher*, S. 16 f.
15. Vgl. Redmond, *QlikView Server and Publisher*, S. 111 ff.

1.3 Extension Objects in Qlik Sense

Die Integration von Extension Objects in Qlik Sense ist deutlich besser, als dies noch in QlikView der Fall war. Qlik Sense verwendet die Technologien AngularJS und RequireJS, was die Entwicklung von Erweiterungen deutlich vereinfacht.[16] Dieses Kapitel beschreibt, welche Vorteile der neue webbasierte Ansatz bietet, welche Struktur Extension Objects in Qlik Sense haben und wie sie sich von den Sheet Objects unterscheiden. Weiterhin wird die Erstellung bzw. Installation von Extension Objects über das Dateisystem bzw. über die Qlik Sense Workbench erläutert.

1.3.1 Webbasierte Anzeige und Entwicklerwerkzeuge

Wie in Kapitel 1.2.4 beschrieben, ist in QlikView die Fehlersuche durch das Fehlen von Entwicklerwerkzeugen im *WebView* deutlich erschwert.

In Qlik Sense ist dies nicht mehr der Fall, denn die Anzeige im QlikView Desktop Client wird über einen integrierten Chromium Browser realisiert.[17] Mit der Tastenkombination *Steuerung* + *Shift* + *Rechtsklick* im Fenster des Qlik Sense Desktop Clients öffnet sich ein Dialog, in dem über den Eintrag *Show DevTools* die Entwicklerwerkzeuge geöffnet werden können. Ist Qlik Sense gestartet, ist es jedoch auch möglich mit jedem anderen modernen Webbrowser über die Adresse `http://localhost:4848/hub/` den Qlik Sense Desktop-Hub zu öffnen.[18]

1.3.2 RequireJS

In der Vorgängerversion QlikView wurde das Anfordern von weiteren Dateien noch über herkömmliche Ajax-Anfragen über absolute Adressen realisiert.[19] Qlik Sense verwendet dagegen eine Bibliothek für die Definition von Modulen.

> Qlik Sense uses RequireJS for loading JavaScript modules [...].[20]

Die Bibliothek erlaubt es, JavaScript-Module mit ihren Abhängigkeiten von anderen Modulen zu definieren.

1.3.3 Extension Objects gleichen Sheet Objects

In QlikView bestand noch ein klarer Unterschied zwischen Extension Objects und Sheet Objects - siehe Kapitel 1.2.3. Allerdings ist es in Qlik Sense schwer, Unterschiede zwischen diesen beiden Kategorien festzustellen. Ein leicht zu erkennender Unterschied ist jedoch, dass

16. Vgl. QlikTech, *Third Party Software Attributions, Copyrights, Licenses and Disclosure - Qlik® Sense*
17. Vgl. QlikTech, *Qlik Sense Architecture*, o. S.
18. Vgl. QlikTech, *Qlik Sense Architecture*, o. S.
19. Vgl. QlikTech, *QlikView Ajax JavaScript Library - LoadExtensionScripts*
20. QlikTech, *Qlik Sense Workbench requirements and assumptions*

die herkömmlichen Sheet Objects nicht im Dateiordner zu finden sind, in dem die Extension Objects abgelegt werden.

Während Qlik Sense ausgeführt wird und ein Arbeitsblatt im Webbrowser geöffnet ist, kann in der JavaScript-Konsole über den Ausdruck *requirejs.s.contexts._.defined* eine Liste aller definierten Module ausgegeben werden.[21] In der Liste tauchen unter anderem alle Extension Objects auf, die zum aktuellen Arbeitsblatt hinzugefügt wurden. So ist zum Beispiel auch das im Kapitel 2.3 erstellte Kreisdiagramm in der Liste unter dem Eintrag *extensions/C3Kreisdiagramm/C3Kreisdiagramm* zu finden. Sofern das durch Qlik Sense mitgelieferte Kreisdiagramm dem Arbeitsblatt ebenso zugewiesen ist, lässt sich auch der Eintrag *extensions.qliktech/piechart/piearea* finden. Dies lässt vermuten, dass die mitgelieferten Sheet Objects sowie die Extension Objects die gleichen Technologien verwenden und somit kein Unterschied in ihrer Anwendung besteht. Die Nachteile der Extension Objects in QlikView, welche im Kapitel 1.2.3 erläutert wurden, existieren in Qlik Sense nicht. So ist es beispielsweise möglich, auch für Extension Objects in Qlik Sense Limitierungen für Dimensionen zu definieren.[22]

1.3.4 Qlik Sense Workbench

Ist Qlik Sense Desktop installiert und gestartet, so lässt sich über die URL `localhost:4848/workbench/` die Qlik Sense Workbench aufrufen.[23] Im Anhang C in Abbildung **??** ist ein Screenshot der Workbench zu sehen. Über die Schaltfläche *Create New* kann ein neues Extension Object oder auch ein neues Mashup - diese werden in der vorliegenden Arbeit nicht erläutert - erstellt werden. Die Workbench erzeugt mithilfe von Schablonen alle benötigten Dateien selbstständig. Der Entwickler kann die Dateien direkt im Anschluss innerhalb der Workbench editieren.

1.3.5 Qlik Sense Extension Objects auf dem Dateisystem

Wird eine Erweiterung über die Qlik Sense Workbench erstellt, so tauchen die generierten Dateien im Dateisystem unter *%UserProfile%\Documents\Qlik\Sense\Extensions* auf.[24]

Wenn der Benutzername der Windows-Anmeldung *user1* und die Daten des Benutzerkontos auf dem Laufwerk C gespeichert sind, dann ist der Pfad *C:\Users\user1\Documents\Qlik\Sense\Extensions*.

Für die Erstellung der Dateien muss nicht zwingend die Qlik Sense Workbench verwendet werden. Sie können auch unabhängig von der Workbench durch beliebige Programme erstellt und modifiziert werden.

21. Vgl. Cowart, *Five Helpful Tips When Using RequireJS - Tech.pro*
22. Vgl. O'Donovan, »Qlik Sense For Beginners«, S. 89 ff.
23. Vgl. QlikTech, *Qlik Sense Workbench*; QlikTech, *Launching Qlik Sense Workbench*; Redmond, *Qlik Tips: Extensions in Qlik Sense*
24. Vgl. Redmond, *Qlik Tips: Extensions in Qlik Sense*

1.3.6 Struktur von Qlik Sense Extension Objects

Die beiden wichtigsten Dateien sind die *.qext*-Datei und die *.js*-Datei. Als Dateinamen erhalten beide den exakten Namen des Extension Objects. Weiterhin gibt es eine *wbfolder.wbl*-Datei, welche optional ist. Darüber hinaus können beliebig viele weitere Dateien hinzugefügt werden, die für das Skript verwendet werden sollen. Die Dateien erfüllen unterschiedliche Funktionen.

.qext-Datei Der Inhalt der *.qext*-Datei ist im JSON-Format beschrieben und speichert grundlegende Informationen des Extension Objects.[25] Dazu gehören unter anderem der Name, die Beschreibung, der Typ oder aber ein Vorschaubild. In der Liste der verfügbaren Extension Objects tauchen diese Informationen entweder direkt in der Liste oder in einer Kurzinfo nach dem Klick auf einen der Einträge auf.

.js-Datei Der Haupteinstiegspunkt des Skriptes befindet sich in der Datei mit dem Namen der Erweiterung und der Endung *.js*.[26] Anders als in QlikView kann die Definition der Dimensionen, der Formeln, der Parameter und deren Eingabefelder vollständig in dieser Datei erfolgen. Alternativ können auch weitere Module angelegt werden, die das RequireJS-Skript anfordert.

wbfolder.wbl-Datei Bei der *wbfolder.wbl*-Datei handelt es sich um eine Liste der zu dieser Erweiterung gehörenden Dateien. Sie hat den Zweck, die Reiter in der Qlik Sense Workbench zu verwalten. Außerdem werden beim Duplizieren der Erweiterung innerhalb der Workbench ausschließlich die Dateien, die in dieser Liste eingetragen sind, kopiert. Alle anderen Dateien werden ignoriert.

1.4 Die Programmiersprache Dart

1.4.1 Allgemeines zur Programmiersprache Dart

Dart ist eine durch Google entwickelte Programmiersprache für die Entwicklung von komplexen und zugleich hochperformanten Webapplikationen. Die Sprache wurde entwickelt, da aus der Sicht von Google die Etablierung einer weiteren Sprache neben JavaScript für die Webentwicklung nötig war.

> JavaScript is great for small scripts, and it has the performance chops to run large apps. But when a script evolves into a large web app, debugging and modifying that app can be a nightmare, especially when you have a large team.[27]

Ziel der Programmiersprache Dart war es unter anderem, Entwicklern, die mit Sprachen wie Java und C# vertraut sind, die Programmierung für das Web zu ermöglichen. Gleichzeitig

25. Vgl. Redmond, *Qlik Tips: Extensions in Qlik Sense*
26. Vgl. Redmond, *Qlik Tips: Extensions in Qlik Sense*
27. Vgl. Walrath und Ladd, *Dart: Up and Running*, S. 1

sollte der Einstieg für JavaScript-Entwickler dadurch nicht erschwert werden. Um dem zu begegnen, ist die Verwendung von Typen in Dart optional. Wie in JavaScript gewohnt, kann eine Variable mit dem Schlüsselwort *var* deklariert werden. Gleichermaßen kann anstelle des *var*-Schlüsselwortes auch ein Typ wie zum Beispiel *String*, *int* oder *Object* stehen.[28]

Dart verfügte bereits zum Zeitpunkt der Standardisierung als ECMA-408 Standard um einige Konzepte, die aus anderen objektorientierten Programmiersprachen wie beispielsweise C++, Java und C# bekannt sind. Dazu gehören unter anderem Klassen, Interfaces, Mixins, Properties, generische Datentypen und Annotationen.[29]

Darüber hinaus äußerte die Community weitere Wünsche nach Funktionalitäten, die in der Sprache noch nicht enthalten waren. So wurden beispielsweise Enum-Typen gewünscht, wie sie in Java, C# und C++ vorkommen.[30] Ein weiterer Vorschlag war das Hinzufügen der Schlüsselwörter *async* und *await* für eine deutlich verbesserte Lesbarkeit im Umgang mit asynchroner Programmierung.[31]

Das Dart-Entwicklerteam antwortete darauf und so wurden die gewünschten Funktionalitäten der zweiten Edition des Standards bereits sechs Monate nach der initialen Standardisierung hinzugefügt.[32]

1.4.2 Kompilierbarkeit zu JavaScript

Um Dart auch auf jedem modernen Webbrowser ausführen zu können, kann der Compiler dart2js verwendet werden. Er übersetzt den Dart-Quellcode, inklusive aller Abhängigkeiten, in JavaScript. Auch nach der Kompilierung kann die Ausführungsgeschwindigkeit des generierten JavaScript-Codes schneller sein als eine äquivalente Implementierung in reinem JavaScript. Dies konnte durch Benchmarks, wie zum Beispiel dem sogenannten Tracer-Test, ermittelt werden.[33] Weitere Benchmarks sind auf folgender Webseite einsehbar: `dartlang.org/performance/`.

1.4.3 Werkzeuge

Für die Programmierung mit Dart wurde der Dart Editor geschaffen. Dabei handelt es sich um eine auf Eclipse basierende integrierte Entwicklungsumgebung. Der Editor unterstützt den Entwickler unter anderem mit statischer Quellcodeanalyse und Debuggingtools. Mithilfe von Plug-ins ist die Programmierung von Dart-Anwendungen auch innerhalb von anderen Entwicklungsumgebungen möglich. Es existieren Plug-ins für IntelliJ IDEA, WebStorm, Sublime Text 3, Emacs und Vim.[34]

28. Vgl. Buckett, *Dart in Action*, S. 5
29. Vgl. ECMA International, *Dart Programming Language Specification*, o. S.
30. Vgl. *Issue 88 - dart - Enhancement: Enum - Dart - Structured Web Programming*
31. Vgl. *Issue 104 - dart - Support for await in Dart - Dart - Structured Web Programming*
32. Vgl. ECMA International, *Dart Programming Language Specification*, S. 43, 88
33. Vgl. Belchin und Juberias, *Web Programming with Dart*, S. 2 f
34. Vgl. Akopkokhyants, *Mastering Dart*, o. S.

1.4.4 Bibliotheken in Dart

Dart ist nicht nur eine Programmiersprache, sondern vielmehr eine Plattform. Dart wird bereits mit einer Reihe von integrierten Bibliotheken ausgeliefert. Kollektionen, reguläre Ausdrücke, mathematische Operationen und die Interaktion mit dem DOM sind nur einige von den vielen Funktionen, die durch diese Bibliotheken abgedeckt werden.[35]

Sollte dennoch eine Bibliothek benötigt werden, so kann sie über den Paketmanager Pub angefordert werden. Er ermöglicht es, Abhängigkeiten zu anderen Bibliotheken hinzuzufügen. Welche aus unterschiedlichen Quellen stammen können. Google selbst bietet einen Service an, solche sogenannten packages auf der Seite `pub.dartlang.org` zu hinterlegen. Alternativ kann auch ein git repository gewählt werden und natürlich lassen sich auch packages im lokalen Dateisystem ablegen und einbinden. Dart wird mit einem Kommandozeilenprogramm ausgeliefert, welches ebenfalls den Namen *pub* trägt. Jedes Dart-Projekt enthält eine Datei mit dem Dateinamen *pubspec.yaml*. Darin werden alle Abhängigkeiten gelistet, die durch den Paketmanager verwaltet werden sollen. Über den Befehl *pub get* werden alle eingetragenen Abhängigkeiten des Projektes aufgelöst und gegebenenfalls heruntergeladen. Dabei werden auch die Abhängigkeiten der angeforderten packages ermittelt und ebenfalls hinzugefügt.[36]

1.4.5 Build Tools

Die Automatisierung immer wiederkehrender Aufgaben beim Entwicklungsprozess kann eine Zeit sparende Erleichterung sein. Die Anwendungsgebiete sind vielfältig. Beispiele sind das Durchlaufen von Modultests, die automatische Kompilierung von Sass-Quelldateien in CSS-Code oder die Verringerung der Dateigröße der generierten Dateien.[37] Doch es kann sich auch einfach um das Kopieren der generierten Dateien auf einen entfernten Server handeln. Dieses Szenario wird auch bei der Entwicklung der QlikView und Qlik Sense Erweiterungen mithilfe von Dart eine Rolle spielen. Für JavaScript existiert ein sogenanntes build tool mit dem Namen Grunt.[38] Doch um dieses build tool zu verwenden, müssen Node.js und anschließend unterschiedliche Node.js packages installiert werden.[39]

Dart bietet dagegen die sogenannten Pub Transformer als ein build tool an, welches ohne weitere Installation verwendet werden kann. Die gewünschten Transformer werden in der *pubspec.yaml*-Datei angegeben und bei jedem Build-Prozess durch den Befehl *pub build* automatisch angestoßen.[40]

In den Kapiteln 3.2 und 3.3 ist die Anwendung der Pub Transformer zu sehen.

35. Vgl. Walrath und Ladd, *Dart: Up and Running*, S. 4
36. Vgl. Kopec, *Dart for Absolute Beginners*, S. 183
37. Vgl. Odell, *Pro JavaScript Development: Coding, Capabilities, and Tooling*, S. 391
38. Vgl. Odell, *Pro JavaScript Development: Coding, Capabilities, and Tooling*, S. 392
39. Vgl. Odell, *Pro JavaScript Development: Coding, Capabilities, and Tooling*, S. 393
40. Vgl. *Assets and Transformers | Dart: Structured web apps*

2 Vorbereitende Entwicklung von Erweiterungen mit JavaScript

In diesem Kapitel werden ein QlikView Extension Object und eine QlikView Document Extension sowie ein Qlik Sense Extension Object mit JavaScript erstellt. Es dient Entwicklern, die Extension Objects bzw. Document Extensions in JavaScript oder auch Dart entwickeln möchten, gleichermaßen, denn viele der Schritte sind identisch. Lediglich bei der Erstellung des Skriptes gibt es Unterschiede. Daher sollten auch Entwickler, die in die Extension Entwicklung mit Dart einsteigen möchten, sich über die Grundlagen der Entwicklung von Extension Objects und Document Extensions in diesem Kapitel vertraut machen.

2.1 Leitfaden zur Erstellung eines QlikView Extension Objects mit JavaScript

In diesem Kapitel wird die Entwicklung eines Extension Objects am Beispiel eines Kreisdiagramms mit JavaScript erklärt. Um das Beispiel möglichst knappzuhalten und dennoch auf die wichtigen Details bei der Extension Erstellung einzugehen, wurde die Bibliothek C3 verwendet. Sie verwendet im Kern D3, welcher eine sehr flexible und funktionsreiche Bibliothek zur Visualisierung von Daten ist. C3 bietet eine Reihe von wiederverwendbaren Diagrammen, die mit einem verhältnismäßig geringen Entwicklungsaufwand verwendet und konfiguriert werden können.[41]

2.1.1 Besonderheiten

In der Mitte der Abbildung **??** auf Seite **??** ist das Kreisdiagramm in einem QlikView-Dokument dargestellt. Die selektierten Daten der links im Bild befindlichen ListBox werden durch das Kreisdiagramm visualisiert. Rechts im Bild ist zum Vergleich ein durch QlikView mitgeliefertes Kreisdiagramm zu sehen. Das Dokument wird im Browser mit einem angewendeten Zoom dargestellt. Das von QlikView mitgelieferte Kreisdiagramm erscheint durch die erkennbaren Rasterpunkte weniger ästhetisch. Die Darstellung des C3 Kreisdiagramms wird nicht verschlechtert, da es sich um eine Vektorgrafik handelt.

2.1.2 Erstellen der Dateistruktur

Das Extension Object soll den Namen C3Kreisdiagramm erhalten. Daher erhält der Dateiordner genau diesen Ordnernamen, da er den Namen des Extension Objects festlegt. Alle im Rahmen des Beispiels erstellten Dateien werden in diesem Ordner gespeichert. Weiterhin müssen die benötigten Dateien der Bibliotheken in diesen Ordner heruntergeladen werden. Die Dateien sind *d3.min.js* [42], *c3.min.js* [43] und *c3.min.css* [44].

41. Vgl. *C3.js | D3-based reusable chart library*
42. https://github.com/mbostock/d3/blob/master/d3.min.js
43. https://github.com/masayuki0812/c3/blob/master/c3.min.js
44. https://github.com/masayuki0812/c3/blob/master/c3.min.css

2.1.3 Erstellen eines QlikView-Testdokumentes

Zur Visualisierung werden Daten benötigt. Weiterhin werden die Extension Objects bzw. die Document Extensions einem QlikView-Dokument zugewiesen. Daher muss zunächst ein Dokument im QlikView Desktop Client angefertigt werden. Für dieses Beispiel werden im QlikView load script Daten zum Test erstellt. Das Listing **??** wird am Ende des Skriptes eingefügt. Es dient der Erzeugung von drei Datensätzen, die für die Visualisierung verwendet werden können.

2.1.4 Hinzufügen eines CSS-Stylesheet

Für die korrekte Darstellung des Kreisdiagramms wird eine CSS-Anweisung benötigt. Sie ist im Listing **??** zu sehen und wird als Datei mit dem Namen *style.css* im Ordner der Erweiterung abgelegt. Sie sorgt dafür, dass die generierte Vektorgrafik korrekt im zu zeichnenden Bereich erstellt wird. Fehlt die Anweisung, ist die generierte Vektorgrafik größer als der Bereich, welcher für die Erweiterung vorgesehen ist.

2.1.5 Erstellung der Definition.xml-Datei

In dem Listing **??** ist die *Definition.xml*-Datei für das Extension Object zu sehen.

Dimensionen festlegen Das Kreisdiagramm soll genau eine Dimension visualisieren, in diesem Beispiel die Dimension *Produkt*. In Zeile 6 wird die Dimension erstellt. Dafür kann keine Id festgelegt werden, denn die Dimensionen werden nach ihrer Reihenfolge des Auftretens in dieser Datei durchnummeriert. Die erste Dimension erhält deshalb die Referenz *Chart.Dimension.0*, während eine weitere Deklaration einer Dimension die Referenz *Chart.Dimension.1* erhalten würde.

Der Knoten Dimension enthält das Attribut *Initial*, welches gesetzt werden muss. Ist das Attribut nicht auffindbar, so wird die Dimension nicht deklariert und somit ist sie in der Extension nicht verfügbar. Allerdings ist es aber auch unerheblich, welcher Wert dem Attribut hinzugefügt wird. Die für diese Arbeit durchgeführten Tests zeigten, dass dieses Attribut den Wert des Feldes nicht initialisiert. Es reicht aus, das Attribut zu deklarieren und ihm eine leere Zeichenkette als Wert zu übergeben. Das Initialisieren des Wertes der Dimension ist jedoch möglich und wird in der Zeile 13 mit dem Wert *Produkt* vorgenommen.

Formeln festlegen In Zeile 8 erfolgt die Deklaration der Formel. Auch hier ist das Festlegen einer Id nicht möglich, sondern wird genau wie bei den Dimensionen über die Reihenfolge der Deklarationen in der Datei bestimmt. Allerdings gibt es hier einen Unterschied in der Syntax der Referenzierung. Die in der Zeile definierte Formel erhält die Referenz *Chart.Expression.0.0*, während eine weitere Formel die Referenz *Chart.Expression.1.0* erhalten würde. Die zweite Nummer scheint bei der Referenz immer eine *0* zu sein. In Zeile 15 wird die Formel über die

korrekte Referenz initialisiert. Zu jedem Datensatz soll die dazugehörige *Menge* angezeigt werden.

Parameter festlegen Ob die Legende des Diagramms angezeigt wird, soll über einen Parameter, der in Zeile 10 definiert wird, konfigurierbar sein. Das Attribut *Label* ist nur von Bedeutung, wenn der Konfigurationsdialog automatisch generiert wird. Während der Entwicklung der Extension wird jedoch ein eigener Konfigurationsdialog erstellt und somit ist das Attribut bedeutungslos. Das Attribut *Type* legt fest, welche Art von Eingabefeld für den Parameter angewendet werden soll. In diesem Beispiel ist es *checkbox*, da es sich um die Deklaration eines Wahrheitswertes handelt.

Darüber hinaus gibt es weitere Typen von Eingabefeldern. Für Zeichenketten erhält das Attribut den Wert *text*, für Farben den Wert *color* und für eine Dropdown-Liste den Wert *select*.[45]

Allerdings muss darauf hingewiesen werden, dass die Auswahlbox fehlerhaft ist. Damit die Dropdown-Liste verwendet werden kann, muss zusätzlicher Code zur JavaScript-Datei der Extension hinzugefügt werden.[46]

Genau wie bei den Dimensionen und Formeln wird die Identifikation der Parameter über die Reihenfolge ihrer Definition in der Datei bestimmt. Die Referenz des ersten Parameters lautet *Chart.Text.0*, während die des zweiten Parameters *Chart.Text.1* wäre. In Zeile 17 wird der Wahrheitswert des Parameters mit der Nummer 0 - für unwahr - initialisiert.

Das Festlegen eines Bezeichners für die Parameter wäre wünschenswert. Wird nämlich der erste Parameter entfernt, so werden auch die Identifikationsnummern wieder neu durchnummeriert. Wenn das passiert, müssen alle Parameter im Konfigurationsdialog, welcher in der *Properties.qvpp*-Datei beschrieben wird, erneut verknüpft werden. Dort erhält der Parameter, welcher zuvor an zweiter Stelle war und nun auf die erste Stelle aufgerückt ist, die Referenz *Chart.Text.0* statt vorher *Chart.Text.1*. Ebenso müssen die Aufrufe der Parameter in der *Script.js*-Datei angepasst werden. In dieser Datei erhält der Parameter, der zuvor der zweite Parameter war, nun die Referenz *Layout.Text0* statt vorher *Layout.Text1*. Dies wird mit einer zunehmenden Anzahl von Parametern immer mühsamer und verleitet dazu, so wenig Parameter wie möglich zu integrieren oder beim Löschen eines Parameters in der *Definition.xml*-Datei die Lücke der Identifikationsnummern durch einen Parameter zu ersetzen, der keine Funktion hat.

2.1.6 Properties.qvpp

Listing **??** auf Seite **??** ist ein Ausschnitt der *Properties.qvpp*-Datei. Die komplette Datei lässt sich im Listing **??** in Anhang B einsehen. Im Anhang A ist der Konfigurationsdialog in seiner Darstellung im Browser in der Abbildung **??** zu sehen.

45. Vgl. QlikTech, *QlikView Extension Defintion file*
46. Vgl. QlikTech, *HTML Select in Extensions | Qlik Community*

In dieser Datei wird der Konfigurationsdialog des Kreisdiagramms beschrieben. Durch einige bereits vordefinierte CSS-Klassen wird das Aussehen des Dialoges umgesetzt. Die CSS-Klassen realisieren das Einpassen der Eingabefelder und Labels in ein Gestaltungsraster. Unterschiedliche Kombinationen von Eingabefeldern mit Labels erfordern unterschiedliche Ausmaße, um im Gestaltungsraster eingebettet werden zu können. Eine einfache Möglichkeit ist es, sich an den *Properties.qvpp*-Dateien anderer QlikView Extension Objects zu orientieren, da das Erstellen sonst sehr mühselig werden könnte.

Das Attribut *avq* bestimmt das Verhalten des jeweiligen Elementes für die Erweiterung. In dem Attribut wird der Typ des Elementes beschrieben. Ist weiterhin noch eine Angabe einer Dimension, einer Formel oder eines Parameters nötig, so erfolgt sie durch einen Doppelpunkt getrennt nach dem Typ des Elements. Die restlichen Attribute dienen im Wesentlichen der Darstellung des Dialoges.

In Zeile 5 wird die Überschrift des ersten Reiters mit *C3 Kreisdiagramm - Data Options* gesetzt. In den Zeilen 11-14, 23-26 und 37-40 werden Labels für die Dimension, die Formel und für das Anzeigen der Legende erstellt. Die Labels der Dimension und der Formel sind jeweils vor dem Eingabefeld eingefügt, während das Label für die Darstellung der Legende nach dem Eingabefeld erscheint. Für die Labels hat das Attribut *avq* den Wert prop_label.

In den Zeilen 16-17 wird für die Dimension das Eingabefeld definiert. Das Attribut *avq* erhält hier den Wert *prop_dynamicDropdown:.Chart.Dimension.0.Field*. Es handelt sich um ein Eingabefeld mit einer automatischen Vervollständigungsfunktion. Im Anhang A ist in Abbildung **??** ein Screenshot davon zu sehen. Wird in diesem Eingabefeld etwas eingegeben, werden mögliche Dimensionen des Datenmodells vorgeschlagen. Dafür ist der erste Teil vor dem Doppelpunkt *prop_dynamicDropdown* verantwortlich. Nach dem Doppelpunkt ist die gewünschte Dimension anzugeben, in diesem Fall *.Chart.Dimension.0.Field*.

In den Zeilen 18-20 wird vor dem Eingabefeld noch eine Schaltfläche hinzugefügt, die einen Dialog für zusätzliche Optionen der Dimension bereitstellt. Darüber lässt sich unter anderem die Sortierung konfigurieren. Ein Screenshot des Dialogs ist im Anhang A in der Abbildung **??** zu sehen.

Das Eingabefeld für die Formel wird in den Zeilen 28-30 beschrieben. Dort wird das Attribut *avq* mit dem Wert *prop_editexpression:.Chart.Expression.0.0.Definition* gefüllt. Wie bereits bei der Initialisierung der Formeln in der *Definition.xml*-Datei ändert sich bei dem Zugriff auf weitere Formeln lediglich die Zweite der beiden Ziffern in diesem Ausdruck.

Die Checkbox für den Wahrheitswert zum Anzeigen der Legende wird in den Zeilen 35 und 36 erstellt. Der Wert des Attributes *avq* ist hierfür *prop_checkbox:.Chart.Text.0.Content*.

2.1.7 Erstellung der Script.js-Datei

Im Listing **??** auf Seite **??** ist das Skript für das Kreisdiagramm zu sehen.

In Zeile 1 wird eine Variable mit dem Namen der Extension gespeichert, da er sich in dem Skript wiederholt. Änderungen an dem Namen sollten nur hier durchgeführt werden.

Anfordern von weiteren Dateien Bei der Entwicklung von QlikView Erweiterungen ist das Nachladen von weiteren Dateien nicht sehr intuitiv gelöst. Es ist nicht möglich, über eine relative Adresse - beispielsweise eine Datei im selben Ordner - auf Dateien zuzugreifen. Die Dateien müssen über eine absolute Adresse angefordert werden. Der Adresse sind Parameter zur Identifikation der Erweiterung zu übergeben.

In Zeile 3 und 4 wird die absolute Adresse erstellt. Zunächst wird die absolute Adresse des Servers über die API-Funktion *Qva.Remote* ausgelesen. Anschließend wird festgestellt, ob die Adresse bereits das Fragezeichen als Trennsymbol für die Parameterliste enthält oder ob es noch hinzugefügt werden muss. Anschließend werden die Parameter übertragen. Der Parameter *public* enthält standardmäßig den Wert *only* und der Parameter *name* erhält die Zeichenkette *Extensions/* gefolgt von dem Namen der Erweiterung. Über die Adresse ist es möglich, auf die Dateien, die in dem Ordner der Erweiterung liegen, zuzugreifen.

In den Zeilen 5-8 werden CSS-Dateien einem Array von absoluten Adressen hinzugefügt und anschließend in einer Schleife nacheinander angefordert. Die QlikView Ajax API stellt dafür die Funktion *Qva.LoadCSS* bereit.

In den Zeilen 9 und 10 erfolgt nun Gleiches mit den JavaScript-Dateien mit dem Unterschied, dass hierfür die API-Funktion *Qv.LoadExtensionScripts* verwendet wird und dass sie als zweiten Parameter eine Callback-Funktion verlangt. Da die Funktionen in dieser Erweiterung auf die Bibliotheken angewiesen sind, ist es nötig, auf das Laden der Skripte zu warten. Daher müssen alle Anweisungen, die auf die Funktionen der angeforderten Bibliotheken zurückgreifen, innerhalb der übergebenen Funktion erfolgen.

Funktionsobjekt des Extension Objects Der Name der Extension wird in Zeile 11 der Funktion *AddExtension* der *QlikView Ajax API* zusammen mit einem Funktionsobjekt übergeben. Hier ist besonders wichtig darauf zu achten, dass der hier übergebene Name mit dem Namen der Extension übereinstimmt. Die hier übergebene Zeichenkette muss also dem Namen des Dateiordners entsprechen, in dem die Extension gespeichert ist.

Das der Funktion *AddExtension* als zweites Argument übergebene Funktionsobjekt wird bei jedem Erstellen und bei jedem Zeichnen des Extension Objects aufgerufen. Alle Anweisungen außerhalb dieser Funktion werden nur einmalig ausgeführt, nämlich das erste Mal, wenn das Skript angefordert wird. Dies ist der Fall, wenn die erste Instanz der Erweiterung entweder auf einem Arbeitsblatt gefunden oder durch Platzieren auf der Arbeitsfläche erstellt wurde. Bei jedem weiteren Auffinden bzw. Erstellen einer neuen Instanz der Erweiterung werden die Anweisungen außerhalb der in Zeile 11 übergebenen Funktion nicht erneut ausgeführt. In diesem Fall wird nun immer diese übergebene Funktion aufgerufen. Befindet sich also bereits eine Instanz der Erweiterung auf dem Arbeitsblatt und wird eine neue hinzugefügt, so wird diese Funktion für beide Instanzen erneut aufgerufen. Gleiches gilt für Veränderungen an anderen Objekten auf dem Arbeitsblatt oder für die Selektionen von Daten.

Das Funktionsobjekt enthält einige Attribute, die für die Visualisierung der Daten benötigt werden. So sind in diesem Funktionsobjekt beispielsweise bei jedem Zeichnen die übergebenen Daten und ebenfalls die definierten Parameter mit ihren Werten enthalten. Zugriff auf die

Attribute wird über die Referenz *this* gewährt. In Zeile 12 wird das Funktionsobjekt über die Referenz *this* in eine lokale Variable mit dem Namen *qvaWrapper* gespeichert. Das Objekt selbst trägt keinen Namen und wurde daher vom Autor der vorliegenden Arbeit *qvaWrapper* genannt, da es sich vom Prototyp *Qva.Public.Wrapper* ableitet. Entwickler anderer Extension Objects speichern das Funktionsobjekt unter anderem unter dem Bezeichner *_this* ab.[47] Das Zwischenspeichern dieser Referenz ist wichtig, da das Definieren einer neuen Funktion die Referenz *this* innerhalb des Funktionskörpers verdecken würde.

Jede Instanz eines Extension Objects ist über eine sogenannte ObjectId eindeutig identifizierbar. Die Id wird in Zeile 13 ausgelesen und in Zeile 14 verwendet, um das Funktionsobjekt auch als globale Variable zu speichern. Doch die ObjectId enthält einen Rückschrägstrich, der innerhalb eines Bezeichners für ein Attribut unzulässig ist. Daher wird das Sonderzeichen in Zeile 13 entfernt. Bereits jetzt hat der Entwickler die Möglichkeit in den Entwicklerwerkzeugen seines Webbrowsers das Objekt zu inspizieren und somit einen Überblick über die Funktionsweise von QlikView Extensions zu bekommen. Da die Dokumentation für die Entwicklung von Extension Objects und Document Extensions für QlikView sehr rar ist, ist dies eine hervorragende Methode, um Informationen über die API zu erhalten. Diese Entwicklerwerkzeuge stehen allerdings nur dann zur Verfügung, wenn ein QlikView Server vorhanden ist. Allein mit dem QlikView Desktop Client ist dies nicht möglich, da der WebView des Desktop Clients nicht durch andere Webbrowser aufgerufen werden kann und der WebView selbst keine Entwicklerwerkzeuge zur Verfügung stellt. Dies ändert sich allerdings in Qlik Sense, denn bei Starten des Qlik Sense Desktop Clients wird ein lokaler Server gestartet, der sich auch von anderen Webbrowsern aufrufen lässt.

Inspizieren des Funktionsobjektes eines Extension Objects Wurde auf dem QlikView Server das Extension Object installiert und in einem Dokument zu einem Arbeitsblatt hinzugefügt, so kann in der JavaScript-Konsole nach dem Objekt gesucht werden. In der Abbildung **??** ist zu sehen, dass das Funktionsobjekt durch automatische Vervollständigung sehr einfach gefunden werden kann. Die automatische Vervollständigung ist sehr nützlich, da die eindeutige *ObjectId* nicht zwingend bekannt ist.

Wenn der Bezeichner des Funktionsobjektes richtig vervollständigt wurde, kann nach Bestätigung der Eingabe das Objekt inspiziert werden. Dies ist in Abbildung **??** auf Seite **??** zu sehen.

Inspizieren der übermittelten Daten Ein weiterer Blick in das Funktionsobjekt der Erweiterung mithilfe der Entwicklertools eines Webbrowsers gibt einen Überblick, wie die Daten der Erweiterung strukturiert sind. In der Abbildung **??** sind drei Datensätze zu sehen, die der Erweiterung zur Visualisierung bereitgestellt wurden. Innerhalb des Funktionsobjektes der Erweiterung befinden sich die Daten im Attribut *Data*. Darin ist unter anderem das Attribut *Rows* gespeichert, bei dem es sich um ein Array handelt, in dem jedes Element einen Datensatz darstellt. Bei jedem dieser Datensätze handelt es sich wiederum um ein Array,

47. Vgl. Walther, *qvD3BulletCharts/Script.js at master · QlikDev/qvD3BulletCharts · GitHub*

in dem das erste Element die Dimension und das zweite Element den dazugehörigen Wert beschreibt.

Das Element der Dimension enthält unter anderem das Attribut *text*, welches den anzuzeigenden Namen der Dimension repräsentiert, und das Attribut *value*, welches nicht den Wert, aber die Identifikationsnummer des Datensatzes speichert. Die Id ist vor allem für die Selektion der Daten wichtig.

Das Element des Wertes enthält die Attribute *data* und *text*, welche beide den gleichen Wert, aber mit einem unterschiedlichen Datentyp bereitstellen.

In den Zeilen 16-28 werden die Daten in eine Datenstruktur überführt, die von der Bibliothek C3.js verstanden wird. In Zeile 16 wird ein Array definiert, dem die Datensätze übergeben werden sollen. In den Zeilen 25 und 26 werden die Datensätze in Form von einem weiteren Array übergeben. C3.js interpretiert das erste Element in diesen Arrays als den Bezeichner und alle weiteren Elemente als Werte, die für das Kreisdiagramm aufsummiert werden würden.[48] Da jedoch lediglich ein Wert als zweites Element hinzugefügt wird, ist eine Summierung nicht nötig.

Das Übergeben der Identifikationsnummer als Bezeichner der Spalte ist nötig, da die Selektion der Daten über den Namen nicht ausreicht. Die Namen könnten zur Identifikation des Datensatzes übergeben werden. Allerdings wird die Id zur Selektion der Daten benötigt. Für den beim Klick auf den gewünschten Datensatz ausgegebenen Namen müsste daher die dazugehörige Id ermittelt werden. Es ist daher mit weniger Implementationsaufwand verbunden, wenn die Id direkt übergeben wird. Die Identifikationsnummern werden jedoch auch in der Legende des Diagramms angezeigt. Stattdessen sollen jedoch die Namen der entsprechenden Dimensionen verwendet werden. Dafür bietet C3.js die Möglichkeit, für jeden Datensatz über die Id einen anzuzeigenden Namen zu deklarieren.[49] Dies geschieht in der Zeile 17, in welcher die Map erstellt wird und in der Zeile 27, in welcher jeder Id der dazugehörige Name der Dimension zugewiesen wird.

Auslesen der Parameter Alle Parameter, die in der *Definition.xml*-Datei deklariert werden, sind im Funktionsobjekt des Extension Objects im Attribut *Layout* zu finden. In der Abbildung **??** wird der Parameter zum Anzeigen der Legende in den Entwicklertools des Chrome Browsers dargestellt.

In der Zeile 29 der *Script.js*-Datei wird der Parameter *Text0* über sein Attribut *text* eingelesen und direkt in einen Wahrheitswert konvertiert. Verwendet wird der Wahrheitswert in Zeile 47, um das Anzeigen der Legende ein- bzw. auszuschalten.

Vorbereiten der Darstellung Jedes Extension Object erhält über das Funktionsobjekt mit der Referenz *Element* Zugriff auf das *div*-Element, in welchem das HTML-Markup gespeichert wird. In Zeile 31 wird dieses *div*-Element zunächst geleert, da sonst Elemente aus dem vorherigen Zeichenvorgang erhalten bleiben würden. In den Zeilen 32-34 wird ein neues *div*-Element mit

48. Vgl. *C3.js | D3-based reusable chart library - Column Oriented Data*
49. Vgl. *C3.js | D3-based reusable chart library - Data Name*

der CSS-Klasse *kreisdiagramm* erstellt und im Extension Object eingefügt. Es wird bei der Generierung des Diagramms verwendet.

Generierung des Diagramms Die Zeilen 36-49 zeigen die Generierung des Diagramms. In Zeile 37 wird übermittelt, in welchem Element das Diagramm angezeigt werden soll. Die Daten werden in Zeile 39 übergeben. In Zeile 40 wird festgelegt, dass die Daten als Kreisdiagramm visualisiert werden sollen und in Zeile 44 werden die Ids der Spalten mit den jeweils anzuzeigenden Namen verknüpft.

Selektion der Daten In den Zeilen 41-43 wird eine Funktion übergeben, die aufgerufen werden soll, wenn auf einen der Abschnitte des Kreisdiagramms geklickt wird. In Zeile 42 wird die Funktion *SelectValuesInColumn* der QlikView Ajax API aufgerufen, um eine Selektion der angeklickten Daten vorzunehmen. Der erste Parameter dabei ist die Id der Dimension. Für die Erweiterung wurde ausschließlich eine Dimension deklariert und diese hat die Id 0. Der zweite Parameter erhält die Id des Datensatzes, der ausgewählt werden soll. Die Id wurde in der Datenstruktur von C3 in dem Attribut *id* abgelegt.

Der letzte Parameter trägt den Namen *isFinal*. Die QlikView Ajax Referenz beschreibt den Parameter folgendermaßen:

> Whether to toggle the selection.[50]

Doch diese Erklärung kann für viel Verwirrung sorgen, denn die Funktion des Parameters ist es nicht, die Selektion umzukehren. Tatsächlich legt der Parameter fest, ob die vorige Selektion beibehalten werden soll. Wird als Argument *false* übergeben, ersetzt die neue Selektion die vorige Selektion. Wird *true* übergeben, bleibt die vorige Selektion erhalten und die neue Selektion wird hinzugefügt.

2.2 Leitfaden zur Erstellung einer QlikView Document Extension mit JavaScript

In diesem Kapitel wird die Entwicklung einer Document Extension beschrieben, deren Ziel es ist, das aktuell sichtbare Arbeitsblatt mit dem Drücken der F10-Taste als PNG-Datei herunterzuladen. Sie nutzt die Bibliothek *html2canvas.js*, um die sichtbaren Elemente im Dokument Object Model in ein *canvas*-Element zu zeichnen. Das erstellte Bild soll dann automatisch heruntergeladen werden.

2.2.1 Erstellen der Dateistruktur

Die Document Extension wird im Ordner mit dem Namen *DownloadArbeitsblattAlsPng* abgelegt. Die für jede Document Extension benötigten Dateien sind die *Definition.xml*- sowie

50. QlikTech, *JsDoc Reference - Qv.Document.Object.Data - SelectValuesInColumn*

die *Script.js*-Datei. Darüber hinaus müssen die für die im Folgenden erstellte Document Extension benötigten Bibliotheken in den Ordner heruntergeladen werden. Diese sind *html2canvas.js*[51], *canvg.js*[52], *rgbcolor.js*[53] und *StackBlur.js*[54].

2.2.2 Erstellen eines QlikView-Testdokumentes

Zum Testen der Document Extension kann das gleiche Dokument zur Anwendung kommen, welches schon im Kapitel 2.1.3 erstellt wurde. Alternativ kann jedoch ein völlig anderes QlikView-Dokument verwendet werden, da die Document Extension nicht von dem Datenmodell des Dokuments beeinflusst wird. Um die Document Extension zu testen oder bereitzustellen, muss sie dem QlikView-Dokument im Menü über *Einstellungen* > *Eigenschaften des Dokuments...* > *Erweiterungen* hinzugefügt werden. Im Anhang A ist dies in den Abbildungen **??** und **??** zu sehen.

2.2.3 Erstellung der Definition.xml-Datei

Im Listing **??** ist die Definition der Document Extension zu sehen. Ein Label wird nicht angegeben, denn dieses wird in der Liste der verfügbaren Document Extensions nicht angezeigt. Lediglich die Beschreibung kann dort eingesehen werden und wird daher über das Attribut *Description* gesetzt. Anders als bei Extension Objects ist hier das Attribut *Type* mit dem Wert *document* anzugeben, um es als Document Extension zu deklarieren.

2.2.4 Erstellung der Script.js-Datei

Das Skript der Document Extension ist im Listing **??** auf Seite **??** abgebildet. Einige der Schritte ähneln der Erstellung der *Script.js*-Datei des Extension Objects im Kapitel 2.1.7.

Da der Name der Erweiterung mehrmals Verwendung findet, wird er in Zeile 1 einer Variablen zugewiesen. Weiterhin wird in Zeile 2 eine Variable für den Tastencode der F10-Taste gespeichert.

Anfordern von weiteren Dateien In den Zeilen 4-9 erfolgt das Nachladen der zusätzlich benötigten JavaScript-Dateien. Beim Zusammensetzen der absoluten Adresse des Ordners, in dem die Erweiterung abgelegt ist, gibt es einen Unterschied zu den Extension Objects. Für Document Extensions muss hier zusätzlich der Parameter *type* mit dem Wert *document* übergeben werden, wie in Zeile 5 zu sehen ist. Die Reihenfolge der in den Zeilen 6 und 7 gelisteten Bibliotheken ist wichtig, da *canvg.js* von *rgbcolor.js* und *StackBlur.js* abhängt und daher nach diesen geladen werden muss.

51. https://github.com/niklasvh/html2canvas/releases/download/0.4.1/html2canvas.js
52. https://raw.githubusercontent.com/gabelerner/canvg/master/canvg.js
53. https://raw.githubusercontent.com/gabelerner/canvg/master/rgbcolor.js
54. https://raw.githubusercontent.com/gabelerner/canvg/master/StackBlur.js

Funktionsobjekt der Document Extension Auch in der Zeile 10 ist ein Unterschied beim Hinzufügen der Erweiterung zu erkennen. Statt *Qva.AddExtension* hat die hier verwendete Methode den Namen *Qva.AddDocumentExtension*. Die beiden Parameter sind jedoch gleich, denn auch hier repräsentiert der erste Parameter den Namen der Erweiterung und der zweite Parameter die dazugehörige Funktion. Diese Funktion wird allerdings im Unterschied zu den Extension Objects nur einmal beim Laden des Dokumentes aufgerufen.

In der Zeile 11 erfolgt die Deklaration der Funktion, die beim Druck der F10-Taste ausgeführt werden soll. Für solche Events kann, wie in diesem Beispiel zu sehen, *JQuery* verwendet werden. Die Bibliothek wird bereits von der QlikView API importiert und muss daher nicht zusätzlich der Erweiterung hinzugefügt werden.

In Zeile 13 werden die Maße des Bereiches berechnet, indem sich sichtbare Objekte befinden. Die dazugehörige Funktion ist in den Zeilen 21-32 definiert.

Die Bibliothek *html2canvas* arbeitet fehlerhaft beim Zeichnen von SVG-Elementen im Dokument. Das im Kapitel 2.1 erstellte Kreisdiagramm erscheint daher nicht im Bild, da es ebenfalls als SVG-Grafik gezeichnet wird. Mit dem Aufruf der Methode *canvg* in Zeile 14 erfolgt eine Umwandlung aller *svg*-Elemente im Document Object Model in *canvas*-Elemente, die dann korrekt durch *html2canvas* gezeichnet werden. Allerdings ist die Umwandlung ungenau und daher behalten die *svg*-Elemente oft nicht ihre exakte Darstellung, wie in Anhang D in Abbildung **??** zu sehen ist.

Nach der Umwandlung wird in Zeile 15 die Funktion *downloadSheetAsPng* zum Zeichnen der sichtbaren Elemente aufgerufen. Ihr werden die Maße als erstes Argument übergeben. Die Funktion ist in den Zeilen 37-47 deklariert. Dem ersten Argument der Methode *html2canvas* wird in Zeile 38 das zu zeichnende Element übergeben. Der gesamte Körper des Dokumentes soll gezeichnet werden und daher wird *document.body* übergeben. Beim zweiten Argument handelt es sich um weitere Parameter in Form eines assoziativen Arrays. In den Zeilen 39 und 40 wird der zu zeichnende Bereich auf die bereits berechnete Fläche von sichtbaren Objekten begrenzt. Die Zeilen 41-46 beschreiben die Funktion, die aufgerufen werden soll, wenn das Zeichnen abgeschlossen ist. Die Funktion erstellt ein *a*-Element, welches den Download der im Attribut *href* verwiesenen Datei ermöglicht. Anders als bei herkömmlichen Hyperlinks erhält das Attribut keine Adresse, sondern die Bilddaten in Form einer Base64-Zeichenkette. Für die Umwandlung sorgt die Funktion *toDataURL* des *canvas*-Objektes. Das Attribut *download* beschreibt den Namen, mit dem das Bild heruntergeladen werden soll. Der Link wird dem Dokument nicht hinzugefügt und ist daher für den Benutzer nicht sichtbar. Dies ist auch nicht nötig, da der Klick auf diesen Link in der Zeile 45 automatisch simuliert wird. Somit startet der Download sofort.

2.3 Leitfaden zur Erstellung eines Qlik Sense Extension Objects mit JavaScript

Für das im Kapitel 2.1 erstellte Kreisdiagramm wird im Folgenden ein Äquivalent in Qlik Sense erstellt. Im Anhang C in Abbildung **??** ist die Erweiterung in einem Qlik Sense Arbeitsblatt zu sehen.

Für das Beispiel der vorliegenden Arbeit wird die Qlik Sense Workbench verwendet, welche im Kapitel 1.3.4 näher erläutert wurde. Nach dem Klick auf die Schaltfläche *Create New* wird ein Dialog geöffnet. Er ist im Anhang C in Abbildung **??** dargestellt. Im Eingabefeld *Name* wird *C3Kreisdiagramm* eingetragen. Im Eingabefeld *Template* kann eine beliebige Schablone für die Erweiterung gewählt werden. Für dieses Extension Object wird der Eintrag *Chart template* gewählt. Nach einem Klick auf die Schaltfläche *Create* erscheint das Extension Object unter dem eingegebenen Namen in der Liste der Erweiterungen. Mit einem Klick auf die Erweiterung öffnet sich ein Dialog, in welchem man über die Schaltflächen *Edit* in den Bearbeitungsmodus für die Erweiterung gelangt. Es öffnet sich eine neue Seite mit zwei Reitern für die Dateien *C3Kreisdiagramm.qext* und *C3Kreisdiagramm.js*.

2.3.1 C3Kreisdiagramm.qext

An der Datei *C3Kreisdiagramm.qext* sollten einige Änderungen durchgeführt werden. Das Attribut *icon* erhält statt dem Wert *bar-chart-vertical* den Wert *pie-chart,* damit die Erweiterung in der Liste der verfügbaren Erweiterungen mit einem Kreisdiagramm als Vorschaubild erscheint. Alternativ kann für das Attribut *preview* der Name einer Bilddatei angegeben werden. Für diese Erweiterung wird das Attribut entfernt. Außerdem erhält das Attribut *author* den Namen des Verfassers dieser Arbeit. Das Listing **??** zeigt die durchgeführten Änderungen an der Datei.

2.3.2 Weitere Module erstellen

Da Qlik Sense die Bibliothek RequireJS verwendet, ist es sehr einfach möglich, die Definition und die initialen Eigenschaften der Erweiterung in externe JavaScript-Dateien auszulagern. Dies ist in Vorbereitung auf die Portierung der Erweiterung nach Dart sehr nützlich, da die JavaScript-Dateien nicht verändert werden müssen. Es handelt sich bei den Parametern lediglich um assoziative Arrays, bei denen es keinen Vorteil hat, sie in die Dart-Syntax zu überführen. Vielmehr erfordert es einen zusätzlichen Aufwand, wie in Anhang E im Listing **??** zu sehen ist. Außerdem wird der Erweiterung ein CSS-Stylesheet hinzugefügt. Dieser kann ebenso durch RequireJS eingebunden werden.

Über die Schaltfläche *New File* können weitere Dateien hinzugefügt werden. In dem Eingabefeld *Name* wird daher zunächst *initialProperties.js* anschließend *definition.js* und schließlich *style.css* eingegeben und jeweils mit der Schaltfläche *Create* bestätigt. Im Folgenden wird in beiden JavaScript-Dateien jeweils mit dem Befehl *define* die Definition des Moduls eingeleitet. Bei der CSS-Datei ist dies nicht nötig, da diese nur als Text eingelesen wird.

CSS-Stylesheet Wie schon bereits im Kapitel 2.1.4 für das QlikView Extension Object beschrieben, bedarf es einer CSS-Anweisung für die Erweiterung. Sie sorgt dafür, dass die generierte Vektorgrafik korrekt im zu zeichnenden Bereich erstellt wird. Der Inhalt, der im Reiter *style.css* eingetragen wird, ist im Listing **??** zu sehen.

Initiale Eigenschaften Im Listing **??** werden die initialen Eigenschaften des Extension Objects deklariert. Hier können beispielsweise Standardwerte für die Dimensionen - in Zeile 4 - und für die Formeln - in Zeile 5 - festgelegt werden. Durch den Befehl *define* in Zeile 1 wird das assoziative Array unter dem Namen der JavaScript-Datei definiert. Es gibt keine Abhängigkeiten von anderen Modulen, daher erhält die Funktion nur einen Parameter, nämlich die aufzurufende Funktion des Moduls. Sie gibt das Array zurück.

Definition der Dimensionen, Formeln, Parameter und Eingabefelder Eine Datei zum Beschreiben des Konfigurationsdialoges, wie es sie unter dem Namen *Properties.qvpp* in QlikView gab - siehe Kapitel 1.2.1 -, existiert in Qlik Sense nicht. Das Aussehen des Dialoges ist bereits festgelegt und lediglich die gewünschten Eingabefelder müssen angegeben werden.

Listing **??** zeigt die Definition des Extension Objects. Die Hauptnavigation ist eine Liste von Menüpunkten - deklariert in Zeile 3 - die in einem *accordion* - zugewiesen in Zeile 4 - dargestellt werden. In einigen Einträgen ist die Zuweisung des Attributes *uses* angegeben. Damit wird die Standardkonfiguration geladen. Alle weiteren Eigenschaften konfigurieren diese Standardeinstellungen.

Es sollen genau eine Dimension und eine Formel für das Extension Object existieren. Daher werden das Minimum und das Maximum für die Dimensionen - in Zeile 8 - und für die Formeln - in Zeile 12 - auf *1* gesetzt. Neben der Standardkonfiguration soll eine komplett neue Eigenschaft hinzugefügt werden, nämlich der Wahrheitswert zum Anzeigen der Legende. Diese soll in der Rubrik der Einstellungen erscheinen und wird daher dem Eintrag *settings* in den Zeilen 17-27 hinzugefügt. Durch den Eintrag *items* kann eine weitere Unternavigation hinzugefügt werden. In dieser ist nur das Eingabefeld *Legend* präsent. In den Zeilen 21 und 23 wird festgelegt, dass es sich um einen Wahrheitswert handelt, der standardmäßig den Wert *wahr* hat und der anzuzeigende Text *Zeige Legende* lauten soll. In Zeile 24 wird ein Bezeichner dem Attribut *ref* zugewiesen, unter welchem die Eingabe des Feldes im Skript zu finden sein wird.

2.3.3 Herunterladen der Bibliotheken

Wie bereits für das Extension Object C3Kreisdiagramm für QlikView müssen ebenfalls die Dateien der Bibliotheken D3 und C3 in den Ordner der erstellten Erweiterung heruntergeladen werden. Anders als im Kapitel 2.1.7 wird diesmal nicht die verkleinerte *c3.min.js*-Datei, sondern die vollständige *c3.js*-Datei verwendet. Der Grund dafür ist, dass diese Datei für das

Funktionieren mit RequireJS modifiziert werden muss. Die im Folgenden notwendigen Dateien sind also: *d3.min.js* [55], *c3.js* [56] und *c3.min.css* [57].

Modifizieren der c3.js-Datei Die Definition der Module erfolgt durch RequireJS nicht nur über den Dateinamen, sondern auch über den Dateipfad.[58] So wird beispielsweise die Datei *C3Kreisdiagramm.js* mit dem Pfad *extensions/C3Kreisdiagramm/C3Kreisdiagramm* erstellt. Die *d3.min.js*-Datei befindet sich in demselben Ordner und wird daher auch relativ zu diesem Pfad definiert, und zwar unter *extensions/C3Kreisdiagramm/d3.min*. Dies ist jedoch problematisch, da die Bibliothek C3 die Abhängigkeit D3 im Pfad *d3* erwartet. Dies ist im Listing **??** in der Zeile 6902 zu sehen.

Unter dem Pfad *d3* ist die Abhängigkeit nicht zu finden, was dazu führt, dass C3 nicht initialisiert werden kann und die Referenz auf die Bibliothek im Skript *null* sein wird. Dies wird bei der Generation des Diagramms zu einem Fehler führen. Daher ist es nötig den Ausdruck zu modifizieren, indem der tatsächliche Pfad der Bibliotheken angegeben wird. Im Listing **??** auf Seite **??** ist die modifizierte Zeile 6902 einsehbar.

Anpassen der wbfolder.wbl-Datei Damit die soeben hinzugefügten Dateien auch in der Qlik Sense Workbench als Reiter auftauchen, müssen sie in der *wbfolder.wbl*-Datei gelistet werden. Im Listing **??** ist die *wbfolder.wbl*-Datei nach dem Hinzufügen der betroffenen Dateien zu sehen.

2.3.4 C3Kreisdiagramm.js

Im Listing **??** auf Seite **??** ist der Haupteinstiegspunkt des Extension Objects zu sehen. In den Zeilen 1 und 2 werden die erstellten Module für die Definition und die initialen Eigenschaften sowie die benötigten Bibliotheken als Abhängigkeiten dieses Moduls definiert. Der zweite Parameter in der Zeile 3 ist eine Funktion, welche die angegebenen Abhängigkeiten als Parameter erhält. Aus diesem Grund ist die Anzahl der Parameter gleich der Anzahl der Einträge der Liste von Abhängigkeiten. Weiterhin ist auch die Reihenfolge gleich, sodass der erste Eintrag der Abhängigkeiten auch dem ersten Parameter der Funktion zugewiesen wird.

Bei den beiden letzten Abhängigkeiten *text!./c3.min.css* und *text!./style.css* handelt es sich um CSS-Dateien, welche nicht als JavaScript-Objekte eingelesen werden können und daher als Text interpretiert werden müssen. Aufgrund dessen beginnen beide Abhängigkeiten mit dem Ausdruck *text!*, welcher angibt, dass der Inhalt der Datei als Zeichenkette dem Parameter übergeben werden soll.[59]

In den Zeilen 6 und 7 werden die Inhalte dieser CSS-Dateien über die Parameter *c3Css* und *styleCss* im *head*-Element des Dokuments als *style*-Elemente eingefügt. Auffällig an den beiden

55. https://github.com/mbostock/d3/blob/master/d3.min.js
56. https://github.com/masayuki0812/c3/blob/master/c3.js
57. https://github.com/masayuki0812/c3/blob/master/c3.min.css
58. Vgl. *RequireJS API - Load JavaScript Files*
59. Vgl. *Specify a Text File Dependency*

Codezeilen ist die Ähnlichkeit der Syntax mit JQuery. Qlik Sense bindet diese Bibliothek nicht ein, dennoch sind JQuery-Befehle verfügbar, da Qlik Sense das AngluarJS-Framework verwendet.

> AngularJS [...] builds on the core functionality of jQuery. In fact, AngularJS contains a cut-down version of jQuery called jqLite [...].[60]

Beginnend ab der Zeile 9 wird das assoziative Array des in dieser Datei definierten Moduls zurückgegeben. In den Zeilen 10 und 11 können nun die in den vorherigen beiden Schritten erstellten Module für das *initialProperties*- und das *definition*-Attribut übergeben werden.

Das letzte wichtige Attribut für das Extension Object ist *paint*, welches die Funktion erhält, die für das Zeichnen verwendet werden soll. Sie wird beispielsweise bei der Erstellung oder Manipulation der Instanz der Erweiterung und bei der Selektion von Daten aufgerufen.

Die Funktion erhält zwei Parameter. Der erste Parameter enthält das *div*-Element, welches dem Extension Object zum Zeichnen zur Verfügung gestellt wird. Über diese Referenz können JQuery-Funktionen aufgerufen werden. Daher wurde dem Namen der Referenz ein Dollarzeichen vorangestellt, um zu verdeutlichen, dass es sich um eine JQuery-Referenz und nicht um eine herkömmliche Referenz auf ein *div*-Element handelt. Dem zweiten Parameter werden die aktuellen Daten und die aktuelle Selektion, sowie weitere Informationen zum Extension Object übergeben.

Darüber hinaus gibt es jedoch noch weitere Informationen des Extension Objects, welche über die Referenz *this* im Kontext der Funktion verfügbar sind. Damit die Referenz nicht von einem weiteren Funktionskörper überschrieben wird, erfolgt eine Zuweisung von *this* zur lokalen Variable *extensionObject* in Zeile 13.

Das Attribut *extensionData.qInfo.qId* wird in Zeile 14 ausgelesen, um sie einer lokalen Variable *objectId* zuzuweisen. Sie wird in den Zeilen 15 und 16 verwendet, um die Information zum Objekt sowie die Information zu den aktuellen Daten und Selektionen im globalen Kontext *window* zu speichern. Auf diese Weise ist es möglich, im Browser durch Inspektion der Objekte Hinweise für das weitere Vorgehen zu erhalten.

Inspektion der Daten Nachdem die Erweiterung installiert und einer Anwendung hinzugefügt wurde, ist es möglich, in den Entwicklungswerkzeugen des Browsers weitere Informationen über die globalen Variablen zu erhalten. Für dieses Beispiel wird dem Extension Object die Dimension *Produkt* und die Formel *Menge* hinzugefügt.

Durch Eintippen der Zeichenkette *C3KreisdiagrammData* in der JavaScript-Konsole wird das konkrete Objekt vorgeschlagen. Dies ist ratsam, da die Id, die für die Referenz verwendet wird, variabel ist. Das Objekt enthält unter anderem das Attribut *qHyperCube.qDataPages*, in welchem die aktuellen Daten einsehbar sind. Dies ist in Abbildung **??** auf Seite **??** zu sehen.

Die Daten sind ähnlich aufgebaut wie in QlikView, was im Kapitel 2.1.7 nachgeschlagen werden kann. Jeder Datensatz ist ein Array mit zwei Elementen. Im ersten Element ist unter

60. Freeman, *Pro AngularJS* -, S. 47

anderem die Id, welche zur Selektion des Datensatzes notwendig ist, im Attribut *qElemNumber* gespeichert. Weiterhin ist die Bezeichnung der Dimension im Attribut *qText* zu finden.

Im zweiten Element des Arrays ist der Wert des Datensatzes in unterschiedlichen Datenformaten vorhanden. Das Attribut *qText* speichert den Wert als Zeichenkette während *qNum* den Wert als Zahl beinhaltet.

Auslesen der Daten In den Zeilen 18-28 des Listings **??** auf Seite **??** wird das Auslesen der Daten vorgenommen. Wie bereits für das Kreisdiagramm Extension Object unter QlikView werden auch hier die Daten in ein Format überführt, die von der C3-Bibliothek verwendet werden können. Ebenso erfolgt wiederum eine Zuweisung des Dimensionsnamens, zu der dazugehörigen Id. Dies kann im Kapitel 2.1.7 nachgelesen werden. Für Qlik Sense unterscheiden sich hier lediglich der Zugriff auf die Daten mittels einer Funktion statt einer Schleife und die Bezeichner der Attribute.

In Qlik Sense kann über die Funktion *eachDataRow* des Attributes *backendApi* des Funktionsobjektes *extensionObject* auf die Daten zugegriffen werden.[61] Ihr wird eine Funktion übergeben, welcher im ersten Parameter die Zeilennummer des Datensatzes und im zweiten Parameter der Datensatz selbst übergeben wird. In den Zeilen 21-23 werden die Attribute dieses Datensatzes ausgelesen. Die Zeilennummer findet hier keine Anwendung.

Zeichnen des Diagramms In den Zeilen 29-32 wird das *div*-Element zunächst geleert und ihm anschließend ein weiteres *div*-Element mit der CSS-Klasse *kreisdiagramm* hinzugefügt. In den Zeilen 34-46 erfolgt die Generierung des Diagramms. Wie schon im Kapitel 2.1.7 beschrieben, werden auch hier die Daten, der Diagrammtyp und die Dimensionsnamen übergeben.

Die Unterschiede zum Vorgehen in QlikView sind die folgenden:

Da es sich bei dem *$contentDiv* nicht um eine herkömmliche Referenz auf ein *div*-Element handelt, sondern um eine Referenz auf ein JQuery-Objekt, welches das *div*-Element referenziert, ist die direkte Zuweisung der Referenz nicht möglich. Bei dem Objekt handelt es sich um eine Liste, welche genau ein Element enthält, nämlich die Referenz auf das tatsächliche *div*-Element. Daher wird dem Attribut *bindto* dieses erste Element mit dem Ausdruck *$contentDiv[0]* übergeben. Die Selektion der Daten wird in Qlik Sense mit der Methode *selectValues* des Attributes *backendApi* des Funktionsobjektes *extensionObject* umgesetzt.

Die Parameter sind weitestgehend die gleichen geblieben. Der erste Parameter ist die Id der Dimension. Der zweite Parameter ist diesmal eine Liste an auszuwählenden Ids. Da nur eine Id ausgewählt werden soll, enthält die Liste nur ein Element. In QlikView war es möglich, ausschließlich eine Id anzugeben. Sollten mehrere Ids ausgewählt werden, so bedarf es in QlikView der Übergabe einer einzelnen Zeichenkette, welche die Ids durch Kommata getrennt enthält. In Qlik Sense ist dagegen nur noch die Auswahl über eine Liste von Ids möglich. Der letzte Parameter gibt den Selektionsmodus an. Ist der Wert wahr, wird die neue Selektion

61. Vgl. QlikTech, *Qlik Sense for Developers - eachDataRow method*

der vorherigen Selektion hinzugefügt. Ist der Wert unwahr, so ersetzt die neue Selektion die vorherige.[62]

3 Entwicklung von Erweiterungen mit Dart

Dieses Kapitel beschäftigt sich mit der Entwicklung von Extension Objects und Document Extensions für QlikView und Qlik Sense mithilfe der Programmiersprache Dart. Zu diesem Zweck wird jedoch zunächst eine Klassenbibliothek entwickelt, mit der die QlikView Ajax API sowie die Qlik Sense Extensions API abstrahiert werden soll. Die für die Entwicklung von Erweiterungen essenziellen Funktionen der jeweiligen Implementationen sollen in der Klassenbibliothek über dieselben Schnittstellen erreichbar sein. Anschließend erfolgt die Implementierung zweier sogenannter Pub Transformer zur automatisierten Bereitstellung der Erweiterungen auf den jeweiligen Servern. Abschließend werden die Erweiterungen, die in dem Kapitel 2 entwickelt wurden, in die Dart Programmiersprache portiert.

3.1 Entwicklung einer Klassenbibliothek zur Verallgemeinerung der QlikView und Qlik Sense API

Die QlikView Ajax API sowie die Qlik Sense Extensions API lassen kaum Gemeinsamkeiten erkennen. Die Objekte, deren Attribute und Funktionen haben unterschiedliche Namen. Auch die Parameter unterscheiden sich teilweise in ihrem Datentyp. Im weiteren Verlauf dieses Kapitels werden einige dieser Funktionen vorgestellt und ihre Unterschiede erläutert.

Die enorme Ungleichheit hat zwei Nachteile zur Folge. Hat man bereits den Umgang mit einer API kennengelernt, hilft dies nur sehr wenig, um sich in die andere einzuarbeiten. Weiterhin ist das Portieren einer Erweiterung zwischen den beiden Plattformen sehr schwierig, da fast jede Anweisung ausgetauscht werden muss.

Darüber hinaus weist die QlikView Ajax API einige Inkonsistenzen und fehlleitende Bezeichner für Funktionen und Parameter auf. So gibt es beispielsweise die Methode *GetCurrentSelections*. Statt der aktuellen Selektion gibt die Methode *undefined* zurück. Ungleich einer gewöhnlichen Getter-Funktion muss ihr ein Objekt im ersten Parameter übergeben werden. Das Objekt sollte ein Attribut *onChange* haben, bei dem es sich um eine Callback-Funktion handelt. Die Funktion wird immer dann aufgerufen, wenn sich an der aktuellen Selektion etwas ändert.[63] Es handelt sich daher also nicht um eine Getter-Funktion, obwohl ihr Name dies vermuten lässt.

Ein weiteres Beispiel dafür, dass die QlikView Ajax API sich nur sehr schwer bedienen lässt, ist im Anhang H zu sehen.

Aufgrund der Ungleichheit beider APIs sowie der erschwerten Anwendung wurde für diese Arbeit eine Klassenbibliothek erstellt, die die Entwicklung von Erweiterungen stark ver-

62. Vgl. QlikTech, *JsDoc Reference - Qv.Document.Object.Data - SelectValuesInColumn*
63. Vgl. QlikTech, *JsDoc Reference - Qv.Document - GetCurrentSelections*

einfachen soll. Ein Klassendiagramm der Klassenbibliothek ist im Anhang F in Abbildung **??** zu sehen.

3.1.1 Erstellen des Projektes

Das Projekt wird als sogenanntes Pub Package erstellt. Der Dart Editor kann dabei sehr hilfreich sein, da es für diese Art von Projekt eine Vorlage mit dem Namen *package* gibt. Das Projekt erhält den Namen qlikview_qlik_sense_extensions. Für die Programmiersprache Dart existiert ein Styleguide, der empfiehlt, Bibliotheken und Dateinamen mit Kleinbuchstaben und Unterstrichen zu benennen.[64]

Das Pub Package enthält unter anderem den Ordner *lib*. Es handelt sich um den Ordner, in dem die Quelldateien abgelegt werden sollen, die von anderen Projekten verwendet werden. Andere Ordner, wie beispielsweise die Ordner *test* und *example*, können ebenfalls Dart-Quelldateien enthalten, welche jedoch nicht in andere Projekte eingebunden werden.[65]

Ein Dartprojekt enthält im Wurzelordner eine *pubspec.yaml*-Datei. Dabei handelt es sich sozusagen um die Projektdatei, in welcher grundlegende Informationen zum Projekt gespeichert werden. Sie enthält auch eine Liste aller Abhängigkeiten des Projektes, die automatisch heruntergeladen werden können. Darüber hinaus gibt es die Möglichkeit, sogenannte Pub Transformer anzugeben, die bei jedem Bildprozess aufgerufen werden.[66] Im Listing **??** ist die *pubspec.yaml*-Datei des soeben erstellten Projektes mit den nötigen Änderungen zu sehen.

In Zeile 7 wird mit dem Kommandozeilenparameter *–enable-enum* angegeben, dass auch Enum-Typen verwendet werden sollen.

> To use dart2js on code that includes enumerated types, opt in by specifying the –enable-enum flag.[67]

3.1.2 Dateiorganisation

In Listing **??** auf Seite **??** ist die Haupteinstiegsdatei *qlikview_qlik_sense_extensions.dart* der Bibliothek zu sehen. In der Datei werden lediglich der Name der Bibliothek – Zeile 3 - und alle dazugehörigen Dateien angegeben. Es wird eine Bibliothek zur Interoperabilität mit JavaScript in Zeile 5 angefordert.

Alle anderen Dateien gehören zu der Bibliothek und befinden sich im Unterordner *src*. Die Schlüsselwörter *import* und *part* sorgen dafür, dass die angegebenen Dateien zum Projekt hinzugefügt werden. Sie unterscheiden sich jedoch etwas in ihrer Funktion.

Ein durch das Schlüsselwort *import* angefordertes package ist nur in dieser Bibliothek verwendbar und wird vor Projekten, die diese Bibliothek einbinden, versteckt. Das package

64. Vgl. *Dart Style Guide | Dart: Structured web apps*
65. Vgl. *Pub Package Layout Conventions | Dart: Structured web apps*
66. Vgl. *Pubspec Format | Dart: Structured web apps*
67. *dart2js: The Dart-to-JavaScript Compiler*

dart:js ist daher nur in diesem package sichtbar. Die in Zeile 6 angeforderte Datei beinhaltet Basisklassen, die durch den Anwender der Bibliothek nicht sichtbar sein sollen. Dennoch ist in der Datei auch das Enum *SelectionMode* zu finden, welches von dem Anwender verwendet werden soll. Daher wird die Datei in Zeile 8 durch das Schlüsselwort *export* verfügbar gemacht. Durch *hide* können jedoch Klassen ausgeschlossen werden. *Extension* und *ExtensionObject* sind Basisklassen, auf die der Anwender keinen Zugriff haben sollte. Durch das Verstecken tauchen diese Klassen in der automatischen Vervollständigung nicht auf.

Eine Datei, die mit dem Schlüsselwort *part* angegeben ist, wird so behandelt, als gehöre sie zur selben Datei. Ein Projekt, welches diese Bibliothek einbindet, hat Zugriff auf alle Klassen und Funktionen, welche in diesen Dateien deklariert sind.

3.1.3 Allgemeine Basisklassen

Listing **??** auf Seite **??** zeigt die Datei *qlikview_qlik_sense_extensions_base.dart*. In Zeile 9 wird die abstrakte Klasse Extension deklariert. Jede Document Extension sowie jedes Extension Object verfügt über eine *onCreate*-Methode. Die Methode wird für jede Instanz nur einmal während seiner Erstellung aufgerufen. Sollte die Methode in den erbenden konkreten Klassen nicht implementiert werden, wird es dementsprechende Fehlermeldungen geben. Die Methode selbst erhält kein *abstract* Schlüsselwort. Dadurch, dass sie sich in einer abstrakten Klasse befindet und ihr Methodenkörper fehlt, ist sie automatisch abstrakt.

Das Enum *SelectionMode* in Zeile 13 wird bei der Selektion der Daten den Parameter *isFinal* bzw. *toogleMode* ersetzen. Durch die beiden Werte *replacePrevious* und *keepPrevious* ist nun klar erkennbar, welche Selektionsoptionen auswählbar sind.

ExtensionObject erbt von *Extension*. Die Klasse wird in Zeile 18 deklariert und ist ebenfalls abstrakt. In den Zeilen 25 und 26 werden die *objectId* und der *contentDiv* als Felder der Klasse deklariert. Das Schlüsselwort *final* gibt an, dass die Referenz zum Zeitpunkt der Objekterstellung gesetzt werden muss und im späteren Verlauf nicht verändert werden kann. In Zeile 20 wird über den Konstruktor mithilfe einer verkürzten Schreibweise die Zuweisung dieser beiden Felder vorgenommen. Jedes Extension Object muss die Getter-Methoden *dataAsList* und *idNameMap* implementieren. Diese sind in den Zeilen 28 und 29 zu finden. Getter-Methoden haben für gewöhnlich keine Parameter. In Dart werden solche Methoden mit dem Schlüsselwort *get* gekennzeichnet. Dann muss die Parameterliste entfallen.

Ein Extension Object muss im Gegensatz zu einer Document Extension auch die Methode *onPaint* implementieren. Nach der Erstellung eines Extension Objects wird die Methode bei jedem Zeichnen aufgerufen. Weiterhin soll jedes Extension Object auch die Methode *selectValues* – Zeile 30 - implementieren. In Zeile 31 ist zu sehen, dass sie einen optionalen dritten Parameter namens *selectionMode* besitzt. Er hat den Standardwert *replacePrevious* und muss daher nur gesetzt werden, wenn ein anderer Selektionsmodus gewünscht ist.

In Zeile 22 wird der Klasse *ExtensionObject* ein assoziatives Array als statische Variable mit dem Namen *extensionMap* hinzugefügt. Die dazugehörige Methode *registerExtensionObject* ist in Zeile 36 deklariert. Sie steuert den Mechanismus, ob ein neues Extension Object erzeugt

wird oder ob es bereits existiert und nur neu gezeichnet werden muss. Dazu erhält es die *objektId*. Diese existiert in QlikView als auch in Qlik Sense. In Zeile 38 wird ermittelt, ob die *objectId* dem assoziativen Array schon einmal hinzugefügt wurde. Ist dies der Fall, wird das dazugehörige Extension Object neu gezeichnet – Zeile 41. Ist die *objectId* neu, wird ein neues Extension Object in den Zeilen 43-45 erstellt. Zu diesem Zweck muss der Funktion ebenfalls die Fabrikmethode *factoryFunc* übergeben werden, welche ein Objekt der konkreten Klasse zurückgeben muss.

Die abstrakte Klasse *QlikViewExtension* behandelt alle Gemeinsamkeiten von QlikView Extensions, demnach alles, was QlikView Document Extensions und QlikView Extension Objects gemeinsam haben. Die Klasse ist im Listing **??** auf Seite **??** abgebildet. Zunächst taucht hier zum ersten Mal in der Zeile 3 der Ausdruck *part of* auf. Jede Datei, die in einer Bibliothek mit dem Schlüsselwort *part* hinzugefügt wird, muss eine *part of* Anweisung besitzen. Dabei wird nach der Anweisung der Name ihrer zugehörigen Bibliothek angegeben.

Zuvor wurde die Basisklasse *ExtensionObject* deklariert und nun ist in Zeile 5 die Deklaration der abstrakten Basisklasse *QlikViewExtension* zu sehen. Hier kündigt sich bereits an, dass dies zu einer Mehrfachvererbung führen muss.

Mixins Eine Art Mehrfachvererbung ist in Dart mithilfe der sogenannten Mixins möglich. Um einer Vererbung eine weitere Klasse hinzuzufügen, muss diese jedoch einige Voraussetzungen erfüllen. Die Klasse darf keinen explizit deklarierten Konstruktor haben, muss direkt von *Object* erben und darf nicht die Referenz *super* verwenden.

Zum Zeitpunkt der Erstellung dieser Arbeit führte ein Verstoß gegen diese Regeln zu Fehlern während der Kompilierzeit. Das Dart-Entwicklerteam gibt im ECMA Standard 408 an, dass die Einschränkungen temporär sind und mit kommenden Versionen entfernt werden.[68]

In Zeile 7 wird die Getter-Methode *remoteAddress* deklariert. Auch hier handelt es sich um eine abstrakte Methode, da sich die absolute Adresse bei QlikView Extension Objects und QlikView Document Extensions unterscheidet. Dennoch kann die Referenz innerhalb dieser Klasse verwendet werden, da sie hier als Schnittstelle deklariert wird.

Über die Methoden *loadScripts* – Zeile 10 – und *loadStyleSheets* – Zeile 12 - sollen weitere Ressourcen geladen werden. Bei beiden Funktionen wurde eine abkürzende Schreibweise verwendet. Das *return*-Statement sowie die öffnenden und schließenden Klammern fehlen. Stattdessen wird die Rückgabe mit einem fetten Pfeil eingeleitet. Die Deklaration der Methode endet mit einem Semikolon. Diese kompakte Schreibweise empfiehlt sich für solche Methoden, welche nur eine Anweisung besitzen.

Asynchrone Programmierung Darüber hinaus ist hier zum ersten Mal der Datentyp *Future* zu sehen. Er wird in Dart für die asynchrone Programmierung verwendet. Die Funktionsweise soll an der Methode *_appendNodeToHeadAndLoad* in den Zeilen 44-49 erklärt werden. Sie hat die Aufgabe, ein beliebiges HTML-Element dem *head*-Element des Dokumentes hinzuzufügen.

68. Vgl. ECMA International, *Dart Programming Language Specification*, S. 41

Dies wird in Zeile 47 durchgeführt. Wird dabei beispielsweise ein Stylesheet oder ein Skript eingefügt, so muss dieses erst heruntergeladen und geparst werden. Auf einen Abschluss dieses Vorgangs wird in Zeile 47 jedoch nicht gewartet. Das Skript wird weiter ausgeführt. Zu diesem Zweck wird in Zeile 45 ein *Completer*-Objekt erstellt. Es dient dazu, nach Abschluss einer asynchronen Funktion, eine Callback-Funktion aufzurufen. In der Zeile 48 wird das Attribut *future* sofort zurückgegeben. In dem Objekt ist die aufzurufende Callback-Funktion enthalten. In Zeile 46 wird festgelegt, dass die Callback-Funktion aufgerufen werden soll, wenn das *onLoad*-Ereignis des zu ladenden HTML-Elements ausgelöst wird. Die gewünschte Funktion, die nach dem Erledigen der asynchronen Aufgabe aufgerufen werden soll, wird dem *Future*-Objekt über die Funktion *then* übergeben.

Die Funktion *_loadDependencies* welche in Zeile 30 deklariert wurde, zeigt ein Beispiel für das Warten auf mehrere asynchrone Aufgaben. In der Zeile 38 wird der Methode *Future.wait* eine Liste von *Future*-Objekten übergeben. Die Rückgabe ist wiederum ein *Future*-Objekt. Wenn alle asynchronen Aufgaben innerhalb der Liste abgearbeitet wurden, wird für alle zusammenfassend eine Callback-Funktion aufgerufen. Diese wird der Methode *then* als Parameter übergeben.

Öffentlichkeit Die Methoden *_loadDependencies* und *_appendNodeToHeadAndLoad* beginnen aus dem Grund mit einem Unterstrich, da sie für fremde Bibliotheken nicht sichtbar sein sollen. Es gibt eine verbreitete Konvention, privaten Attributen und Methoden einen Unterstrich voranzustellen, damit sie beim Lesen des Quellcodes sofort auffallen. In den verwendeten Programmiersprachen, wie beispielsweise Java und C#, muss jedoch trotzdem die Deklaration dieser Attribute und Methoden mit dem Schlüsselwort *private* erfolgen. In Dart gibt es das Schlüsselwort *private* nicht. Es ist der beginnende Unterstrich, der die Attribute und Methoden als nicht öffentlich deklariert. Die Sichtbarkeit bezieht sich in Dart jedoch nicht auf die Klassen, sondern auf die Bibliothek.[69]

Cascade operator Dart verfügt über einen sogenannten cascade Operator, dessen Anwendung in den Zeilen 17-19 und 24-26 zu sehen ist. Der Operator erspart das wiederholende Eingeben des Bezeichners des Objektes, dessen Attribute manipuliert werden. Diese Syntax trägt auch den Namen method chaining und wird beispielsweise in den Bibliotheken jQuery und d3 verwendet. Um in JavaScript eine solche Syntax zu ermöglichen, müssen die Funktionen die Referenz *this* zurückgeben. In Dart ist das nicht nötig, da der cascade Operator in die Sprache selbst integriert ist. Aus diesem Grund kann das Verketten von Methodenaufrufen bzw. Zuweisungen von Attributen an jedem Objekt durchgeführt werden.

String interpolation Es befindet sich noch eine weitere syntaktische Besonderheit von Dart in dem Listing **??**. Sie ist in Zeile 42 zu sehen. Es werden die Zeichenketten *remoteAddress* und *resource* durch einen Schrägstrich voneinander getrennt miteinander konkateniert. Soll eine Variable innerhalb einer Zeichenkette eingetragen werden, so muss sie nur von geschweiften Klammern und einem führenden Dollarzeichen umgeben werden. Auf diese Weise bleibt die Übersichtlichkeit erhalten.

69. Vgl. ECMA International, *Dart Programming Language Specification*, S. 10

3.1.4 Die Basisklasse QlikViewExtensionObject

Listing **??** auf Seite **??** zeigt die angesprochene Klasse *QlikViewExtensionObject*. Der Anwender der Klassenbibliothek wird von dieser Klasse erben, wenn er eine eigene konkrete Klasse für ein QlikView Extension Object erstellen möchte. Sie erbt direkt von der Klasse *ExtensionObject*. Mithilfe des Schlüsselwortes *with* wird in Zeile 6 definiert, dass sie auch Funktionalitäten und Schnittstellen der Klasse *QlikViewExtension* erhalten soll.

Initialisierungsliste des Konstruktors In den Zeilen 7-9 ist der Konstruktor zu sehen, der genau einen Parameter erhält. An dieser Stelle konnte die kompakte Schreibweise *this.qvaWrapper* nicht angewendet werden, da der gleiche Parameter auch für den Konstruktor der Basisklasse benötigt wird. Diesem muss die *objektId* und das *div*-Element zum Zeichnen übergeben werden. Beide Parameter sind als Attribute im *qvaWrapper*-Objekt zu finden. Mit einem Doppelpunkt nach der Parameterliste des Konstruktors wird der Beginn der Initialisierungsliste gekennzeichnet. Durch das Schlüsselwort *super* kann der Konstruktor der Basisklasse aufgerufen werden. Felder der Klasse, die mit dem Schlüsselwort *final* gekennzeichnet sind, können nicht im Körper des Konstruktors zugewiesen werden. Die Initialisierung muss bereits im statischen Kontext der Objekterstellung durchgeführt werden. Daher wird nach dem Aufruf des Konstruktors der Basisklasse, das Feld *qvaWrapper* in der Initialisierungsliste zugewiesen. Deklariert ist das Feld in der Zeile 11.

Nachdem die Objektinitialisierung abgeschlossen ist, kann nun die in Zeile 12 deklarierte Getter-Methode *name* aufgerufen werden. Der Name des Extension Objects kann in JavaScript über den Ausdruck *qvaWrapper.ObjectMgr.Extension* gefunden werden. Doch soll in Dart auf das Attribut zugegriffen werden, so ist es mit dieser Syntax nicht möglich.

Zugriff auf Attribute von JavaScript Objekten Der Typ des Objektes *qvaWrapper* ist *JsObject* aus dem package *dart:js*. Der Typ ermöglicht die Interaktion mit Objekten, welche in JavaScript erstellt wurden. In JavaScript gibt es zwei mögliche Vorgehensweisen, um auf Attribute von Objekten zuzugreifen. Zum Einen können der Bezeichner des Objektes und die Bezeichner der Attribute mit jeweils einem Punkt voneinander getrennt aufgelistet werden, also *qvaWrapper.ObjectMgr.Extension*. Zum Anderen können die Attribute als Einträge in einem assoziativen Array verstanden werden, demnach *qvaWrapper['ObjectMgr']['Extension']*. Wenn auf Attribute von JavaScript Objekten innerhalb von Dart zugegriffen werden soll, ist das nur über die letztere Syntax möglich.

In den Zeilen 14-19 ist die überschriebene Getter-Methode *remoteAddress* zu sehen. Die Methode wurde in der Klasse *QlikViewExtension* deklariert und wird auch nur dort verwendet, um Ressourcen der Erweiterung nach zu laden.

Da die Klasse als Mixin verwendet wird, darf sie keinen Konstruktor mit Parametern deklarieren. Um allerdings die absolute Adresse des Ordners der Erweiterung zu konstruieren, müssen der Typ und der Name der Erweiterung feststehen. Beide können aber nicht an die Klasse übergeben werden und somit muss die Konstruktion der absoluten Adresse an einer anderen Stelle vorgenommen werden. Daher wurde in der Klasse *QlikViewExtension* lediglich

die Schnittstelle der Methode *remoteAddress* deklariert. Die Implementierung der Methode übernehmen dann die Klassen *QlikViewExtensionObject* und *QlikViewDocumentExtension*. In beiden Klassen steht bereits der Typ der Erweiterung fest. Allein der Name fehlt für die Zusammensetzung der Zeichenkette. Die in Zeile 12 implementierte Getter-Methode *name* wird durch string interpolation in Zeile 18 verwendet, um den Pfad zu vervollständigen.

In den Zeilen 21-31 wurde die Schnittstelle *dataAsList* der abstrakten Klasse *ExtensionObject* überschrieben. Sie erleichtert den Zugriff auf die zu visualisierenden Daten erheblich. Dem Anwender wird erspart, die Struktur, der durch die QlikView Ajax API gelieferten Daten, zu analysieren. Durch die Annotation *@override* wird erreicht, dass der Compiler eine Warnung ausgibt, sollte sich die Methode nicht in den Klassen befinden, von denen geerbt wird. Die Annotation ist optional, doch sie ist hilfreich, wenn der Methodenname oder die Parameterliste innerhalb der Hierarchie verändert wird.

In der Methode wird zunächst eine Liste für die Datensätze angelegt. Bei den Elementen der Liste handelt es sich wiederum um Listen, welche die einzelnen Datensätze repräsentieren. Die Art und Weise, in der die Daten gegliedert sind, kann im Kapitel 2.1.7 nachgelesen werden. In einer Schleife werden die Id und der Wert ausgelesen. Für jeden Datensatz wird eine Liste erstellt, in der das erste Element die Id und das zweite Element den Wert erhält. Der Datensatz wird der Liste *data* über die Methode *add* hinzugefügt.

Damit für die anzuzeigenden Werte auch die Namen der Dimensionen auftauchen, wird in den Zeilen 33-43 die Getter-Methode *idNameMap* implementiert. Für jeden Datensatz werden die Id und der Dimensionsname in einem assoziativen Array gespeichert. Die Assoziation der Dimensionsnamen vom Typ *String* erfolgt über die Id vom Typ *int*.

In den Zeilen 45-51 wird der Aufruf der QlikView Methode *SelectValuesInColumn* vereinfacht. Dafür gibt es zwei Gründe. Zum einen soll es mithilfe des Enums *SelectionMode* transparenter werden, welche Funktion der dritte Parameter hat. Zum anderen handelt es sich um eine JavaScript-Methode, dessen Aufruf innerhalb von Dart schwierig ist.

Aufruf von JavaScript-Funktionen Objekte vom Typ *JsObject* besitzen die Methode *callMethod* die zum Aufrufen von JavaScript-Funktionen des Objektes verwendet wird. Der erste Parameter repräsentiert den Namen der Methode und hat den Typ *String*. Der zweite Parameter repräsentiert die Parameterliste vom Typ *List*.[70]

Die Methode *selectValues* erwartet im zweiten Parameter die auszuwählenden Ids als Liste. Der Methode *SelectValuesInColumn* der QlikView Ajax API müssen diese Ids jedoch als Zeichenkette übergeben werden, indem die Ids durch Leerzeichen voneinander getrennt sind. Objekte der Klasse *List* haben dafür eine Methode namens *join*, die dazu dient, alle Listeneinträge in eine Zeichenkette umzuwandeln. Das Zeichen, welches zwischen den Listeneinträgen eingetragen werden soll, kann als Parameter übergeben werden. Die Anwendung der Methode ist in Zeile 50 zu sehen.

70. Vgl. *dart:js.JsObject API Docs - callMethod*

Um eine QlikView Erweiterung zu registrieren, soll die Funktion *registerQlikViewExtension Object* verwendet werden. Sie ist im Listing **??** zu sehen. Auch diese Funktion hat vor allem den Zweck, dem Anwender die komplexen Aufrufe von JavaScript-Methoden zu ersparen. Auf die globale JavaScript-Variable *Qva* kann über die Syntax *context['Qva']* zugegriffen werden. Über das Objekt wird die Methode *AddExtension* aufgerufen. Sie erwartet den Namen der Erweiterung und die Funktion, welche bei jedem Zeichnen aufgerufen werden soll.

Benannte Konstruktoren In Zeile 48 ist der Ausdruck *new JsFunction.withThis* zu sehen. Es handelt sich hier um einen Konstruktor der Klasse *JsFunction* mit dem zusätzlichen Namen *withThis*. Den Konstruktoren Namen zu geben, ist in Dart nötig, da das Überladen von Methoden und somit auch von Konstruktoren durch Manipulation der Parameterliste nicht möglich ist. Für Methoden sollte daher ein anderer Bezeichner verwendet werden und Konstruktoren erhalten zusätzliche Namen.[71]

Verwendung der Referenz this von JavaScript-Funktionen Die Funktion, welche der Methode *AddExtension* übergeben werden muss, erhält beim Aufruf das *qvaWrapper*-Objekt in der *this* Referenz. In Dart referenziert das Schlüsselwort *this* jedoch auf das Objekt der Klasse, in dem es sich befindet. Auf das Funktionsobjekt kann in Dart mit *this* nicht zugegriffen werden. Um die Referenz auf das Funktionsobjekt zu erhalten, muss das Objekt mit dem Konstruktor *JsFunction.withThis* erstellt werden. Es muss eine Funktion übergeben werden, welcher das Funktionsobjekt im ersten Parameter mitgegeben wird.

In Zeile 50 wird die Funktion *registerExtensionObject* aus Listing **??** von Seite **??** aufgerufen. Sie verwaltet auch Qlik Sense Erweiterungen, welche andere Parameter als das *qvaWrapper*-Objekt benötigen. Die Methode wurde daher so abstrahiert, dass ihr eine Funktion übergeben werden muss, die keine Parameter hat und ein Objekt vom Typ *ExtensionObject* zurückgibt. Die Verarbeitung der Parameter muss also vor dem Aufruf vorgenommen werden. Daher wird dem zweiten Parameter von *registerExtensionObject* eine anonyme Funktion übergeben, welche die Fabrikmethode *factoryFunc* mit dem *qvaWrapper*-Objekt als Argument aufruft und die erstellte Instanz direkt zurückgibt. Die anonyme Funktion hat keine Parameter und die zurückgegebene Instanz ist vom Typ *QlikViewExtensionObject*, welcher eine Erweiterung des Typs *ExtensionObject* ist. Somit muss in der Implementation von *registerExtensionObject* lediglich die übergebene Funktion aufgerufen werden, denn das *qvaWrapper*-Objekt ist in ihr schon vorhanden.

3.1.5 Die Basisklasse QlikViewDocumentExtension

In Listing **??** auf Seite **??** ist die Klasse *QlikViewDocumentExtension* abgebildet. Der Anwender der Klassenbibliothek kann von dieser Klasse erben, um eigene QlikView Document Extensions zu erstellen. Die Klasse erbt diesmal direkt von *Extension*. Außerdem erhält sie die Funktionalitäten und Schnittstellen der Klasse *QlikViewExtension*.

71. Vgl. Kopec, *Dart for Absolute Beginners*, S. 133

Der Name der Erweiterung, der in Zeile 10 deklariert ist, kann im Konstruktor in der Zeile 9 über die kompakte Schreibweise *this.name* initialisiert werden. Anders als in der Klasse *QlikViewExtensionObject* handelt es sich dieses Mal um ein Feld. Trotzdem kann es auf die gleiche Weise zur Vervollständigung des Pfades der Getter-Methode *remoteAddress* verwendet werden. Üblicherweise ist die Parameterliste von Getter-Methoden leer. In Dart wird die Parameterliste nicht angegeben. Damit hat der Aufruf die gleiche Syntax wie der Zugriff auf ein Feld. Gleiches gilt für Setter-Methoden.

Die Methode zum Registrieren der QlikView Document Extensions ist im Listing **??** auf Seite **??** zu sehen. Nur die Fabrikmethode für die Erstellung der Erweiterung muss übergeben werden. Anders als bei Extension Objects existiert für QlikView Document Extensions keine Methode zum Zeichnen. Die Funktion der Erweiterung wird genau einmal nach dem Laden der Seite aufgerufen. Darüber hinaus kann eine Document Extension auch nicht mehrmals einem Dokument zugewiesen werden. Somit gibt es auch keine *objectId*, über welche die Identifizierung stattfinden könnte. Aus diesem Grund wird die Fabrikmethode lediglich einmal in Zeile 20 aufgerufen, um das Objekt zu erzeugen.

Die Funktion, welche *AddDocumentExtension* übergeben werden soll, wird in der Zeile 21 deklariert und initialisiert. Es handelt sich um eine gewöhnliche Dart-Funktion, welche die *onCreate*-Methode der Erweiterung aufruft. Es muss kein Objekt vom Typ *JsFunction* erstellt werden, da der Methode auch kein *qvaWrapper*-Objekt in der *this* Referenz übergeben wird.

Der Name der Erweiterung muss bereits im Konstruktor der Klasse *QlikViewDocument Extension* übergeben werden. Genauso wie die Methode *AddExtension* zum Registrieren von QlikView Extension Objects erwartet *AddDocumentExtension* den Namen der Erweiterung als ersten Parameter. Er wird in Zeile 23 aus der bereits erstellten Instanz *createdExtension* über das Feld *name* ausgelesen und der Parameterliste übergeben.

3.1.6 Die Basisklasse QlikSenseExtensionObject

Listing **??** auf Seite **??** zeigt die Basisklasse *QlikSenseExtensionObject*. Sie hat viele Gemeinsamkeiten mit der Basisklasse *QlikViewExtensionObject*, welche im Kapitel 3.1.4 beschrieben wird. Sie erbt ebenso von der Klasse *ExtensionObject*.

Der in Zeile 7 deklarierte Konstruktor erhält zwei Parameter, nämlich *extensionObject* und *extensionData*. Die *objectId* und der *contentDiv* werden in Zeile 8 übergeben. Dabei gibt es einige Unterschiede im Vergleich zu QlikView Extension Objects. Die *objectId* befindet sich im Attribut *['qInfo']['qId']* des Objektes *extensionData*. Der *contentDiv* wird im *extensionObject* über *[r'$element'][0]* adressiert. In Dart kennzeichnet ein klein geschriebenes *r* vor dem beginnenden Anführungsstrich die Zeichenkette als sogenannten raw string.[72] Das Dollarzeichen, welches in Dart für die string interpolation - siehe 3.1.3 - verwendet wird, hat daher in dieser Zeichenkette keine Bedeutung. Da es sich bei *$element* um ein JQuery-Objekt handelt, muss die tatsächliche Referenz des *div*-Elementes extrahiert werden. Sie befindet sich im ersten Element des Arrays.

72. Vgl. Kopec, *Dart for Absolute Beginners*, S. 19

Nachdem die Felder *extensionObject* und *extensionData* in den Zeilen 9 und 10 initialisiert wurden, kann auf die Getter-Methode *backendApi* – Zeile 12 - zugegriffen werden. Sie wird in den Implementierungen *dataAsList* und *idNameMap* verwendet, um durch die gelieferten Daten zu iterieren. Im Unterschied zur Basisklasse *QlikViewExtensionObject* erfolgt die Iteration nicht über eine *for*-Schleife, sondern über eine Funktion, welche in den Zeilen 18 bzw. 29 der Funktion *eachDataRow* übergeben wird. Außerdem werden andere Attributnamen zur Adressierung der Id, des Dimensionsnamens und des Wertes verwendet.

Die in den Zeilen 37-43 implementierte Methode *selectValues* ähnelt ebenso der Implementierung in der Klasse *QlikViewExtensionObject*. Die aufzurufende Methode der Qlik Sense Extensions API ist *selectValues*, welche ebenfalls über das Objekt der Getter-Methode *backendApi* aufgerufen wird. Im Gegensatz zur Funktion *SelectValuesInColumn* wird im zweiten Parameter ein Array anstatt einer Zeichenkette erwartet. Es soll die zu selektierenden Ids enthalten. Beim Übergeben von Kollektionen als Argumente von JavaScript-Funktionen muss jedoch zuerst eine Umwandlung durchgeführt werden.

Dart-Kollektionen als Argumente von JavaScript-Funktionen Bei einem Großteil der Objekte, welche an JavaScript-Funktionen übergeben werden, erfolgt die Umwandlung automatisch. Bei Kollektionen vom Typ *Map* und *Iterable* ist das nicht der Fall. Sie müssen manuell in ein Objekt vom Typ JsObject umgewandelt werden. Dazu wird die Kollektion dem Konstruktor *JsObject.jsify* übergeben.[73]

In Listing **??** sind die zwei Methoden zu sehen, welche der Registrierung des Qlik Sense Extension Objects dienen. Es soll weiterhin RequireJS verwendet werden. Dem Anwender soll jedoch das Erarbeiten der korrekten Syntax für den Aufruf der Funktion *define* in Dart erspart bleiben. Die Funktion *defineModule* in den Zeilen 46-49 erleichtert den Aufruf. Sie erwartet die Namen der Module als Liste vom Typ String und die aufzurufende Callback-Funktion. Beide Parameter werden der Funktion *define* übergeben, wobei die Liste der Abhängigkeiten zunächst manuell mithilfe des Konstruktors *JsObject.jsify* in ein JavaScript-Objekt umgewandelt wird.

In den Zeilen 51-55 ist die Methode *registerQlikSenseExtensionObject* zu sehen. Sie soll innerhalb der Funktion des Attributes *paint* vom RequireJS-Modul aufgerufen werden. Dabei soll das Objekt *extensionData* sowie die Fabrikmethode *factoryFunc* übergeben werden. Nachdem die *objectId* extrahiert wurde, wird genau wie für die QlikView Extension Objects die Funktion *registerExtensionObject* aufgerufen. Sie verwaltet das Erstellen bzw. das Zeichnen des Extension Objects über die *objectId*.

3.2 Transformer zum Anpassen der source maps für QlikView Erweiterungen

Wie bereits im Kapitel 2.1.7 beschrieben, können auf Dateien von QlikView Erweiterungen nicht über einen relativen Pfad zugegriffen werden. In der aus dem Dart-Quellcode generierten JavaScript-Datei befindet sich jedoch der Verweis auf eine relative Adresse der dazugehörigen source map. Listing **??** im Anhang G zeigt ein Beispiel für einen solchen Verweis. Auf derselben

73. Vgl. *dart:js.JsObject API Docs – JsObject-jsify*

Seite ist im Listing **??** ein Beispiel einer solchen source map zu sehen. Bei den Einträgen im Knoten *sources* handelt es sich ebenfalls um relative Adressen auf die jeweiligen Dart-Quelldateien. Diese müssen bei jedem build angepasst werden, was auf Dauer eine wirklich zeitintensive Aufgabe sein kann. Aus diesem Grund wird ein Pub Transformer implementiert, der den Prozess automatisiert.

Das zu erstellende package erhält den Namen *qlikview_modify_source_map*. In der *pubspec.yaml*-Datei muss die Abhängigkeit *barback* hinzugefügt werden, wie im Listing **??** zu sehen ist.

Das Listing **??** auf Seite **??** zeigt den Transformer *QlikViewModifySourceMap*. Es wird unter anderem das package *barback* in Zeile 5 importiert, welches notwendig ist, um von der Klasse Transformer erben zu können. Darüber hinaus taucht zum ersten Mal das package *path* in Zeile 8 auf. Es vereinfacht die Konstruktion von Dateipfaden.

Der Konstruktor mit dem Namen *.asPlugin* muss vorhanden sein und kann verwendet werden, um ein Parameterobjekt für den Transformer zu definieren.[74] In den Zeilen 14-19 wird mithilfe der übergebenen Parameter *extensionName* und *extensionType* die absolute Adresse für den Ordner der Erweiterung konstruiert.

Sofern kein Filter für den Transformer implementiert ist, wird die Methode *apply* für jede Quelldatei angewendet. Da der Transformer nach dem Aufruf des dart2js-Compilers aufgerufen wird, handelt es sich dabei auch um die generierten JavaScript-Dateien.

In den Zeilen 25-27 wird ermittelt, ob es sich um die generierte JavaScript-Datei oder die dazugehörige source map handelt. Alle anderen Dateien werden ignoriert.

Handelt es sich um die JavaScript-Datei, werden die Zeilen 29-33 ausgeführt. Es wird zunächst ein regulärer Ausdruck erstellt, der den Verweis auf die source map finden soll. Nach dem Ausdruck *sourceMappingURL=* folgt eine beliebige Anzahl von Zeichen, bis die Zeile mit *.dart.js.map* abgeschlossen wird. Die absolute Adresse des Ordners der Erweiterung wird mit dem Namen der generierten JavaScript-Datei und der zusätzlichen Dateiendung *.map* konkateniert. Die konstruierte Adresse wird anstelle der vorherigen eingetragen. Unter derselben Id der Quelldatei wird über die Methode *addOutput* der generierte Inhalt als neue Ausgabe definiert.

Wird stattdessen die Datei der source map behandelt, werden die Zeilen 35-40 ausgeführt. Die JSON-Datei wird in ein assoziatives Array umgewandelt, um den Zugriff zu erleichtern. Die Verweise auf die Quelldateien befinden sich in einer Liste, welche im Attribut *sources* gespeichert ist. Die absolute Adresse des Ordners der Erweiterung wird allen Dateipfaden in einer Schleife vorangestellt. Das manipulierte assoziative Array wird zurück in das JSON-Format überführt und ebenfalls unter derselben Id der Quelldatei als Ausgabe hinzugefügt.

74. Vgl. *Writing a Pub Transformer | Dart: Structured web apps*

3.3 Transformer zur Bereitstellung der QlikView und Qlik Sense Erweiterungen

Nach der Kompilierung erscheinen die generierten Dateien im *build*-Verzeichnis des Projektordners. Der Inhalt der generierten JavaScript-Datei muss jedoch mit einem vorgegebenen Namen im Ordner der Erweiterung gespeichert werden. Darüber hinaus müssen die source maps der QlikView und Qlik Sense Erweiterungen leicht manipuliert werden.

Obwohl alle verwendeten packages im *build*-Verzeichnis in den Ordner *packages* kopiert werden, weist die source map unterschiedliche Adressierungen auf. Alle durch pub oder git angeforderten packages sind im Pfad *packages/* angegeben, während die im Kapitel 3.1 erstellte Klassenbibliothek unter dem Pfad *../packages/* erscheint. Der Transformer hat daher auch die Aufgabe, die Pfade aller packages anzugleichen.

Das für den Transformer erstellte package erhält den Namen *qlikview_qlik_sense_deployment*. Wie jedem Transformer wird ihm die Abhängigkeit *barback* in der *pubspec.yaml*-Datei hinzugefügt.

Die Klasse QlikViewQlikSenseDeployment, die von Transformer erbt, ist in Listing **??** auf Seite **??** zu sehen. In der Zeile 7 taucht das package *dart:io* auf. Es wird für Operationen mit dem Dateisystem benötigt.

In Zeile 16 wird der Parameter *deploymentFolder* ausgelesen, der den Ordner der Erweiterung auf dem Zielserver beschreibt. Der Parameter *destinationScriptFile* enthält den Dateinamen, den die Skript-Datei im Ordner der Erweiterung tragen soll. In Zeile 18 wird der absolute Dateipfad aus beiden Parametern konstruiert.

In der auf alle Ausgabedateien angewendete Methode *apply* erfolgt eine Unterscheidung nach JavaScript- und source map-Datei sowie allen übrigen Dateien.

Für die generierte JavaScript-Datei wird ausschließlich der Befehl in den Zeilen 29 und 30 ausgeführt. Es erfolgt lediglich eine Speicherung des gelieferten Inhaltes in den Zieldateipfad mit dem im Parameter übergebenen Dateinamen.

Die Zeilen 32-37 enthalten die Anweisungen für die source map. Der Inhalt der JSON-Datei wird in ein assoziatives Array konvertiert. Alle Einträge der Liste im Attribut *sources* werden in einer Schleife durchlaufen. Sollte sich die Zeichenkette *../* in dem Pfad befinden, wird sie entfernt. Das Array landet abschließend als JSON-Datei mit unverändertem Namen im Ordner der Erweiterung.

Alle weiteren Dateien werden in Zeile 39 unverändert im Zielordner gespeichert.

3.4 Leitfaden zur Erstellung eines QlikView Extension Objects mit Dart

Das Kapitel 2.1 beschrieb die Erstellung eines QlikView Extension Objects mit JavaScript. Dabei handelte es sich um ein Kreisdiagramm, welches mithilfe der Bibliothek C3 gezeichnet wurde. Analog dazu behandelt dieses Kapitel die Implementierung der gleichen Funktionalität

in der Programmiersprache Dart. Dafür werden die im Kapitel 3.1 erstellte Klassenbibliothek sowie die in den Kapiteln 3.2 und 3.3 erstellten Pub Transformer verwendet.

3.4.1 Vorbereitung

Die einzige Datei, die sich ändert, ist *Script.js*. Aus diesem Grund kann der gesamte Ordner *C3Kreisdiagramm* kopiert und unter dem neuen Namen *DartQvC3Kreisdiagramm* gespeichert werden. Das dazugehörige Dart-Projekt erhält den Namen *dart_qv_c3_kreisdiagramm*. Es kann über die Vorlage *ubersimplewebapp* erstellt werden. Darüber erhält es bereits den für die Entwicklung erforderlichen Ordner *web*. Die darin befindliche Datei *index.html* sowie der Ordner *styles* können gelöscht werden. Neben der bereits enthaltenen *main.dart*-Datei sollte für dieses Beispiel noch die Datei *dart_qv_c3_kreisdiagramm.dart* erstellt werden.

3.4.2 pubspec.yaml

Die *pubspec.yaml*-Datei muss für die Anwendung der benötigten packages und für die Konfiguration der Transformer manipuliert werden. Ein Auszug der Datei ist im Listing **??** auf Seite **??** zu sehen.

In der Zeile 9 wird das package *async_await* aufgeführt. Die Quelle, von der es bezogen wird, ist ein git-Repository, welches in Zeile 10 angegeben ist. Bei dem package handelt es sich um einen Transformer, der verwendet werden muss, wenn die Schlüsselwörter *async* und *await* verwendet werden sollen.

> The Dart-to-JS compiler (dart2js) doesn't currently support asynchrony [...]. However, you can try out asynchrony in your Dart web app by using the async-await pub transformer when building the JavaScript version of your app.[75]

Beide Transformer und die Klassenbibliothek werden in den Zeilen 11-16 aufgeführt. Mit dem Eintrag *path* wird jeweils der Dateiordner des Projektes angegeben. Wenn es nicht anders konfiguriert wurde, werden alle Projekte durch den Dart Editor in demselben Ordner angelegt. Mit der Zeichenkette *..*, gefolgt von dem Ordnernamen des Projektes können die packages eingebunden werden.

In den Zeilen 18-26 werden die Transformer aufgelistet und konfiguriert. Der Quellcode wird für die Anwendung der Schlüsselwörter *async* und *await* vorbereitet. Anschließend erfolgt die Kompilierung mit aktivierter Unterstützung für Enum-Typen. Dem Transformer *qlikview_modify_source_map* werden der Name der Erweiterung sowie dessen Typ übergeben. Beide Parameter sind nötig, um die absoluten Adressen der Quelldateien in der source map eintragen zu können. Abschließend erhält der Transformer *qlikview_qlik_sense_deployment* den Pfad des Ordners sowie den Namen der Skript-Datei für die Bereitstellung der Erweiterung.

75. *A Tour of the Dart Language - Asynchrony support*

3.4.3 Die Klasse DartQvC3Kreisdiagramm

Die konkrete Klasse *QlikViewExtensionObject* ist in Listing **??** auf Seite **??** zu sehen. Sie weist im Vergleich zu ihrem JavaScript-Äquivalent aus Kapitel 2.1 einige Verbesserungen auf.

In der Zeile 8 wird einmalig das *div*-Element zum Zeichnen der Erweiterung erstellt. Die Getter-Methode *showLegend*, welche in Zeile 10 deklariert ist, erleichtert den Zugriff auf den Parameter zum Anzeigen der Legende.

In den Zeilen 13-23 wird die Methode *onCreate* implementiert, die nur einmalig für jede Instanz der Erweiterung aufgerufen wird. Die Methode ist mit dem Schlüsselwort *async* gekennzeichnet, was sie zu einer asynchronen Methode macht und ihr erlaubt, das *await*-Statement zu verwenden. Zunächst wird die Instanz der Klasse als globale Variable gespeichert. In den Entwicklerwerkzeugen des Browsers kann auf alle Attribute des Objektes zugegriffen werden. In Zeile 17 wird dem *contentDiv* das *div*-Element zum Zeichnen des Kreisdiagramms hinzugefügt. In der JavaScript-Version dieser Erweiterung wurden zuvor alle Kinderelemente entfernt, um den Ausgangszustand bei jedem Zeichnen wiederherzustellen. Darauf kann nun verzichtet werden. In Zeile 19 erfolgt das asynchrone Laden der Stylesheets. Auf den Abschluss wird jedoch nicht gewartet. Im Gegensatz dazu wird in Zeile 20 das *await* Schlüsselwort verwendet, um mit der weiteren Ausführung des Codes zu warten, bis beide JavaScript-Bibliotheken geladen wurden. Erst dann wird zum ersten Mal die *onPaint*-Methode aufgerufen.

Die *onPaint*-Methode wird in den Zeilen 25-44 implementiert. Wie bereits für die JavaScript-Version der Erweiterung wird die Konfiguration des Kreisdiagramms in einem assoziativen Array angelegt. Die Liste der Daten sowie die zu den Ids dazugehörigen Dimensionsnamen müssen dieses Mal nicht erstellt werden. Sie werden in den Zeilen 30 bzw. 36 direkt übergeben. Das Attribut *onclick* erhält eine anonyme Funktion mit genau zwei Parametern. Auch wenn der zweite Parameter nicht verwendet wird, muss er angegeben werden. Sollte die Funktion mit weniger oder mehr Parametern aufgerufen werden, führt dies zu einem Fehler. In der Zeile 33 wird mithilfe der Methode *selectValues* die Selektion durchgeführt. In der Zeile 34 ist die Übergabe eines optionalen benannten Parameters zu sehen. Auf die Übergabe kann verzichtet werden, da es sich bei dem Argument um den Standardwert handelt. Abschließend muss die Funktion *generate* der Bibliothek C3 aufgerufen werden. Da es sich bei dem Parameter um eine Kollektion handelt, wird sie mithilfe des Konstruktors *JsObject.jsify* in ein JavaScript-Objekt umgewandelt.

3.4.4 Registrierung der Erweiterung

In Listing **??** wird die Erweiterung registriert. Dazu wird eine Fabrikmethode übergeben, in der das qvaWrapper-Objekt als Parameter erwartet und direkt als Argument für den Konstruktor von DartQvC3Kreisdiagramm verwendet wird.

3.5 Leitfaden zur Erstellung einer QlikView Document Extension mit Dart

Die im Kapitel 2.2 mit JavaScript erstellte Document Extension wird im Folgenden mit der Programmiersprache Dart entwickelt.

3.5.1 Vorbereitung

Wie bereits im Kapitel 3.4.1 für die Erweiterung DartQvC3Kreisdiagramm soll der gesamte Ordner *DownloadArbeitsblattAlsPng* kopiert und unter dem Namen *DartDownloadArbeitsblatt AlsPng* gespeichert werden. Die einzige Datei, die sich ändert, ist *Script.js*. Das Dart-Projekt erhält den Namen *dart_download_arbeitsblatt_als_png*.

3.5.2 pubspec.yaml

Es gibt viele Gemeinsamkeiten mit der im Kapitel 3.4.2 erstellten *pubspec.yaml*-Datei. Die gleichen packages werden eingebunden und lediglich zwei Transformer werden anders konfiguriert. Die nötigen Änderungen sind in Listing **??** zu sehen. Für den Transformer *qlikview _modify_source_map* muss der Name der Erweiterung sowie der Typ angegeben werden. Weiterhin unterscheidet sich der Bereitstellungspfad des Transformers *qlikview_qlik_sense _deployment*. Der Name der Skript-Datei ist dagegen für alle QlikView Erweiterungen gleich.

3.5.3 Die Klasse DartDownloadArbeitsblattAlsPng

Listing **??** auf Seite **??** zeigt die Klasse *DartDownloadArbeitsblattAlsPng*. Im Vergleich zur JavaScript-Variante dieser Erweiterung gibt es nur wenige Verbesserungen. Eine davon ist das erleichterte Laden der Abhängigkeiten, was in Zeile 12 zu sehen ist. Der absolute Pfad muss nicht vom Entwickler konstruiert werden. Stattdessen reicht die Angabe der zu ladenden Dateien über relative Pfade. Darüber hinaus erleichtert das *await*-Schlüsselwort den Umgang mit dem asynchronen Methodenaufruf. Erst nachdem alle Bibliotheken geladen wurden, wird auch die Anweisung in Zeile 14 ausgeführt. Sie registriert *onKeyDown* als die aufzurufende Funktion beim Drücken einer Taste durch den Benutzer. Sollte der Benutzer noch vor dem Laden der Bibliotheken eine Taste drücken, wird die Funktion nicht aufgerufen. Die Erweiterung verwendet zwei JavaScript-Bibliotheken. Das Verwenden der Funktionen ist durch die komplizierte Syntax von *dart:js* erschwert. Die Aufrufe sind in den Zeilen 20 und 26 zu sehen.

Der Vorteil der optionalen Typisierung in Dart wird an der Getter-Methode *sheetSize* deutlich. Sie ist in den Zeilen 37-45 implementiert. Der Rückgabetyp ist *Rectangle*. Die Funktion *downloadSheetAsPng*, welche in Zeile 25 definiert ist, erwartet einen Parameter von diesem Typ. In Zeile 21 wird die Funktion mit der Getter-Methode *sheetSize* als Argument aufgerufen. Würde das Argument nicht vom Typ *Rectangle* sein, so würde es eine Warnung geben. In Zeile 40 wird darüber hinaus die Variable *element* vom Typ *Element* deklariert. In den Zeilen 41 und 42 erfolgt ein Zugriff auf dessen Attribut *offset*. Sollte sich in dem Typ *Element* weder

ein Feld, noch eine Methode mit dem Namen *offset* befinden, gäbe es ebenso eine Warnung. In JavaScript gibt es keine Typen und so würde es auch keine Benachrichtigung bei einem solchen Verstoß geben.

3.5.4 Registrierung der Erweiterung

Listing **??** zeigt die Registrierung der Document Extension. Die Funktion *registerQlikView-DocumentExtension* hat im Gegensatz zu der im Listing **??** auf Seite **??** zu sehenden Funktion *registerQlikViewExtensionObject* nur einen Parameter. Der Name der Erweiterung wird nicht im ersten Argument, sondern im Konstruktor der Fabrikmethode übergeben. Anders als für QlikView Object Extensions wird kein *qvaWrapper*-Objekt für die Fabrikmethode benötigt, denn es existiert für Document Extensions nicht.

3.6 Leitfaden zur Erstellung eines Qlik Sense Extension Objects mit Dart

Für das Qlik Sense Extension Object, das im Kapitel 2.3 mit JavaScript erstellt wurde, wird in den folgenden Ausführungen dieses Kapitels eine Variante in Dart erstellt.

3.6.1 Vorbereitung

Der gesamte Ordner *C3Kreisdiagramm*, der über die Qlik Sense Workbench erstellt wurde, sollte kopiert und unter dem Namen *DartQsC3Kreisdiagramm* gespeichert werden. Allerdings sind durch das Umbenennen des Ordnernamens einige weitere Änderungen nötig.

Auch die Dateien *C3Kreisdiagramm.qext* und *C3Kreisdiagramm.js* müssen in *DartQsC3Kreis-diagramm.qext* und *DartQsC3Kreisdiagramm.js* umbenannt werden. Damit die Dateien als Reiter in der Qlik Sense Workbench auftauchen, ist das Ändern der Dateinamen in der *wbfolder.wbl*-Datei erforderlich. In der Datei *DartQsC3Kreisdiagramm.qext* muss der Name der Erweiterung ebenfalls in *DartQsC3Kreisdiagramm* geändert werden.

Schließlich ist, wie im Kapitel 2.3.3 beschrieben, die Anpassung der Modulpfade in der Datei *c3.js* nötig. Wie im Listing **??** in Zeile 6902 und 6903 zu sehen, werden die Pfade in *extensions-/DartQsC3Kreisdiagramm/c3* und *extensions/DartQsC3Kreisdiagramm/d3.min* umgeändert. Das zu erstellende Dart-Projekt erhält den Namen *dart_qs_c3_kreisdiagramm*.

3.6.2 pubspec.yaml

Listing **??** zeigt einen Auszug der *pubspec.yaml*-Datei des Dart-Projektes. Die Klassenbibliothek *qlikview_qlik_sense_extensions* ist auch für die Anwendung mit Qlik Sense Erweiterungen entwickelt worden. Das package wird in Zeile 11 als Abhängigkeit hinzugefügt.

Im Gegensatz zu den QlikView Erweiterungen ist nur noch ein Transformer nötig. Es handelt sich um den Transformer *qlikview_qlik_sense_deployment*, welcher in der Zeile 9 als Abhängigkeit aufgelistet und in den Zeilen 16-18 konfiguriert wird. In Zeile 17 wird der Ordner für die

Bereitstellung angegeben. Der Name der JavaScript-Datei mit dem Haupteinstiegspunkt ist in Zeile 18 zu sehen. Anders als bei QlikView Erweiterungen ist der Name der Datei nicht immer gleich. Er entspricht dem Namen der Erweiterung.

3.6.3 Die Klasse DartQsC3Kreisdiagramm

Listing **??** auf Seite **??** zeigt die Klasse *DartQsC3Kreisdiagramm*. Die Basisklasse verlangt die Parameter *extensionObject* und *extensionData*. Daher werden sie in der Zeile 7 im Konstruktor erwartet und in Zeile 9 dem Konstruktor der Basisklasse übergeben. Darüber hinaus werden in Zeile 8 die Felder *c3Js*, *c3Css* und *styleCss* initialisiert. Es handelt sich um die durch RequireJS angeforderten Module. *c3Js* wurde als JavaScript-Modul geparst. Das dazugehörige Feld wird in Zeile 11 mit dem Typ *JsObject* deklariert. Bei den Modulen *c3Css* und *styleCss* handelt es sich um Text und daher werden beide auch als Feld vom Typ *String* in Zeile 12 deklariert.

Weiterhin erfolgt die Deklaration und Initialisierung des *chartDivElement* in der Zeile 13. Es wird als Container für das zu zeichnende Diagramm verwendet. Noch während der Initialisierung wird mithilfe des cascade operator die CSS-Klasse *kreisdiagramm* hinzugefügt.

In der JavaScript-Variante wurden die Objekte *extensionObject* und *extensionData* über jeweils eine globale Variable verfügbar gemacht. In Zeile 17 wird dagegen lediglich das Objekt der Klasse über die Referenz *this* als globale Variable gespeichert. Darüber lassen sich alle Felder der Klasse aufrufen und somit ist auch der Zugriff auf die beiden Objekte möglich.

In den Zeilen 19 und 20 erfolgt das Hinzufügen der Stylesheets zum *head*-Element des Dokumentes. Das package *dart:html* bietet Klassen und Funktionen für solche Anwendungsfälle an.

Das *chartDivElement* zum Zeichnen des Diagramms wird dem *contentDiv* in Zeile 22 hinzugefügt. Wie alle Anweisungen innerhalb der *onCreate*-Methode wird auch diese nur einmalig aufgerufen. Bei jedem Zeichnen muss daher nicht darauf geachtet werden, den Ausgangszustand wiederherzustellen. Der Vorteil ergibt sich aus der Nutzung der Klassenbibliothek *qlikview_qlik_sense_extensions*. Ohne die strukturgebenden Methoden *onCreate* und *onPaint* ist es erforderlich, bei jedem Zeichnen alle Elemente im *contentDiv* erneut zu erstellen. Die Notwendigkeit wird am Listing **??** auf Seite **??** des Kapitels 2.3.4 erkennbar. In Zeile 12 wird das *div*-Element erstmalig als Parameter der Funktion *paint* verfügbar. Eine initiale Konfiguration kann nicht erfolgen. Daher wird in den Zeilen 29-32 der Inhalt des Elementes entfernt und anschließend neu konstruiert.

In der *onPaint*-Methode, welche in den Zeilen 27-43 des Listings **??** implementiert ist, wird ausschließlich C3 verwendet um das Diagramm zu generieren. Wie bereits bei der Klasse *DartQvC3Kreisdiagramm* aus dem Kapitel 3.4.3 bleibt dem Anwender das Analysieren der gelieferten Daten erspart. Die Daten werden durch Qlik Sense anders strukturiert, doch die Klassenbibliothek *qlikview_qlik_sense_extensions* abstrahiert die Struktur. Auf die Daten kann mithilfe der Getter-Methoden *dataAsList* und *idNameMap* zugegriffen werden. Sie werden in den Zeilen 32 und 37 aufgerufen.

3.6.4 Registrierung der Erweiterung

Listing **??** zeigt die Definition des Moduls sowie die Registrierung der Erweiterung. In den Zeilen 12 und 13 werden die anzufordernden Module als Zeichenketten in einer Liste übergeben. Die geparsten JavaScript-Objekte sowie die eingelesenen Texte werden jeweils den Parametern in den Zeilen 14 und 15 als Objekte vom Typ *JsObject* bzw. *String* übergeben.

In den Zeilen 16-25 erfolgt die Rückgabe des definierten Moduls. Dazu werden den Attributen *initialProperties* - in Zeile 17 – und *definition* - in Zeile 18 – die deserialisierten JavaScript-Objekte der JSON-Dateien aus dem Kapitel 2.3.2 übergeben.

In den Zeilen 19-24 wird dem Attribut *paint* eine JavaScript-Funktion zugewiesen, in der die Registrierung stattfindet. Da sich das benötigte *extensionObject* in der *this*-Referenz der Funktion befindet, muss der Konstruktor *JsFunction.withThis* verwendet werden. Im ersten Parameter *paintThis* wird die *this*-Referenz gespeichert. Weiterhin werden der Funktion die Parameter *$contentDiv* und *extensionData* in den Zeilen 19 bzw. 20 übergeben.

Die Funktion *registerQlikSenseExtensionObject* erwartet zwei Parameter. Das Objekt *extensionData* muss bereits als erstes Argument übergeben werden. Es enthält die *objectId*, welche benötigt wird, um die Erstellung bzw. das Zeichnen der Erweiterung zu steuern. Der zweite Parameter ist die Fabrik-Methode für die Erstellung der Instanzen der Klasse *DartQsC3Kreisdiagramm*. Im Konstruktor werden bereits alle benötigten Module übergeben.

4 Ergebnisse

Die Intention des Autors dieser Arbeit war es, herauszufinden, ob und in welchem Ausmaß die Programmiersprache Dart für die Entwicklung von Erweiterungen für QlikView und Qlik Sense genutzt werden kann. Bei der Erstellung der Erweiterungen gibt es strenge Richtlinien. Qlik Sense verwendet RequireJS, um die Erweiterungen als Module zu definieren. QlikView verwaltet die Erweiterungen hingegen mittels einer eigenen Implementation. Zu Beginn der Erstellung dieser Arbeit war unklar, ob der aus Dart generierte JavaScript-Quellcode in diese Strukturen eingefügt werden kann. Das Ergebnis der Arbeit ist, dass die Interaktion ohne Einschränkungen möglich ist.

In den weiteren Ausführungen dieses Kapitels werden die erarbeiteten Vor- und Nachteile der Entwicklung von Erweiterungen mit Dart erläutert. Abschließend wird eine Empfehlung für die Anwendungsgebiete ausgesprochen.

4.1 Abstraktion der APIs

Mithilfe der Programmiersprache Dart wurde eine Bibliothek entwickelt, die es ermöglicht, über dieselben Schnittstellen auf unterschiedliche Implementierungen in QlikView und Qlik Sense zurückzugreifen. Dies ist als klarer Vorteil zu sehen, denn beim Wechsel von QlikView zu Qlik Sense ist auch die Einarbeitung in die veränderte API nötig. Mit der entwickelten Bibliothek

bleibt die erneute Einarbeitung weitestgehend erspart. Außerdem können bereits für QlikView entwickelte Erweiterungen sehr einfach auch in Qlik Sense integriert werden. Lediglich die Deklaration der Dimensionen, Formeln und Parameter muss in das unterschiedliche Format überführt werden.

4.1.1 Leichtere Einarbeitung durch automatische Vervollständigung

Bei der Entwicklung gegen die QlikView Ajax API bzw. die Qlik Sense Extensions API gibt es zwei Möglichkeiten der Einarbeitung.

Die Dokumentation kann verwendet werden, um nach den benötigten Funktionen zu suchen. Doch besonders im Fall von QlikView ist diese Dokumentation teilweise unvollständig oder sogar fehlerhaft. Dies lässt sich am Beispiel der Methode *SelectValuesInColum* erkennen. Dort fehlen die Position und die Beschreibung des Parameters *colNo*.[76]

Alternativ können im Skript globale Variablen angelegt werden, um bei der Ausführung der Erweiterung in der JavaScript-Konsole mithilfe der automatischen Vervollständigungsfunktion Auskunft über die verfügbaren Funktionen zu erhalten. Diese Vorgehensweise hat jedoch den Nachteil, dass die zu übergebenden Parameter nicht ersichtlich sind.

Durch die Funktionalität der automatischen Vervollständigung im Dart Editor ist die Entwicklung der Erweiterung deutlich vereinfacht, auch wenn sich der Vorteil nur auf die im Rahmen dieser Arbeit entwickelten Klassenbibliothek bezieht. Doch die Bibliothek kann beliebig erweitert werden und somit in Zukunft die Entwicklung und Einarbeitung deutlich verkürzen. Im Anhang I sind Beispiele der automatischen Codegenerierung sowie Vorschläge für automatische Vervollständigungen aufgeführt.

4.1.2 Verbesserte Lesbarkeit und nachvollziehbarere Struktur

Besonders die QlikView Ajax API sorgt durch fehlleitende Funktionsnamen und die fehlende Struktur für Verwirrung bei der Entwicklung von QlikView Erweiterungen. Der Funktion *AddExtension* muss beispielsweise eine Funktion übergeben werden, die bei jedem Zeichnen aufgerufen wird. Einige Entwickler machen den Fehler, in dieser Funktion die von der Erweiterung benötigten Abhängigkeiten zu laden. Somit werden bei jedem Zeichnen dem *head*-Element redundante Skripte und Stylesheets angehängt. Ein Beispiel dafür kann im Anhang H nachgelesen werden. Durch das Verwenden von RequireJS in Qlik Sense wurde der Fehler weitestgehend entschärft. Der Eintrag *paint* in der Definition des Moduls signalisiert eindeutig, dass die diesem Attribut übergebene Funktion zum Zeichnen verwendet werden soll. Dennoch ist es auch hier nur durch Kenntnis der Bibliothek RequireJS möglich, herauszufinden, in welcher Region des Skriptes die Abhängigkeiten verarbeitet werden sollen.

Durch den Entwurf der Schnittstellen *OnCreate* und *OnPaint* in der in Dart entwickelten Bibliothek, ist eindeutig erkennbar, welche Operationen in welcher Methode ausgeführt werden sollen.

76. Vgl. QlikTech, *JsDoc Reference - Qv.Document.Object.Data - SelectValuesInColumn*

Auch die Lesbarkeit wird durch die mögliche Verwendung des *await*-Schlüsselwortes deutlich positiv beeinflusst. Die Häufigkeit der öffnenden und schließenden geschweiften Klammern wird dadurch auf eine leicht überschaubare Anzahl reduziert.

4.2 Build tools

Die durch Dart bereitgestellten Transformer ermöglichten es, die kompilierten Dateien in den Ordner der Erweiterungen auf dem jeweiligen Server zu kopieren. Transformer können als package eingebunden werden und lassen sich über die Konfigurationsmöglichkeiten in der *pubspec.yaml*-Datei sehr einfach an das Zielsystem anpassen. Daher muss ein solcher Transformer nicht immer wieder neu implementiert werden. Weiterhin konnte durch den Transformer eine Modifizierung an den kompilierten Dateien vorgenommen werden, um den generierten JavaScript-Code mithilfe der SourceMap einfacher zu debuggen. Doch das Modifizieren und Kopieren wurde erst dadurch nötig, da sowohl QlikView als auch Qlik Sense eine JavaScript-Datei für das Verhalten der Erweiterung erwarten. Würde die Entwicklung direkt an der JavaScript-Datei erfolgen, bliebe das Kompilieren, Modifizieren und Kopieren der Dateien erspart. Daher kann die Möglichkeit der Entwicklung der Transformer in Dart nicht wirklich als Vorteil gewertet werden. Vielmehr dienen die Transformer als Werkzeuge, welche die Nachteile der Entwicklung in Dart zumindest teilweise aufheben können.

4.3 Empfehlung

Durch die strengen Vorgaben bei der Entwicklung von Erweiterungen für Qlik Sense und QlikView bedarf es einem höheren Zeitaufwand, wenn die Entwicklung mit Dart stattfinden soll. Außerdem ist die Verwendung von JavaScript-Bibliotheken in Dart erschwert und erfordert eine zusätzliche Einarbeitung.

Die Entwicklung von Erweiterungen mithilfe von JavaScript lässt sich sehr schnell umsetzen. Dies gilt insbesondere für Qlik Sense, da dort die Erweiterungen mit einem Klick über die Qlik Sense Workbench erstellt werden können. Nachdem Änderungen im Skript gespeichert wurden, reicht ein neues Laden der Seite aus, um die Resultate im Browser zu testen. Im Gegensatz dazu muss im Dart Editor erst ein Projekt erstellt und für die Bereitstellung in den Ordner der Erweiterung konfiguriert werden. Nach jeder Änderung muss eine erneute Kompilierung erfolgen. Während der Anfertigung dieser Arbeit lag die durchschnittliche Kompilierzeit bei etwa 15 Sekunden. Dadurch kann der Entwickler in seiner Produktivität bereits deutlich eingeschränkt werden. Sollte es also möglich sein, die gewünschte Erweiterung sehr einfach mit den durch JavaScript bereitgestellten Funktionalitäten zu entwickeln, kann auf den erheblichen zusätzlichen Aufwand verzichtet werden.

Die Entwicklung mit Dart lohnt sich dann, wenn komplexe und leicht skalierbare Erweiterungen entwickelt werden sollen. Darüber hinaus ist Dart besonders für die Entwicklung im Team geeignet. Bei der Integration von Code können einige Fehler bereits vor der Kompilierung ausfindig gemacht werden. Sollten Methoden aufgerufen werden, die nicht deklariert oder

implementiert sind, so erscheint eine dementsprechende Fehlermeldung. Gleiches gilt für die Übergabe von Argumenten, deren Datentypen mit denen der Parameter inkompatibel sind.

5 Ausblick

Die in dem Rahmen dieser Arbeit erstellte Klassenbibliothek zur Vereinfachung der QlikView und Qlik Sense API umfasst verhältnismäßig wenige Funktionen. Sie kann jedoch beliebig erweitert werden, um auch alle weiteren Funktionalitäten der jeweiligen Implementationen auszunutzen.

Die Pub Transformer könnten genutzt werden, um die Ladezeiten der QlikView Erweiterungen deutlich zu reduzieren. Für gewöhnlich wird bei einer QlikView Erweiterung zuerst die *Script.js*-Datei angefordert. Erst nachdem sie eingelesen wurde, werden die restlichen JavaScript-Dateien über weitere HTTP-Anfragen geladen. Es ist sogar möglich, dass die angeforderten JavaScript-Dateien wiederum weitere Dateien anfordern. Die Dateien werden im schlimmsten Fall alle sequenziell angefragt. Dabei muss jedes Mal auf die Antwort des Servers gewartet werden. Die gesamte Ladezeit der Erweiterung ist dadurch erheblich erhöht. Mithilfe eines Pub Transformers könnten die Inhalte aller JavaScript-Dateien zu einer einzigen Datei konkateniert werden.

Darüber hinaus ist es auch denkbar, die Deklaration von Dimensionen, Formeln und Parametern weiter zu vereinfachen. Die Art und Weise, in der Qlik Sense solche Deklarationen vornimmt, ist weitaus kompakter, als es in QlikView der Fall ist. Wünschenswert wäre es, die gleiche Syntax auch für QlikView anwendbar zu machen. Ein Pub Transformer könnte das JSON-Format in die von QlikView verwendete Struktur übersetzen. Die *Definition.xml*- sowie die *Properties.qvpp*-Datei würde daraus generiert werden. QlikView erlaubt es nicht, Bezeichner für Parameter festzulegen. Die Referenzierung erfolgt über die Reihenfolge, in welcher die Parameter in der *Definition.xml*-Datei aufgelistet sind. Die Bezeichner könnten jedoch in einem Skript angelegt werden, welches ebenfalls generiert wird. So würde beispielsweise die von QlikView verwendete Referenz *Text0.text* zum Anfang des Skriptes der Variable *showLegend* zugewiesen werden. Eine solche Implementierung wäre sehr zeitintensiv, da es einige Unterschiede bei der Erstellung des Konfigurationsdialoges gibt. Doch sollte eine solche Umsetzung gelingen, wäre es möglich, in QlikView entwickelte Extension Objects auch in Qlik Sense zu verwenden. Die Einarbeitung in die Erstellung der *Definition.xml*- sowie der *Properties.qvpp*-Datei könnte somit komplett erspart bleiben. Die einzigen Änderungen, die darüber hinaus noch nötig sind, wären das Anpassen der Dateinamen und das Anfordern der weiteren Ressourcen über RequireJS.

Glossar

QlikView	eine Business Intelligence-Software des Softwareunternehmens Qlik zur sekundenschnellen Analyse von Geschäftsdaten
Qlik Sense	eine Business Intelligence-Software des Softwareunternehmens Qlik, die als Nachfolger für QlikView verstanden werden kann
Sheet Object	positionierbare Objekte in QlikView und Qlik Sense, die zur Visualisierung von Daten mithilfe von Diagrammen, Tabellen und Listen eingesetzt werden
Document Extension	eine Erweiterung, die das Verhalten bzw. das Aussehen eines QlikView Dokumentes manipulieren kann
Extension Object	eine Erweiterung für QlikView und Qlik Sense die sich ähnlich verhält wie ein Sheet Object. Eine solche Erweiterung erhält Daten der Anwendung, welche von ihr visualisiert und zur Selektion angeboten werden können
HTML	Die Hypertext Markup Language ist eine textbasierte Auszeichnungssprache zur Strukturierung von Webseiten
CSS	Die Cascading Style Sheets dienen der textbasierten Erstellung von Darstellungsregeln von beispielsweise HTML-Dokumenten und helfen Darstellung und Strucktur der Inhalte zu trennen
JavaScript	Eine Skriptsprache für das Erstellen von Webanwendungen
JSON	Die JavaScript Object Notation ist ein textbasiertes Datenformat zum Datenaustausch zwischen Anwendungen
XML	Die Extensible Markup Language ist eine textbasierte Auszeichnungssprache für hierarchisch strukturierte Daten
Dart	Eine Programmiersprache für das Erstellen von Anwendungen, die im Browser und auf dem Server mit integrierter Dart-VM lauffähig sind.
Dart2js	Ein Transcompiler zur Übersetzung von Dart-Quellcode nach JavaScript um in Dart entwickelte Webanwendungen auch im Browser ohne Dart-VM lauffähig zu machen

Literatur

A Tour of the Dart Language - Asynchrony support. Feb. 2015. URL: https://www.dartlang.org/docs/dart-up-and-running/ch02.html#asynchrony.

Akopkokhyants, S.
Mastering Dart. Packt Publishing, 2014. ISBN: 9781783989577. URL: https://books.google.de/books?id=hYGOBQAAQBAJ.

Assets and Transformers | Dart: Structured web apps. Jan. 2015. URL: https://www.dartlang.org/tools/pub/assets-and-transformers.html.

Belchin, Moises und Patricia Juberias.
Web Programming with Dart. Springer, 2015.

Buckett, C.
Dart in Action. Manning, 2013. ISBN: 9781617290862. URL: http://books.google.com.ng/books?id=ECGJMAEACAAJ.

C3.js | D3-based reusable chart library. Dezember 2014. URL: http://c3js.org/.

C3.js | D3-based reusable chart library - Column Oriented Data. Dezember 2014. URL: http://c3js.org/samples/data_columned.html.

C3.js | D3-based reusable chart library - Data Name. Dezember 2014. URL: http://c3js.org/samples/data_name.html.

Cowart, Jim.
Five Helpful Tips When Using RequireJS - Tech.pro. Jan. 2015. URL: http://tech.pro/blog/1561/five-helpful-tips-when-using-requirejs.

Dart Style Guide | Dart: Structured web apps. Jan. 2015. URL: https://www.dartlang.org/articles/style-guide/.

dart:js.JsObject API Docs - callMethod. Jan. 2015. URL: https://api.dartlang.org/apidocs/channels/be/dartdoc-viewer/dart:js.JsObject#id_callMethod.

dart:js.JsObject API Docs – JsObject-jsify. Jan. 2015. URL: https://api.dartlang.org/apidocs/channels/be/dartdoc-viewer/dart:js.JsObject#id_JsObject-jsify.

dart2js: The Dart-to-JavaScript Compiler. Feb. 2015. URL: https://www.dartlang.org/tools/dart2js/.

ECMA International.
Dart Programming Language Specification. 1. Aufl. Juni 2014. URL: http://www.ecma-international.org/publications/files/ECMA-ST-ARCH/ECMA-408%201st%20edition%20June%202014.pdf.

— *Dart Programming Language Specification*. 2. Aufl. Dez. 2014. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-408.pdf.

Freeman, Adam.
Pro AngularJS -. 1. Aufl. New York: Apress, 2014. ISBN: 978-1-430-26449-1.

Issue 104 - dart - Support for await in Dart - Dart - Structured Web Programming. Jan. 2015. URL: https://code.google.com/p/dart/issues/detail?id=104.

Issue 88 - dart - Enhancement: Enum - Dart - Structured Web Programming. Jan. 2015. URL: https://code.google.com/p/dart/issues/detail?id=88.

Kopec, D.
Dart for Absolute Beginners. Expert's voice in Web development. Apress, 2014. ISBN: 9781430264811. URL: http://books.google.de/books?id=GytKngEACAAJ.

O'Donovan, Mark.
»Qlik Sense For Beginners«. In: (2014).

Odell, D.
Pro JavaScript Development: Coding, Capabilities, and Tooling. Expert's voice in Web development. Apress, 2014. ISBN: 9781430262695. URL: https://books.google.de/books?id=pyZIBAAAQBAJ.

Pub Package Layout Conventions | Dart: Structured web apps. Jan. 2015. URL: https://www.dartlang.org/tools/pub/package-layout.html.

Pubspec Format | Dart: Structured web apps. Jan. 2015. URL: https://www.dartlang.org/tools/pub/pubspec.html.

QlikTech.
Business Intelligence and Data Visualization Software Company | Qlik. Feb. 2014. URL: http://www.qlik.com/en/company.

— *HTML Select in Extensions | Qlik Community*. Dezember 2014. URL: http://community.qlik.com/thread/53335.

— *JsDoc Reference - Qv.Document - GetCurrentSelections*. Jan. 2015. URL: http://qlikcommunity.s3.amazonaws.com/misc/symbols/Qv.Document.html#GetCurrentSelections.

— *JsDoc Reference - Qv.Document.Object.Data - SelectValuesInColumn*. Jan. 2015. URL: http://qlikcommunity.s3.amazonaws.com/misc/symbols/Qv.Document.Object.Data.html#SelectValuesInColumn.

— *Launching Qlik Sense Workbench*. Jan. 2015. URL: https://help.qlik.com/sense/en-us/developer/Subsystems/Workbench/Content/qswb_wb_launch.htm.

— *Qlik Sense Architecture*. Jan. 2015. URL: http://qvfiles.s3.amazonaws.com/landing_pages/DK/DK-VYW_Qlik_SenseArchitecture.pdf.

— *Qlik Sense for Developers - eachDataRow method*. Jan. 2015. URL: https://help.qlik.com/sense/en-us/developer/Subsystems/Workbench/Content/BuildingExtensions/API/backendApi/MethodEachDataRow.htm.

— *Qlik Sense Workbench*. Jan. 2015. URL: https://help.qlik.com/sense/en-us/developer/Subsystems/Workbench/Content/qswb_wb_intro.htm.

— *Qlik Sense Workbench requirements and assumptions*. Jan. 2015. URL: https://help.qlik.com/sense/en-us/developer/Subsystems/Workbench/Content/qswb_wb_requirements.htm.

— *QlikView Ajax JavaScript Library - AddExtension Function*. Jan. 2015. URL: http://qlikcommunity.s3.amazonaws.com/misc/symbols/Qv.html#AddExtension.

— *QlikView Ajax JavaScript Library - LoadExtensionScripts*. Jan. 2015. URL: http://qlikcommunity.s3.amazonaws.com/misc/symbols/Qv.html#LoadExtensionScripts.

— *QlikView Extension Defintion file*. Dezember 2014. URL: http://qlikcommunity.s3.amazonaws.com/misc/Qv11/definition.htm.

— *QlikView Properties Pages*. Dezember 2014. URL: http://qlikcommunity.s3.amazonaws.com/misc/Qv11/qvpp.htm.

— *Third Party Software Attributions, Copyrights, Licenses and Disclosure - Qlik® Sense*. Jan. 2015. URL: http://www.qlik.com/~/media/Files/info/license-terms-third-party/Qlik-Sense-Third-Party-License-Terms.ashx.

Redmond, Stephen.
Qlik Tips: Extensions in Qlik Sense. Jan. 2015. URL: http://www.qliktips.com/2014/07/extensions-in-qlik-sense.html.
— *QlikView for Developers Cookbook*. Packt Publishing Ltd, 2013.
— *QlikView Server and Publisher*. Packt Publishing Ltd, 2014.
RequireJS API - Load JavaScript Files. Jan. 2015. URL: http://requirejs.org/docs/api.html#jsfiles.
Specify a Text File Dependency. Dezember 2014. URL: http://requirejs.org/docs/api.html#text.
Stasieńko, Justyna.
»BI IN-MEMORY–THE NEW QUALITY OF BUSINESS INTELLIGENCE SYSTEMS«. In: *INFORMATION SYSTEMS IN MANAGEMENT IX* ().
Walrath, K. und S. Ladd.
Dart: Up and Running. Nutshell handbook. O'Reilly Media, 2012. ISBN: 9781449330897. URL: https://books.google.de/books?id=xLxqmzlZHU4C.
Walther, Stefan.
»A Comprehensive List of QlikView Object Extensions (01/2013)«. In: (2013). URL: http://www.qlikblog.at/1939/a-comprehensive-list-of-qlikview-object-extensions-012013/.
— *qvAutoRefresh/Definition.xml at master · QlikDev/qvAutoRefresh · GitHub*. Dezember 2014. URL: https://github.com/QlikDev/qvAutoRefresh/blob/master/AutoRefresh/Definition.xml.
— *qvD3BulletCharts/Script.js at master · QlikDev/qvD3BulletCharts · GitHub*. Dezember 2014. URL: https://github.com/QlikDev/qvD3BulletCharts/blob/master/D3BulletCharts/Script.js.
Writing a Pub Transformer | Dart: Structured web apps. Jan. 2015. URL: https://www.dartlang.org/tools/pub/transformers/.

Anhang

A Screenshots von QlikView

B Properties.qvpp-Datei des C3Kreisdiagramm Extension Objects

C Screenshots von Qlik Sense

D Beispiel-Screenshot der DownloadArbeitsblattAlsPng-Erweiterung

E initialProperties für ein Qlik Sense Extension Object in Dart-Syntax

Im Listing **??** ist die testweise Portierung der *initialProperties.js*-Datei zu sehen. In den Zeilen 4, 5 und 6 ist die Erstellung von JavaScript-Proxy-Objekten mithilfe des Konstruktors *JsObject.jsify* zu sehen. Dies es notwendig, denn in Dart erstellte Kollektionen sind nicht mit

JavaScript kompatibel, auch wenn es sich um leere Listen wie in den Zeilen 4 und 5 handelt. Daher müssen sie konvertiert werden.

F Klassendiagramm des Projektes qlikview_qlik_sense_extensions

G Listings zu source maps

H Fehlerhafte Verwendung der QlikView Ajax API

Im Listing **??** ist ein Auszug aus der *Script.js*-Datei der QlikView Extension *Animated Scatter Chart* zu sehen. In der Zeile 3 wird der Funktion *AddExtension* die Callback-Funktion übergeben, welche unter anderem auch den Aufruf der Funktionen *LoadCSS* in Zeile 30 und *LoadExtensionScripts* in Zeile 37 enthält. Doch da die Callback-Funktion bei jedem Zeichnen aufgerufen wird, werden diese Ressourcen auch bei jedem Zeichnen erneut dem *head*-Element des Dokuments hinzugefügt.

Die Abbildung **??** zeigt, dass diese Ressourcen tatsächlich mehrfach im *head*-element des Dokuments auftauchen. In Abbildung **??** ist darüber hinaus zu sehen, dass diese Dateien bei deaktiviertem Browsercache sogar mehrfach vom Server angefragt und erneut heruntergeladen werden.

I Abbildungen zur erleichterten Entwicklung mit Dart

Mit einem Rechtsklick im Editor öffnet sich ein Kontextmenü, in dem unter anderem der Eintrag *Quick Fix* verfügbar ist. Alternativ kann die Funktion mit der Tastenkombination Steuerung + 1 aufgerufen werden. Bei Fehlermeldungen und Warnungen bietet sie Lösungsvorschläge an und generiert automatisch den passenden Code.

In Abbildung **??** ist zu sehen, dass der Klasse *MyQlikViewExtensionObject* der Aufruf des Konstruktors der Basisklasse fehlt. Die automatische Implementierung des Konstruktors wird angeboten. Anschließend wird eine Warnung ausgegeben, dass zwei Schnittstellen in der konkreten Klasse noch immer nicht implementiert sind.

Abbildung **??** zeigt, dass in diesem Fall die Generierung der Methodenköpfe angeboten wird. Die vollständige Struktur der Klasse ist in Abbildung **??** zu sehen. Nach dem Eingeben von *this.* erscheint eine Liste von Vorschlägen für automatische Vervollständigungen. In der Liste sind lediglich die im Kontext sichtbaren Felder und Methoden aufgeführt. Diese Hilfestellungen erleichtern die Einarbeitung, beschleunigen die Entwicklung und beugen Fehlern vor.