RUHR-UNIVERSITÄT BOCHUM

RUB

RUHR-UNIVERSITÄT BOCHUM

Bridging the Gap: Verlustfreie und sichere Umwandlung von XML-Datenstrukturen ins JSON-Format

Jan Holthuis

Bacherlorarbeit – 28. Juni 2017.
Lehrstuhl für Netz- und Datensicherheit.

Betreuer: Prof. Dr. Jörg Schwenk
Berater: B. Sc. Paul Rösler

hgi Lehrstuhl für Netz- und Datensicherheit

Zusammenfassung

Bei XML und JSON handelt es sich um insbesondere im Mobile- und Web-Bereich konkurrierende, menschenlesbare Formate für die Speicherung und den Austausch hierarchisch strukturierter Daten. Je nach eingesetzter API, Programmiersprache oder Programmbibliothek kann es dabei sinnvoll sein, die Daten zunächst in das jeweils andere Format zu überführen. Die vorliegende Arbeit befasst sich mit der verlustfreien Translation von XML-Daten in JSON-Strukuren. Dazu werden bestehende Verfahren auf Genauigkeit und Sicherheit hin untersucht und schlussendlich ein verlustfreies Verfahren zur Übersetzung von XML-Daten in JSON sowie in Rückrichtung entwickelt und vorgestellt.

Eidesstattliche Erklärung

Ich erkläre, dass ich keine Arbeit in gleicher oder ähnlicher Fassung bereits für eine andere Prüfung an der Ruhr-Universität Bochum oder einer anderen Hochschule eingereicht habe.

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen, die anderen Quellen dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen kenntlich gemacht. Dies gilt sinngemäß auch für verwendete Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen.

Ort, Datum	Unterschrift

Erklärung zum Urheberrecht

Ich erkläre mich damit einverstanden, dass meine Abschlussarbeit am Lehrstuhl *Netz- und Datensicherheit (NDS)* dauerhaft in elektronischer und gedruckter Form aufbewahrt wird und dass die Ergebnisse aus dieser Arbeit unter Einhaltung guter wissenschaftlicher Praxis in der Forschung weiterverwendet werden dürfen.

Ort, Datum	Unterschrift

Inhaltsverzeichnis

1 Einleitung

Für den implementationsunabhängigen Austausch von zugleich menschen- als auch maschinenlesbaren Daten hat sich die Extensible Markup Language (XML) bewährt. In bestimmten Bereichen wie Web-APIs hat die JavaScript Object Notation (JSON) das bewährte XML-Format jedoch inzwischen überflügelt.

Dabei kann neben der spezifischen Situation auch die eingesetzte Programmiersprache, die Unterstützung durch das zugrundeliegende Framework oder die persönliche Präferenz des Entwicklers den Ausschlag geben, welches Format ein Webservice oder eine Programmbibliothek unterstützt.

Zwecks Interoperabilität zwischen verschiedenen Teilen einer Anwendung kann es daher notwendig werden, Daten von einem Format temporär in das jeweils andere zu überführen und später wieder in das ursprüngliche Format zu bringen.

Dabei sollen Daten, die von der Anwendungslogik nicht verändert wurden, auch nach der Rückübersetzung ins Ursprungsformat unverändert bleiben. Damit dies gewährleistet ist, darf das zugrunde liegende Konversionverfahren keine Informationen bei der Umwandlung verwerfen – es muss also verlustlos arbeiten.

Das konkrete Konversionsverfahren ist dabei abhängig vom jeweiligen Ausgangsformat, d. h. die Umwandlungsverfahren für die beiden Richtungen

1. JSON $\rightarrow$ XML $\rightarrow$ JSON und
2. XML $\rightarrow$ JSON $\rightarrow$ XML

haben jeweils eigene Anforderungen und sind getrennt voneinander zu betrachten.

Während die Abbildung beliebiger JSON-Datenstrukturen in XML – zumindest bei oberflächlicher Betrachtung – trivial erscheint, ist dies beim verlustlosen Transfer von XML-Daten ins JSON-Format keineswegs der Fall.

Ziel der vorliegenden Arbeit ist es daher, ein sicheres und verlustloses Verfahren zur Konversion von XML-Dokumenten in das JSON-Format zu finden.

Dazu wird eine Reihe von bereits verfügbaren Programmbibliotheken analysiert, die XML-Daten in JSON abbilden und aus der resultierenden JSON-Datenstruktur wieder ein XML-Dokument erstellen können.

Da keines der analysierten Verfahren die zuvor aufgestellten Kriterien in Gänze erfüllt, wird das vollständigste Konversionsprogramm weiterentwickelt und im Anschluss erneut evaluiert.

1.1 Motivation

Das Aufkommen des sogenannten *Web 2.0* und die zunehmenden Vernetzung durch das Internet of Things (IoT) ging mit einer erhöhten Verfügbarkeit von öffentlichen Web-APIs einher. Als Datenformat wird dabei häufig JSON oder XML verwendet.

Neben XML-basierten Webservices, die beispielsweise das SAML-Framework, SOAP, oder XML Remote Procedure Call (XML-RPC) verwenden, wird XML auch in einer Vielzahl weiterer Einsatzbereiche eingesetzt. So dient XML den Dateiformaten RSS/ASF, MathML, Scalable Vector Graphics (SVG) oder Extensible HyperText Markup Language (XHTML) als Basis. Auch die gängigen Office-Dateiformate – das OpenDocument Format (ODF), Microsofts Office Open XML (OOXML) und Apples iWorks – bauen auf XML auf.

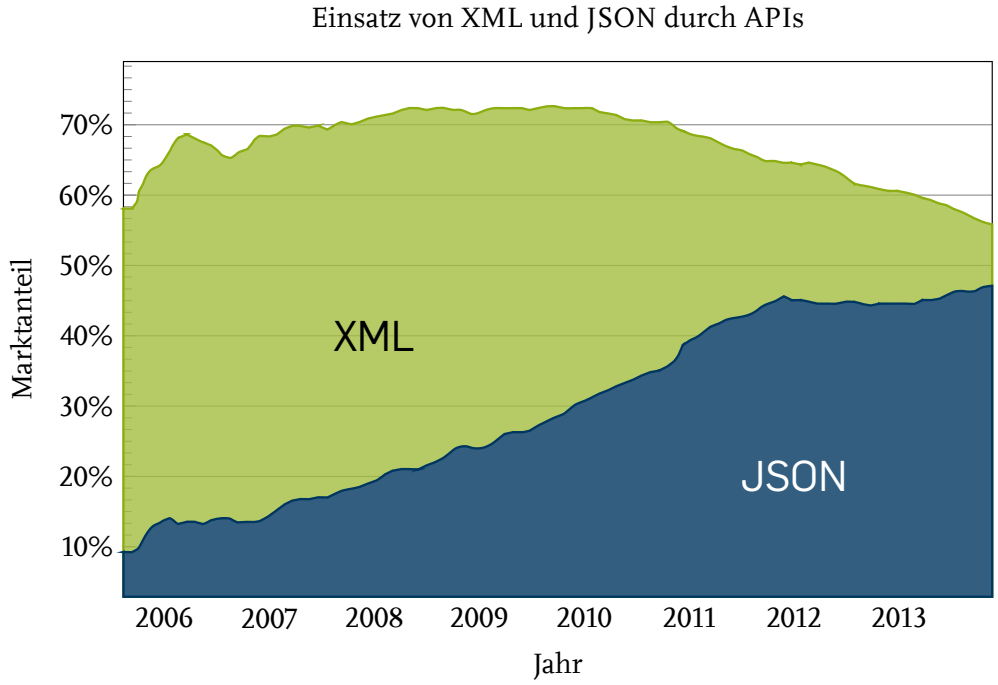


Abbildung 1.1: Sowohl JSON als auch XML wurden 2013 von jeweils mehr als 45 Prozent der Web-APIs unterstützt.

Inzwischen gewinnt jedoch JSON vor allem im Mobile- und Web-Bereich immer mehr an Bedeutung (vgl. Abb. 1.1). Laut dem API-Verzeichnis *ProgrammableWeb*[1] unterstützten im Jahr 2013 ca. 60 Prozent aller neu hinzugefügten APIs das JSON-Format, während XML im selben Zeitraum lediglich von 37 Prozent der neuen APIs unterstützt wurde. [DuV13]

JSON ist bei bestimmten Aufgaben in Bezug auf Geschwindigkeit und Ressourcenauslastung deutlich effizienter als XML [Nur09]. Inzwischen setzen auch einige populäre NoSQL-Datenbanken wie *CouchDB* oder *MongoDB* auf JSON zur Datenspeicherung. Auch MySQL verfügt seit Version 5.7.8 über einen nativen JSON-Datentyp.

Die Umwandlung zwischen den beiden Formaten kann aus vielen Gründen notwendig werden. Soll beispielsweise ein SOAP-Webservice als moderne JSON-ReST-Ressource angeboten werden, muss zwischen XML und JSON konvertiert werden. Auch bei der Speicherung von XML-Daten in den oben genannten NoSQL-Datenbanken ist dies der Fall. Zudem ist die Unterstützung der Formate durch Programmiersprachen, Frameworks und Applikationen nicht immer gleich gut.

1.2 Verwandte Arbeiten

Muschett, Salz und Schenker [JSONx] schlugen 2011 ein Format zur Repräsentation beliebiger JSON-Daten in XML 1.0 names JSONx vor. In dem inzwischen ausgelaufenen Internet Draft (I-D) werden eine Reihe von Regeln festgelegt um Instanzen der verschiedenen JSON-Daten in XML-Strukturen zu konvertieren. Die vorliegenden Arbeit untersucht hingegen die Konversion in umgekehrter Richtung, d.h. die Umwandlung beliebiger XML-Dokumente in das JSON-Format.

Ein weiteres XML-basiertes Format, dass das Datenmodell der JavaScript Object Notation (JSON) in XML nachbildet und dadurch die verlustlose Abbildung beliebiger JSON-Daten in XML ermöglicht, ist JsonXML (JXML) [JXML].

Christensen *et. al.* [Chr10] halten ein US-Patent an der Übersetzung zwischen XML und „dynamic language data expressions“, womit unter anderem auch JSON gemeint ist. Es nennt mehrere verschiedene denkbare Ausführungen, beispielsweise die Konversion von JSON in typannotiertes XML mittels Attributen oder Element-Namen, aber auch die Round-Trip-Umwandlung von XML zu JSON und zurück, ohne diese jedoch im Detail zu beschreiben.

David Lee [Lee11] entwickelte 2011 ein Verfahren, um anhand annotierter XSD-Schemata XSLT-Dateien zu erzeugen, die in der Lage sind, bidirektional zwischen XML und JXML

[1] `https://www.programmableweb.com/`

zu konvertieren. Damit ist also – mit einem Umweg über JXML – die verlustlose Umwandlung von XML in JSON und wieder zurück möglich. Durch die Nutzung des jeweiligen XSD-Schemas der zu konvertierenden XML-Dokumente sind bei der Umwandlung die verwendeten Datentypen bekannt, wodurch ein zuverlässigen Mapping auf die nativen JSON-Typen möglich wird.

Im Gegensatz dazu beschränkt sich die vorliegende Arbeit jedoch auf Konverter, die beliebige XML-Dokumente auch ohne Zuhilfenahme weiterer Metadaten wie XSD-Schemata umwandeln können.

Goessner schlägt ein Mapping zwischen strukturierten XML-Daten und JSON vor, dass möglichst kompakt und simpel sein soll. Dafür werden jedoch Abstriche im Bereich Verlustlosigkeit und Umkehrbarkeit der Konversion hingenommen, d.h. in bestimmten Fällen wie dem Auftreten mehrerer Kindelemente gleichen Namens kann ein Verlust (*hier:* Verlust der Elementreihenfolge) auftreten. [Goe06]

Ein US-Patent von Williamson [Wil14] beschreibt die Überführung von Daten aus einem XML-Dokument in eine JSON-Struktur, indem bestimmte Teile des XML-Baums selektiert (z. B. mittels XPath-artigen Ausdrücken) und einem Ziel-Schlüssel im resultierenden JSON-Objekt zugeordnet werden. Dadurch werden jedoch lediglich Teile der im XML-Dokument enthaltenen Informationen in die Ausgabe übernommen, sodass verlustloses Round-Tripping unmöglich wird.

In einem Paper vergleicht Wang [Wan11] die beiden Formate und entwickelt einen Übersetzungsalgorithmus zwischen den beiden Formaten. Allerdings wird dabei von datenorientiertem XML ausgegangen und somit keine Probleme betrachtet, die bei der Umwandlung von spezfischen Strukturen einer dokumentorientierten Auszeichnungssprache wie XML in das JSON-Format entstehen können, beispielsweise die korrekte Abbildung von Mixed Content (vgl. Abschn. 3.1.2.8). Der von Wang vorgeschlagene Algorithmus ist zudem auch bei datenorientierem XML unter Umständen verlustbehaftet [Wan11, S. 184].

Ein System zur Umstellung des von einer Webapplikation auf JavaScript-Basis genutzten Datenformats wird von Ying und Miller [Yin13] vorgeschlagen. Neben der Konversion von XML in JSON wird dabei auch der JavaScript-Quelltext der Applikation automatisch an das neue Format angepasst. Der Fokus der Arbeit liegt dabei auf dem Code-Refactoring – einer genauere Evaluation des eingesetzten Konversionsverfahrens in Bezug auf Sicherheit und Verlustlosgkeit findet nicht statt.

1.3 Aufbau der Arbeit

Im zweiten Kapitel wird zunächst eine kurze Einführung in die Struktur der Formate XML und JSON gegeben. Zudem werden auch relevante Technologien aus dem Umfeld der Formate beschrieben und die bekannten Angriffe gegen XML- und JSON-Parser erläutert.

Im Anschluss daran wird der Versuchsaufbau beschrieben. Zunächst werden die Anforderungen konkret definiert, d. h. es werden überprüfbare Kriterien aufgestellt, denen ein Konversionsverfahren genügen muss. Dazu gehört auch, Verlustlosigkeit genauer zu definieren, d.h. es wird die Relevanz der durch verschiedene XML-Strukturen dargestellten Informationen untersucht sowie möglicherweise zu Datenverlust führende Probleme bei der Umwandlung beschrieben.

In Abschnitt 3.2 wird die Auswahl der getesteten Konverter vorgestellt und daran anschließend das Überprüfungsverfahren erläutert.

Kapitel 4 stellt die Ergebnisse der Tests vor und diskutiert diese. Dabei wird erörtert, ob und welche Verfahren die zuvor genannten Bedingungen erfüllen.

Das Konversionsverfahren, das bei der Überprüfung die XML-Strukturen am besten abgebildet hat, wird im folgenden Kapitel dann zu einem vollständig verlustlosen Verfahren weiterentwickelt. Das so verbesserte Verfahren wird mithilfe der Kriterien aus Kapitel 3.1 einer erneuten Überprüfung unterzogen.

Im Fazit werden die aktuellen Möglichkeiten zur Umwandlung von XML in JSON und wieder zurück abschließend eingeschätzt und bewertet sowie ein Ausblick auf weitere mögliche Forschungfelder gegeben.

2 Hintergrund

2.1 Extensible Markup Language (XML)

Bei der Extensible Markup Language (XML) handelt es sich um eine weit verbreitete Auszeichnungssprache. Die Entwicklung von XML begann im Jahr 1996, zwei Jahre später wurde die Spezifikation erstmals als Empfehlung des World Wide Web Consortium (W3C) veröffentlicht. [Bra98, Abschn. 1.1]

XML ist eine plattformunabhängige Metasprache, die für den Einsatz im Internet und den Datenaustausch zwischen Anwendungen konzipiert wurde und als Basis für neue Datenformate genutzt werden kann.

Die Auszeichnungsprache basiert auf der im ISO-Standard 8879 beschriebenen Standard Generalized Markup Language (SGML) und stellt eine Teilmenge von SGML dar, weshalb XML-Dokumente zugleich immer auch SGML-Dokumente sind. Eines der Designziele war die Reduzierung der Komplexität im Vergleich zu SGML, denn optionale Zusatzfeatures und das vom Standard erlaubte Auslassen von Teilen des Markups (s. Abschn. 2.1.1) machen das korrekte Parsing von SGML-Dokumenten vergleichsweise schwierig. Im Gegensatz dazu soll XML einfacher zu verarbeiten sein – auch wenn das zu Lasten der Dateigröße geht. [XML; Bra98, Abschnitt 1.1 H15]

Aktuell steht XML in zwei unterschiedlichen Versionen zu Verfügung:

- XML 1.0 (Fifth Edition) [XML], veröffentlicht am 26. November 2008 und
- XML 1.1 (Second Edition) [XML11], veröffentlicht am 15. August 2006.

XML 1.1 ist in der Praxis kaum verbreitet und wird daher in der vorliegenden Arbeit nicht betrachtet.

Als Metasprache bildet XML die Grundlage für eine große Anzahl von Dateiformaten, beispielsweise die verbreiteten Office-Formate OpenDocument Format (ODF) und Office Open XML (OOXML) ebenso wie das Vektorgrafikformat SVG.

Im Web wird XML z.B. in Form von XHTML oder Rich Site Summary (RSS) für die Darstellung von Inhalten eingesetzt, vor allem aber für den Datenaustausch mit dynamischen AJAX-Applikationen und Webservices genutzt.

Beispiel 1 (XML-Dokument). Dieses einfache XML-Dokument enthält verschiedene Knoten-Typen.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="style.css"?>
3  <!-- Nice album! -->
4  <albums>
5    <album catno="ARGO LP-628">
6      <artist>Ahmad Jamal Trio</artist>
7      <title>At The Pershing</title>
8      <recording>Recorded <date>January 16,
       ↪  1958</date>.</recording>
9    </album>
10 </albums>
```

2.1.1 Grundlagen

XML-Daten bestehen aus Text, deren Struktur durch die Produktionsregeln der XML-Spezifikation vorgegeben ist. Der Text eines XML-Dokuments besteht aus einem Gemisch von *Markup* und *Character Data*, wobei das Markup eine Vielzahl von Formen annehmen kann, sich jedoch in jedem Fall syntaktisch von Character Data unterscheidet. Es wird dazu genutzt, den Inhalt des Dokumentes zu strukturieren und verschiedene Teile des Dokuments mit Metainformationen zu versehen. Text, der *kein* Markup enthält, nennt man *Character Data*.

Beispielsweise sind in Zeile 8 des in Beispiel 1 gezeigten XML-Dokuments „`<recording>`“, „`<date>`“, „`</date>`“ und „`</recording>`“ *Markup*, während „`Recorded␣`“, „`January␣16,␣1958`“ und „`.`“ *Character Data* ist.

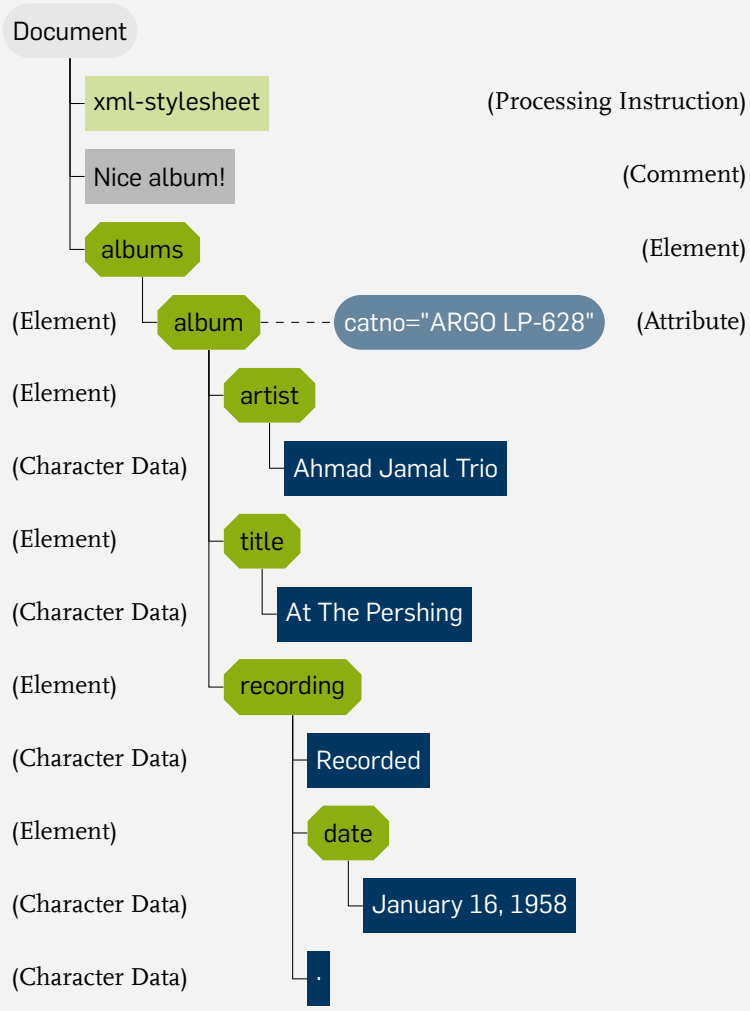
Auf die in XML-Dokumenten enthaltenen Informationen wird in der Regel über Schnittstellen zugegriffen, die das Markup auswerten und das Dokument in Form einer baumartigen Datenstruktur darstellen. Ein prominentes Beispiel dafür ist das Document Object Model [DOM], das u. a. von *JavaScript*-Programmen in Webbrowsern eingesetzt wird, um HTML-Seiten auszuwerten und zu manipulieren.

Das Beispiel 1 zeigt ein XML-Dokument, das aus mehreren verschiedenen Knotentypen besteht:

Document ist der Wurzelknoten der Baumstruktur und kommt daher genau ein Mal im Baum vor. Es repräsentiert das XML-Dokument selbst – ist also nicht Teil des XML-Quelltextes – und macht die darin enthaltenen Informationen über seine Kindknoten zugänglich. [XMLInfo, Abschnitt 2.1]

Elemente sind Knoten, die weitere Elemente oder andere Knoten als Kinder enthalten können. Auf der Dokumentebene muss es immer genau ein Element (das *Document Element*) geben. Zudem können sie mit Attributen versehen werden. Elemente bestehen entweder aus einem Start- und einem End-Tag (z.B. „`<foo>`“ und „`</foo>`“ oder – falls sie keine Kindknoten enthalten – aus einem einzelnem leeren Element-Tag (z.B. „`<foo />`“).

Beispiel 2 (XML-Baumstruktur). Das XML-Dokument aus Beispiel 1 kann als Baumstruktur dargestellt werden. Die durch Einrückung entstandenen Whitespace-Textknoten wurden zwecks Übersichtlichkeit nicht miteinbezogen.



Attribute sind Schlüssel-Wert-Paare, die einem einzelnen Element-Knoten zugeordnet sind. Sie werden im Start-Tag eines Elements angegeben. So enthält das „`<album>`“-Tag im Beispiel 1 das Attribut „`catno="ARGO LP 628"`“.

Kommentare können Text (z.B. Beschreibungen zu Teilen des XML-Dokuments) enthalten, sind jedoch nicht Teil der *Character Data* des Dokuments. Sie werden mit der Zeichenfolge „`<!--`“ eingeleitet und mit „`-->`“ beendet. Um die Rückwärtskompatibilität mit SGML sicherzustellen, dürfen sie die Zeichenfolge „`--`“ nicht enthalten. [XML, Abschnitt 2.5]

Processing Instructions (PIs) sind Steueranweisungen an ein bestimmtes, das XML-Dokument verarbeitendes Programm. Sie bestehen aus einem *Ziel* (einer Zeichenkette, die nicht „`xml`“ sein darf) und *Daten*, die häufig in einer Attributen nachempfundenen Form angegeben sind. PIs werden mit der Zeichenfolge „`<?`“ eingeleitet und mit „`?>`“ beendet. [XML, Abschnitt 2.6]

Text sind Blattknoten, die Character Data enthalten. Im Beispiel 1 enthält das „`<artist>`“-Element einen Textknoten, der den Text „`Ahmad Jamal Trio`“ repräsentiert.

Es existieren jedoch auch abweichende Datenmodelle um die logische Strukur von XML-Daten zu beschreiben, beispielsweise das XQuery and XPath Data Model [XDM] oder das XML Information Set [XMLInfo]. Die verschiedenenen Modelle unterscheidenen sich dabei vor allem im Abstraktionsgrad – so stellt das Document Object Model (DOM) Textknoten und CDATA-Sektionen als separate Knotentypen dar, während das XML Information Set beide Knotentypen unter *Character Information Data* subsumiert [XMLInfo, Abschnitt 2.6].

CDATA-Abschnitte kennzeichnen größere Bereiche von Text, die Markup-Zeichen wie „`<`“ oder „`&`“ enthalten dürfen, ohne dass diese quotiert werden müssten. Sie werden mit der Zeichenkette „`<![CDATA[`“ eingeleitet und mit „`]]>`“ beendet. [XML, Abschnitt 2.7]

DOM, XDM und XML Information Set kennen noch eine Vielzahl weiterer Konstrukte, die Teil eines XML-Dokuments sein können, z.B. Notation-Knoten. Detaillierte Beschreibungen können in den jeweiligen Spezifikationen gefunden werden. [DOM; XMLInfo; XDM; XML]

2.1.2 Document Type Definitions (DTDs)

Document Type Definitions (DTDs) sind eine einfache Möglichkeit, um den Aufbau eines XML-Dokuments genauer zu definieren. Innerhalb einer DTD wird dann beispielsweise festgelegt, ob ein Element Zeichen oder weitere Elemente enthalten darf und ob es einfach oder mehrfach in einem Dokument auftreten kann. Ein Dokument,

das über eine DTD verfügt und deren Regeln vollständig erfüllt, wird als *valide* bezeichnet.

Eine weitere Funktion von DTDs ist die Möglichkeit zum Festlegen von Platzhaltern, sogenannten *Entities*. Wird eine Entity-Referenz im XML-Dokument verwendet, fügt der XML-Parser an dieser Stelle stattdessen die ihr zugewiesenen Daten ein.

Es existieren zwei Arten von Entities:

General Entities können innerhalb des Wurzelelements eines XML-Dokuments referenziert werden. Sie werden beispielsweise genutzt, um häufig im Dokument vorkommende Zeichenfolgen einfacher anpassbar zu machen, da diese so nur in der Entity-Definition verändert werden müssen.

Parameter Entities sind hingegen nur innerhalb der DTD des Dokuments verfügbar und werden z. B. eingesetzt, um Mehrfachnennungen bei Element-Deklarationen zu vermeiden.

Neben internen Entity-Deklarationen können sowohl General Entities als auch Parameter Entities die ihnen zugewiesenen Daten auch aus einer Datei bzw. einer Uniform Resource Locator (URL) nachladen. Zudem verfügt XML auch über eine Reihe vordefinierter General Entities, die für die Quotierung von Markup-Trennzeichen – wie z.B. dem Größerzeichen – verwendet werden können.

Die DTD selbst kann entweder direkt in das XML-Dokument eingebettet oder aus einer externen Resource nachgeladen werden.

2.1.3 XML Schema Definition (XSD)

Anstelle von DTDs können auch sogenannte Schemata eingesetzt werden, um die Struktur von XML-Dokumenten zu beschreiben und festzulegen. Diese legen fest, welche Elemente in einem Dokument vorhanden sein dürfen oder müssen. Ebenso definieren sie Anzahl und Reihenfolge von Elementen, aber auch die Datentypen von Attributwerten und Textknoten. Attributen eines Elements können mithilfe eines Schemas auch Standardwerte zugewiesen werden. Zudem können Attributwerte als unveränderbar deklariert werden.

Neben der vom World Wide Web Consortium (W3C) spezifizierten XML Schema Definition (XSD) gibt es noch eine Reihe ähnlicher Technologien wie RELAX NG, Schematron, Document Structure Description (DSD) etc.

2.1.4 XML-Namespaces

XML-Dokumente beinhalten häufige eine Vielzahl verschiedener Element- und Attributnamen, die die verschiedenen Inhalte des Dokuments strukturieren und semantisch voneinander unterscheidbar machen sollen. Namespaces bieten die Möglichkeit, logisch verwandte Namen zu gruppieren – sie legen also eine Menge von Element- und Attributnamen fest, die innerhalb der jeweiligen Namespaces eindeutig sein müssen.

Namespaces werden mittels eines Bezeichners identifiziert, der die Form eines Uniform Resource Identifier (URI) haben muss. Dieser ist allerdings nur als Name zu verstehen – eine Nutzung des URI zum Abruf von Daten, etwa zum Nachladen eines XML-Schemas aus dem Internet, ist nicht vorgesehen [XMLNS, Abschn. 3].

Beispiel 3 (XML-Namespaces)**.** Anhand des Namespaces kann die semantische Bedeutung verschiedener Elemente trotz gleichen Tag-Namens unterschieden werden.

```
1  <song xmlns="http://example.com/song"
2        xmlns:cp="http://example.com/composition"
3        xmlns:rc="http://example.com/recording">
4    <title>Love for Sale</title>
5    <artist>Cannonball Adderley</artist>
6    <cp:artist>Cole Porter</cp:artist>
7    <cp:year>1930</cp:year>
8    <album>Somethin' Else</album>
9    <rc:date xmlns="http://example.com/date">
10     <month>March</month> <day>9</day>, <year>1959</year>
11   </rc:date>
12 </song>
```

Um in einem XML-Dokument zu deklarieren, kann das `xmlns`-Attribut verwendet werden. Dabei gibt es zwei Formen:

Deklaration eines Namespace-Prefixes Durch Angabe eines Attributes in der Form `xmlns:prefix` kann einem Namespace ein Prefix zugewiesen werden. Um zu signalisieren, dass das Element oder eines seiner Kindelemente zu diesem Namespace gehört, wird der Tag-Name mit diesem Prefix versehen (z.B. in Zeile 6 von Beispiel 3). Dies ermöglicht auch die Nutzung und Vermischung des Vokabulars verschiedener Namespaces in einem einzelnen Dokument.

Deklaration des Default Namespace Sind Tags mit keinem Prefix versehen, gehören diese zum Default Namespace. Dieser kann mithilfe des `xmlns`-Attributes ei-

nes Elements für das Element und seine Kindelemente festgelegt werden (z.B. in Zeile 9 von Beispiel 3).

2.1.5 XML-Kanonisierung (C14N)

Logisch äquivalente XML-Dokumente können sich in der konkreten Darstellung stark unterscheiden. Das W3C hat daher Richtlinien zur Umwandlung in eine einheitliche Darstellungsweise – die sogenannte *Kanonische Form* – festgelegt. Der Umwandlungprozess wird XML-Kanonisierung (C14N) genannt. [C14N]

Beispiel 4 (Kanonisierung). Ein XML-Dokument vor und nach der Kanonisierung. Beide Versionen sind logisch äquivalent.
Das unkanonisierte XML-Dokument enthält überflüssigen Whitespace und z.T. einfache Hochkommata als Attributwert-Trennzeichen.

```
1 <?xml version="1.0"?>
2 <entries>
3   <entry     foo = "bar" />
4   <entry foo='baz'  ></entry>
5 </entries>
```

Durch die Kanonisierung des Dokuments wurde insignifikanter Whitespace entfernt, durchgehend das Anführungszeichen als Attributtrennzeichen verwendet und lediglich aus einem Start-Tag bestehende leere Elemente durch das entsprechende Start-End-Tag-Paar ersetzt.

```
1 <entries>
2   <entry foo="bar"></entry>
3   <entry foo="baz"></entry>
4 </entries>
```

2.1.5.1 Motivation und Anwendungsbereich

Durch die Flexibilität des XML-Standards besteht die Möglichkeit, Informationen durch eine Vielzahl von XML-Dokumenten darzustellen [Sid02a]. Die im Beispiel 4 abgebildeten XML-Dokumente haben einen identischen Informationsgehalt, lediglich die Darstellungsweisen unterscheiden sich. Allein die Möglichkeit, innerhalb eines Start-Tags eine beliebige Anzahl von Whitespace einzusetzen [XML, Produktionsregeln 3 und 40] führt bereits zu einer unendlichen Anzahl von logisch äquivalenten XML-Darstellungen einer Information.

Um die in verschiedenen XML-Dokumenten enthaltene Logik vergleichbar zu machen, muss daher von der konkreten Darstellung abstrahiert werden. XML-Kanonisierung überführt daher beliebige XML-Dokumente in eine *Kanonische Form*, indem eine Reihe von Arbeitsschritten abgearbeitet wird, die in Abschnitt 2.1.5.2 beschrieben sind.

XML-Kanonisierung wird vor allem im Bereich der Signaturerstellung und -verifikation eingesetzt. Dabei wird nicht das zu siginierende Datenobjekt selbst, sondern lediglich ein *Digest*, d. h. ein Hashwert des Objekts, signiert [XMLSig, Abschnitt 2.0]. Subtile und für die Dokumentlogik irrelevante Unterschiede – wie der Einsatz von abweichenden Zeilenumbrüchen oder die Nutzung von einfachen statt doppelten Hochkommata als Trennzeichen für Attribute – würden dabei den Hashwert ändern und die Signatur ungültig werden lassen. Daher werden die Datenobjekte standardmäßig mittels XML-Kanonisierung transformiert. [XMLSig, Abschnitt 4.3.3.2]

2.1.5.2 Arbeitsschritte

Bei der XML-Kanonisierung werden grob folgende Änderungen am XML-Dokument vorgenommen [C14N, Abschnitt 1.1]:

1. Kodierung der Eingabe mit dem UTF-8-Zeichensatz
2. Normalisieren der Zeilenumbrüche

   Die Konventionen für die Speicherung von Zeilenenden (`EOL`) sind je nach Betriebssystem unterschiedlich. So nutzen unixode Betriebssysteme wie Linux und BSD beispielsweise einen Zeilenumbruch[1] (`\n`), Windows und DOS hingegen nutzen die Kombination Wagenrücklauf[2] und Zeilenumbruch (`\r\n`). Ältere Mac-OS-Versionen setzen hingegen die Kombination Zeilenumbruch und Wagenrücklauf (`\n\r`) ein [Unicode9, S. 212]. Bei der Kanonisierung werden alle Varianten durch einen einfachen Zeilenumbruch ohne Wagenrücklauf (`\n`) ersetzt.
3. Normalisieren der Attributwerte

   Dabei werden einzelne oder mehrere aufeinanderfolgende Whitespace-Zeichen wie Leerzeichen, Tabulatoren und Zeilenumbrüche in Attributwerten zu einem einzelnen Leerzeichen zusammengefasst.
4. Zeichen- und Entity-Referenzen werden ersetzt
5. `CDATA`-Abschnitte werden in normale Text-Knoten umgewandelt
6. XML-Deklaration und DTD werden entfernt

[1] Unicode Code-Point `U+000A`: `LINE FEED (LF)`
[2] Unicode Code-Point `U+000D`: `CARRIAGE RETURN (CR)`

7. Leere Elemente werden in Paare, bestehend aus Start- und End-Tags umgewandelt
8. Whitespace außerhalb des Wurzelelements und innerhalb von Start- und End-Tags wird normalisiert
9. Jeglicher Whitespace innerhalb des Wurzelelements wird (mit Ausnahme der o.g. normalisierten Zeilenenden) unverändert belassen
10. Alle Attributwert-Trennzeichen werden in Anführungszeichen („double quotes“)[3] umgewandelt
11. Sonderzeichen in Attributwerten und Zeicheninhalt werden durch XML Character References ersetzt.
12. Überflüssige Namespace-Deklarationen werden entfernt.
13. In der `ATTLIST` angegebene Vorgabeattribute werden zu allen Elementen hinzugefügt.
14. Attribute und Namespace-Deklarationen aller Elemente werden in lexikographische Ordnung gebracht

2.1.5.3 Kommentare

Die W3C-Empfehlung für kanonisches XML erlaubt sowohl das ersatzlose Entfernen als auch das Beibehalten der Kommentare, wobei das Resultat in letzterem Fall als *Kanonisches XML mit Kommentaren* (engl. „canonical XML with comments“) [C14N, Abschnitt 2.1] bezeichnet wird. Implementierungem müssen in der Lage sein, kanonisches XML ohne Kommentare zu erstellen, während die Unterstützung für kanonisches XML mit Kommentaren zwar empfohlen, jedoch nicht vorgeschrieben ist.

2.1.5.4 Exklusive XML-Kanonisierung

Einen Sonderfall stellt die Exklusive XML-Kanonisierung dar, die in einer separaten Empfehlung des W3C beschrieben ist [ExcC14N]. Während bei der regulären (inklusiven) XML-Kanonisierung der Vorfahren-Kontext (z.B. Namespaces und Attribute im XML-Namespace) von Teildokumenten eines XML-Dokuments erhalten bleibt, wird dieser bei der exklusiven C14N verworfen [Sid02b, Abschnitt 18]. Dies kann insbesondere dann sinnvoll sein, wenn Teildokumente aus dem Quelldokument herausgelöst und ggf. andere Dokumente eingebettet werden sollen (Re-Enveloping), ohne dass sich ihre digitale Signatur ändert.

[3] Unicode Code-Point `U+0022: QUOTATION MARK`

2.1.6 Angriffe

Einige der Features des XML-Ökosystems können für Angriffe auf Parser missbraucht werden und insbesondere DTDs haben sich in der Vergangenheit als potentiell gefährlich herausgestellt. [Mor14, S. 4]

Obwohl Angriffe zum Teil seit Jahren bekannt sind, sind viele XML-Parser weiterhin verwundbar – Forscher der Ruhr-Universität Bochum fanden Sicherheitslücken in den Standardkonfigurationen von 25 der 33 gestesteten XML-Parser [Spä16].

2.1.6.1 File System Access

Bei den Angriffen aus dem Bereich File System Access soll der XML-Parser dazu gebracht werden, eine normalerweise nicht zugängliche Datei auf dem lokalen Dateisystem in ein XML-Dokument einzubetten. Kann der Angreifer die Zielapplikation dazu bringen, ihm Zugriff auf das Dokument zu gewähren, erlangt er Kenntnis vom Inhalt der Datei.

XML External Entity (XXE) XXE-Angriffe sind bereits seit 2002 bekannt [Ste02]. Der Angreifer gibt dabei den Pfad zu einer lokalen Datei auf dem Server in Form einer File-URL in einer External Entity an und refenziert diese Entity dann im Dokument. Enthält der eingebette Inhalt jedoch Zeichen, die von XML als Markup interpretiert werden, kann dies unter Umständen dazu führen, dass das resultierende Dokument nicht mehr wohlgeformt ist und der Angriff fehlschlägt. [Spä16, Abschnitt 5.1]

Beispiel 5 (Klassischer XXE-Angriff). Der XML-Parser wird angewiesen, den Inhalt der Datei `/etc/passwd` in das Dokument einzubetten.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
  <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<root>&xxe;</root>
```

Parameter-Entity-based XML External Entity (XXE) Im Gegensatz zu klassischen XXE-Angriffen kann der Dateiinhalt bei XML External Entity (XXE) mit *Parameter Entities* in eine CDATA-Sektion eingebettet werden. Dies wird durch Anlegen mehrerer solcher Entities erreicht, die Beginn und Ende der CDATA-Sektion sowie den Dateiinhalt enthalten und im Anschluss zusammengesetzt werden. [Mor14, S. 10]

Beispiel 6 (XXE-Angriff mit Parameter Entities). In der DTD des XML-Dokuments definiert der Angreifer Parameter Entities für den Inhalt der Datei `/etc/passwd` sowie für Beginn und Ende einer CDATA-Sektion.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE root [
  <!ENTITY % start "<![CDATA[">
  <!ENTITY % file SYSTEM "file:///etc/passwd">
  <!ENTITY % end "]]>">
  <!ENTITY % dtd SYSTEM "http://example.com/evil.dtd">
  %dtd;
]>
<root>&all;</root>
```

Mithilfe einer externen DTD-Datei, die vom Angreifer unter der im Dokument angegeben URL abgelegt wurde, werden die einzelnen Parameter Entities zusammengesetzt.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY all "%start;%file;%end;">
```

XInclude Auch bei Angriffen auf Basis von XML Inclusions (XInclude) enthält das zu parsende XML-Dokument eine Anweisung zum Einbetten einer lokalen Datei [XInclude]. Im Gegensatz zu klassischen XXE-Angriffen kann der eingebette Inhalt jedoch beliebiger Text sein, darf also z.B. auch ungültiges XML-Markup enthalten. [Spä15, S. 80]

Beispiel 7 (XInclude-Angriff). Der XML-Parser wird angewiesen, den Inhalt der Datei `/etc/passwd` in das Dokument einzubetten [XInclude, Abschn. 3.1].

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="file:///etc/passwd" parse="text" />
</root>
```

Auch XSL Transformation (XSLT) erlaubt die Einbindung von Dateien, beispielsweise über die `fn:document`-Funktion [XSLT, Abschn. 20.1] und eigenet sich daher prinzipell für FSA-Angriffe.

2.1.6.2 Denial of Service (DoS)

Denial of Service ist ein Angriffstyp, bei dem ein Angreifer darauf abzielt einen Dienst unerreichbar für andere Nutzer zu machen, indem dieser – oder der Server, auf dem er läuft – überlastet wird. Dazu wird in der Regel versucht, den zur Verfügung stehenden Arbeitsspeicher zu füllen oder die CPU über längere Zeit auszulasten. Zudem können Angriffsvektoren aus dem Bereich File System Access für DoS-Angriffe eingesetzt werden, indem der Parser angewiesen wird, extrem große bzw. niemals endende Dateien wie `/dev/random` zu lesen [Mor14, S. 13].

Entity Recursion Attack Hierbei werden Entities so angelegt, dass sie auf sich selbst verweisen. Zwar sind direkte und indirekte Selbstreferenzierung von Entites in der XML-Spezifikation explizit verboten [XML, Abschnitt 4.1], naive Parser könnten aber trotzdem in eine Endlosschleife verfallen.

Beispiel 8 (Entity Recursion). Die Entität `&a;` verweist indirekt auf sich selbst.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
  <!ENTITY a "&b;">
  <!ENTITY b "&a;">
]>
<root>&a;</root>
```

Exponential Entity Attack (Billion Laughs) Indem interne Entities über mehrere Ebenen ineinander verschachtelt werden, wird der Parser dazu gebracht, eine große Menge von Arbeitsspeicher zu allozieren. Eine mögliche Gegenmaßnahme ist die Begrenzung der Anzahl von erlaubten Entity-Referenzen.

Quadratic Blowup Attack Dieser Angriff ist eine Abwandlung der *Billion Laughs Attack*. Es wird eine einzelne Entität angelegt, die einen extrem langen String enthält. Dieser wird nun mehrmals im Dokument referenziert, die Anzahl der verwendeten Referenzen ist jedoch im Vergleich zu *Billion Laughs* gering, sodass eine mögliche Obergrenze für erlaubte Entity-Referenzen umgangen werden kann.

2.1.6.3 Server Side Request Forgery

Durch manipulierte XML-Dateien soll ein Parser dazu gebracht werden, eine Anfrage an einen anderen Netzwerkteilnehmer zu senden. Dies ist üblicherweise ein Dienst, auf den der Angreifer aufgrund einer Firewall keinen direkten Zugriff hat, der sich

Beispiel 9 (Billion Laughs). Nach der rekursiven Dereferenzierung belegt die expandierte Entity `&x9;` $3 \cdot 10^9$ B ≈ 2.7939 GiB im Arbeitsspeicher.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root [
  <!ENTITY x "lol">
  <!ENTITY x1 "&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;">
  <!ENTITY x2 "&x1;&x1;&x1;&x1;&x1;&x1;&x1;&x1;&x1;&x1;">
  <!-- [...] -->
  <!ENTITY x9 "&x8;&x8;&x8;&x8;&x8;&x8;&x8;&x8;&x8;&x8;">
]>
<root>&x9;</root>
```

aber in der gleichen Firewall-Zone wie die XML-Applikation befindet. So könnten beispielsweise HTTP-Requests an einen Webserver im Unternehmens-Intranet gesendet werden.

Beispiel 10 (Klassische SSRF). Über den `SYSTEM`-Identifikator wird der XML-Parser dazu gebracht, die URL `http://example.com/internal` aufzurufen.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root SYSTEM "http://example.com/internal">
<root />
```

Weitere SSRF-Angriffsvektoren sind die XSD-Attribute[4] „`schemaLocation`“ und „`noNamespaceSchemaLocation`“. Auch sie erlauben die Angabe von URLs, die u. U. von einem Parser aufgerufen werden.[Mor14, S. 8f]

XInclude- und XXE-Angriffe aus dem Bereich File System Access können teilweise auch mit URLs durchgeführt und somit für Server Side Request Forgery eingesetzt werden.

2.2 JavaScript Object Notation (JSON)

Vor allem im Bereich des Datenaustauschs hat sich die JavaScript Object Notation (JSON) als Alternative zum gängigen XML-Format etabliert. Im Gegensatz zur dokumentorientierten Auszeichnungssprache XML wird JSON häufig als *datenorientiert* bezeichnet. [Gup07]

[4] Die vollständige Namespace-URI lautet: `http://www.w3.org/2001/XMLSchema-instance`.

Das ursprünglich Anfang der 2000er Jahre von Douglas Crockford spezifizierte Format [Cro09] liegt inzwischen in zwei konkurrierenden Standards vor:

- ECMA-404 (1st Edition) [ECMA404], veröffentlicht im Oktober 2013 und
- RFC7159 der Internet Engineering Task Force (IETF) [RFC7159], veröffentlicht im März 2014.

ECMA-404 beschränkt sich vor allem auf die JSON-Syntax, während bei der Konzeption von RFC7159 auch Interoperabilität eine Rolle spielte [Bra14].

2.2.1 Datenmodell und Syntax

Das JSON-Format verfügt über mehrere verschiedene Datentypen, darunter die Containertypen *Objekt* und *Array* [ECMA404]:

Objekte werden mit geschweiften Klammern angegeben und enhalten eine kommaseparierte ungeordnete Liste von beliebig vielen Schlüssel-Wert-Paaren. Die Schlüssel müssen vom Typ String sein und dürfen im Objekt nur einmal vorkommen. Die Werte dürfen beliebigen Typs sein (auch weitere Objekte).

Arrays sind neben *Objekten* die zweite Containerklasse des JSON-Formats und enhalten eine kommaseparierte geordnete Liste von Werten beliebigen Typs. Diese Liste wird von eckigen Klammern umschlossen und darf auch leer sein.

Strings sind einfache Zeichenketten, die aus beliebig vielen Zeichen bestehen und von doppelten Anführungszeichen umschlossen sind. Kontrollzeichen, doppelte Anführungszeichen und der Backslash müssen mit einem Backslash quotiert werden.

Zahlen werden als Dezimalwert ohne führende Null dargestellt. Für negative Werte kann ein Minuszeichen vorangestellt werden. Bei der Angabe von Nachkommastellen werden diese mittels eines Punkts vom Ganzzahlteil getrennt. Auch die wissenschaftliche „E-Notation“ ist erlaubt.

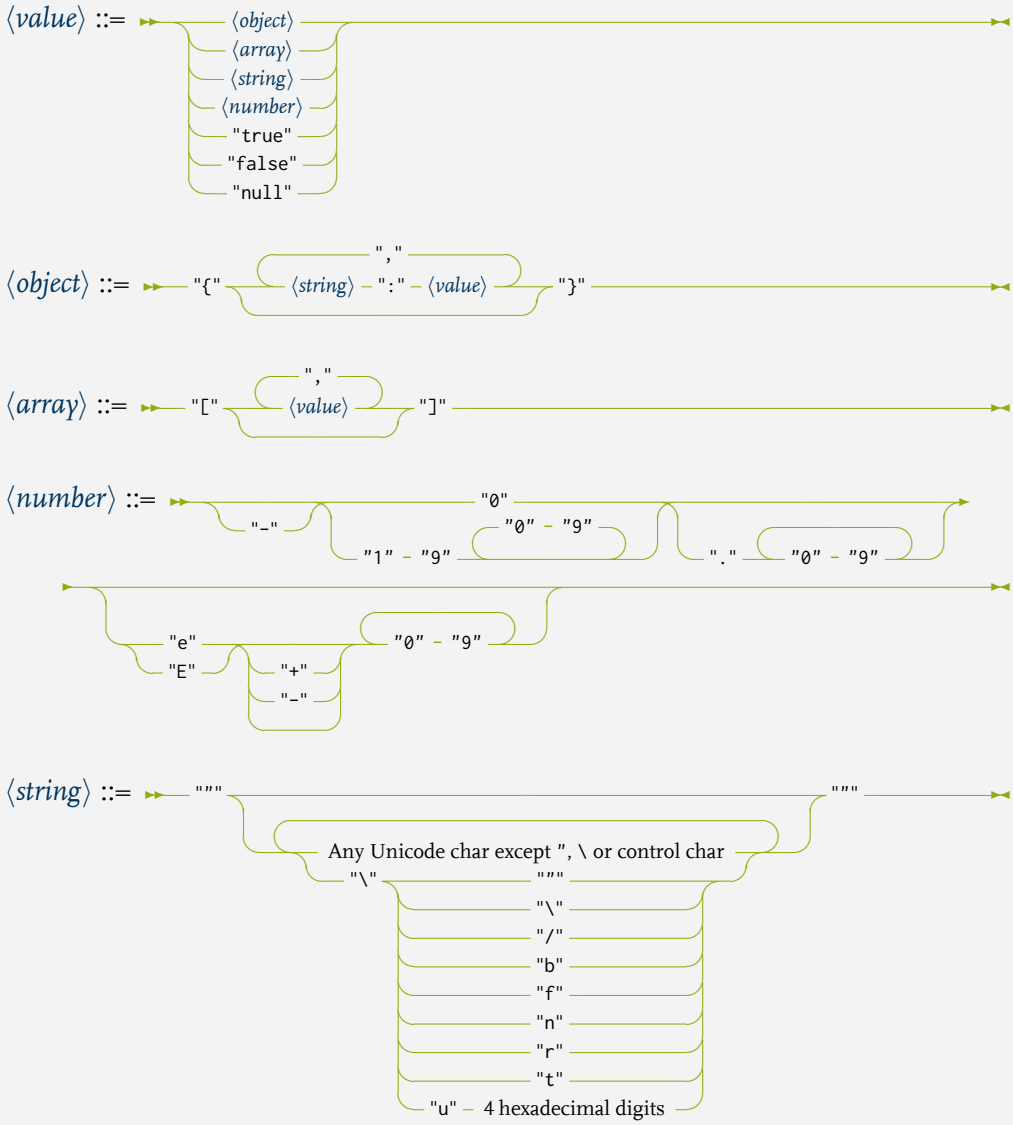
`true` / `false` bezeichnet die Booleschen Werte „wahr“ und „falsch“.

`null` steht für den Nullwert.

2.2.2 Angriffe

Das JSON-Format ist sehr einfach aufgebaut – komplexere Features wie die Referenzierung von internen oder externen Werten und Dateien fehlen in der Spezifikation. Damit gibt es kein Äquivalent zu den *Entities* des XML-Formats, die Grundlage der meisten Angriffe auf XML-Parser sind.

Definition 1 (Formale Syntax der JavaScript Object Notation (JSON)). Unerheblicher Whitespace (Tabulator[a], Zeilenumbruch[b], Wagenrücklauf[c] und Leerzeichen[d]) kann vor oder nach allen Token ("{"[e], "}"[f], "["[g], "]"[h], ":"[i], ","[j]) eingefügt werden.



[a] Unicode-Codepoint U+0009: CHARACTER TABULATION
[b] Unicode-Codepoint U+000A: LINE FEED (LF)
[c] Unicode-Codepoint U+000D: CARRIAGE RETURN (CR)
[d] Unicode-Codepoint U+0020: SPACE
[e] Unicode-Codepoint U+007B: LEFT CURLY BRACKET
[f] Unicode-Codepoint U+007D: RIGHT CURLY BRACKET
[g] Unicode-Codepoint U+005B: LEFT SQUARE BRACKET
[h] Unicode-Codepoint U+005D: RIGHT SQUARE BRACKET
[i] Unicode-Codepoint U+003A: COLON
[j] Unicode-Codepoint U+002C: COMMA

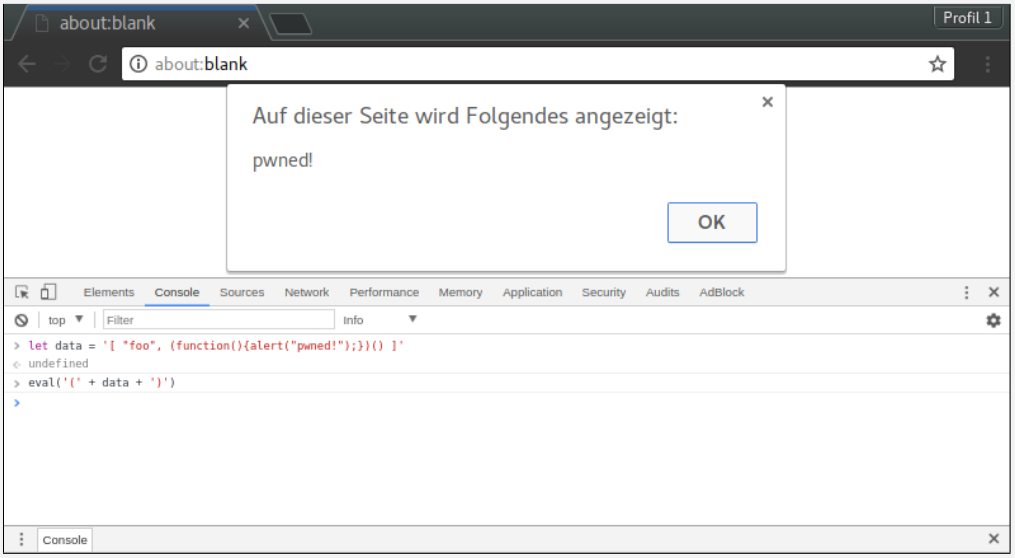
Ein Internet Draft (I-D) der Internet Engineering Task Force (IETF) von September 2012 schlug eine solche Funktionalität unter dem Titel „JSON Reference“ zwar vor, diese kam aber nie über das Entwurfsstadium hinaus. [JSONRef]

Eine Spezifikation für die Nutzung von Schemata ist aktuell noch im Entwurfsstadium. Daher werden die Sicherheitsaspekte von *JSON Schema* im Rahmen dieser Bacherlorarbeit nicht betrachtet. Sie beschreibt ebenfalls eine Möglichkeit zur Referenzierung von Schemata und JSON-Werten [JSONSchema, Abschnitt 8]. Im Gegensatz zu XML sieht die Spezifikation allerdings zur Zeit keine Möglichkeit zur Einbettung eines Schemas in ein Datendokument vor. Auch der Verweis auf ein Schema im Datendokument selbst ist nicht vorgesehen. Stattdessen können aber HTTP-Header [JSONSchema, Abschnitt 10.1] verwendet werden, um auf ein Schema zu verweisen.

Ein möglicher Angriffspunkt beim Einsatz von JSON-verarbeitenden Applikationen auf JavaScript-Basis ist der Einsatz der `eval()`-Funktion [ECMA262, Abschnitt 18.2.1]

Beispiel 11 (Unsicheres JSON-Parsing). Wird JSON mittels `eval()`-Funktion geparst, kann ein Angreifer u.U. Schadcode ausführen. Hier öffnet der im String eingebette Funktionsaufruf ein `alert()`-Benachrichtungsfenster im Browser.

```
1 // Maliciously constructed JSON string
2 let data = '[ "foo", (function(){alert("pwned!");})() ]';
3 // This executes the attacker's payload and returns the Array [
  ↪  "foo", undefined ]
4 eval('(' + data + ')');
```



sein. Als Teilmenge von JavaScript[5] ist die Nutzung der Funktion zwar möglich, kann jedoch ohne weitergehende Validierung die Einschleusung von Schadcode ermöglichen [RFC7159, Abschn. 12].

[5] Auch wenn diese Aussage vom JSON-Entwickler selbst stammt [Cro06], ist sie umstritten, da JavaScript auf Zahlen im IEEE-754 binary64-Format [ECMA262, Abschnitt 6.1.6] beschränkt ist. Im Gegensatz dazu enthält die JSON-Spezifikation keinerlei Einschränkungen bezüglich der Fließkommagenauigkeit [ECMA404, Abschnitt 8] – JSON-Dokumente können daher auch Zahlen enthalten, die in JavaScript nicht darstellbar sind.

3 Versuchsaufbau

Im Folgenden wird der Versuchsaufbau, bestehend aus

1. der Erstellung einer Reihe von Kriterien zur Bewertung der Verlustlosigkeit und Sicherheit von Konversionverfahren,
2. einer Auswahl von zu überprüfenden Konvertern, und
3. der Methodik zur Überprüfung der Konversionsverfahren auf Einhaltung der zuvor festgelegten Kriterien

vorgestellt.

3.1 Bewertungskriterien für Konversionsverfahren

Ziel ist das Finden eines Konversionsverfahrens, welches sowohl *sicher* als auch *verlustlos* arbeitet. Dazu müssen beide Anforderungen zunächst genauer definiert werden.

3.1.1 Sicherheit

Die Sicherheit eines Konverters hängt in erster Linie von der Sicherheit des eingesetzten XML-Prozessors ab. Dieser darf gegen keinen der im Abschnitt 2.1.6 beschriebenen Angriffe auf XML-Parser verwundbar sein.

In Bezug auf Sicherheit stellt die Unterstützung eines XML-Parsers für Entities das größte Einfallstor für Angriffe dar. Viele der Angriffsvektoren sind nur dann möglich, wenn der Parser General bzw. Parameter Entities auswertet und expandiert.

Das Open Web Application Security Project (OWASP) empfiehlt daher, die Unterstützung für Document Type Definitions (DTDs) komplett zu deaktivieren, da dies sowohl XXE-Angriffe als auch DoS-Attacken wie *Billion Laughs* oder *Quadratic Blowup* wirksam verhindert [Wic17, Abschn. 1.1]. Diese Gegenmaßnahme empfiehlt auch Nazim Lala von Microsofts IIS Security Team [Lal13].

Aus Sicherheitsgründen ist die Verarbeitung von DTDs für die in der vorliegenden Arbeit überprüften Konverter daher nicht erforderlich.

Dies gilt jedoch weder für die vordefinierten General Entities „`&`“, „`<`“, „`>`“, „`&apos;`“ und „`&quot;`“ [XML, Abschn. 4.6], noch für numerische Character References [XML, Abschn. 4.1], da diese nicht innerhalb einer DTD angegeben werden müssen und sie zum Quotieren von Markup in Text-Knoten und Attributen nötig sind.

3.1.2 Verlustlosigkeit

Damit ein Konversionsprozess als *verlustlos* angesehen werden kann, müssen folgende Bedingungen erfüllt sein:

1. Eingabedokument und Ausgabedokument müssen logisch äquivalent sein.
2. Sowohl das JSON-Zwischenprodukt als auch das XML-Ausgabedokument müssen gültige Dokumente sein.

Logisch äquivalent (vgl. Bed. 1) heißt nicht, dass das XML-Dokument nach dem Konversions-Round-Trip bitidentisch mit dem Ursprungsdokument sein muss – es ist ausreichend, dass die Struktur der beiden Dokumente übereinstimmt. Daher werden beide Dokumente zunächst kanonisiert[1] und im Anschluss verglichen. Stimmt die kanonische Form beider XML-Dokumente überein, war die Konversion verlustlos.

Definition 2 (Verlustlosigkeit von Konversion). Sei $V := \{x \mid x \text{ ist ein valider JSON-Wert}\}$ und $W := \{x \mid x \text{ ist ein wohlgeformtes XML-Dokument}\}$.
Ein Konversionverfahren $K = (f_{enc}, f_{dec})$ heißt *verlustlos* genau dann wenn

$$(f_{dec} \circ f_{enc})(x) \overset{\text{c14n}}{=} x \tag{3.1}$$

$$f_{enc} : W \mapsto V \tag{3.2}$$

$$f_{dec} : V \mapsto W \tag{3.3}$$

für alle $x \in W$.

Für die Feststellung, dass ein ein Konversionsverfahren verlustbehaftet ist, ist es ausreichend, *ein einzelnes* wohlgeformtes XML-Dokument zu finden, das das Verfahren nicht verlustlos konvertiert. Um jedoch hinreichend zu beweisen, dass ein Konversionsverfahren vollständig verlustlos arbeitet und somit die Bedingungen aus Def. 2 erfüllt, müsste die Umwandlung für *alle* wohlgeformten XML-Dokumente überprüft werden. Da die Menge der möglichen wohlgeformten XML-Dokumente – auch bei Beachtung der Einschränkungen aus Abschnitt 3.1.1 – unendlich ist, ist eine deduktive Art der Beweisführung jedoch nicht möglich.

[1] Vgl. Abschnitt 2.1.5

Um zu entscheiden, ob ein Konversionsverfahren verlustlos ist, verfolgt die vorliegende Arbeit daher einen empirisch-induktiven Ansatz: Wenn für eine ausreichend große Anzahl verschiedener Fälle gezeigt wird, dass die Bedingung erfüllt ist[2], kann eine allgemeine Erfüllung der Bedingung induziert werden [Rud53, S. 2].

Daher wird die Verlustlosigkeit einer Konversion im Folgenden anhand einzelner XML-Strukturen und Problemstellungen erläutert. Diese bilden die Grundlage für die anschließende Erstellung konkreter Testfälle in Form von XML-Dokumenten (vgl. Abschn. 3.3.1).

3.1.2.1 Elemente und Attribute

Als Kerninhalt eines jeden XML-Dokuments müssen Elemente und deren Beziehung zueinander erhalten bleiben. Ebenso dürfen Attribute bei der Konversion nicht verloren gehen. Auch die Tag-Namen der Elemente transportieren relevante Information, daher müssen sie beibehalten werden – dies gilt auch für den Tag-Namen des Wurzelelements.

3.1.2.2 Namespaces

Namespaces bieten eine Möglichkeit, Teilen eines XML-Dokuments eine bestimmte Semantik zuzuweisen. Die Zuordnung zwischen Elementen und Namespaces im Dokument darf daher nicht verändert werden.

Obwohl Namespace-Prefixe eigentlich frei gewählt werden können, ist es möglich, dass auch sie wichtige Informationen enthalten. Dies wäre beispielsweise dann der Fall, wenn ein im Dokument vorhandener XPath-Ausdruck ein Namenraumprefix referenziert [C14N, Abschn. 4.4]. Bei einer Umbenennung von Prefixen wäre die korrekte Evaluation des XPath-Ausdrucks nicht mehr möglich. Die Prefix-Bezeichner müssen also erhalten bleiben.

3.1.2.3 Character Data

Character Data ist in XML-Dokumente ein ebenso wichtiger Träger von Information wie Elemente. Dabei kann er im Inhaltsteil von Elementen in zwei Varianten vorkommen: Als normaler Text-Knoten oder als CDATA-Abschnitt.

Mithilfe von CDATA-Abschnitten lässt sich Text, der Markup-Zeichen wie beispielsweise das Kleiner-Zeichen[3] enthält, direkt in ein XML-Dokument einbetten, ohne dass diese Zeichen als Markup interpretiert werden.

[2] *hier:* die XML-Dokumente vor und nach der Konversion sind logisch äquivalent
[3] Unicode-Codepoint `U+003C`: `LESS-THAN SIGN`

Dies ist vor allem dann sinnvoll, wenn es unpraktikabel ist, alle Markup-Zeichen im Text einzeln durch die jeweilige Zeichen- oder Entity-Referenz zu ersetzen. CDATA stellt somit eine weitere Möglichkeit dar, Zeichendaten in einem XML-Dokument anzugeben. [XML, Abschn. 2.4]

Der Unterschied zwischen Zeichendaten aus CDATA-Abschnitten und solchen, bei denen dies nicht der Fall ist, ist jedoch lediglich ein syntaktischer. Daher werden bei der XML-Kanonisierung alle CDATA-Abschnitte im Eingabedokument durch den entsprechenden *Character Content* ersetzt [C14N, Abschn. 2.1].

Für die Verlustlosigkeit der Konversion ist es somit unerheblich, ob die CDATA-Abschnitte im Ursprungsdokument als solche erhalten bleiben, oder lediglich die Zeichendaten beibehalten werden.

3.1.2.4 Kommentare

XML verfügt über die Möglichkeit, Dokumente mit Kommentaren zu versehen. Diese sind jedoch nicht Teil der Zeichendaten des XML-Dokuments. Die Möglichkeit, Kommentare programmatisch auszuwerten, können XML-Parser zwar bereitstellen, dies ist jedoch optional.

Zudem ist auch bei der Implementierung der XML-Kanonisierung die Unterstützung von *Kanonischem XML mit Kommentaren* lediglich empfohlen, während die Möglichkeit der Umwandlung in *Kanonisches XML* ausschließlich aller Kommentare zwingend erforderlich ist. [C14N, Abschn. 2.1]

Folglich ist es nicht nötig, dass sich die Kommentare im XML-Eingabedokument nach dem XML→JSON→XML Roundtrip auch in der Ausgabe wiederfinden.

3.1.2.5 Processing Instructions (PIs)

Wie bereits SGML unterstützt XML die Einbettung von Anweisungen, die für die verarbeitetende Applikation bestimmt sind. Diese werden Processing Instruction (PI) genannt (vgl. Abschn. 2.1.1).

In der Praxis werden PIs eher selten eingesetzt. Genutzt werden sie unter anderem um Darstellungsinformationen in Form von Stylesheets mit XML-Dokumenten zu verknüpfen [XMLStyle, Abschn. 4].

Ein weiteres prominentes Beispiel für eine Praxisanwendung ist die Microsoft Office Suite, die seit der Version 2003 Office-Dokumente als einzelne XML-Dateien speichern kann. Diese setzen die unspezifische Dateiendung `*.xml` ein und würden daher mit einem generischem XML-Anzeigeprogramm geöffnet werden. Mithilfe der Processing Instruction „`<?mso-application progid="Word.Document"?>`“ werden Windows- bzw.

Internet Explorer angewiesen, sie stattdessen als Microsoft-Word-Dokumente zu behandeln. [Tve08, Abschn. 3.2]

Auch die XSLT-Stylesheets des DocBook-Formats nutzen Processing Instructions, um spezielle Formatierungen für die verschiedenen Ausgabeformate festzulegen. [DBXSL, Kapitel „User Reference: PIs“]

Ebenso wie Kommentare sind sie nicht Teil der Zeichendaten eines XML-Dokuments, müssen jedoch laut Spezifikation zwingend vom XML-Parser an die verarbeitende Applikation weitergereicht werden [XML, Abschn. 2.6].

Auch bei der XML-Kanonisierung bleiben die PIs erhalten. [C14N, Abschn. 2.3]

3.1.2.6 Dokumentordnung

Elementordnung Die C14N-Spezifikation bezieht sich bezüglich der Ordnung auf die W3C-Empfehlung zur XML Path Language (XPath) [C14N, Abschn. 2.2]:

> There is an ordering, document order, defined on all the nodes in the document corresponding to the order in which the first character of the XML representation of each node occurs in the XML representation of the document after expansion of general entities. Thus, the root node will be the first node. Element nodes occur before their children. Thus, document order orders element nodes in order of the occurrence of their start-tag in the XML (after expansion of entities). [XPath, Abschn. 5]

Folglich muss die Reihenfolge der Elemente bei der Konversion beibehalten werden.

Ordnung von Attributen und Namespaces Der Reihenfolge, in der die Attribute eines Elements im Start-Tag bzw. Leeres-Element-Tag angegeben wurden, kommt laut XML-Spezifikation keine Bedeutung zu [XML, Abschn. 3.1].

Dies wird auch von der XPath-Spezifikation untermauert, die die Reihenfolge von Namespace-Deklarationen und Attributen als implementierungsabhängig festlegt und somit keine feste Ordnung garantiert [XPath, Abschn. 5]. Die Reihenfolge der Attribute eines Elements nach einem XML→JSON→XML-Roundtrip ist daher beliebig und muss nicht mit der Reihenfolge vor der Umwandlung identisch sein.

3.1.2.7 Whitespace

XML-Parser müssen grundsätzlich jeglichen Whitespace – also Leerzeichen, Tabs und Zeilenumbrüche – innerhalb des Wurzelelements eines Dokuments an die verarbeitende Applikation weiterreichen [XML, Abschn. 2.10].

Tim Bray, einer der Autoren des XML-Standards, schreibt dazu:

> XML has an incredibly simple rule about how to handle white space, that is contained in this one sentence: "If it ain't markup, it's data." Under no circumstances will an XML processor discard some white space because, in the processor's opinion, it is not "significant". [Bra98, Abschn. 2.1 T-2]

Ausgenommen hiervon ist jedoch Whitespace außerhalb des Wurzelelements, dem keine Bedeutung zukommt [C14N, Abschn. 2.1].

Zudem wird Whitespace in bestimmten Fällen vor der Weiterverarbeitung durch den XML-Prozessor *normalisiert*:

1. Zeilenenden, d.h. die verschiedenen Kombinationen der Wagenrücklauf- und Zeilenvorschub-Zeichen, werden zu einem einfachen Zeilenvorschub umgewandelt [XML, Abschn.. 2.11].
2. In Attributen vorkommende Whitespace-Zeichen (keine Character References) werden zu Leerzeichen umgewandelt [XML, Abschn. 3.3.3]

Zwar kann Whitespace durch DTDs oder XSD-Schemata auch als insignifikant markiert werden [Pag05], dies wird jedoch nur von *validierenden* XML-Prozessoren beachtet und geht somit über den Rahmen der vorliegenden Arbeit hinaus.

3.1.2.8 Mixed Content

Eine Besonderheit von XML und auch SGML ist, dass die Spezifikation sogenannten *Mixed Content* erlaubt. Dieser liegt vor, wenn ein Element sowohl Character Data als auch Kindelemente enthält [XML, Abschn. 3.2.2].

Beispiel 12 (Mixed Content). Ist *Character Data* mit Elementen durchsetzt, wird dies *Mixed Content* genannt.

```
<mixed>This is an element<br /> containing <emph>mixed
↪  content</emph>.<mixed>
```

Mixed Content stellt XML-Parser vor besondere Herausforderungen [McG02]. Zudem existiert kein JSON-Äquivalent von *Mixed Content*, was eine Konversion erschwert.

3.1.2.9 Typinferenz bei der Konversion zu JSON

Zwar ist es möglich, mittels einer XML Schema Definition (XSD) die in einem XML-Dokument enthaltenten Datentypen genauer festzulegen, direkte syntaktische Unterstützung für Zahlen bietet XML jedoch im Gegensatz zur JavaScript Object Notation nicht.

Wird kein Schema verwendet bzw. legt ein Schema den Datentyp eines Elements nicht anderweitig fest, so ist es standardmäßig vom Typ „`xsd:anyType`“:

> A special complex type definition, (referred to in earlier versions of this specification as 'the ur-type definition') whose name is **anyType** in the XSD namespace, is present in each ·*XSD schema*·. The **definition of anyType** serves as default type definition for element declarations whose XML representation does not specify one. [XSD, Abschn. 2.2.1.1]

Elemente dieses Typs unterliegen keinen Beschänkungen. Daher ist es ohne Zuhilfenahme eines Schemas nicht möglich, Aussagen über den Wertebereich eines Elements, eines Attributs oder eines Text-Knotens zu treffen.

Beispiel 13 (Type Inference). Das folgende XML-Dokument enthält einen numerischen Wert, wobei in XML kein syntaktischer Unterschied zwischen Strings und Zahlen besteht.

```xml
<price>5.99</price>
```

Die JSON-Entsprechung des XML-Dokuments *ohne* Typinferenz enthält den Zahlenwert als String:

```json
{ "price": "5.99" }
```

Wird jedoch Typinferenz verwendet, enthält die JSON-Struktur die Daten als Wert vom Typ `Number`:

```json
{ "price":  5.99  }
```

Ist das Schema eines XML-Dokuments nicht bekannt, scheint es daher naheliegend, die Datentypen aus den im Dokument enthaltenen Werten abzuleiten. Enthält ein Text-Knoten oder ein Attribut beispielsweise die Zeichenkette `123`, könnte daraus auf einen numerischer Datentyp geschlossen werden. Dieses Prinzip der *Type Inference* nutzt beispielweise Microsoft im Rahmen des *.NET Frameworks* um das XML-Schema auf Basis von einem oder mehreren XML-Dokumenten zu „erraten“ [Mic17].

Im Rahmen der XML zu JSON-Konversion ist es zwar wünschenswert, die nativen Datentypen der JSON-Spezifikation voll auszunutzen, ein solches Vorgehen bringt jedoch mehrere Probleme mit sich:

Fehlerkennung von Typen Die oben beschriebene Vorgehensweise zur Typableitung kann zur Fehlerkennung von Datentypen führen.

Darf ein Text-Knoten beispielsweise eigentlich beliebige Zeichen enthalten, enthält aber *zufälligerweise* ausschließlich Ziffern, würde fälschlicherweise ein Zahlentyp erkannt werden. Dies könnte zu Problemen mit der verarbeitetenden Applikation führen, die stattdessen eine Zeichenkette erwartet.

Einschränkungen durch Wertebereiche Ein weiteres Problem bei der *Type Inference* kann durch die unterschiedlichen Wertebereichsgrenzen der verschiedenen Datentypen entstehen. Während Zeichenketten in vielen Programmiersprachen mehrere tausend Zeichen lang sein dürfen, ist der Wertebereich von numerischen Datentypen in der Regel deutlich eingeschränkter. Wird ein Zahlenwert vom Konversionsprogramm also in einen nativen Datentyp umgewandelt, für den ein kleinerer Wertebereich gilt als durch Strings darstellbar sind, führt dies zu Fehlern oder Informationsverlust.

Die Programmiersprache *JavaScript* erlaubt beispielsweise Strings mit einer Länge von bis zu $n = 2^{53} - 1 \approx 9 \cdot 10^{15}$ Zeichen, d.h. als Strings abgelegte Zahlen können rund 9 Billarden Stellen haben [ECMA262, Abschn. 6.1.4]. Für Zahlen im Dezimalsystem entspräche dies dem Wertebereich $\left\{x \in \mathbb{Z} \mid -\left(10^{n-1} - 1\right) \leq x \leq 10^n - 1\right\}$.

Für den Datentyp `Number` nutzt JavaScript 64-bit-Fließkommazahlen nach dem IEEE-754-Standard[4] [IEEE754; ECMA262, Abschn. 4.3.20]. Für Ganzzahlen gilt daher der „sichere“ Wertebereich $\left\{x \in \mathbb{Z} \mid |x| \leq 2^{53} - 1\right\}$ [ECMA262, Abschn. 20.1.2.5]. Dies entspricht einer Zahl mit lediglich $\lfloor \log_{10}\left(2^{53} - 1\right) \rfloor + 1 = 16$ Ziffern im Dezimalsystem.

Zahlen außerhalb dieses Wertebereichs können nicht mehr fehlerfrei eingesetzt werden. Auf den möglichen Informationsverlust bei dem Einsatz von Zahlentypen in JSON wird in der IETF-Spezifikation daher explizit hingewiesen [RFC7159, Abschn. 6].

[4] Fließkommazahlen mit doppelter Genauigkeit (`binary64`-Typ nach IEEE-754)

Beispiel 14 (Informationsverlust durch Typumwandlung in JavaScript). Die Umwandlung von numerischen Zeichenketten in den `Number`-Typ kann in JavaScript bei Zahlen $> 2^{53} - 1$ zu Problemen führen.

```
1 var num1 = '9007199254740993'; // = 2^53 + 1
2 var num2 = parseInt(num1).toString();
3 if(num1 == num2) {
4     console.log(num1 + " == " + num2);
5 } else {
6     console.log(num1 + " != " + num2);
7 }
8 // Output: 9007199254740993 != 9007199254740992
```

Ähnliches gilt für Konversionsverfahren, die aus Werten wie „yes“ und „true“ den Boole'schen Datentyp ableiten: Nach der Konversion ist nicht mehr feststellbar, ob der Ursprungswert nun „yes“ oder „true“ lautete.

3.1.2.10 Unterstützung des Zeichenbereichs

Die XML-Spezifikation erlaubt eine große Anzahl verschiedener Unicode-Zeichen weit über den ASCII-Raum hinaus [XML, Regel 2]. Da *Character Data* bei der Konvertierung nicht verloren gehen darf (vgl. Abschn. 3.1.2.3), müssen Konverter in der Lage sein, diese Zeichen zu JSON zu übersetzen und ggf. korrekt zu quotieren.

Dies gilt auch für die Zeichen, von deren Einsatz in der XML-Spezifikation ausdrücklich abgeraten wird [XML, Abschn. 2.2], da es sich dennoch um wohlgeformtes XML handelt.

3.2 Auswahl der XML-JSON-Konverter

Inzwischen sind eine große Anzahl an Applikationen und Programmbibliotheken für die Umwandlung zwischen JSON und XML verfügbar.

Aufgrund des in dieser Arbeit zur Anwendung kommenden Prüfverfahrens konnten jedoch nur solche Konverter betrachtet werden, die

1. XML-Dokumente in JSON umwandeln können und
2. aus den so erhaltenen JSON-Strukuren wieder XML-Dokumente erstellen können.

Da für viele XML-Strukturen kein Äquivalent in der JSON-Spezifikation existiert, gibt es meist verschiedene Möglichkeiten der Abbildung von XML-Inhalten in JSON. Die gebräuchlichsten Varianten haben sich in Form verschiedener Konvertierungskonventionen herausgebildet, die jeweils beschreiben, wie XML-Strukturen in JSON dargestellt werden [Op311]. Dabei unterscheidet sich das anhand der verschiedenen Konventionen produzierte JSON zum Teil stark, beispielsweise darin, welcher JSON-Containertyp zur Darstellung eines XML-Elements genutzt wird oder wie Attribute dargestellt werden. Die für die Analyse im Rahmen der vorliegenden Arbeit ausgewählten Konverter sollten daher nach Möglichkeit verschiedene Umwandlungsverfahren und Konventionen implementieren.

Da die Formate insbesondere im Webbereich eingesetzt werden und Webservices sowie APIs ein relevantes Einsatzfeld für XML-JSON-Konversion darstellen, wurde bei der Auswahl zudem darauf geachtet, ein breites Spektrum verschiedener populärer Programmiersprachen aus diesem Bereich abzudecken. Sowohl die klassischen Programmiersprachen für Web-Applikationen – Java, PHP und JavaScript – als auch C#, Ruby

und Python, die im Web in Form von ASP.NET, Ruby-on-Rails bzw. Django oder Flask zum Einsatz kommen, werden durch die Auswahl abgedeckt.

Cobra vs Mongoose Diese Implementierung ist in Form eines Ruby-Gems verfügbar und übersetzt XML-Dokumente in Ruby-Datenstrukturen (Hashes), kann aber auch JSON-Daten ausgeben. Bei der Umwandlung setzt der Konverter auf die sogenannte *Badgerfish*-Konvention [Bat06; Op311, Abschn. 3].

GreenCape XML Converter Der in PHP implementierte Konverter kann XML-Daten in assoziative PHP-Arrays umwandeln. Diese können dann zu JSON-Werten se-

Tabelle 3.1: Übersicht der überprüften Konverter.

Konverter	Autor	Lizenz	Sprache	Version
Cobra vs Mongoose[a]	Paul Battley	MIT	Ruby	`0.0.2` 27.06.2006
GreenCape XML Converter[b]	Niels Braczek	MIT	PHP	`a830542` 02.07.2015
Json-lib[c]	Andres Almiray[1]	Apache 2.0	Java	`2.4` 14.12.2010
JsonML[d]	Stephen M. McKamey	MIT	JavaScript	`2.0.0` 09.04.2016
JXON[e]	Martin Raifer, Mozilla	GNU GPL 3.0	JavaScript	`2.0.0-beta.4` 22.11.2016
Json.NET[f]	James Newton-King	MIT	C#	`10.0.3` 18.06.2017
org.json.XML[g]	Sean Leary / JSON.org	MIT	Java	`20160810` 10.08.2016
Pesterfish[h]	Jacob Smullyan	MIT	Python	`1578db9` 22.11.2010
x2js[i]	Abdulla G. Abdurakhmanov	Apache 2.0	JavaScript	`185e410` 04.01.2016
x2js (Fork)[j]	Sander Saares / Axinom[2]	Apache 2.0	JavaScript	`3.1.0` 05.12.2016
xmljson[k]	S. Anand	MIT	Python	`0.1.7` 09.05.2017

[1] Basiert auf Software von Douglas Crockford.
[2] Fork der *x2js*-Bibliothek von Abdulla G. Abdurakhmanov.
[a] `https://rubygems.org/gems/cobravsmongoose`
[b] `https://github.com/GreenCape/xml-converter` [c] `http://json-lib.sourceforge.net/`
[d] `http://www.jsonml.org/` [e] `https://github.com/tyrasd/jxon`
[f] `https://github.com/stleary/JSON-java` [g] `https://github.com/JamesNK/Newtonsoft.Json`
[h] `https://bitbucket.org/smulloni/pesterfish/` [i] `https://github.com/abdmob/x2js`
[j] `https://github.com/x2js/x2js` [k] `https://github.com/sanand0/xmljson`

rialisiert werden.

Json-lib Die Java-Bibliothek `net.sf.json-lib` baut auf Software des JSON-Entwicklers Douglas Crockford auf und wird in über 400 anderen Programmen und Bibliotheken eingesetzt [Mvn1]. Das Paket beinhaltet unter anderem auch eine Klasse zum (De-)Serialisieren von XML-Daten.

JsonML Die JSON Markup Language (JsonML) ist ein JSON-basiertes Format zur Speicherung von XML-Markup. Neben der JavaScript-Bibliothek existieren auch Implementierungen in anderen Programmiersprachen, z. B. in Java[5] oder PHP[6].

JXON Bei dem JavaScript-Modul handelt es sich um eine bidirektionale Bibliothek für die Lossless JavaScript XML Object Notation und eine Weiterentwicklung der ursprünglich von Mozilla veröffentlichten Implementierung.

Newtonsoft Json.NET Das JSON-Framework für C#.NET wurde insgesamt mehr als 64 Millionen Mal von Paket-Repository NuGet heruntergeladen [Fou17a] und war zum Testzeitpunkt das am meisten heruntergeladene .NET-Paket [Fou17b]. Die darin enthaltene `JsonConvert`-Klasse ermöglicht dem Framework die Konversion von XML-Dokumenten in das JSON-Format.

org.json.XML Das Paket `org.json` ist die Referenzimplementierung des JSON-Formats für Java und bietet auch ein Konversionsverfahren in Form einer dedizierten XML-Klasse. Die Java-Bibliothek wird von mehr als 1 600 Projekten eingesetzt, darunter auch die Google Android Library, das Google Web Toolkit (GWT) oder die Objektserialisierungs-Bibliothek XStream [Mvn2].

Pesterfish Das Python-Modul Pesterfish konvertiert zwischen XML und Python-Dictionaries, die dann zu JSON serialisiert werden können. Laut dem Autor wurde das Konversionsverfahren als „Reaktion auf die Badgerfish-Konvention“ entwickelt[7]. Für die Verarbeitung der XML-Daten baut das Modul auf die `ElementTree`-API auf und erlaubt Entwicklern auch die Angabe eines eigenen Parsers, beispielsweise `defusedxml`.

x2js Die JavaScript-Bibliothek ist mit 475 Sternen und 219 Forks auf GitHub recht populär [Abd17] und erlaubt eine große Zahl von Einstellungsmöglichkeiten.

x2js (Fork) Hierbei handelt es sich um einen für die Verwendung mit NodeJS ausgelegten Fork der xj2s-Bibliothek, der seit der Abspaltung im Oktober 2015 unabhängig weiterentwickelt wird.

xmljson Das Python-Paket *xmljson* wurde mehr als 11 500 mal vom Python-Repository PyPI heruntergeladen [PyPI17]. Eine Besonderheit ist, dass es die Konversion von

[5] Das `org.json`-Paket stellt eine entsprechende Klasse bereit.

[6] Als Teil des FluentDOM-Projekts, siehe: `https://github.com/FluentDOM/FluentDOM/blob/master/src/FluentDOM/Serializer/Json/JsonML.php`

[7] Der Autor schreibt dies in einem Kommentar im Quelltext des Moduls.

XML-Daten in Python-Dictionaries/Lists bzw. JSON anhand von 6 verschiedenen Konventionen (*Abdera*, *Badgerfish*, *Cobra*, *GData*, *Parker* und *Yahoo*) unterstützt.

3.3 Methodik

3.3.1 Überprüfung der Konversionqualität

Zur Überprüfung der Konversionsqualität wurden für alle der in Abschnitt 3.1.2 festgelegten Kriterien und diskutierten Probleme ein oder mehrere Testfälle erstellt. Jeder Testfall besteht aus einer XML-Datei, anhand der bestimmte XML-Features oder Problemstellungen nachvollzogen werden können. Die Dateien werden dazu zunächst mit

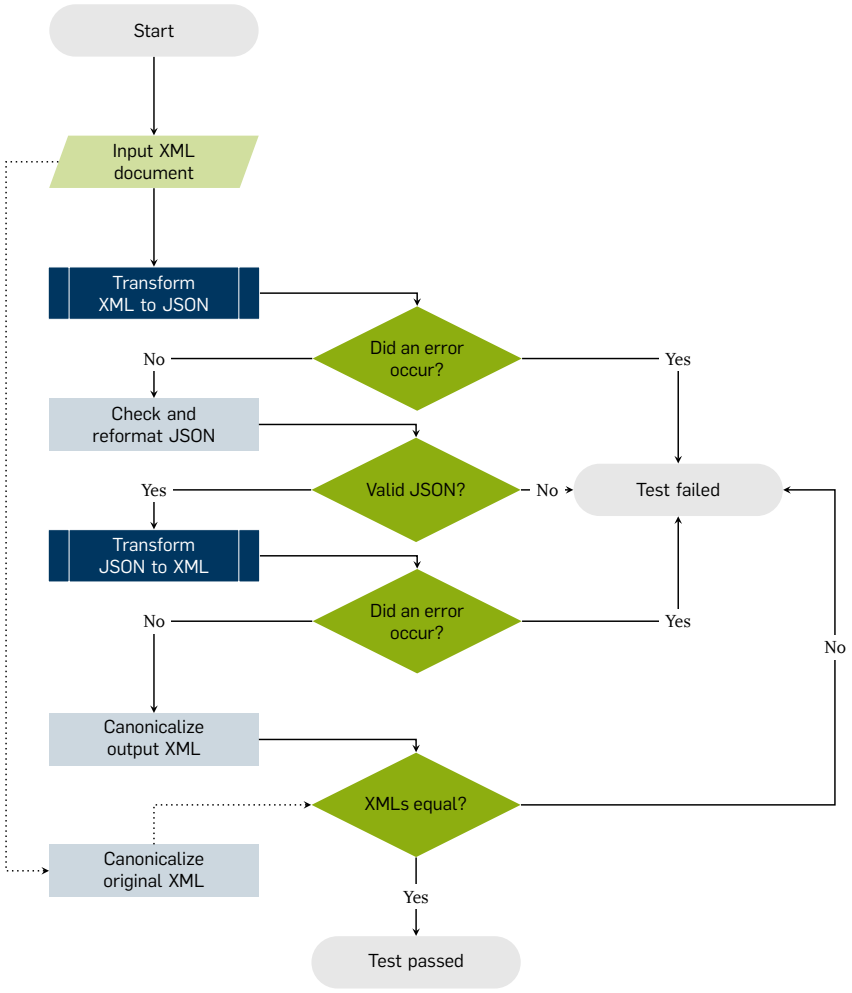


Abbildung 3.2: Ablauf der Konversionstests.

einem Konverter in das JSON-Format umgewandelt. Die daraus resultierenden JSON-Daten werden im Anschluss wieder zurück in das XML-Format konvertiert, d. h. es wird ein kompletter Round-Trip vollzogen.

Gemäß Definiton 2 in Abschnitt 3.1.2 gibt ein Abgleich zwischen den resultierenden XML-Daten und dem ursprünglichen XML-Dokument dann Aufschluss über den eventuellen Verlust signifikanter Informationen. Dazu wird überprüft, ob beide Dokumente logisch äquivalent sind, indem beide in *kanonisches XML* umgewandelt werden. Ob Informationsverlust aufgetreten ist, kann dann mit einem simplen Vergleich geprüft werden.

Es muss zudem sichergestellt sein, dass bei der Konversion auch tatsächlich valides JSON bzw. wohlgeformtes XML zurückgegeben wird. Da das vom Konverter zurückgelieferte Dokument im Zuge der XML-Kanonisierung geparst wird, werden Verstöße gegen die Wohlgeformtheitsanforderungen der XML-Spezifikation erkannt und führen zum Nichtbestehen des Tests.

Die JSON-Zwischendaten werden ebenfalls geparst und auf Verstöße gegen die JSON-Spezifikation untersucht. Zudem werden die Daten vor der Rückkonvertierung neu formatiert, sodass nur Informationen weitergegeben werden, die auch tatsächlich signifikant im Sinne der Spezifikation sind, d. h. dass beispielsweise Whitespace außerhalb von Zeichenketten oder die Reihenfolge von Wertpaaren innerhalb eines JSON-Objekts verloren gehen.

3.3.2 Überprüfung der Sicherheit

Um die Sicherheit der Konversion zu überprüfen, wird eine modifizierte Vorgehensweise genutzt. Die auf Sicherheitslücken in XML-Parsern abzielenden Test-Dokumente werden ebenfalls zunächst in das JSON-Format und im Anschluss wieder zurück in XML konvertiert. Allerdings ist dabei ausschließlich von Belang, dass keiner der in Abschnitt 2.1.6 beschriebenen Angriffe ausgelöst wird – die Korrektheit der Konversion wird hierbei nicht überprüft. Auch ein Abbruch des Programms hierbei gilt als Erfolg.

Stattdessen wird ein Sicherheitstest als Fehlschlag gewertet, wenn eine oder mehrere der folgenden Bedingungen zutrifft:

1. Der Konversionsvorgang überschreitet zuvor festgelege Obergrenzen für die Allozierung von Arbeitsspeicher, verbraucht zuviel CPU-Zeit oder dauert zu lange.
2. Während der Konversion sendet der Konverter Anfragen an einen Webserver.
3. Die vom Konverter ausgegebenen JSON- oder XML-Dateien enthalten eine bestimmte Zeichenkette.

Ist der eingesetzte XML-Parser verwundbar gegenüber Angriffen aus dem Bereich Denial of Service (DoS), führt das Parsen der Testdateien zu einem erhöhten Verbrauch

an Systemressourcen. Die Obergrenzen für Arbeitsspeicherbelegung, CPU-Auslastung und Zeitdauer sind so angelegt, dass ein erfolgreicher Angriff zwangsläufig gegen die erste Bedingung verstößt und damit zum Nichtbestehen des Tests führt.

Um zu prüfen, ob Server Side Request Forgery (SSRF) möglich ist, wird ein HTTP-Server gestartet, der von den entsprechenden Testfällen referenziert wird. Ruft ein Konverter während des Parsings eine der URLs auf, wird die Anfrage vom Server aufgezeichnet. Sind bei dem Server bis zum Abschluss der Konversion keine Anfragen eingegangen (Bed. 2), ist der Konverter offenbar nicht anfällig für solche Angriffe.

Die Testfälle für die Suche nach Schwachstellen der Kategorie File System Access ver-

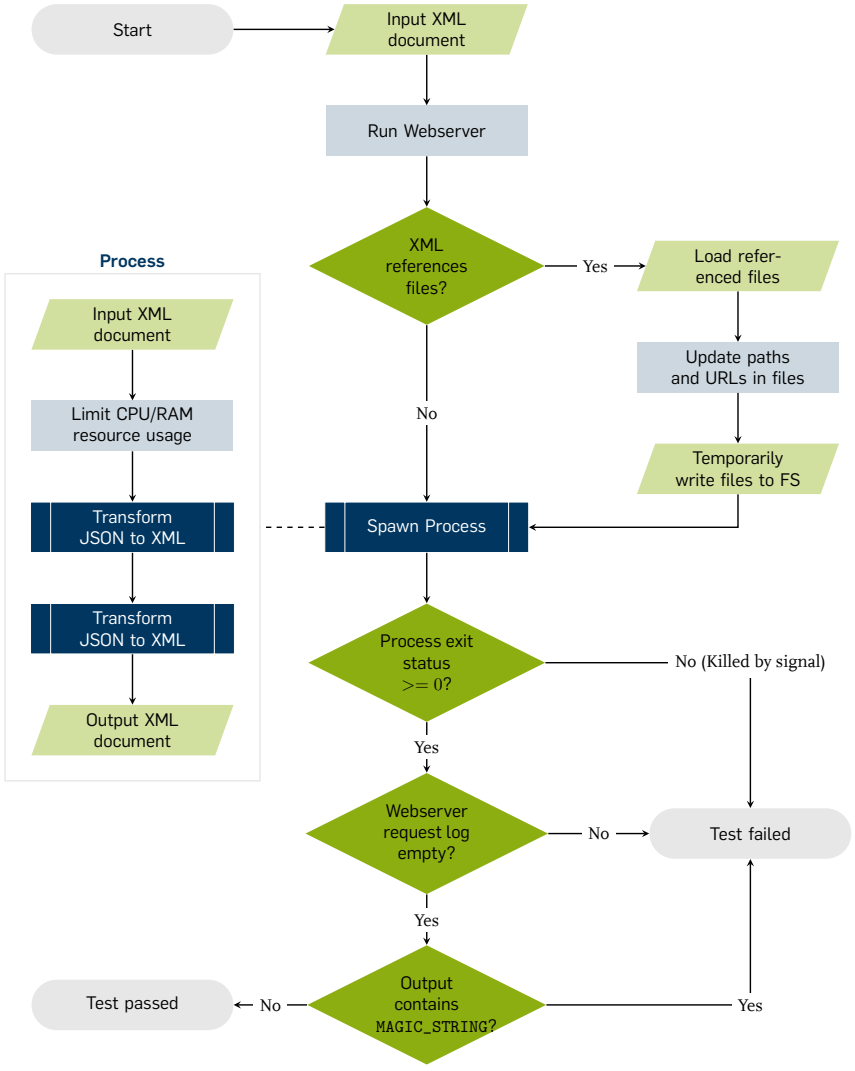


Abbildung 3.3: Ablauf der Sicherheitstests.

weisen auf Dateien, die eine bestimmte Zeichenkette enthalten. Ist diese Zeichenkette in den vom Konverter ausgegebenen JSON- oder XML-Daten enthalten, wurde vom Parser auf das Dateisystem zugegriffen und der Konverter ist verwundbar.

Alle eingesetzen Sicherheits-Testfälle basieren auf den XML-Testdokumenten aus dem von Christoph Späth betreuten „DTD-Attacks“-Repository des Lehrstuhls für Netz- und Datensicherheit der Ruhr-Universität Bochum [DTDAttack].

3.3.3 Technische Umsetzung

Zur einfacheren Durchführung der Testreihen wurde ein Evaluations-Framework auf Basis von Python 3.5 implementiert, das ca. 2600 Zeilen Programmcode (LOC) umfasst. Der Testprozess wird dabei weitestgehend automatisiert.

Um die XML-Kanonisierung durchzuführen, nutzt das Framework die `write_c14n()`-Methode aus der *lxml*-Bibliothek[8]. Die Umwandlung in kanonisches XML erfolgt dabei nicht-exklusiv (vgl. Abschn. 2.1.5.4) und ohne Kommentare – davon ausgenommen ist aber ein Testfall, der die Unterstützung für Kommentare überprüft.

Für das XML-Parsing selbst kommt *defusedxml*[9] zum Einsatz, das die gängigen Python-APIs zum Parsen von XML-Dokumenten einem Security-Hardening unterzieht und daher nicht anfällig gegenüber fast allen Angriffsvektoren ist [Spä16, Abschn. 9.5].

Zur Prüfung und Formatierung der JSON-Daten wird das Python-Modul *demjson*[10] eingesetzt, das über umfangreiche Möglichkeiten zum Finden von Verstößen gegen die JSON (einen sogenannten *Linter*) verfügt. Zusätzlich zum `strict`-Modus werden einige besondere Einstellungen vorgenommen: Byte-Order-Marks sind in Übereinstimmung mit der JSON-Spezifikation explizit verboten. Null-Bytes und andere von der Spezifikation nicht verbotene, aber möglicherweise Kompatibilitätsprobleme verursachende Zeichen sind erlaubt, da es sich hierbei um korrektes JSON handelt.

Für die Sicherheitstests wird ein eigener Prozess gestartet. Da als Host-System Linux zum Einsatz kommt, wurde die Begrenzung der Systemressourcen mithilfe des `setrlimit()`-Syscalls umgesetzt. Werden die so festgelegten Obergrenzen für die zur Verfügung stehende CPU-Zeit und Größe des adressierbaren Speichers des Prozesses überschritten, wird der Prozess vom Kernel durch das Signal `SIGXCPU` bzw. `SIGKILL` oder `SIGSEGV` beendet. Das Framework prüft nach Beendigung des Prozesses, ob der Exitcode auf die Terminierung durch ein Signal hinweist oder ob der Prozess normal beendet wurde.

[8] http://lxml.de/
[9] https://pypi.python.org/pypi/defusedxml
[10] http://deron.meranda.us/python/demjson/

4 Ergebnisse

Es wurden insgesamt 123 Testfälle verwendet und 11 verschiedene Konverter überprüft. Allerdings konnte keiner der Konverter alle Anforderungen aus Abschnitt 3.1 erfüllen. Im Folgenden werden die Ergebnisse der verschiedenenen Konverter vorgestellt. Ausgabebeispiele zur Veranschaulichung befinden sich in Anhang A.

		Cobra vs Mongoose	GreenCape XML	Json-lib	JsonML	JsonML (patched)[1]	JXON	Json.NET	org.json.XML	Pesterfish	Pesterfish (defusedxml)	x2js (fork)	x2js	xmljson (Abdera)	xmljson (Badgerfish)	xmljson (Cobra)	xmljson (GData)	xmljson (Parker)	xmljson (Yahoo)
1	Attribute (multiple)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓		✓
2	Attribute	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓		✓
3	CDATA close in Text Node	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4	Escaped CDATA section	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5	CDATA section with Markup	✓	✗		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
6	CDATA support	✓	✗		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
7	Comments[2]	✗		✗															
8	Deep nesting	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	
9	Duplicate Child tag Names (Alternating Order)		✓		✓	✓				✓	✓								
10	Duplicate child tag names (different NS prefix)	✓	✓	✓	✓	✓	✓	✓				✓							
11	Duplicate child tag names	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓			✓		✓	✓	✓
12	Element Order		✓		✓	✓				✓	✓								
13	Empty Elements	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓		✓
14	Mixed Content	✗			✓	✓				✓	✓		✗						
15	Processing Instructions (Attibute Data)	✗		✗		✓		✓											
16	Processing Instructions (Markup)	✗		✗		✓		✓											
17	Processing Instructions outside root	✗				✓		✗											
18	Processing Instructions (Arbitrary Data)	✗		✗		✓		✓											
19	Processing Instructions (Whitespace)	✗		✗		✓													
20	Processing Instructions (Basic)	✗		✗		✓		✓											
21	Root Element Attribute	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓		
22	Root Element Tag Name	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	
23	Simple Element List	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	
24	Type Inference with floats (Attr)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓						✓
25	Type Inference with floats	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓						
26	Type Inference with doubles (Attr)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓						✓
27	Type Inference with doubles	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓						
28	Type Inference with Boolean values (Attr)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓		✓
29	Type Inference with Boolean values	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓	✓	
30	Type Inference with Big Integers (Attr)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓		✓
31	Type Inference with Big Integers	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓		✓	✓	
32	Whitespace (Indentation)				✓	✓				✓	✓		✗						
33	Whitespace (Mixed Content)	✗			✓	✓				✓	✓		✗						
34	Whitespace (Clean/Dirty)			✓	✓	✓				✓	✓	✓	✓					✓	
35	Namespace declaration (multiple)		✓	✓	✓	✓		✓				✓	✓						
36	Namespace declaration with prefix	✓	✓	✓	✓	✓	✓	✓				✓	✓						
37	Namespace declaration	✓	✓	✓	✓	✓	✓	✓				✓	✓						
	Gesamt	**23**	**24**	**19**	**30**	**36**	**23**	**28**	**9**	**26**	**26**	**25**	**23**	**4**	**16**	**4**	**16**	**11**	**12**

[1] Erweiterte JsonML-Version (vgl. Kapitel 5)
[2] optional, Unterstützung nicht notwendig

Tabelle 4.1: Grundlegende Konversions-Testergebnisse.

		Cobra vs Mongoose	GreenCape XML	Json-lib	JsonML	JsonML (patched)[1]	JXON	Json.NET	org.json-XML	Pesterfish	Pesterfish (defusedxml)	x2js (fork)	x2js	xmljson (Abdera)	xmljson (Badgerfish)	xmljson (Cobra)	xmljson (GData)	xmljson (Parker)	xmljson (Yahoo)
38	C0 set (u000009, 00000A & 00000D)				✓	✓				✓	✓	✓	✓				✓		
39	Space Char (u000020)			✓	✓	✓				✓	✓	✓	✓				✓		
40	ASCII printable (u000021-00007E)	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓		
41	Discouraged (u00007F-000084)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
42	NEL Control Char (u000085)	✗	✓		✓	✓	✓			✓	✓	✓	✓				✓		
43	Discouraged (u000086-00009F)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
44	BMP plane I (u0000A0-000FFF)	✗	✓		✓	✓				✓	✓	✓	✓						
45	BMP plane II (u001000-001FFF)	✗	✓		✓	✓				✓	✓	✓	✓						
46	BMP plane III (u002000-002FFF)	✗	✓		✓	✓				✓	✓	✓	✓				✓		
47	BMP plane IV (u003000-003FFF)	✗	✓		✓	✓				✓	✓	✓	✓				✓		
48	BMP plane V (u004000-004FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
49	BMP plane VI (u005000-005FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
50	BMP plane VII (u006000-006FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
51	BMP plane VIII (u007000-007FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
52	BMP plane IX (u008000-008FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
53	BMP plane X (u009000-009FFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
54	BMP plane XI (u00A000-00AFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓						
55	BMP plane XII (u00B000-00BFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
56	BMP plane XIII (u00C000-00CFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
57	BMP plane XIV (u00D000-00D7FF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
58	BMP plane XV (u00E000-00EFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
59	BMP plane XVI (u00F000-00FDCF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
60	BMP plane Discouraged (u00FDD0-00FDEF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
61	BMP plane XVII (u00FDF0-00FFFD)	✗	✓		✓	✓				✓	✓	✓	✓						
62	SMP plane (u010000-01FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
63	SMP plane Discouraged (u01FFFE-01FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
64	SIP plane (u020000-02FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
65	SIP plane Discouraged (u02FFFE-02FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
66	Unassigned plane 3 (u030000-03FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
67	Plane 3 Discouraged (u03FFFE-03FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
68	Unassigned plane 4 (u040000-04FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
69	Plane 4 Discouraged (u04FFFE-04FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
70	Unassigned plane 5 (u050000-05FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
71	Plane 5 Discouraged (u05FFFE-05FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
72	Unassigned plane 6 (u060000-06FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
73	Plane 6 Discouraged (u06FFFE-06FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
74	Unassigned plane 7 (u070000-07FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
75	Plane 7 Discouraged (u07FFFE-07FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
76	Unassigned plane 8 (u080000-08FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
77	Plane 8 Discouraged (u08FFFE-08FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
78	Unassigned plane 9 (u090000-09FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
79	Plane 9 Discouraged (u09FFFE-09FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
80	Unassigned plane 10 (u0A0000-0AFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
81	Plane 10 Discouraged (u0AFFFE-0AFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
82	Unassigned plane 11 (u0B0000-0BFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
83	Plane 11 Discouraged (u0BFFFE-0BFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
84	Unassigned plane 12 (u0C0000-0CFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
85	Plane 12 Discouraged (u0CFFFE-0CFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
86	Unassigned plane 13 (u0D0000-0DFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
87	Plane 13 Discouraged (u0DFFFE-0DFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
88	SSP plane (u0E0000-0EFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
89	SSP plane Discouraged (u0EFFFE-0EFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
90	SPUA-A plane (u0F0000-0FFFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
91	SPUA-A plane Discouraged (u0FFFFE-0FFFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
92	SPUA-B plane (u100000-10FFFD)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
93	SPUA-B plane Discouraged (u10FFFE-10FFFF)	✗	✓		✓	✓	✓			✓	✓	✓	✓	✓		✓	✓		
	Gesamt	**1**	**54**	**2**	**56**	**56**	**49**	**1**	**0**	**56**	**56**	**56**	**56**	**0**	**47**	**0**	**47**	**52**	**0**

[1] Erweiterte JsonML-Version (vgl. Kapitel 5)

Tabelle 4.2: Ergebnisse der Tests bezüglich Unterstützung der von der XML-Spezifikation erlaubten Zeichen.

		Cobra vs Mongoose	GreenCape XML	Json-lib	JsonML	JsonML (patched)[1]	JXON	Json.NET	org.json.XML	Pesterfish	Pesterfish (defusedxml)	x2js (fork)	x2js	xmljson (Abdera)	xmljson (Badgerfish)	xmljson (Cobra)	xmljson (GData)	xmljson (Parker)	xmljson (Yahoo)
94	MSWord 2003 XML file	✗		✗		✓		✗											
95	MSWord XML file	✗		✗		✓		✗											
96	OpenDocument sample file	✗	✗	✗	✓	✓							✗						
97	XML 1.0 5th Edition (XHTML version)	✗		✗	✓	✓		✗					✗						
98	XML 1.0 5th Edition (XML version)	✗	✗	✗		✓		✗					✗			✗		✗	
99	SVG Car Demo	✗		✗	✓	✓							✗						
100	DocBook V5.0 transition guide	✗	✗	✗	✓	✓		✗					✗						
101	MusicBrainz.org API Response	✗	✓	✗	✓	✓													
102	SVG Photos Demo		✗		✓	✓							✗						
103	RSS 0.91 sample document				✓	✓				✓	✓		✗						
104	RSS 0.92 sample document	✗			✓	✓				✓	✓		✗						
105	RSS 2.0 sample document				✓	✓				✓	✓		✗						
106	XML tree Example (Thesis)	✗				✓							✗						
	Gesamt	**0**	**1**	**0**	**9**	**13**	**0**	**0**	**0**	**3**	**3**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

[1] Erweiterte JsonML-Version (vgl. Kapitel 5)

Tabelle 4.3: Testergebnisse der Konverter bei komplexen Dokumenten.

		Cobra vs Mongoose	GreenCape XML	Json-lib	JsonML	JsonML (patched)[1]	JXON	Json.NET	org.json.XML	Pesterfish	Pesterfish (defusedxml)	x2js (fork)	x2js	xmljson (Abdera)	xmljson (Badgerfish)	xmljson (Cobra)	xmljson (GData)	xmljson (Parker)	xmljson (Yahoo)
107	Billion Laughs Attack (PE Version)	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
108	Billion Laughs Attack	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
109	Entity Recursion Attack (PE)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
110	Entity Recursion Attack	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
111	Quadratic Blowup Attack	✓	✓		✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
112	Doctype Parameter Entity FSA Attack	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
113	External PE DTD FSA Attack	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
114	XInclude FSA Attack	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
115	XSLT FSA Attack	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
116	XXE FSA Attack (PE)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
117	XXE FSA Attack	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
118	DOCTYPE URL Invocation	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
119	SchemaLocation NoNamespace URL Invocation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
120	SchemaLocation URL Invocation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
121	XInclude URL Invocation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
122	XXE URL Invocation (PE)	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
123	XXE URL Invocation	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Gesamt	**17**	**17**	**11**	**17**	**17**	**17**	**14**	**17**	**15**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**	**17**

[1] Erweiterte JsonML-Version (vgl. Kapitel 5)

Tabelle 4.4: Ergebnisse der Tests auf Sicherheitslücken.

4.1 Cobra vs Mongoose

Die Reihenfolge der Elemente sowie Whitespace werden von *Cobra vs Mongoose* verworfen. Das Auftreten von Mixed Content, PIs und Kommentaren im Urspungsdokument führt zu Fehlern bei der Rückkonvertierung von JSON zu XML.

Enthält ein XML-Dokument Zeichen außerhalb des ASCII-Bereichs, führt dies zum Absturz des Programms.

Bei mehreren verschiedenen Default Namespaces innerhalb eines Dokuments werden diese zusammengefasst und lediglich die zuletzt genannte Namespace-Deklaration beibehalten. Zudem geht die Position der Namespace-Prefix-Deklarationen im Dokument verloren.

Cobra vs Mongoose setzt den XML-Parser *REXML* aus der Ruby-Standardbibliothek ein und ist für keinen der überprüften Angriffe verwundbar.

4.2 GreenCape XML Converter

Bei der Umwandlung von JSON zu XML-Daten fügt der Konverter hinter allen Elementen automatisch Zeilenumbrüche und Einrückungen mit einer Breite von vier Leerzeichen ein. Dies führt dazu, dass der Konverter in unmodifizierter Form keinen der Tests besteht. Um die Überprüfung der anderen, davon unabhängigen Aspekte des Konverters gewährleisten zu können, musste dieses Verhalten durch einen Patch entfernt werden (siehe Anhang C.1).

Die Konversion der umfangreichen ODF-Spezifikation in Form einer `*.fodt`-Datei mithilfe des *GreenCape XML Converters* schlägt fehl. Der entsprechende Versuch wurde abgebrochen, nachdem der PHP-Prozess seit rund 3 Stunden bei voll ausgelasteter CPU eingefroren war. Eine Analyse mithilfe des Tools `strace` zeigte, dass sich der PHP-Interpreter in einer Endlosschleife aus aufeinanderfolgenden `mmap()`- und `mummap()`-Syscalls befand (siehe Abb. 4.5).

Abgesehen von diesem möglicherweise für DoS-Attacken ausnutzbaren Verhalten ist der *GreenCape XML Converter* für keinen der getesteten Angriffe verwundbar.

CDATA-Sektionen in XML-Dokumenten führen zu Fehlern im von der PHP-Bibliothek selbst implementierten XML-Parser und bringen das Programm zum Absturz.

Bei der Konversion von Mixed Content wird sämtlicher Textinhalt außerhalb der Kindelemente verworfen. Whitespace geht ebenso verloren. PIs werden ignoriert. Abgesehen von Whitespace bleiben alle Sonderzeichen bei der Umwandlung jedoch erhalten.

Der *GreenCape XML Converter* kann – nach Deaktivierung der automatischen Einrückung – als einziges Programm neben JsonML (vgl. Abschn. 4.4) die XML-Antwort auf eine Anfrage an die MusicBrainz-API verlustlos umwandeln.

4.3 Json-lib

Bei Kommentaren oder PIs innerhalb des Wurzelelements eines XML-Dokuments stürzt der Konverter ab. PIs außerhalb des Wurzelelements werden von *Json-lib* ignoriert. Bei der Konversion geht zudem der Tag-Name des Wurzelelements verloren. Tritt Mixed Content auf, werden bei der Rückkonversion alle Text-Knoten zusammengefasst.

Druckbare ASCII-Zeichen sowie Leerzeichen werden korrekt umgewandelt, alle anderen Zeichen werden von *Json-lib* verworfen.

Mehrere aufeinanderfolgende Elemente selben Namens werden von *Json-lib* in ein JSON-Array konvertiert. Dabei wird jedoch lediglich der Inhalt der Elemente übernommen, während der Tag-Name verloren geht.

Enthält das Wurzelelement eines XML-Dokuments lediglich *Character Data*, so werden diese bei einem Round-Trip zum Inhalt eines Kindelements des Wurzelelements – in diesem Fall fügt *Json-lib* also eine Elementebene hinzu, die im Ursprungsdokument nicht existiert. Eine genauere Analyse der ausgegebene Daten zeigt, das dieses Verhalten auch der Grund für das Scheitern der CDATA-Testfälle ist. CDATA-Sektionen werden von *Json-lib* sehr wohl unterstützt.

Leider ist es nicht möglich, *Json-lib* ohne besondere Konfiguration in einem Prozess mit begrenztem virtuellen Adressraum zu verwenden, da die Java Virtual Machine (JVM) in diesem Fall nicht gestartet werden kann [JVM]. Um dieses Problem zu umgehen,

```
$ strace -p  3560
mmap(NULL, 4984832, PROT_READ|PROT_WRITE,
 ↪  MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff29521e000
munmap(0x7ff2956df000, 4984832)         = 0
mmap(NULL, 4984832, PROT_READ|PROT_WRITE,
 ↪  MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff2956df000
munmap(0x7ff29521e000, 4984832)         = 0
[...]
mmap(NULL, 4984832, PROT_READ|PROT_WRITE,
 ↪  MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff29521e000
munmap(0x7ff2956df000, 4984832)         = 0
mmap(NULL, 4984832, PROT_READ|PROT_WRITE,
 ↪  MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff2956df000
munmap(0x7ff29521e000, 4984832)         = 0
^Cstrace: Process 3560 detached
$ kill 3560
```

Abbildung 4.5: Eine Endlosschleife im *GreenCape XML Converter* musste mittels `SIGTERM` unterbrochen werden.

muss beim Start des Java-Prozesses die maximale Größe des Heap-Adressraums sowie die Größe des für Zeiger auf Metadaten von Java-Klassen zur Verfügung stehenden Adressraums angegeben werden.

Eine sinnvolle Aussage über die Anfälligkeit gegenüber Angriffen aus dem Bereich Denial of Service ist unter diesen Umständen jedoch nicht möglich, sodass die entsprechenden Tests für *Json-lib* manuell wiederholt werden mussten.

Dabei ergab sich eine Verwundbarkeit gegenüber *Billion-Laughs*-Angriffen sowohl mit General Entities als auch mit Parameter Entities. *Json-lib* ist ebenfalls anfällig für *Quadratic-Blowup*-Attacken. Zudem ist der Parser anfällig für XXE-Angriffe mittels General Entites bzw. Parameter Entities, die sowohl für File System Access als auch für Server Side Request Forgery genutzt werden können. Ebenso erlaubt *Jsonlib* die Einbettung lokaler Dateien mittels Parameter Entities in DTDs, sowohl in der ursprünglichen Version des Angriffs [Mor14, S. 10], als auch in einer modifizierten Variante, die 2016 von Sicherheitsforschern der Ruhr-Universität Bochum vorgestellt wurde. [Spä16, Abschn. 5.2].

Ein Blick in den Quellcode der Klasse `net.sf.json.XMLSerializer` zeigt, dass Json-lib den XML-Parser XOM einsetzt.

4.4 JsonML

Das Konversionverfahren JsonML ist vergleichsweise vollständig. Lediglich die in den Ursprungsdokumenten enthaltenen PIs werden vom Konverter ignoriert und nicht in die JSON-Ausgabe übernommen.

JsonML ist von allen getesten Konvertern als einziger in der Lage, alle Test-Dokumente verlustlos zu konvertieren, solange diese keine PIs enhielten.

4.5 Json.NET

Der *Json.NET*-Konverter nutzt die Klasse `System.Xml.XmlDocument` aus der C#-Standardbibliothek für die Verarbeitung von XML-Daten.

Probleme hat der Konverter mit der Beibehaltung der ursprünglichen Elementreihenfolge des Dokuments. Bei der Umwandlung von Mixed Content tritt ebenfalls Informationsverlust auf. Whitespace wird bei der Konversion verworfen.

Json.NET ist als einziger Konverter im Test in der Lage, PIs korrekt umzuwandeln – zumindest solange sie keinen Whitespace enthalten, da dieser verloren geht. Treten PIs außerhalb des Wurzelelements auf, führt dies bei der Rückkonvertierung der JSON-Daten zu einem Absturz des Programms.

Im Gegensatz zu allen anderen überprüften Konvertern wandelt *Json.NET* auch XML-Kommentare um. Der Konverter fügt diese als JavaScript-Blockkommentare, d.h. in der Form „`/* Comment */`“, in die JSON-Ausgabe ein. Da Kommentare jedoch nicht Teil der JSON-Syntax sind (vgl. Def. 1 in Abschn. 2.2.1), ist die von *Json.NET* in diesem Fall produzierte Ausgabe kein gültiges JSON [ECMA404; RFC7159] und führt bei den meisten Parsern zu Problemen [Ser16, Abschn. 2.1, 4.1].

Alle Zeichen, die nicht mittels 7-Bit-ASCII-Kodierung darstellbar sind (d. h. alle Zeichen, deren Unicode-Codepoint hinter U+00007F liegt) werden von *Json.NET* bereits bei der Konversion in JSON durch Fragezeichen ersetzt, deren Anzahl sich an der Anzahl der Bytes bemisst, die für die Kodierung des Zeichens in UTF-8 nötig sind.

Der Konverter ist anfällig für SSRF-Angriffe mittels XXE, sowohl mit General Entities als auch mit Parameter Entities. Ebenso ruft der Konverter externe DTDs über den im `SYSTEM`-Identifiktator angegebene URI ab.

4.6 JXON

JXON fehlt die Unterstützung von Mixed Content. Zudem geht die Reihenfolge der Elemente verloren. Processing Instructions werden bei der Konversion ignoriert.

Auch Whitespace geht bei der Konversion verloren – neben Tab, Zeilenvorschub, Wagenrücklauf und Leerzeichen umfasst das jedoch auch Unicode-Whitespace wie Ogham-Leerzeichen, mongolische Volkaltrennzeichen oder ideographische Leerzeichen. Eine komplette Liste ist in Anhang B.1.

Eine Verwundbarkeit für die getesteten Angriffe auf XML-Parser wurde bei der Überprüfung nicht gefunden.

4.7 org.json.XML

Da es sich bei `org.json.XML` um ein Java-Package handelt, ist die Untersuchung von DoS-Angriffen wegen der Beschränkungen der Java Virtual Machine ebenso problematisch wie bei *Json-lib* (vgl. Abschn. 4.3). Auch bei org.json.XML musste die Verwundbarkeit gegenüber solchen Angriffen daher manuell verifiziert werden. Der XML-Parser wird von dem Paket selbst implementiert und ist für keinen der überprüften Angriffe verwundbar.

Attribute gehen bei der Konvertierung von XML-Daten in JSON zwar nicht verloren, bei der Rückkonvertierung kann der Konverter jedoch nicht mehr erkennen, dass es sich um Attribute handelt und interpretiert diese stattdessen als Elemente.

XML-Namespace-Prefixes für Tag-Namen werden zwar grundsätzlich unterstützt, Namespaces können aber aufgrund der fehlerhaften Behandlung von Attributen dennoch

nicht genutzt werden. Mixed Content wird nicht unterstützt – enthält ein Element neben einem Kindelement auch Text, wird dieser verworfen.

Ebenfalls gehen bei der Konversion PIs und die Ordung der Dokumente verloren. Die Elementreihenfolge ist dabei anscheinend zufällig: Bei der Rückkonvertierung werden Elemente weder in alphabetischer noch in der Reihenfolge ausgegeben, in der sie in der JSON-Datei angeordnet sind.

Der Konverter unterstützt lediglich die Konversion von druckbaren ASCII-Zeichen. Unicode-Zeichen außerhalb dieses Bereichs werden bei der Umwandlung in das JSON-Format in eine der Byteanzahl ihrer UTF-8-Kodierung entsprechnde Menge von Fragezeichen umgewandelt.

Zudem werden Zahlenwerte sowie die Zeichenketten „`true`“ und „`false`“ in die nativen Java-Datentypen konvertiert, wobei Informationsverlust auftreten kann. Die Zeichenkette „`1e-324`“ wird beispielweise bei der Übersetzung zu JSON als Zahl interpretiert und erscheint daher in der Ausgabe gerundet als „`0`“.

4.8 Pesterfish

Bei der Konversion gehen die Namen der Namespace-Prefixe verloren und werden durch generische Bezeichnungen (`ns0`, `ns1`, ...) ersetzt. Diese werden auch dann verwendet, wenn im Ursprungsdokument keinerlei Namespace-Prefixes verwendet wurden, sondern ein eigener Default-Namespace genutzt wurde. Grund dafür ist die `ElementTree`-API, die Namespace-Prefixes beim Parsing von XML-Dokumenten automatisch zur vollen URI expandiert und den ursprünglichen Prefixnamen verwirft [ETree, Abschn. 20.5.1.7]. Zudem gehen bei der Konvertierung auch PIs verloren.

Dennoch ist es mit Pesterfish-Konverter möglich, komplette RSS-Dateien verlustlos in das JSON-Format und wieder zurück zu konvertieren.

Bei den Sicherheitstests zeigte sich, dass *Pesterfish* bei Nutzung der Vorgabeeinstellungen verwundbar für die DoS-Angriffe *Billion Laughs* und *Quadratic Blowup* ist. Da die Bibliothek jedoch die Verwendung einer eigenen `ElementTree`-Implementierung erlaubt, kann diesem Problem durch den Einsatz eines sicheren Ersatzes – beispielsweise aus der *defusedxml*-Bibliothek – entgegengewirkt werden.

4.9 x2js

Bei der Umwandlung mittels *x2js* geht die Reihenfolge der Elemente verloren. PIs werden komplett ignoriert.

Verfügt ein Element über mehrere Kindelemente mit dem gleichen Tag-Namen, tritt bei der Konversion ein Fehler auf, wodurch mehrere JSON-Arrays ineinander geschachtelt

werden. Bei der Rückkonversion enstehen dann zusätzliche Kindelemente, bei denen die Array-Indizes als Tag-Name verwendet werden.

Mixed Content – auch durch Einrückungen verursachter – führt zum Absturz von *x2js*. Sonderzeichen stellen für den Konverter jedoch kein Problem dar.

Bei komplexeren Dokumenten entstehen durch die Konversion mittels *x2js* extrem tief verschachtelte JSON-Strukturen mit mehr als 2 200 Ebenen (vgl. Abb. 5.2 in Abschn. 5.1). Dies kann bei Parsern zu Problemen und Abstürzen führen, was auch bei dem zur Reformatierung der JSON-Daten eingesetzten Parser der Fall war. Die entsprechenden Testfälle mussten daher manuell wiederholt werden.

4.10 x2js (Fork)

Die Beibehaltung der Dokumentreihenfolge ist beim Einsatz des *x2js*-Forks nicht gegeben.

Mixed Content führt im Gegensatz zum Ursprungsprojekt nicht zum Absturz, jedoch wird dabei der gesamte Character Content eines Elements zu einem einzelnen Textknoten zusammengefasst.

Whitespace bleibt zwar grundsätzlich erhalten, Einrückungen werden jedoch wie anderer Mixed Content auch zusammengefasst, sodass bei Dokumenten mit Einrückungen nach der Konversion jedes Element mit zuvor eingerücktem Inhalt stattdessen einen einzelnen, nur aus Whitespace bestehenden Textknoten enthält.

Auch PIs werden nicht unterstützt – treten diese innerhalb des Wurzelelements auf, wird anstelle der PI ein leeres Element namens „undefined“ eingefügt.

Der Bug, der bei *x2js* Probleme mit Elementen verursacht, die mehrere Kindelemente mit dem gleichen Tag-Namen enhalten, ist in der geforkten Version behoben. Die Kindelemente werden dabei jedoch nach Namen gruppiert, sodass die Reihenfolge beispielsweise bei alternierenden Tag-Namen verloren geht.

4.11 xmljson

Im Gegensatz zu anderen Konvertern verfügt das Python-Package *xmljson* über keine Möglichkeit, einen XML-Daten enthaltenden String direkt zu konvertieren, sondern akzeptiert lediglich bereits geparste `ElementTree`-Objekte. Da der Nutzer selbst für das Parsen des XML-Dokuments zuständig ist und es im Unterschied zu Pesterfish (Vgl. Abschn. 4.8) auch keine Voreinstellung gibt, wird eine Sicherheitsanalyse dieses Konverters hinfällig. Zum Durchführen der Tests wurde `defusedxml.lxml` verwendet.

Der *xmljson*-Konverter kann mehrere verschiedene Konvertierungskonventionen nutzen (vgl. Abschn. 3.2), die unabhängig voneinander getestet wurden.

Durch den Einsatz der `ElementTree`-API hat *xmljson* keinen Zugriff auf die im XML-Dokument verwendeten Prefixnamen, sodass diese wie beim *Pesterfish*-Konverter durch generische Namen ersetzt werden (vgl. Abschn. 4.8). PIs werden von allen Konventionen ignoriert.

Lediglich die *Cobra*- und *Yahoo*-Konventionen (Abschn. 4.11.3 und 4.11.6) nutzen keine Typinferenz. Alle anderen Konverter wandeln die Zeichenketten „`true`“ und „`false`“ in boolsche Werte um. Auch Zahlenwerte werden als numerische Datentypen interpretiert, wodurch beispielsweise Rundungsfehler auftreten können oder Formatierung verloren geht (z.B. `1e+39` anstatt `"1E39"`). Sehr große Zahlen werden als **`Infinity`**-Literal dargestellt, der zwar valides JavaScript wäre, von der JSON-Spezifikation jedoch nicht erlaubt ist.

Ähnlich wie auch *JXON* (vgl. 4.6) entfernt die *Badgerfish*- bzw. *GData*-Konvention bei der Konversion Unicode-Whitespace-Zeichen (vgl. Anhang B.1). Zudem werden bei den Konventionen *Badgerfish*, *GData* und *Parker* einige dezimale Zahlzeichen aus der BMP-Ebene des Unicode-Standards [Unicode9, S. 49] in ihr ASCII-Äquivalenta umgewandelt – aus einer bengalischen Ziffer Acht[1] wird so beispielsweise die lateinische „8“[2]. Die vollständige Liste der veränderten Zahlzeichen befindet sich in Anhang B.2.

4.11.1 Abdera

Attribute werden bei der JSON-Konvertierung mithilfe der *Abdera*-Konvention zwar in die Ausgabe übernommen, allerdings ist *xmljson* bei der Rückkonvertierung nicht in der Lage, diese von normalen Elementen zu unterscheiden. Daher befinden sich Attribute nach der Rückkonvertierung nicht mehr an der ursprünglichen Stelle im Dokument, sondern in einem „`<attributes>`“-Element, das als Kindelement des Ursprungselements eingefügt wird. Ein zusätzliches „`<children>`“-Element enthält die ursprünglichen Kindelemente des Elements.

Außerdem werden teilweise Elemente und Textknoten zu Attributwerten umgedeutet. So wird aus „`<a><b>hello</b></a>`“ durch einen Round-Trip „`<a b="hello" />`“

Da der Konverter nicht in der Lage ist, anhand der JSON-Daten verlässlich zwischen Elementen und Attributen zu unterscheiden und diese dadurch später nicht mehr korrekt rekonstruieren kann, schlagen auch Tests fehl, die der Konverter eigentlich bestehen könnte. So nutzt der Konverter bei mehreren Kindelementen JSON-Arrays, d.h. die Elementreihenfolge ist auch nach der Konversion noch ersichtlich. Auch Tag-Name und Attribute des Wurzelelements gehen eigentlich nicht verloren.

[1] Unicode-Codepoint `U+09EE`: `BENGALI DIGIT EIGHT`
[2] Unicode-Codepoint `U+0038`: `DIGIT EIGHT`

Mixed Content wird nicht unterstützt, es ist lediglich der erste Textknoten im Dokument auffindbar. Führender oder anhängender Whitespace wird bei der Konversion verworfen.

4.11.2 Badgerfish

Die Ergebnisse stimmen im Wesentlichen mit denen des *Cobra-vs-Mongoose*-Konverters (vgl. Abschn. 4.1) überein, der ebenfalls auf die sogenannte Badgerfish-Konvention zur Darstellung von XML-Strukturen in JSON setzt. Nicht unterstützte Features wie Mixed Content, PIs oder Kommentare führten bei *xmljson* jedoch nicht zu einem Absturz des Programms. Zudem kommt es zu Problemen durch den Einsatz von Typinferenz und den Verlust der Prefixnamen von Namespaces.

4.11.3 Cobra

Bei *Cobra* handelt es sich um eine modifizierte Variante der *Abdera*-Konvention, die sich hauptsächlich in der Frage, welche Schlüssel in der JSON-Objekt-Repräsentation eines XML-Dokuments optional sind, von *Abdera* unterscheidet. Daher hat auch *Cobra* große Probleme mit der Rückkonvertierung zu XML und der Unterscheidung zwischen Elementen und Attributen.

Ein weiterer Unterschied ist, dass bei Cobra keine Datentypen erraten werden, sondern alles als String behandelt wird.

4.11.4 GData

Durch die Nutzung von ungeordneten JSON-Objekten geht bei dem Einsatz der *GData*-Konvention die Reihenfolge der im XML-Dokument enthaltenen Elemente verloren. Bei Mixed Content wird lediglich der erste Textknoten übernommen, alle weiteren werden verworfen.

Bei der Konversion kann es zu Informationsverlust durch Typinferenz kommen.

4.11.5 Parker

In der Standardeinstellung verwirft diese Konvention das Wurzelelement – dieses Verhalten ist jedoch über einen Parameter abschaltbar.

Alle Attribute gehen bei der Konversion verloren, ebenso wie PIs. Als einzige Konvention des *xmljson*-Konverters wurde bei ausschließlich Text enhaltenden Elementen führender oder nachfolgender Whitespace nicht verworfen – Mixed Content wird jedoch trotzdem nicht unterstützt.

4.11.6 Yahoo

Die *Yahoo* hat ebenfalls Probleme bei der Unterscheidung zwischen Elementen und Attributen, fügt aber im Gegensatz zu *Abdera* und *Cobra* nicht neue, im Originaldokument nicht existierende Elemente in das Dokument ein.

Die Elementreihenfolge geht bei der Konvertierung verloren.

Im XML-Dokument enthaltener Character Content wird in allen getesteten Fällen zu JSON-Strings konvertiert, eine Typumwandlung findet nicht statt.

5 Weiterentwicklung eines Konversionsverfahrens

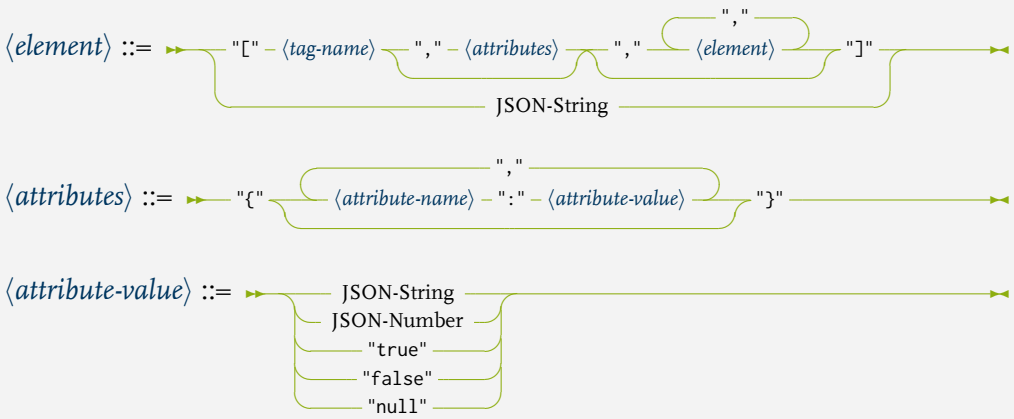
Mit insgesamt 112 von 123 bestandenen Testfällen erfüllt die JSON Markup Language (JsonML) von allen Konversionverfahren die meisten Kriterien im Test.

In diesem Kapitel wird daher zunächst ein Überblick über die von JsonML eingesetzte Syntax gegeben. Darauf aufbauend werden die notwendigen Modifikationen am JsonML-Verfahren beschrieben, die im Zuge dieser Arbeit entwickelt wurden, um das Ziel eines vollständig verlustlosen Konversionsverfahrens zu erreichen.

5.1 Syntax

Im Gegensatz zu anderen Konvertern nutzt JsonML ungeordnete JSON-Objekte ausschließlich für Attributlisten [JsonML]. Die Baumstruktur eines XML-Dokuments wird mittels JSON-Arrays dargestellt, wobei ein Array immer genau ein Element repräsentiert. Textknoten bzw. CDATA-Sektionen werden zu einfachen JSON-Strings umgewandelt.

Definition 3 (Formale Syntax der JSON Markup Language (JsonML)). Sowohl ⟨*tag-name*⟩ als auch ⟨*attribute-name*⟩ sind JSON-Werte vom Typ String. Die Whitespace-Regeln sind identisch mit denen von JSON (vgl. Def. 1).



Beispiel 15 (JsonML-Dokument). In der JsonML-Repräsentation des XML-Dokuments aus Beispiel 1 fehlt lediglich die Processing Instruction.

```
["albums", "\n  ",
    ["album", {"catno": "ARGO LP-628"} ,"\n    ",
        ["artist",  "Ahmad Jamal Trio"], "\n    ",
        ["title", "At The Pershing"], "\n    ",
        ["recording", "Recorded ",
            ["date", "January 16, 1958"], "."
        ], "\n  "
    ], "\n"
]
```

JsonML sieht keine gesonderte Verarbeitung von Namespace-Deklarationen oder mit Namespace-Prefixes versehenen Tag-Namen vor, sondern behandelt diese wie normale Attribute bzw. wie einen Teil des Tag-Namens.

JsonML stellt die XML-Inhalte recht effizient dar: Die JSON-Repräsentation eines umfangreichen Office-Dokuments im FODT-Format benötigt rund 6,6 Prozent weniger Speicherplatz als die äquivalente Darstellung durch kanonisches XML.

Im Vergleich zu anderen Konvertern ist der Overhead bei JsonML deutlich geringer. So waren die vom Pesterfish-Konverter ausgegebenen Daten trotz Informationsverlust auch nach Entfernung von optionalem Whitespace und unnötiger Quotierung von Unicode-Zeichen mehr als dreieinhalb Mal so groß wie bei JsonML (vgl. Abb. 5.1).

Tabelle 5.1: Größenvergleich von JsonML ggü. XML und Pesterfish anhand der Spezifikation des OpenDocument Format im FODT-Format.

	Größe (in bytes)	Verhältnis zu XML (in %)	
		Größe	Veränderung
Kanonisches XML	5787196	100,0	0
JsonML[a]	5405329	93,4	−6,6
Pesterfish[a]	15061634	260,3	+160,3
Pesterfish[b]	14480612	250,2	+150,2

[a] JSON unverändert
[b] Optionaler JSON-Whitespace entfernt

Neben der Dateigröße ist auch die Verschachtelungstiefe der JSON-Container ein guter Indikator für Overhead. Meist wird für die Verarbeitung der verschiedenen Ebenen

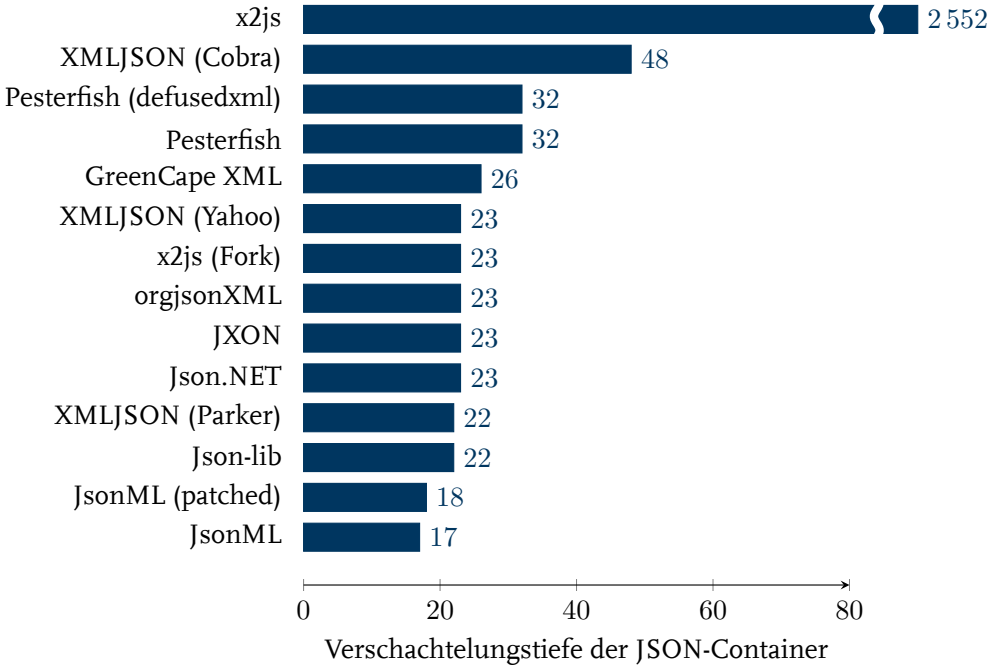


Abbildung 5.2: Verschachtelungstiefe der aus einem Word-XML-Dokument erzeugten JSON-Dateien (weniger ist besser).

von JSON-Objekten und -Arrays ein Stapelspeicher (Stack) eingesetzt, der bei jeder zusätzlichen Verschachtelungsebene anwächst. Damit der Speicherverbrauch des Stacks im Rahmen bleibt, erlaubt es die IETF daher die Verschachtelungstiefe von JSON zu begrenzen [RFC7159, Abschn. 9]. So hat die `json_decode()`-Funktion von PHP standardmäßig eine Obergrenze von 512 Ebenen [PHP] und Ruby limitiert die maximale Tiefe sogar auf nur 100 Ebenen [Ruby].

Bei einem Vergleich der Verschachtelungstiefe anhand eines Microsoft-Office-Dokuments kann JsonML ebenfalls gut abschneiden (vgl. Abb. 5.2). Mit 17 Ebenen ist die Komplexität von JsonML am geringsten und unterschreitet damit den Medianwert um rund 26 Prozent.

5.2 Unterstützung von Processing Instructions

Probleme hat der Konverter jedoch mit der Umwandlung von Processing Instructions (PIs). Diese werden bei der Umwandlung in JSON vollständig ignoriert. Stephen McKamey, der Entwickler von JsonML, begründet dies damit, das es keine sinnvolle Entsprechung von PIs in JSON gäbe [McK06].

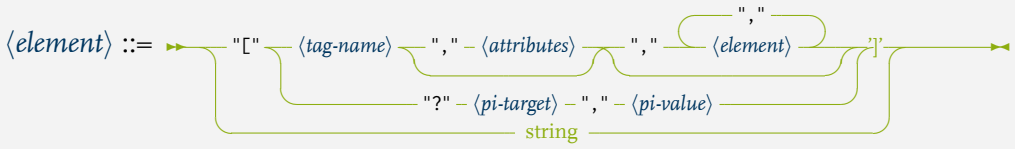
Zwar bietet JSON tatsächlich keinen vergleichbaren Mechanismus, eine Unterstützung von PIs kann für bestimmte Einsatzzwecke aber sinnvoll sein, beispielsweise wenn sonst die Verknüpfung mit XML-Stylesheets oder Formatierungsinformationen in DocBook-Dateien verloren gehen könnte. Im Rahmen der vorliegenden Arbeit wurde die JsonML-Syntax daher um Unterstützung von Processing Instructions ergänzt.

PIs bestehen aus einem *Ziel* und *Daten* (Vgl. Abschn. 2.1.1), bilden also das 2-Tupel $P := \langle target, data \rangle$. Der Datenteil kann dabei auch leer sein.

Das Ziel muss ein gültiger Name im Sinne der XML-Spezifikation sein [XML, Regel [17]]. Das heißt, dass der Name einer PI ebenso wie auch der Tag-Name von Elementen [XML, Regel [40]] mit einem sog. `NameStartChar` beginnen muss. Dadurch wird ausgeschlossen, dass Tag-Namen mit bestimmten Zeichen beginnen – darunter auch das Fragezeichen, da dies dazu führen würde, dass Start-Tags mit PIs verwechselt werden könnten. Insbesondere in SGML – zu dem XML vollständig kompatibel sein soll – wären solche Tags nicht mehr von PIs zu unterscheiden, da laut SGML-Spezifikation lediglich ein einfaches Größerzeichen anstatt der Kombination aus Fragezeichen und Größerzeichen („?>“) zum Schließen der PI ausreicht.

Dadurch wird es möglich, PIs in JsonML eindeutig in Form eines JSON-Arrays „`["?target", "data"]`“ darzustellen (vgl. Definition 4), das dem 2-Tupel P (s.o.) entspricht. Die Repräsentation von PIs ähnelt damit der eines Elementknotens, der einen einzelnen Textknoten (*Character Data*) enthält. Eine Verwechslung ist jedoch durch das dem Zielnamen vorangestellte Fragezeichen ausgeschlossen – ein Tag-Name darf nicht mit einem Fragezeichen beginnen, wodurch die Kategorisierung als PI eindeutig ist.

Definition 4 (Formale Syntax der JSON Markup Language (JsonML) mit *PIs*)**.** Die um Unterstützung von *PIs* erweiterte Syntax ist mit Ausnahme der Produktionsregeln für ⟨*element*⟩ identisch mit der Syntax aus Definition 3. ⟨*tag-name*⟩, ⟨*pi-target*⟩ und ⟨*pi-data*⟩ sind JSON-Werte vom Typ String.



Enthält das Dokument *PIs* auf Dokument-Ebene (d.h. als Top-Level-Konstrukt), dann ist das JsonML-Wurzelelement ein ⟨*element*⟩ mit einem leeren String als ⟨*tag-name*⟩, das die Kindknoten des Dokuments (d.h. *PIs* auf Dokumentebene und das Wurzelelement des Dokuments) als Unterelemente enthält.

5.3 Überprüfung der Änderungen

Die syntaktischen Änderungen aus Definition 4 wurden in die JavaScript-Referenzimplementierung von Stephen McKamey eingearbeitet. Entsprechende *Unittests* zur Sicherstellung der korrekten Umwandlung von PIs wurden ebenfalls hinzugefügt.

Beispiel 16 (JsonML-Dokument mit PIs)**.** Die JsonML-Repräsentation des XML-Dokuments aus Beispiel 2 kann nun die PI darstellen – auch solche, die sich außerhalb des Wurzelelements befinden.

```
["", "\n",
    [ "?xml-stylesheet", "href=\"style.css\"" ],"\n","\n",
    ["albums", "\n  ",
        ["album", {"catno": "ARGO LP-628"}, "\n    ",
            ["artist", "Ahmad Jamal Trio"], "\n    ",
            ["title", "At The Pershing"], "\n    ",
            ["recording", "Recorded ",
                ["date", "January 16, 1958"], "."
            ], "\n  "
        ], "\n"
    ]
]
```

Bei einer erneuten Überprüfung des JsonML-Konverters unter Berücksichtigung der o. g. Änderungen wurde deren Korrektheit bestätigt: Alle Testdokumente, auch die zuvor fehlgeschlagenen, lassen sich nun verlustlos von XML nach JSON und wieder zurück konvertieren (vgl. Tab. 4.1 bis 4.4).

Alle Änderungen wurden dem JsonML-Projekt zur Verfügung[1] gestellt (siehe Anhang C.2).

[1] Vgl. `https://github.com/mckamey/jsonml/pull/14`

6 Fazit und Ausblick

Ziel der vorliegenden Arbeit war das Finden eines sicheren und verlustlosen Verfahrens zur Konversion von beliebigen XML-Dokumenten in JSON-Datenstrukturen. Dazu wurden die Anforderungen „Sicherheit“ und „Verlustlosigkeit“ zunächst näher bestimmt und ein Kriterienkatalog für ein sicheres und verlustloses Konversionsverfahren erstellt.

Auf dieser Basis wurden dann eine Reihe von Konversionsprogrammen analysiert. Dazu wurde ein Test-Framework implementiert, das die Durchführung der Überprüfung weitestgehend automatisiert.

Es zeigte sich, das keines der überprüften Konversionsverfahren in der Lage ist, XML-Strukturen verlustlos abzubilden. Teile der XML-Spezifikation werden von vielen Konversionsverfahren nicht oder nur ungenügend unterstützt. Darunter sind eher selten genutzte Features wie PIs, aber auch grundlegende XML-Eigenschaften wie die Beibehaltung der Elementreihenfolge oder die Möglichkeit zur Nutzung von Mixed Content.

Die JSON Markup Language (JsonML) erfüllte bei der Analyse die meisten der zuvor aufgestellten Kritieren. Probleme bestanden jedoch bei der Unterstützung von Processing Instructions. Durch eine Weiterentwicklung des Konversionsverfahrens konnte dieser Mangel behoben werden, sodass das nun alle Anforderungen vollumfänglich erfüllt werden. Auch die Umwandlung von komplexen XML-basierten Formaten wie OOXML, Flat ODF oder SVG in JSON ist so verlustlos und sicher möglich.

Das hier gezeigte Prüfverfahren vergleicht die XML-Dokumente vor und nach einem Konversions-Round-Trip. Viele der Konversionsverfahren sind jedoch für eine Analyse dieser Art schon allein deshalb nicht geeignet, da ihnen die Möglichkeit der Rückkonvertierung zu XML fehlt. Ohne eine solche Funktionalität ist eine Prüfung der Verlustlosigkeit in der hier beschriebenen Art jedoch nicht möglich. Projekte wie die PHP-Bibliothek *xml2jsonphp* [Nat07] oder das JavaScript-Modul *xmlToJSON* [Sum16] konnten daher nicht berücksichtigt werden.

Ansatzpunkt weiterer Forschung könnte neben der Evaluierung weiterer Konversionverfahren und -programmen auch die Einbeziehung zusätzlicher Metadaten wie etwa DTDs oder Schema-Informationen sein. Dies könnte Verbesserungen bei der Verwendung von nativen JSON-Datentypen bringen.

Auch die Überprüfung von Verfahren, die beliebige JSON-Daten in ein XML-basiertes Format übersetzen, könnte Gegenstand weiterer Betrachtungen sein, inbesondere im Hinblick auf die Sicherheit gegenüber Angriffen auf XML-Parser.

Das Aufkommen von Technologien wie JSON Schema, JSON Reference oder JSON Include könnte die Komplexität von JSON-Parsern in Zukunft deutlich ansteigen lassen und diese für Angriffe verwundbar machen, für die bislang typischerweise eher XML-Parser anfällig sind. Eine Evaluierung der Verwundbarkeit von Konvertern gegenüber solchen Angriffen erscheint daher sinnvoll.

Abkürzungsverzeichnis

API Application Programming Interface 1–3, 33, 35, 44, 48, 50, 65

ASCII American Standard Code for Information Interchange 44, 45, 47, 48

ASF Atom Syndication Format 2

C14N XML-Kanonisierung (XML Canonicalization) vii, 13–15, 28, 29, 37, 39

CPU Central Processing Unit 18, 38, 44

DOM Document Object Model 8, 10

DoS Denial of Service vii, 18, 25, 37, 44, 46–48

DSD Document Structure Description 11

DTD Document Type Definition vii, 10, 11, 14, 16, 17, 25, 26, 30, 47, 59

FODT Flat OpenDocument Text 54, 66

FSA File System Access vii, 16–19, 38, 46

HTML HyperText Markup Language 8, 62

HTTP HyperText Transfer Protocol 19, 38

I-D Internet Draft 3, 22

IEEE Institute of Electrical and Electronics Engineers 32

IETF Internet Engineering Task Force 20, 22, 32, 55

IoT Internet of Things 2

ISO International Organization for Standardization 7

JSON JavaScript Object Notation iii, vii, viii, 1–5, 19–23, 26, 28–37, 39, 43–57, 59–61, 65, 67

JsonML JSON Markup Language viii, ix, 34, 35, 44, 46, 53–57, 59, 66, 67, 75, 83

JVM Java Virtual Machine 45, 47

JXML JsonXML 3, 4

JXON Lossless JavaScript XML Object Notation 35

MathML Mathematical Markup Language 2

ODF OpenDocument Format 2, 7, 44, 54, 59, 66

OOXML Office Open XML 2, 7, 59

OWASP Open Web Application Security Project 25

PI Processing Instruction vii–ix, 10, 28, 29, 43–51, 54–57, 59, 67, 83

RELAX NG Regular Language Description for XML New Generation 11

ReST Representational State Transfer 3

RSS Rich Site Summary 2, 7, 48

SAML Security Assertion Markup Language 2

SGML Standard Generalized Markup Language 7, 10, 28, 30, 56

SSRF Server Side Request Forgery vii, 18, 19, 38, 46, 47

SVG Scalable Vector Graphics 2, 7, 59

UCS Universal Character Set 62

URI Uniform Resource Identifier 12, 19, 47

URL Uniform Resource Locator 11, 17, 19

UTF-8 8-Bit UCS Transformation Format 47, 48

W3C World Wide Web Consortium 7, 11, 13, 15, 29

XDM XQuery and XPath Data Model 10

XHTML Extensible HyperText Markup Language 2, 7

XInclude XML Inclusions 17, 19, 67

XML Extensible Markup Language iii, vii, viii, 1–5, 7–20, 22, 25–31, 33–37, 39, 42–57, 59–63, 65–67

XML-RPC XML Remote Procedure Call 2

XPath XML Path Language 4, 27, 29, 62

XQuery XML Query Language 62

XSD XML Schema Definition vii, 3, 4, 11, 19, 30

XSL Extensible Stylesheet Language 63

XSLT XSL Transformation 3, 17, 29

XXE XML External Entity 16, 17, 19, 25, 46, 47, 67

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Beispiele

Literatur

[Abd17] Abdulla G. Abdurakhmanov. *x2js - XML to JSON and back for JavaScript*. GitHub.com. 21. Juni 2017. URL: https://github.com/abdmob/x2js (besucht am 21. 06. 2017).

[Bat06] Paul Battley. *Cobra vs Mongoose*. Zugriff via Archive.org Wayback Machine: https://web.archive.org/web/20161220162327/http://cobravsmongoose.rubyforge.org/. RubyForge.org. 29. Juni 2006. URL: http://cobravsmongoose.rubyforge.org/ (besucht am 20. 12. 2016).

[Bra14] Tim Bray. *JSON Redux AKA RFC7159*. Aktualisiert am 6. März 2014. tbray.org. 5. März 2014. URL: https://www.tbray.org/ongoing/When/201x/2014/03/05/RFC7159-JSON (besucht am 25. 06. 2017).

[Bra98] Tim Bray. *The Annotated XML Specification*. Basiert auf der XML 1.0 W3C Empfehlung vom 10. Februar 1998. XML.com. 1998. URL: https://www.xml.com/axml/axml.html (besucht am 07. 06. 2017).

[C14N] John Boyer. „Canonical XML Version 1.0“. W3C Recommendation. W3C, 15. März 2001. URL: http://www.w3.org/TR/2001/REC-xml-c14n-20010315 (besucht am 07. 05. 2017).

[Chr10] Erik B. Christensen et al. „Complete mapping between the XML infoset and dynamic language data expressions“. US Patent 7,761,484 B2. Microsoft Corporation. 20. Juli 2010. URL: https://www.google.com/patents/US7761484 (besucht am 22. 06. 2017).

[Cro06] Douglas Crockford. „JSON: The fat-free alternative to XML“. In: *Proc. of XML*. (Boston, MA). 2006. URL: http://www.json.org/fatfree.html (besucht am 08. 06. 2017).

[Cro09] Douglas Crockford. *The JSON Saga*. Vortrag auf der Øredev Conference 2009 in Malmö, Schweden. Vimeo.com. 4. Nov. 2009. URL: https://vimeo.com/8692019 (besucht am 25. 06. 2017).

[DBXSL] Norman Walsh. *DocBook XSL Stylesheets. Reference Documentation*. The DocBook Project. 28. Aug. 2007. 444 S. URL: http://docbook.sourceforge.net/release/xsl/current/doc/ (besucht am 06. 06. 2017).

[DOM] Anne van Kesteren, Aryeh Gregor, Ms2ger, Alex Russell und Robin Berjon. „W3C DOM4“. W3C Recommendation. W3C, 19. Nov. 2015. URL: https://www.w3.org/TR/2015/REC-dom-20151119/ (besucht am 08. 06. 2017).

[DTDAttack] Christopher Späth. *DTD-Attacks*. Version 6f703e9. GitHub-Repository. Lehrstuhl für Netz- und Datensicherheit. 21. Juli 2016. URL: https://github.com/RUB-NDS/DTD-Attacks (besucht am 15. 06. 2017).

[DuV13] Adam DuVander. *JSON's Eight Year Convergence With XML*. ProgrammableWeb News. 26. Dez. 2013. URL: https://www.programmableweb.com/news/jsons-eight-year-convergence-xml/2013/12/26 (besucht am 06. 03. 2017).

[ECMA262] „ECMAScript® 2016 Language Specification“. 7th Edition. ECMA 262. Geneva, Switzerland: Ecma International, Juni 2016. 586 S. URL: https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf (besucht am 03. 06. 2017).

[ECMA404] „The JSON Data Interchange Format“. 1st Edition. ECMA 404. Geneva, Switzerland: Ecma International, Okt. 2013. 14 S. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (besucht am 08. 05. 2017).

[ETree] *The ElementTree XML API*. xml.etree.ElementTree. Python 3.5.3 Documentation. Python Software Foundation. 16. Mai 2017. URL: https://docs.python.org/3.5/library/xml.etree.elementtree.html (besucht am 15. 06. 2017).

[ExcC14N] John Boyer, Donald E. Eastlake III. und Joseph Reagle. „Exclusive XML Canonicalization Version 1.0“. W3C Recommendation. W3C, 18. Juli 2002. URL: http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718 (besucht am 07. 05. 2017).

[Fou17a] .NET Foundation. *Json.NET*. NuGet Gallery. 21. Juni 2017. URL: https://www.nuget.org/packages/Newtonsoft.Json (besucht am 21. 06. 2017).

[Fou17b] .NET Foundation. *NuGet Gallery Statistics*. NuGet Gallery. 21. Juni 2017. URL: https://www.nuget.org/stats (besucht am 21. 06. 2017).

[Goe06] Stefan Goessner. *Converting Between XML and JSON*. ©1998-2008 by O'Reilly Media, Inc. XML.com Archive. 31. Mai 2006. URL: http://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html (besucht am 17. 06. 2017).

[Gup07] Arun Gupta. *Language-neutral data format: XML and JSON*. Oracle Blogs. 1. März 2007. URL: https://blogs.oracle.com/arungupta/language-neutral-data-format:-xml-and-json (besucht am 22. 05. 2017).

[IEEE754] *IEEE Standard for Floating-Point Arithmetic*. Version 754-2008. IEEE Computer Society. New York, NY, 29. Aug. 2008. 70 S. ISBN: 978-0-7381-5752-8. URL: http://ieeexplore.ieee.org/servlet/opac?punumber=4610933 (besucht am 23. 06. 2017).

[JsonML] Stephen M. McKamey. „JsonML-Syntax“. Techn. Ber. URL: http://www.jsonml.org/syntax/ (besucht am 15. 06. 2017).

[JSONRef] Paul Bryan und Kris Zyp. „JSON Reference“. *draft-pbryan-zyp-json-ref-03*. Internet-Draft. IETF Secretariat, 16. Sep. 2012. URL: https://tools.ietf.org/html/draft-pbryan-zyp-json-ref-03 (besucht am 09. 06. 2017).

[JSONSchema] Austin Wright und Henry Andrews. „JSON Schema: A Media Type for Describing JSON Documents“. *draft-wright-json-schema-01*. Internet-Draft. IETF Secretariat, 15. Apr. 2017. URL: https://tools.ietf.org/html/draft-wright-json-schema-01 (besucht am 09. 06. 2017).

[JSONx] Brien Muschett, Rich Salz und Michael Schenker. „JSONx, an XML Encoding for JSON“. *draft-rsalz-jsonx-00*. Internet-Draft. IETF Secretariat, 3. Nov. 2011. URL: https://tools.ietf.org/html/draft-rsalz-jsonx-00 (besucht am 17. 06. 2017).

[JVM] *After "ulimit -v", the JVM can not start without extra GC command line args*. JDK-8043516. Oracle Bug Database. 20. Mai 2014. URL: http://bugs.java.com/view_bug.do?bug_id=8043516 (besucht am 15. 06. 2017).

[JXML] David Lee. *JSON XML Schema (JXML)*. 4. Feb. 2011. URL: http://xml.calldei.com/JsonXML (besucht am 17. 06. 2017).

[Lal13] Nazim Lala. *Safely handling untrusted XML server-side*. IIS.NET Community Blog. Microsoft. 13. März 2013. URL: https://blogs.iis.net/nazim/safely-handling-untrusted-xml-server-side (besucht am 20. 06. 2017).

[Lee11] David Lee. „JXON: an architecture for schema and annotation driven JSON/XML bidirectional transformations“. In: *Proceedings of Balisage: The Markup Conference*. 2011. URL: http://www.calldei.com/pubs/Balisage2011/JXON.pdf (besucht am 21. 03. 2017).

[McG02] Sean McGrath. *Mixed content myopia*. Zugriff via Archive.org Wayback Machine: http://web.archive.org/web/20061017170707/http://open.itworld.com/nl/xml_prac/07112002/pf_index.html. ITWorld „XML in Practice“ Newsletter. 11. Juli 2002. URL: http://open.itworld.com/nl/xml_prac/07112002/pf_index.html (besucht am 17. 10. 2006).

[McK06] Stephen M. McKamey. *Converting XML to JsonML*. 16. Nov. 2006. URL: http://www.jsonml.org/xml/ (besucht am 13. 06. 2017).

[Mic17] Microsoft Corporation. *Inferring Schemas from XML Documents*. .NET Framework 4.7, 4.6, and 4.5 Documentation. Microsoft Developer Network (MSDN). 2017. URL: https://msdn.microsoft.com/en-us/library/xz2797k1(v=vs.110).aspx (besucht am 04. 06. 2017).

[Mor14] Timothy D. Morgan und Omar Al Ibrahim. „XML Schema, DTD, and Entity Attacks: A Compendium of Known Techniques“. Techn. Ber. Virtual Security Research LLC, 19. Mai 2014. URL: https://www.vsecurity.com//download/publications/XMLDTDEntityAttacks.pdf (besucht am 21. 03. 2017).

[Mvn1] *Artifacts using Json Lib. net.sf.json-lib*. MvnRepository. 25. Juni 2017. URL: https://mvnrepository.com/artifact/net.sf.json-lib/json-lib/usages (besucht am 25. 06. 2017).

[Mvn2] *Artifacts using JSON In Java. org.json*. MvnRepository. 25. Juni 2017. URL: https://mvnrepository.com/artifact/org.json/json/usages (besucht am 25. 06. 2017).

[Nat07] Senthil Nathan, Edward J. Pring und John Morar. *Convert XML to JSON in PHP. Use server-side code to streamline Ajax*. Erstveröffentlicht am 16. Jan. 2007. IBM developerWorks®. 5. Juni 2007. URL: https://www.ibm.com/developerworks/library/x-xml2jsonphp/index.html (besucht am 13. 06. 2017).

[Nur09] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds und Clemente Izurieta. „Comparison of JSON and XML Data Interchange Formats: A Case Study.“ In: *Caine* 2009 (2009), S. 157–162. URL: https://pdfs.semanticscholar.org/8432/1e662b24363e032d680901627aa1bfd6088f.pdf (besucht am 21. 03. 2017).

[Op311] *XML to JSON Conventions*. Open311.org. 25. Okt. 2014. URL: http://wiki.open311.org/JSON_and_XML_Conversion/ (besucht am 22. 06. 2017).

[Pag05] Sam Page. *White Space in XML Documents*. UsingXML.com. 2005. URL: http://usingxml.com/Basics/XmlSpace (besucht am 25. 06. 2017).

[PHP] Mehdi Achour et al. *PHP Manual*. json_decode. Hrsg. von Peter Cowburn. Version 7.1.4. PHP Documentation Group. 23. Juni 2017. URL: http://php.net/manual/en/function.json-decode.php (besucht am 24. 06. 2017).

[PyPI17] *xmljson. PyPI Statistics*. PyPI-Stats.com. 21. Juni 2017. URL: http://www.pypi-stats.com/package/?q=xmljson (besucht am 21. 06. 2017).

[RFC7159] Tim Bray. „The JavaScript Object Notation (JSON) Data Interchange Format“. RFC 7159. RFC Editor, März 2014. URL: http://www.rfc-editor.org/rfc/rfc7159.txt (besucht am 21. 03. 2017).

[Ruby] *Ruby Documentation. module JSON*. Version 2.4.0. ruby-lang.org. 7. Nov. 2016. URL: https://docs.ruby-lang.org/en/2.4.0/JSON.html (besucht am 24. 06. 2017).

[Rud53] Richard Rudner. „The Scientist Qua Scientist Makes Value Judgments“. In: *Philosophy of Science* 20.1 (Jan. 1953), S. 1–6. ISSN: 00318248, 1539767X. URL: http://www.jstor.org/stable/185617 (besucht am 25. 06. 2017).

[Ser16] Nicholas Seriot. *Parsing JSON is a Minefield*. seriot.ch. 26. Okt. 2016. URL: https://www.tbray.org/ongoing/When/201x/2014/03/05/RFC7159-JSON (besucht am 25. 06. 2017).

[Sid02a] Bilal Siddiqui. *XML Canonicalization*. ©1998-2008 by O'Reilly Media, Inc. XML.com Archive. 18. Sep. 2002. URL: https://www.xml.com/pub/a/ws/2002/09/18/c14n.html (besucht am 05. 05. 2017).

[Sid02b] Bilal Siddiqui. *XML Canonicalization, Part 2*. ©1998-2008 by O'Reilly Media, Inc. XML.com Archive. 9. Okt. 2002. URL: https://www.xml.com/pub/a/ws/2002/10/09/canonicalization.html (besucht am 07. 05. 2017).

[Spä15] Christopher Späth. „Security Implications of DTD Attacks Against a Wide Range of XML Parsers“. Master Thesis. Lehrstuhl für Netz- und Datensicherheit, Ruhr-Universität Bochum, 20. Okt. 2015. URL: http://nds.rub.de/media/nds/arbeiten/2015/11/04/spaeth-dtd_attacks.pdf (besucht am 22. 04. 2017).

[Spä16] Christopher Späth, Christian Mainka, Vladislav Mladenov und Jörg Schwenk. „SoK: XML parser vulnerabilities“. In: *10th USENIX Workshop on Offensive Technologies, WOOT 16, Austin, TX, August 8-9, 2016*. 2016. URL: http://www.nds.ruhr-uni-bochum.de/media/nds/veroeffentlichungen/2016/07/22/woot16-sok_xml_parser_vulnerabilities.pdf (besucht am 21. 03. 2017).

[Ste02] Gregory Steuck. *XXE (Xml eXternal Entity) Attack*. SecurityFocus Bugtraq Mailing list. 29. Okt. 2002. URL: http://www.securityfocus.com/archive/1/297714 (besucht am 03. 03. 2017).

[Sum16] William Summers. *xmlToJSON*. A simple javascript module for converting XML into JSON within the browser. GitHub.com. 19. Feb. 2016. URL: https://github.com/metatribal/xmlToJSON (besucht am 27. 06. 2017).

[Tve08] Jesper Tverskov. *Understanding Processing Instructions in XML*. Aktualisiert am 2. August 2011. XMLplease.com. 12. Apr. 2008. URL: http://www.xmlplease.com/xml/pi/ (besucht am 06. 06. 2017).

[Unicode9] The Unicode Consortium. *The Unicode Standard. Core Specification*. Version 9.0. Unicode Consortium. Mountain View, CA, 21. Juni 2016. 1036 S. ISBN: 978-1-936213-13-9. URL: http://www.unicode.org/versions/Unicode9.0.0 (besucht am 05. 05. 2017).

[Wan11] Guanhua Wang. „Improving Data Transmission in Web Applications via the Translation between XML and JSON“. In: *2011 Third International Conference on Communications and Mobile Computing*. Apr. 2011, S. 182–185. DOI: 10.1109/CMC.2011.25. URL: http://ieeexplore.ieee.org/abstract/document/5931189/ (besucht am 21. 03. 2017).

[Wic17] Dave Wichers, Xiaoran Wang, James Jardine, Tony Hsu und Dean Fleming. *XML External Entity (XXE) Prevention Cheat Sheet*. Open Web Application Security Project (OWASP). 19. Juni 2017. URL: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet (besucht am 20. 06. 2017).

[Wil14] Victor L. Williamson. „Converting XML to JSON with configurable output“. US Patent Application Publication 2014/0244692 A1. Cellco Partnership D/B/A Verizon Wireless. Appl. No. 13/779,052. 24. Aug. 2014. URL: https://www.google.de/patents/US20140244692 (besucht am 22. 06. 2017).

[XDM] Norman Walsh, John Snelson und Andrew Coleman. „XQuery and XPath Data Model 3.1“. W3C Recommendation. W3C, 21. März 2017. URL: https://www.w3.org/TR/2017/REC-xpath-datamodel-31-20170321 (besucht am 08. 06. 2017).

[XInclude] Jonathan Marsh, David Orchard und Daniel Veillard. „XML Inclusions (XInclude) Version 1.0 (Second Edition)“. W3C Recommendation. W3C, 15. Nov. 2006. URL: https://www.w3.org/TR/2006/REC-xinclude-20061115/ (besucht am 11. 06. 2017).

[XML] Eve Maler, Jean Paoli, François Yergeau, Tim Bray und Michael Sperberg-McQueen. „Extensible Markup Language (XML) 1.0 (Fifth Edition)“. W3C Recommendation. W3C, 26. Nov. 2008. URL: https://www.w3.org/TR/2008/REC-xml-20081126/ (besucht am 21. 03. 2017).

[XML11] Tim Bray et al. „Extensible Markup Language (XML) 1.1 (Second Edition)“. W3C Recommendation. Editiert am 29. September 2006. W3C, 16. Aug. 2006. URL: https://www.w3.org/TR/2006/REC-xml11-20060816/ (besucht am 26. 06. 2017).

[XMLInfo] John Cowan und Richard Tobin. „XML Information Set (Second Edition)“. W3C Recommendation. W3C, 4. Feb. 2004. URL: https://www.w3.org/TR/2004/REC-xml-infoset-20040204/ (besucht am 08. 06. 2017).

[XMLNS] Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin und Henry S. Thompson. „Namespaces in XML 1.0 (Third Edition)“. W3C Recommendation. W3C, 8. Dez. 2009. URL: http://www.w3.org/TR/2009/REC-xml-names-20091208/ (besucht am 25. 06. 2017).

[XMLSig] Mark Bartel, John Boyer, Barb Fox, Brian LaMacchia und Ed Simon. „XML Signature Syntax and Processing (Second Edition)“. W3C Recommendation. W3C, 10. Juni 2008. URL: http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/ (besucht am 15. 05. 2017).

[XMLStyle] James Clark, Simon Pieters und Henry S. Thompson. „Associating Style Sheets with XML documents 1.0 (Second Edition)“. W3C Recommendation. W3C, 28. Okt. 2010. URL: https://www.w3.org/TR/2010/REC-xml-stylesheet-20101028/ (besucht am 06. 06. 2017).

[XPath] James Clark und Steve DeRose. „XML Path Language (XPath)“. W3C Recommendation. Version 1.0. Status updated October 2016. W3C, 16. Nov. 1999. URL: https://www.w3.org/TR/1999/REC-xpath-19991116/ (besucht am 08. 05. 2017).

[XSD] Shudi (Sandy) Gao, C.M. Sperberg-McQueen und Henry S. Thompson. „W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures“. W3C Recommendation. W3C, 5. Apr. 2012. URL: https://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/ (besucht am 04. 06. 2017).

[XSLT] Michael Kay. „XSL Transformations (XSLT) Version 3.0“. W3C Recommendation. W3C, 8. Juni 2017. URL: https://www.w3.org/TR/2017/REC-xslt-30-20170608/ (besucht am 13. 06. 2017).

[Yin13] Ming Ying und James Miller. „Refactoring legacy AJAX applications to improve the efficiency of the data exchange component“. In: *Journal of Systems and Software* 86.1 (2013), S. 72–88. ISSN: 0164-1212. DOI: 10.1016/j.jss.2012.07.019. URL: http://www.sciencedirect.com/science/article/pii/S0164121212002129 (besucht am 21. 03. 2017).

A Ausgabebeispiele der Konverter

```json
{
  "": null,
  "albums": {
    "album": {
      "@catno": "ARGO LP-628",
      "artist": {
        "$": "Ahmad Jamal Trio"
      },
      "title": {
        "$": "At The Pershing"
      },
      "recording": {
        "$": [
          "Recorded",
          "."
        ],
        "date": {
          "$": "January 16, 1958"
        }
      }
    }
  }
}
```

(a) Cobra vs Mongoose

```json
{
  "albums": {
    "album": [
      {
        "artist": "Ahmad Jamal Trio"
      },
      {
        "title": "At The Pershing"
      },
      {
        "recording": {
          "date": "January 16, 1958"
        }
      }
    ],
    "@catno": "ARGO LP-628"
  },
  "#comment": [
    "Nice album!"
  ]
}
```

(b) GreenCape XML Converter

```json
[{
  "@catno": "ARGO LP-628",
  "artist": "Ahmad Jamal Trio",
  "title": "At The Pershing",
  "recording":   {
    "#text":     [
      "Recorded ",
      "."
    ],
    "date": "January 16, 1958"
  }
}]
```

(c) Json-lib

```json
[
  "albums", "\n  ",
  [ "album", { "catno": "ARGO
  ↪  LP-628" }, "\n    ",
    [ "artist", "Ahmad Jamal Trio"
    ↪  ], "\n    ",
    [ "title", "At The Pershing" ],
    ↪  "\n    ",
    [ "recording", "Recorded ",
      [ "date", "January 16, 1958"
      ↪  ], "."
    ], "\n  "
  ], "\n"
]
```

(d) JsonML

```
{
  "albums": {
    "album": {
      "artist": "Ahmad Jamal Trio",
      "catno": "ARGO LP-628",
      "recording": {
        "date": "January 16, 1958",
        "content": [
          "Recorded",
          "."
        ]
      },
      "title": "At The Pershing"
    }
  }
}
```

(e) org.json.XML

```
{
  "albums": {
    "album": {
      "artist": "Ahmad Jamal Trio",
      "title": "At The Pershing",
      "recording": {
        "date": "January 16, 1958",
        "__text": "Recorded \n."
      },
      "_catno": "ARGO LP-628",
      "__text": "\n    \n\n    \n\n
      ↪  \n\n  "
    },
    "__text": "\n  \n\n"
  }
}
```

(f) x2js (Fork)

```
{
  "albums": {
    "album": {
      "artist": "Ahmad Jamal Trio",
      "title": "At The Pershing",
      "recording": {
        "date": "January 16, 1958",
        "_": "Recorded."
      },
      "$catno": "ARGO LP-628"
    }
  }
}
```

(g) JXON

```
{
  "?xml": {
    "@version": "1.0",
    "@encoding": "UTF-8"
  },
  "?xml-stylesheet":
  ↪  "href=\"style.css\"",
  /* Nice album! */
  "albums": {
    "album": {
      "@catno": "ARGO LP-628",
      "artist": "Ahmad Jamal Trio",
      "title": "At The Pershing",
      "recording": {
        "#text": [
          "Recorded ",
          "."
        ],
        "date": "January 16, 1958"
      }
    }
  }
}
```

(h) Json.NET

```
{
  "text": "\n  ",
  "tag": "albums",
  "children": [
    {
      "tail": "\n",
      "text": "\n    ",
      "attributes": {
        "catno": "ARGO LP-628"
      },
      "tag": "album",
      "children": [
        {
          "tail": "\n    ",
          "text": "Ahmad Jamal Trio",
          "tag": "artist"
        },
        {
          "tail": "\n    ",
          "text": "At The Pershing",
          "tag": "title"
        },
        {
          "tail": "\n  ",
          "text": "Recorded ",
          "tag": "recording",
          "children": [
            {
              "tail": ".",
              "text": "January 16, 1958",
              "tag": "date"
            }
          ]
        }
      ]
    }
  ]
}
```

(i) Pesterfish

```
{
  "albums": {
    "album": {
      "attributes": {
        "catno": "ARGO LP-628"
      },
      "children": [
        {
          "artist": "Ahmad Jamal Trio"
        },
        {
          "title": "At The Pershing",
          "recording": {
            "children": [
              "Recorded",
              {
                "date": "January 16, 1958"
              }
            ]
          }
        }
      ]
    }
  }
}
```

(j) xmljson (Abdera)

```
{
  "albums": {
    "album": {
      "catno": "ARGO LP-628",
      "artist": "Ahmad Jamal Trio",
      "title": "At The Pershing",
      "recording": {
        "content": "Recorded",
        "date": "January 16, 1958"
      }
    }
  }
}
```

(k) xmljson (Yahoo)

```
{
  "albums": {
    "album": {
      "@catno": "ARGO LP-628",
      "artist": {
        "$": "Ahmad Jamal Trio"
      },
      "title": {
        "$": "At The Pershing"
      },
      "recording": {
        "$": "Recorded",
        "date": {
          "$": "January 16, 1958"
        }
      }
    }
  }
}
```

(l) xmljson (Badgerfish)

```
{
  "albums": {
    "album": {
      "catno": "ARGO LP-628",
      "artist": {
        "$t": "Ahmad Jamal Trio"
      },
      "title": {
        "$t": "At The Pershing"
      },
      "recording": {
        "$t": "Recorded",
        "date": {
          "$t": "January 16, 1958"
        }
      }
    }
  }
}
```

(m) xmljson (GData)

```
{
  "albums": {
    "attributes": {},
    "children": [
      {
        "album": {
          "attributes": {
            "catno": "ARGO LP-628"
          },
          "children": [
            {
              "artist": "Ahmad Jamal
              ↪  Trio"
            },
            {
              "title": "At The
              ↪  Pershing",
              "recording": {
                "attributes": {},
                "children": [
                  "Recorded",
                  {
                    "date": "January
                    ↪  16, 1958"
                  }
                ]
              }
            }
          ]
        }
      }
    ]
  }
}
```

(n) xmljson (Cobra)

```
{
  "albums": {
    "album": {
      "artist": "Ahmad Jamal Trio",
      "title": "At The Pershing",
      "recording": {
        "date": "January 16, 1958"
      }
    }
  }
}
```

(o) xmljson (Parker)

B Problematische Unicode-Zeichen

B.1 Whitespace

Bei der Konversion mittels *JXON* und *xmljson* mit *Badgerfish*- oder *GData*-Konversion gehen die folgenden Unicode-Whitespace-Zeichen verloren:

Name	Unicode-Codepoint
CHARACTER TABULATION	000009
LINE FEED (LF)	00000A
CARRIAGE RETURN (CR)	00000D
SPACE	000020
NEXT LINE (NEL)[1]	U+0085
NO-BREAK SPACE	U+00A0
OGHAM SPACE MARK	U+1680
MONGOLIAN VOWEL SEPARATOR	U+180E
EN QUAD	U+2000
EM QUAD	U+2001
EN SPACE	U+2002
EM SPACE	U+2003
THREE-PER-EM SPACE	U+2004
FOUR-PER-EM SPACE	U+2005
SIX-PER-EM SPACE	U+2006
FIGURE SPACE	U+2007
PUNCTUATION SPACE	U+2008
THIN SPACE	U+2009
HAIR SPACE	U+200A
LINE SEPARATOR	U+2028
PARAGRAPH SEPARATOR	U+2029
NARROW NO-BREAK SPACE	U+202F
MEDIUM MATHEMATICAL SPACE	U+205F
IDEOGRAPHIC SPACE	U+3000
ZERO WIDTH NO-BREAK SPACE[2]	U+FEFF

[1] Nur bei *xmljson* mit *Badgerfish*- und *GData*-Konvention

[2] Nur bei *JXON*

B.2 Zahlen

Bei der Konversion mittels *xmljson* mit *Badgerfish*-, *GData*- oder *Parker*-Konvention wurden folgende Unicode-Zahlenbereiche in ihr ASCII-Äquivalent im Bereich `U+0030` bis `U+0039` umgewandelt:

Name	Unicode-Bereich	
	Beginn	Ende
ARABIC-INDIC DIGITS	U+0669	U+0660
EXTENDED ARABIC-INDIC DIGITS	U+06F9	U+06F0
NKO DIGITS	U+07C9	U+07C0
DEVANAGARI DIGITS	U+096F	U+0966
BENGALI DIGITS	U+09EF	U+09E6
GURMUKHI DIGITS	U+0A6F	U+0A66
GUJARATI DIGITS	U+0AEF	U+0AE6
ORIYA DIGITS	U+0B6F	U+0B66
TAMIL DIGITS	U+0BEF	U+0BE6
TELUGU DIGITS	U+0C6F	U+0C66
KANNADA DIGITS	U+0CEF	U+0CE6
MALAYALAM DIGITS	U+0D6F	U+0D66
SINHALA LITH DIGITS	U+0DEF	U+0DE6
THAI DIGITS	U+0E59	U+0E50
LAO DIGITS	U+0ED9	U+0ED0
TIBETAN DIGITS	U+0F29	U+0F20
MYANMAR DIGITS	U+1049	U+1040
MYANMAR SHAN DIGITS	U+1099	U+1090
KHMER DIGITS	U+17E9	U+17E0
MONGOLIAN DIGITS	U+1819	U+1810
LIMBU DIGITS	U+194F	U+1946
NEW TAI LUE DIGITS	U+19D9	U+19D0
TAI THAM HORA DIGITS	U+1A89	U+1A80
TAI THAM THAM DIGITS	U+1A99	U+1A90
BALINESE DIGITS	U+1B59	U+1B50
SUNDANESE DIGITS	U+1BB9	U+1BB0
LEPCHA DIGITS	U+1C49	U+1C40
OL CHIKI DIGITS	U+1C59	U+1C50
VAI DIGITS	U+A629	U+A620
SAURASHTRA DIGITS	U+A8D9	U+A8D0
KAYAH LI DIGITS	U+A909	U+A900
JAVANESE DIGITS	U+A9D9	U+A9D0
MYANMAR TAI LAING DIGITS	U+A9F9	U+A9F0
CHAM DIGITS	U+AA59	U+AA50
MEETEI MAYEK DIGITS	U+ABF9	U+ABF0
FULLWIDTH DIGITS	U+FF19	U+FF10

C Patches

C.1 Entfernung des Whitespace im GreenCape XML Konverter

```
From 944b1bf386f6388afdef1442f9b0366213b5fef4 Mon Sep 17 00:00:00 2001
From: Jan Holthuis <jan.holthuis@ruhr-uni-bochum.de>
Date: Tue, 13 Jun 2017 19:08:36 +0200
Subject: [PATCH] Disable auto-indentation of XML content

---
 src/GreenCape/XML/Converter.php | 25 ++++++--------------------
 1 file changed, 6 insertions(+), 19 deletions(-)

diff --git a/src/GreenCape/XML/Converter.php b/src/GreenCape/XML/Converter.php
index d7290e7..f24e812 100644
--- a/src/GreenCape/XML/Converter.php
+++ b/src/GreenCape/XML/Converter.php
@@ -78,15 +78,13 @@ private function traverse($node, $level = 0)
         {
                 $xml        = '';
                 $attributes = '';
-                $indent     = str_repeat('    ', $level);
 
                 if (!empty($node['#comment']))
                 {
                         foreach ($node['#comment'] as $comment)
                         {
-                                $comment = "{$indent}<!-- {$comment} -->";
-                                $comment = $this->applyIndentation($comment, $indent);
-                                $xml .= "\n" . $comment . "\n";
+                                $comment = "<!-- {$comment} -->";
+                                $xml .= $comment;
                         }
                         unset($node['#comment']);
                 }
@@ -111,7 +109,7 @@ private function traverse($node, $level = 0)
                         switch (gettype($data))
                         {
                                 case 'array':
-                                        $xml .= "{$indent}<{$tag}{$attributes}>\n";
+                                        $xml .= "<{$tag}{$attributes}>";
```

```
                                        if ($this->isAssoc($data))
                                        {
                                                $xml .= $this->traverse($data,
 ↪  $level + 1);
@@ -123,15 +121,15 @@ private function traverse($node, $level = 0)
                                                        $xml .=
 ↪  $this->traverse($child, $level + 1);
                                                }
                                        }
-                                       $xml .= "{$indent}</{$tag}>\n";
+                                       $xml .= "</{$tag}>";
                                        break;

                                case 'NULL':
-                                       $xml .= "{$indent}<{$tag}{$attributes}
 ↪  />\n";
+                                       $xml .= "<{$tag}{$attributes} />";
                                        break;

                                default:
-                                       $xml .=
 ↪  "{$indent}<{$tag}{$attributes}>{$data}</{$tag}>\n";
+                                       $xml .=
 ↪  "<{$tag}{$attributes}>{$data}</{$tag}>";
                                        break;
                        }
                }
@@ -386,15 +384,4 @@ private function isFile($data)
        {
                return file_exists($data);
        }
-
-       /**
-        * @param $text
-        * @param $indent
-        *
-        * @return mixed
-        */
-       private function applyIndentation($text, $indent)
-       {
-               return preg_replace('~\s*\n\s*~', "\n{$indent}", $text);
-       }
 }
--
2.13.1
```

C.2 Unterstützung für PIs in JsonML

```
From 73e5650c3f6cc9337b32bb93ab11f91a3fbe0b62 Mon Sep 17 00:00:00 2001
From: Jan Holthuis <jan.holthuis@ruhr-uni-bochum.de>
Date: Thu, 11 May 2017 19:01:52 +0200
Subject: [PATCH 1/5] Add support for ProcessingInstruction nodes

---
 jsonml-xml.js | 16 ++++++++++++++-
 1 file changed, 15 insertions(+), 1 deletion(-)

diff --git a/jsonml-xml.js b/jsonml-xml.js
index e8d48c2..3147cf7 100644
--- a/jsonml-xml.js
+++ b/jsonml-xml.js
@@ -62,6 +62,8 @@ if (typeof module === 'object') {

                } else if (tag.charAt(0) === '!') {
                        return document.createComment(tag === '!' ? '' :
 ↪  tag.substr(1)+' ');
+               } else if (tag.charAt(0) === '?') {
+                       return document.createProcessingInstruction(tag === '!' ?
 ↪  '' : tag.substr(1), '');
                }

                return document.createElement(tag);
@@ -99,7 +101,8 @@ if (typeof module === 'object') {
                                if (child.nodeType === 3) { // text node
                                        elem.nodeValue += child.nodeValue;
                                }
-
+                       } else if (elem.nodeType === 7) {
+                               elem.data = child.data;
                        } else if (elem.canHaveChildren !== false) {
                                elem.appendChild(child);
                        }
@@ -264,6 +267,17 @@ if (typeof module === 'object') {
                                // free references
                                elem = null;
                                return str;
+                       case 7: // ProcessingInstruction node
+                               var jml = ['?'+elem.target, elem.data]
+
+                               // filter result
+                               if ('function' === typeof filter) {
+                                       jml = filter(jml, elem);
+                               }
+
+                               // free references
+                               elem = null;
+                               return jml;
                        case 10: // doctype
                                jml = ['!'];
```

```
--
2.13.1


From c6a0b50a17ba4134a8d868bf9c2a1b24f6199d12 Mon Sep 17 00:00:00 2001
From: Jan Holthuis <jan.holthuis@ruhr-uni-bochum.de>
Date: Thu, 11 May 2017 19:05:31 +0200
Subject: [PATCH 2/5] Also use ProcessingInstructions outside the
 documentElement

Unfortunately, this introduces an ugly empty TextNode as parentElement,
but it works for now.
---
 jsonml-xml.js | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

diff --git a/jsonml-xml.js b/jsonml-xml.js
index 3147cf7..1a47c3a 100644
--- a/jsonml-xml.js
+++ b/jsonml-xml.js
@@ -375,7 +375,7 @@ if (typeof module === 'object') {
          */
         JsonML.fromXMLText = function(xmlText, filter) {
                 var elem = parseXML(xmlText);
-                elem = elem && (elem.ownerDocument || elem).documentElement;
+                elem = elem && (elem.ownerDocument || elem);

                 return fromXML(elem, filter);
         };
--
2.13.1


From 0a75a40d3a17009bac904e448e323e65bb8a7e98 Mon Sep 17 00:00:00 2001
From: Jan Holthuis <jan.holthuis@ruhr-uni-bochum.de>
Date: Fri, 12 May 2017 10:47:53 +0200
Subject: [PATCH 3/5] Avoid empty text node parent if documentElement is the
 only top-level node

---
 jsonml-xml.js | 4 ++++
 1 file changed, 4 insertions(+)

diff --git a/jsonml-xml.js b/jsonml-xml.js
index 1a47c3a..9c508b1 100644
--- a/jsonml-xml.js
+++ b/jsonml-xml.js
@@ -253,6 +253,10 @@ if (typeof module === 'object') {

                                 addChildren(elem, filter, jml);

+                                if (jml[0] === '' && jml.length === 2) {
+                                        jml = jml[1]
```

```
+                                }
+
                                 // filter result
                                 if ('function' === typeof filter) {
                                         jml = filter(jml, elem);
--
2.13.1


From edbbc2593943bfc805144076e83c39308eaee4f0 Mon Sep 17 00:00:00 2001
From: Jan Holthuis <jan.holthuis@ruhr-uni-bochum.de>
Date: Fri, 12 May 2017 10:49:28 +0200
Subject: [PATCH 4/5] Add unittests for processing instructions

---
 test/xmlTests.js | 17 +++++++++++++++++
 1 file changed, 17 insertions(+)

diff --git a/test/xmlTests.js b/test/xmlTests.js
index 9b270bf..8561930 100644
--- a/test/xmlTests.js
+++ b/test/xmlTests.js
@@ -340,4 +340,21 @@ test('JsonML.fromXMLText/.toXMLText roundtrip, comments',
 ↪   function() {
         same(actual, expected);
 });

+test('JsonML.fromXMLText/.toXMLText roundtrip, processing instructions',
 ↪   function() {
+
+        var expected =
+                '<?some-pi and its data?>' +
+                '<foo>' +
+                        '<?another-pi with data?>' +
+                '</foo>';
+
+        // JsonML will strip the XML Declaration
+        var input = '<?xml version="1.0"?>' + expected;
+
+        var jml = JsonML.fromXMLText(input);
+        var actual = JsonML.toXMLText(jml);
+
+        same(actual, expected);
+});
+
 }catch(ex){alert(ex);}
--
2.13.1


From aa601ae621cfc92435db38d0472304ea56962a5c Mon Sep 17 00:00:00 2001
From: Jan Holthuis <jan.holthuis@ruhr-uni-bochum.de>
Date: Mon, 22 May 2017 18:45:06 +0200
Subject: [PATCH 5/5] Add support for Processing Instructions to
```

```
 jsonml-utils.js

---
 jsonml-utils.js | 33 +++++++++++++++++++++++++++++++--
 1 file changed, 31 insertions(+), 2 deletions(-)

diff --git a/jsonml-utils.js b/jsonml-utils.js
index 76e352a..9e26a2f 100644
--- a/jsonml-utils.js
+++ b/jsonml-utils.js
@@ -84,13 +84,42 @@ if (typeof module === 'object') {
          * @return {boolean}
          */
         var isElement = JsonML.isElement = function(jml) {
-                return isArray(jml) && ('string' === typeof jml[0]);
+                return isArray(jml) && ('string' === typeof jml[0]) &&
 ↪   (jml[0].charAt(0) !== '?');
         };

         /**
          * @param {*} jml
          * @return {boolean}
          */
+        var isProcessingInstruction = JsonML.isProcessingInstruction =
 ↪   function(jml) {
+                return isArray(jml) && ('string' === typeof jml[0]) &&
 ↪   (jml[0].charAt(0) === '?');
+        };
+
+        /**
+         * @param {*} jml
+         * @return {string}
+         */
+        var getTarget = JsonML.getTarget = function(jml) {
+                if (!isProcessingInstruction(jml)) {
+                        throw new SyntaxError('invalid JsonML');
+                }
+                return jml[0].substring(1);
+        };
+
+        /**
+         * @param {*} jml
+         * @return {string}
+         */
+        var getData = JsonML.getData = function(jml) {
+                if (!isProcessingInstruction(jml)) {
+                        throw new SyntaxError('invalid JsonML');
+                }
+                return jml[1];
+        };
+        /**
+         * @param {*} jml
+         * @return {boolean}
+         */
```

```
        var isAttributes = JsonML.isAttributes = function(jml) {
                return !!jml && ('object' === typeof jml) && !isArray(jml);
        };
@@ -196,7 +225,7 @@ if (typeof module === 'object') {

                } else if (child && 'object' === typeof child) {
                        if (isArray(child)) {
-                               if (!isElement(child)) {
+                               if (!isElement(child) &&
 ↪  !isProcessingInstruction(child)) {
                                        throw new SyntaxError('invalid JsonML');
                                }

--
2.13.1
```