Die einfache Alternative zu XML, JSON und Co.

Einführung in das Datenformat SML

Haben Sie sich auch schon einmal beim Editieren von Konfigurationsdateien gefragt, warum Kommentare in XML so kompliziert geschrieben werden müssen oder warum JSON überhaupt keine Kommentare bietet? Oder mussten Sie in Ihrer YAML-Datei schon einmal einen Einrückungsfehler beheben? Wenn ja, dann kennen Sie bereits einige der Stolpersteine der gängigen textuellen Datenformate. In diesem Artikel wollen wir eine menschenfreundlichere Alternative zu den etablierten Standards betrachten, der Simple Markup Language.

von Stefan John - 18. August 2021

Ob als Konfigurationsdateien, zum Datenaustausch zwischen Client- und Server, oder zur Serialisierung von Objekten, textbasierte Formate wie XML, JSON, YAML und Co. sind allgegenwärtig für Entwickler. Doch nicht nur Entwickler sind konfrontiert mit diesen Formaten. Auch Anwender der Software, Supportspezialisten, Administratoren oder Berater arbeiten für Konfigurationszwecke oder zur Fehleranalyse mit Dateien und Datenströmen in diesen Formaten. Und obwohl deren Grundkonzepte meist relativ leicht zu verstehen sind, findet sich nicht jedermann leichtfertig mit den teils umfangreichen Regelwerken zurecht.

Selbst fortgeschrittene Entwickler kennen nicht alle Details der meist langen Spezifikationen auswendig. Die YAML-Spezifikation mit 84 Seiten ist, neben der XML-Spezifikation, ein Beispiel dafür. Bietet ein Format außerdem alternative Möglichkeiten an, wie etwas umgesetzt werden kann, überlegen selbst Experten. YAML zum Beispiel bietet neun unterschiedliche Möglichkeiten an, wie ein mehrzeiliger String geschrieben werden kann [1] und wer bereits ein XML-Format definiert hat, wird garantiert die ein oder andere Diskussion miterlebt haben, ob ein Wert nun als Element oder Attribut geschrieben werden soll.

Auch die Schreibweise der Formate fällt nicht jedermann leicht. Auch wenn Entwickler durch ihre Programmiersprachen trainiert sind, viele Sonderzeichen zu schreiben, ist vor allem Zehnfingerschreibern oft intuitiv bewusst, dass Sonderzeichen maßgeblich die Schreibgeschwindigkeit beeinflussen. Einige Entwickler wechseln daher extra vom deutschen Tastaturlayout zum US-amerikanischen Layout, nur um Sonderzeichen leichter tippen zu können.

Das Thema Lesbarkeit ist ein weiterer wichtiger Aspekt. Ein, auf eine Zeile minifiziertes JSON-Dokument ist selbst für einen Entwickler nur mit Tools zur Formatierung oder besseren Darstellung sinnvoll lesbar. Wie oft werden Dokumente mit sensiblen Daten in Online-PrettyPrinter-Webseiten eingefügt, ohne zu wissen, wohin die Daten gesendet werden? Und auf ein im Internet-Explorer dargestelltes XML-Dokument ist wohl auch der ein oder andere bereits gestoßen.

Die Frage, die sich nun stellt, ist, lässt sich ein alternatives Datenformat zu XML, JSON und Co. finden, das:

- auf ein Minimum an Regeln reduziert ist und dennoch funktional ebenbürtig bleibt,
- leicht und schnell zu schreiben ist,
- gut lesbar ist, auch ohne spezielle Tools,
- und auch für Nicht-Experten leicht verständlich und intuitiv ist?

Die Simple Markup Language, kurz SML, setzt sich genau diese Punkte zum Ziel. In den folgenden Abschnitten betrachten wir die Grundkonzepte und Schreibweise von SML und beleuchten am Ende mögliche Einsatzgebiete.

Ein erstes Beispiel

Betrachten wir zum Einstieg folgendes Beispiel eines SML-Geodatenformats, das einen markanten geografischen Punkt beschreibt – in diesem Fall dem Wahrzeichen der Stadt Seattle, dem Aussichtsturm Space Needle.

```
PointOfInterest
  City      Seattle
  Name      "Space Needle"
  GpsCoords 47.6205 -122.3493
  # Opening hours should go here
End
```

An dem Beispiel lässt sich erkennen, dass SML ein zeilenbasiertes Format ist. Die erste Zeile startet das Dokument und gibt einen Hinweis auf dessen Inhalt. Die zweite Zeile definiert in welcher Stadt sich der Point-of-Interest befindet und stellt ein Attribut dar. Der Attributname und der Attributwert sind mit mehreren Leerzeichen von einander getrennt und es steht kein Sonderzeichen, wie ein Doppelpunkt oder ein Ist-gleich-Zeichen, zwischen beiden. In Zeile drei steht der Name des Wahrzeichens. Im Gegensatz zu Zeile zwei steht der Wert des Attributs in doppelten Anführungszeichen. Diese werden geschrieben, da der Name selbst ein Leerzeichen enthält. Die nächste Zeile enthält die GPS-Koordinaten des Punktes. Attribute können mehrere Werte enthalten. Diese werden, wie in diesem Fall, einfach mit Leerzeichen oder mit anderen Leerraumzeichen (Whitespace) voneinander getrennt und hintereinander geschrieben. Die vorletzte Zeile enthält außer einem Kommentar, der mit einer Raute gestartet wird und bis zum Ende der Zeile geht, keine weiteren Informationen. Die letzte Zeile enthält das englische Wort End und schließt das Dokument ab. Schon an diesem Beispiel fällt auf, dass SML mit relativ wenigen Sonderzeichen auskommt. Selbst ein Nicht-Experte könnte diesen Text mit relativer Leichtigkeit abtippen und würde wohl nicht allzu lange dafür brauchen.

XML, JSON und YAML

Vergleichen wir nun das SML-Dokument mit einem in XML geschriebenen Dokument, das die selben Informationen enthält.

```
<?xml version="1.0" encoding="UTF-8"?>
<PointOfInterest>
  <City>Seattle</City>
  <Name>Space Needle</Name>
  <GpsCoords lat="47.6205" long="-122.3493"/>
  <!-- Opening hours should go here -->
</PointOfInterest>
```

Als Erstes sei anzumerken, dass dies nur eine Variante ist, wie die Informationen aus dem SML-Beispiel in XML abgebildet werden können. Die GPS-Koordinaten, zum Beispiel, könnten als Unterelemente anstatt mit Attributen dargestellt werden, oder als InnerText mit Sonderzeichen getrennt, der später wieder in zwei Bestandteile gesplittet wird. Hier beeinflusst auch die Präferenz des Entwicklers, für welche Variante sich entschieden wird. XML ist eine mächtige Markupsprache, mit der nicht nur Daten strukturiert abgebildet werden können, sondern auch Text im eigentlichen Sinne einer Markupsprache formatiert werden kann. Für dieses kleine Beispiel eines strukturierten Datensatzes sehen wir jedoch, dass wesentlich mehr Sonderzeichen verwendet werden, Attributwerte in Anführungszeichen geschrieben werden müssen und schließende Tags den gleichen Namen wie öffnenden Tags tragen. Würde man einen kleinen Abtippwettbewerb ohne spezielle Tools durchführen, wäre der SML-Abtippende wohl als erster fertig. Stolperfallen von XML können neben der XML-Deklaration in der ersten Zeile und der Schreibweise der Kommentare, vor allem aber Punkte wie Namensräume oder Details der Spezifikation sein. Sind Zeilenumbrüche zum Beispiel in Attributen erlaubt? Oder können Attribute auskommentiert werden? Was passiert, wenn die Datei in einem anderen, als dem in der Deklaration angegeben Encoding gespeichert wird? Oder können Sie die Syntax eines CDATA-Blocks aus dem Kopf eintippen?

Betrachten wir zum Vergleich einen weiteren weitverbreiteten Standard, der JavaScript Object Notation. Unser Geodatenbeispiel würde in JSON wie folgt aussehen.

```
{ "City":      "Seattle",
  "Name":      "Space Needle",
  "GpsCoords": [47.6205, -122.3493],
  "_comment":  "Opening hours should go here" }
```

JSON ist ein einfach gehaltenes Datenformat, das sich vor allem durch das Zusammenspiel mit JavaScript im Browser großer Beliebtheit erfreut und unter anderem für die Client-/Serverkommunikation, für eigene Datenformate als Alternative zu XML oder auch als Konfigurationsformat verwendet wird. Es baut auf der Verwendung von doppelten Anführungszeichen, Doppelpunkten, Kommas, eckigen und geschweiften Klammern, sowie einigen Schlüsselwörtern zur Beschreibung der Datenstruktur und Datentypen auf. String-

Werte müssen immer in doppelten Anführungszeichen geschrieben werden und durch C-kompatible Escapesequenzen, kann ein JSON-Dokument komplett in einer Zeile geschrieben werden. Ein vergessenes oder am Ende zu viel geschriebenes Komma führt zu einem Syntaxfehler und Kommentare, wie sie in JavaScript mit // und /* */ geschrieben werden können, sind im JSON-Standard nicht erlaubt. Für Serialisierungsformate mag das nicht weiter störend sein. Für Formate, in denen man jedoch gerne Teile ein- und auskommentieren möchte, wie zum Beispiel Konfigurationsdateien, kann dies unpraktikabel werden und hat zu einigen Workarounds oder Alternativformaten geführt. Diese reichen von Schlüssel-Wert-Paaren, bei denen ein spezielles Präfix im Schlüssel das Paar als Kommentar kennzeichnet, über Präprozessoren, die Kommentare vor dem Parsen des Dokuments herausfiltern, bis hin zu Formaten wie Hjson [2] und JSON5 [3]. Anders als bei SML oder XML trägt das JSON Wurzelelement keinen Namen. Ein JSON-Dokument hat damit standardmäßig keinen Identifier, der einen ersten Hinweis auf den Inhalt des Dokumentes gibt.

Eine verbreitete Alternative zu JSON ist YAML. YAML kombiniert die Syntax von JSON mit einer Sonderzeichen-reduzierten Schreibweise, die ähnlich wie die Programmiersprache Python, auf Einrückungsregeln basiert. Unser Geodatenbeispiel sieht in YAML wie folgt aus.

```
City:      Seattle
Name:      Space Needle
GpsCoords: [47.6205, -122.3493]
# Opening hours should go here
```

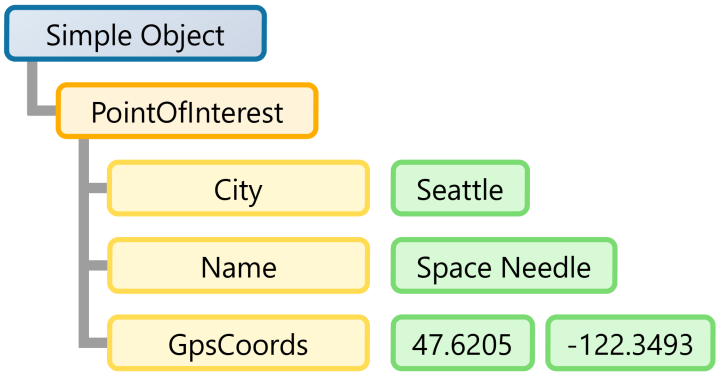
YAML unterstützt wie SML einzeilige Kommentare, die mit einer Raute beginnen. Die GPS-Koordinaten sind wie ein JSON-Array geschrieben, können aber auch einzeln in mehrzeiliger Notation mit vorangestelltem Bindestrich und mindestens einem vorangestellten Whitespace geschrieben werden. Ein Doppelpunkt, ebenfalls gefolgt von mindestens einem Whitespace-Zeichen, wird zur Trennung von Schlüsseln und Werten verwendet. Obwohl YAML auf den ersten Blick einfach erscheint, liegen die Tücken in den vielen und teils recht komplexen Sonderregeln [1]. Und wer Tabs zur Einrückung bevorzugt, der wird bei YAML zwingend Leerzeichen verwenden müssen.

Verallgemeinernd lässt sich sagen, je mehr Sonderzeichen zum Einsatz kommen und je umfangreicher das Regelwerk ist, desto schwieriger finden sich Nicht-Experten, wie auch Experten, zurecht. Je mehr Regeln existieren, desto mehr kann falsch gemacht werden und damit leidet die Robustheit der Formate. Das Regelwerk von SML ist deshalb auf ein Minimum reduziert, genauso wie die Anzahl der verwendeten Sonderzeichen. Das Format ist dadurch robust, leicht und schnell zu tippen und lässt sich einfach erlernen. Betrachten wir nun in den nächsten Abschnitten, wie genau SML funktioniert.

Simple Objekte

Bevor wir die Syntaxregeln der Simple Markup Language näher beleuchten, betrachten wir die Datenstruktur, die hinter SML-Dokumenten steht. Ein SML-Dokument bildet eine hierarchische Datenstruktur ab, die als Simple Object bezeichnet wird. Diese hierarchische Datenstruktur wird aus zwei Typen von Knoten aufgebaut, Elementen und Attributen. Ein

Simple Object besitzt genau ein Wurzelelement. Dieses Wurzelelement kann weitere Elemente oder Attribute enthalten. Elemente dienen dabei der Gruppierung und Attribute enthalten die eigentlichen Daten, die in Form von String-Werten vorliegen. Beide Knotentypen sind benannt. Elemente sind damit benannte Gruppen von Kindknoten und Attribute sind benannte String-Arrays.



Die Abbildung zeigt unser SML-Geodatenbeispiel als hierarchische Datenstruktur visualisiert. Das Simple Object besteht hierbei aus dem Wurzelelement PointOfInterest. Diesem Wurzelelement sind drei Attribute untergeordnet. Die ersten beiden Attribute City und Name enthalten jeweils nur einen Wert. Das dritte Attribut GpsCoords enthält zwei Werte.

Attribute müssen mindestens einen Wert enthalten, können jedoch auch leere Strings oder Null-Werte enthalten. Kindknoten eines Elements sind geordnet und können identische Namen haben. Es ist also zum Beispiel möglich mehrere Attribute mit dem gleichen Namen einem Element unterzuordnen. Elemente müssen nicht zwingend Kindknoten enthalten und können leer sein.

Bezüglich der Benennung der Knoten gibt es, bis auf die Einschränkung keine Null-Werte zu verwenden, keinerlei Restriktionen. Alle Unicode-Zeichen sind erlaubt und auch die Reihenfolge der Zeichen kann beliebig gewählt werden. Wichtig ist, dass Namen case-insensitiv sind, dass heißt die Groß- und Kleinschreibung nicht beachtet wird. In unserem Beispiel könnten wir also auch das Attribut City komplett in Kleinbuchstaben oder das Attribut Name komplett in Großbuchstaben schreiben, ohne dass dies einen Unterschied machen würde. Warum diese Eigenschaften wichtig sind, um ein leicht zu schreibendes Format zu erhalten, betrachten wir noch genauer im weiteren Verlauf des Artikels.

Serialisierung Simpler Objekte

Nun, da wir die Datenstruktur hinter SML-Dokumenten kennen, betrachten wir dessen Serialisierung. Die für die Maschine einfachste Form der Serialisierung, wäre eine Umwandlung in ein Binärformat. Zum Einlesen ist dann kein Parser notwendig und die Bytes müssen nur nach einem vorgegebenen Schema verarbeitet werden. Diese Form der Serialisierung ist zwar maschinenfreundlich, allerdings ganz und gar nicht menschenfreundlich. Alternativ ist eine Serialisierung in ein gängiges textuelles Format wie

XML oder JSON möglich. Da diese jedoch meist Groß- und Kleinschreibung unterscheiden oder, wie XML, Einschränkungen bezüglich der Namensgebung haben, ist eine Abbildung umständlich und bläht die resultierenden Dokumente unnötig auf.

Genau hier setzt die Simple Markup Language an und bietet eine auf das Minimum reduzierte textuelle Repräsentation eines Simple Objects. Das Grundkonzept hierfür ist simpel und basiert auf dem Ansatz, dass in einer textuellen Repräsentation zwei Textentitäten, deren Längen nicht bekannt sind, immer durch mindestens ein Zeichen getrennt werden müssen. Gängige Formate nutzen hierfür Doppelpunkte, Kommas, Ist-gleich-Zeichen und andere Sonderzeichen. Aber warum verwenden wir hierfür nicht einfach das Leerzeichen und die Enter-Taste? Ein Zeilenumbruch lässt sich mit nur einem einzelnen Zeichen abbilden und genauso reicht ein Leerzeichen, um eine visuelle Trennung zweier Werte zu erreichen. Wir sind damit bei einem zeilenbasierten Format angelangt.

Aber wie unterscheiden wir zwischen den beiden Knotentypen, ohne dafür Sonderzeichen zur Kennzeichnung zu verwenden? Der Trick hierfür ist einfach. Wir betrachten jede Zeile als eine Menge von Werten, die mit einzelnen oder zusammenhängenden Whitespace-Zeichen voneinander getrennt werden. Enthält die Zeile nur einen Wert, ist dies ein sich öffnendes oder schließendes Element. Enthält die Zeile mindestens zwei Werte, handelt es sich um ein Attribut. Enthält die Zeile keine Werte, handelt es sich um eine leere Zeile, die nicht zum Inhalt des Simple Object beiträgt.

Whitespace-Separated Values

Dieses Konzept ist vergleichbar mit einer CSV-Datei (Comma-Separated Values). Hier werden Werte mittels eines Trennzeichens, wie einem Komma oder Semikolon, von einander getrennt. Im Fall von SML handelt es sich bei den Trennzeichen um eine Gruppe von Zeichen, den Whitespace-Zeichen, die neben dem Leerzeichen auch Tabs und weitere Unicode-Leerraumzeichen einschließt. Eine Zeile eines SML-Dokuments ist damit eine WSV-Zeile (Whitespace-Separated Values) und das gesamte Dokument ein WSV-Dokument.

Für den Fall, dass ein Wert selbst Whitespace-Zeichen enthält, umschließen wir ihn einfach in doppelten Anführungszeichen. Betrachten wir die weiteren Sonderregeln.

```
WertMitLeerzeichen       "Hallo Welt"
WertMitAnführungszeichen "Hallo ""Welt"""
LeererString             ""
Null                     -
NurEinBindestrich        "-"
MehrzeiligerText         "Zeile 1"/"Zeile 2"
WertMitRaute             "#Das ist kein Kommentar"
```

Einhält ein Wert selbst ein doppeltes Anführungszeichen, muss der Wert in doppelten Anführungszeichen geschrieben werden und das Zeichen durch die Escapesequenz "" ersetzt werden. Ein leerer Wert wird durch zwei, direkt hintereinander folgende doppelte Anführungszeichen dargestellt und ein Nullwert durch einen Bindestrich. Durch diese

Konvention muss ein einfacher Bindestrich als Wert ebenfalls in doppelten Anführungszeichen geschrieben werden, um ihn vom Nullwert zu unterscheiden.

Bezüglich mehrzeiliger Werte, die Zeilenumbrüche enthalten, geht SML folgenden Weg. Um ein wirklich zeilenbasiertes Format zu bleiben, wird das Linefeed Zeilenumbruchzeichen durch die Escapesequenz "/" ersetzt. Dies hat den Vorteil, dass eine Attributszeile auch mit mehrzeiligen Werten in einer Zeile bleibt und die Struktur des Dokuments weiterhin zu erkennen bleibt. Der Ansatz bietet außerdem den Vorteil, dass ein selbstgeschriebener Parser, der den Dokumentenstring durch einen einfachen String-Split-Aufruf in seine Zeilen zerlegt, keine falschen Ergebnisse liefert. Im Beispiel oben sehen wir, wie ein zweizeiliger Wert in einer Zeile geschrieben wird.

Da das Raute-Zeichen in SML den Start eines Kommentars markiert, muss ein Wert, der das Raute-Zeichen enthält, ebenfalls in doppelten Anführungszeichen geschrieben werden. Und mit diesem Punkt endet auch bereits die Liste der Regeln. Keine weiteren Zeichen müssen durch Escapesequenzen ersetzt werden. Die Folge ist, dass viele Werte, auch mit exotischen Sonderzeichen, ohne doppelte Anführungszeichen auskommen.

The End

Wir hatten bereits definiert, dass eine SML-Zeile mit nur einem Wert ein sich öffnendes oder schließendes Element darstellt. Um zu unterscheiden, um welchen der beiden Fällen es sich handelt, benötigen wir ein spezielles Schlüsselwort. Am naheliegendsten ist ein Schlüsselwort zu verwenden, das bereits für viele Jahrzehnte in zeilenbasierten Programmiersprachen zum Einsatz kommt, dem Schlüsselwort End. Dieses wird standardmäßig verwendet. Doch SML geht einen Schritt weiter und erlaubt arbiträre Werte als Endeschlüsselwörter. Grund hierfür ist, dass ein SML-Dokument nicht in englischer Sprache geschrieben werden muss, sondern auch komplett in einer anderen Sprache geschrieben werden kann. Das folgende SML-Dokument zum Beispiel ist komplett in Deutsch geschrieben.

```
Vertragsdaten
  Personendaten
    Nachname Meier
    Vorname Hans
  Ende
  Datum 2021-01-02
Ende
```

Damit das funktioniert, geht der Parser beim Laden des Dokumentes an dessen Ende, bestimmt das Endeschlüsselwort und beginnt die Interpretation der Zeilen von oben. Durch dieses Konzept, ist eine vollständige Lokalisierung und ein komplett automatisches Einlesen möglich, ohne spezifizieren zu müssen, wie das Endeschlüsselwort heißt.

Im Gegensatz zum Simple Object selbst, gibt es durch dieses Konzept bezüglich der Benennung der Elemente eine Einschränkung. Ein Element darf nicht den selben Namen tragen, wie das Endeschlüsselwort, da sonst die hierarchische Struktur nicht korrekt erkannt

werden kann. Ansonsten dürfen, außer dem Nullwert, beliebige Namen gewählt werden, für die die gleichen Schreibweiseregeln gelten, wie für Werte. Das folgende Beispiel zeigt ein SML-Dokument, dessen Wurzelelement und ein untergeordnetes Attribut jeweils Leerzeichen im Namen enthalten und deshalb in doppelten Anführungszeichen geschrieben werden müssen.

```
"My Root Element"
  "My First Attribute" 123
End
```

Der Name für Elemente und Attribute darf nicht Null sein, um die Verarbeitung in Programmen robuster zu gestalten. Ein weiterer Grund hierfür ist die Möglichkeit, dass SML-Dokumente minifiziert werden können, was wir nun genauer beleuchten.

Minifikation

Einrückungen helfen in SML, die hierarchische Struktur besser zu erkennen, wie das folgende Beispiel einer Konfigurationsdatei eines Spiels mit zwei untergeordneten Elementen zeigt.

```
# Game.cfg
Configuration
  Video
    # Hier die Auflösungseinstellungen festlegen
    Resolution   1920 1080  #Alternativ 1280 720
    RefreshRate  60
    Fullscreen   true
  End
  Audio
    Volume 100
    #Music 80
  End
End
```

Im Gegensatz zu YAML ist die Einrückung bei SML nicht verpflichtend und es können alle Whitespace-Zeichen nach Belieben verwendet werden. Werte von Attributen müssen nicht zwingend durch nur ein einzelnes Whitespace-Zeichen vom Attributsnamen und den folgenden Werten getrennt werden, sondern können beliebig eingerückt werden. Auch Kommentare sind überall möglich. Zum Beispiel wurde in der SML-Konfigurationsdatei hinter dem Attribut Resolution eine Anmerkung hinterlassen und das Attribut Music wurde komplett auskommentiert. Vollständig leere Zeilen oder Zeilen, die nur aus einem Kommentar bestehen, sind möglich.

Diese Formatierungsfreiheit trägt zur Robustheit und Menschenfreundlichkeit von SML bei. Für eine Maschine sind die Einrückungen und Kommentare jedoch nicht von Bedeutung und können deshalb entfernt werden. Vor allem im Kontext von Client-/Serverkommunikation, bei der Datengrößen eine Rolle spielen, macht ein, auf das

Nötigste minifiziertes Dokument Sinn. JSON bietet auch die Möglichkeit zur Minifikation, bei der Whitespace komplett entfernt und das Dokument auf einen Einzeiler reduziert wird. Für die Datengröße ist das gut. Die Lesbarkeit leidet darunter jedoch und führt zu dem bereits erwähnten Einsatz von Pretty-Printern. SML auf der anderen Seite bleibt auch noch im minifizierten Zustand lesbar, da die Zeilenumbrüche erhalten bleiben. Dies macht, obwohl es vielleicht gegen die Intuition des ein oder anderen ist, größentechnisch keinen Unterschied, da nur ein Zeichen verwendet wird. Das Konfigurationsbeispiel sieht minifiziert wie folgt aus.

```
Configuration
Video
Resolution 1920 1080
RefreshRate 60
Fullscreen true
-
Audio
Volume 100
-
-
```

Alle Kommentare wurden entfernt und die Einrückung ist nicht mehr vorhanden. Werte und Attributnamen sind nur noch mit einem Leerzeichen von einander getrennt. Das Endeschlüsselwert wurde ersetzt durch einen Nullwert und wird damit, im serialisierten Zustand, nur noch durch einen Bindestrich dargestellt. Da der Name von Elementen nicht Null sein darf, kann es hier zu keinen Namenskollisionen kommen und eine Minifikation ist immer garantiert. Auch ohne Pretty-Printer lässt sich der Inhalt des Dokuments noch gut erkennen.

Kodierung

Wenn Sie SML-Dokumente als Dateien speichern oder als Byte-Array serialisieren, müssen Sie bezüglich des Encodings beachten, dass SML-Dokumente ReliableTXT-Dokumente sind. ReliableTXT [4] ist eine Konvention, die vorgibt, wie Textdateien kodiert, dekodiert und interpretiert werden. Die Vorgaben sind dabei so gewählt, dass gängige Encodingprobleme vermieden werden. Erreicht wird dies durch das verpflichtende Schreiben einer Encoding-Präambel (einer kurzen Bytefolge), die das verwendete Encoding eindeutig identifiziert. Durch diese Präambel muss das Encoding nicht erraten werden und es ist stattdessen ein zuverlässiges Einlesen möglich. ReliableTXT beschränkt sich bei den möglichen Encodings auf vier Unicode-Encodings. Diese sind UTF-8, UTF-16 in Little- und Big-Endian, sowie UTF-32 Big-Endian. Bei allen muss der sogenannte BOM (Byte Order Mark) geschrieben werden. Wird er weggelassen, muss ein SML-Loader zwingend einen Fehler anzeigen und darf das Dokument nicht einlesen. Das mag strikt klingen, ist aber notwendig, um ein zuverlässiges Einladen zu ermöglichen. Wichtig ist zu wissen, dass die Präambel nicht Teil des Textinhaltes ist.

ReliableTXT verwendet außerdem eine andere Bezeichnung für die Bytereihenfolge. Die Little-Endian Reihenfolge wird als Reverse bezeichnet und die Bezeichnung Big-Endian ganz weggelassen. Dies ist eine alternative Eselsbrücke zum Merken der Bytereihenfolge.

ReliableTXT geht bezüglich der Zeilenumbrüche auch einen eigenen Weg. Der Unicode-Standard kennt sieben unterschiedliche Zeichen, die als Zeilenumbrüche interpretiert werden können. ReliableTXT begrenzt sich auf ein einzelnes Zeichen als Zeilenumbruch, dem Zeilenvorschubzeichen (Line Feed). ReliableTXT-Dateien sind dadurch aber keine POSIX/Unix-Textdateien, da Zeilen nicht durch das Linefeed-Zeichen terminiert, sondern separiert werden. Das ist ähnlich dem Konzept von Windows-Textdateien, jedoch ohne die Verwendung des Wagenrücklaufzeichens (Carriage Return). Da das Carriage-Return-Zeichen als Whitespace in einem SML-Dokument angesehen wird, entstehen jedoch keine Probleme, sollte ein SML-Dokument einmal mit Windows-Zeilenumbrüchen geschrieben worden sein.

Einsatzmöglichkeiten

SML ist ein universelles Format und kann in den verschiedensten Bereichen eingesetzt werden. Vor allem im Bereich der strukturierten Daten ist SML leicht einzusetzen. Formate zur Konfiguration und Lokalisierung, 2D- und 3D-Grafikformate, oder UI-, Geodaten-, Multimedia- oder Manifestformate sind mögliche Anwendungsbereiche. Ein einfaches Format zur Beschreibung von Rezepten ist ebenso möglich, wie auch beliebig komplexe Datenstrukturen. Die Verwendung von SML ist vor allem dann sinnvoll, wenn ein Datei direkt im Texteditor betrachtet oder verändert werden soll, z.B. wenn es noch kein Programm mit visueller Oberfläche gibt. Aber auch um Dateien für andere Programme zu erstellen, lässt sich SML leicht einsetzen. Eine Media-Playlist wie die folgende, kann jedes Programm leicht erzeugen.

```
Tracks
  Track Song1 /storage/sdcard0/Music/Song1.ogg
  Track Song2 /storage/sdcard0/Music/Rock.ogg
  Track Song3 https://www.example.com/Pop.ogg
End
```

Eine besondere Stärke von SML ist die Verbindung von hierarchischen Datenstrukturen und tabellarischen Daten. Dadurch, dass Attribute mehrere Werte enthalten können, lassen sich Tabellen leicht in SML-Dokumenten einbetten. Ist die erste Spalte der Tabelle ein Primärschlüssel, der keine Nullwerte annehmen darf, kann der Wert direkt als Attributsname verwendet werden.

Das nächste Beispiel zeigt ein SML-Dokument, das zwei eingebettete Tabellen enthält.

```
Tables
  Table1
    FirstName  LastName  Age  PlaceOfBirth
    William    Smith     30   Boston
    Olivia     Jones     27   Austin
    Lucas      Brown     38   Chicago
  End
  Table2
    City     State
    Boston   Massachusetts
    Austin   Texas
    Chicago  Illinois
  End
End
```

Durch die Flexibilität in der Formatierung können Tabs oder Leerzeichen verwendet werden, um die Daten visuell besser angeordnet darzustellen. Ein solches Dokument lässt sich leicht in jedem Texteditor eintippen, ohne ein einziges Sonderzeichen zu verwenden.

Auch Client-/Serverkommunikation oder Remote Procedure Calls können von SML profitieren. Wurde einst eine XML-Datei mittels AJAX übermittelt und dann später auf JSON umgestellt, könnten nun SML-Dokumente zwischen den Maschinen fließen. Größentechnisch ist SML JSON mindestens ebenbürtig und kann sogar kleinere Größen erzielen, auch ohne Kompression.

Fazit

Am Anfang des Artikels haben wir uns die Frage gestellt, ob sich ein alternatives Datenformat zu XML, JSON und Co. finden lässt, das auf ein Minimum an Regeln reduziert ist und dennoch funktional ebenbürtig bleibt, leicht und schnell zu schreiben und außerdem gut lesbar ist, und auch für Nicht-Experten leicht verständlich und intuitiv ist. Der letzte Punkt ist natürlich rein subjektiv, denn jeder entscheidet für sich persönlich, was verständlich und intuitiv ist. Auch der erste Punkt ist davon abhängig, für welchen Einsatzzweck ein Format verwendet werden soll. Soll ein Text sehr feingranular formatiert werden, ist wohl eine Sprache wie XML oder Markdown eine gute Entscheidung. Geht es jedoch um rein strukturierte Daten, steht SML den weitverbreiteten Standards wie XML, JSON und YAML in nichts nach. SML punktet vor allem dann, wenn es um die manuelle Veränderung von Dokumenten oder deren Darstellung in Texteditoren geht. Hier zeichnet sich SML durch seine einfache Schreibweise aus. Zehnfingerschreiber werden das Format wohl als besonders angenehm empfinden, da es durch die reduzierte Anzahl an Sonderzeichen und die Case-Insensitivität schneller zu tippen ist. Bezüglich der Anzahl der Regeln hat SML ein Minimum erreicht, das wohl nur durch Weglassen von Funktionalität übertroffen werden kann.

Ansonsten bleibt nur zu sagen, probieren Sie es einfach aus. SML kann online direkt im Browser ausprobiert werden [5] und Referenzbibliotheken stehen für TopTen-TIOBE-

Sprachen wie Java, PHP, C#, JavaScript und Python zur Verfügung [6] und viele Anleitungsvideos für die ersten Schritten existieren [7]. Vielleicht ist SML auch für Sie eine runde Sache. In diesem Sinne, frohes SML tippen.

Über den Autor

Stefan John beschäftigt sich seit über 20 Jahren mit Datenformaten jeder Art. Die Suche nach einem einfachen und robusten Datenformat für jedermann hat ihn zur Entwicklung der Simple Markup Language geführt.

Links

[1] https://www.arp242.net/yaml-config.html
[2] https://hjson.github.io
[3] https://json5.org
[4] https://www.reliabletxt.com
[5] https://www.simpleml.com
[6] https://github.com/Stenway
[7] https://www.youtube.com/channel/UCSVt-9JcnxfTFnztnLEQQug

Copyright-Hinweis