Skriptum Kommunikationstechnik

Teil 2 Datenformate

DI(FH) Andreas Pötscher, HTL Litec

1.9.2025

Inhaltsverzeichnis

Datum	Beschreibung der durchgeführten Änderung	Autor
09.04.2025	V1.0 . . . 1.Version Basierend auf FI1 bis FI3 von Gebhard Klinkan	Andreas Pötscher, HTL Linz–Paul-Hahn-Straße (LiTec)

Einführung JavaScript

Einfachen Daten werden in IT Systemen als primitive Datentypen dargestellt. Diese können je nach Datentyp verschiedene Werte enthalten. In dieser Unterlage wird dafür die Programmiersprache JavaScript verwendet. Auf dieser basiert auch Node-Red, das für die Übungsbeispiele verwendet wird. In JS (JavaScript) gibt es folgende primitive Datentypen:

- **Number**: Für numerische Werte, sowohl Ganzzahlen als auch Gleitkommazahlen.
- **String**: Für Zeichenketten, also Text.
- **Boolean**: Für Wahrheitswerte `true` oder `false`.
- **Null**: Wert, der bewusst als Abwesenheit von einem Wert verwendet wird.

Die Syntax von JS basiert auf der C Syntax.

Dynamische Typisierung

Im großen Unterschied zu C verwendet JavaScript keine strenge Typisierung sondern eine dynamische Typisierung, das sogenannte Duck Typing. Duck Typing ist ein Konzept in JavaScript, bei dem der Datentyp eines Objekts oder einer Variable nicht aufgrund der Initialisierung, sondern aufgrund seines Verhaltens oder seiner Eigenschaften bestimmt wird. Dies bedeutet, dass JavaScript die Typüberprüfung zur Laufzeit durchführt, anstatt zur Entwicklungszeit.

Beispiel:

```
let i = 10;
console.log(typeof(i)); //number
```

Das die Variable i vom Typ number ist wird erst dynamisch durch die Zuweisung des Wertes 10 festgelegt.

Beispiel:

```
let i = "10"
console.log(typeof(i)); //string
```

Hier wird die Variable i durch Zuweisung eines Textes automatisch zu einem String.

Diese Verhalten ist an und für sich sehr praktisch beinhaltet aber auch einige Fallstricke. Javascript wandelt (auch casting genannt) wenn möglich die Variablen automatisch auf Datentypen, dass sich die Operation ausführen lassen.

Beispiel:

```
let a = 1;   //number
let b = "1"  //string
console.log(b+a); //Ausgabe 11
```

Hier wird a automatisch auf einen String gecastet, da der + Operator für Strings implementiert ist. Der + Operator verkettet Strings miteinander.

Beispiel:

```
let a = 1;
let b = "1"
console.log(b-a); // Ausgabe 0
```

Der - Operator ist für Strings nicht implementiert. Deshalb wird b automatisch auf den Typ number gecastet.

Um hier Fehler zu vermeiden, können die Variablen aktiv gecastet werden.

Beispiel:

```
let a = 1;
let b = "1"
console.log(a+Number(b)); // Ausgabe 2
```

Kontrollstrukturen in JS

Bei den Kontrollstrukturen gibt es im weitesten Sinne die gleichen wie in C/C++.

- Verzweigungen (`if`,`switch`)
- Schleifen (`for`, `while`)

Die Vergleichsoperatoren wie `&&`, `||`, `!`, usw funktionieren wie in C. Aus der dynamischen Typisierung ergibt sich dennoch ein wichtiger Unterschied. Es gibt Vergleichsoperatoren die nur den Inhalt der Variable vergleichen und Vergleichsoperatoren die auch den Datentyp vergleichen.

Operator	Was wird verglichen	Beispiel	
`==`	Wert	`10 == "10"`	`//true`
`===`	Wert und Datentyp	`10 === "10"`	`//false`
`!=`	Wert	`10 != "10"`	`//false`
`!==`	Wert und Datentyp	`10 !== "10"`	`//true`

Beispiel:

```
let x = 10;
let y = "10";
console.log(x == y);  //true
console.log(x === y); //false
```

Darstellung von komplexeren Datenstrukturen in Software

In diesem Skriptum werden zwei komplexe Datenstrukturen in JavaScript beschrieben. Arrays und Objekte.

Arrays in JS

Arrays in JavaScript sind spezielle Datentypen, die verwendet werden, um eine geordnete Liste von Werten zu speichern. Arrays können Werte unterschiedlicher Datentypen (einschließlich anderer Arrays) enthalten und ermöglichen den Zugriff auf Elemente anhand ihres Index. Theoretisch handelt es sich bei JS Arrays nicht klassische Arrays sondern um Listen. Deshalb gibt es auch Funktion mit denen Elemente hinzugefügt und entfernt werden können.

Deklaration von Arrays: Um ein Array zu erstellen, können Sie die folgenden beiden Methoden verwenden:

```
// Methode 1: Verwenden Sie die Array-Literal-Syntax
const fruits = ["Apfel", "Banane", "Kirsche"];

// Methode 2: Verwenden Sie den Array-Konstruktor
const colors = new Array("Rot", "Grün", "Blau");
```

Zugriff auf Array-Elemente: Sie können auf die Elemente eines Arrays mithilfe ihres Index zugreifen. Beachten Sie, dass die Indexierung in JavaScript bei 0 beginnt:

```
console.log(fruits[0]); // Gibt "Apfel" aus
console.log(colors[2]); // Gibt "Blau" aus
```

Länge eines Arrays: Sie können die Länge eines Arrays mithilfe der `length`-Eigenschaft ermitteln:

```
console.log(fruits.length); // Gibt 3 aus
```

Hinzufügen von Elementen: Sie können ein Element am Ende eines Arrays hinzufügen, indem Sie die `push()`-Methode verwenden:

```
fruits.push("Erdbeere");
console.log(fruits); // Gibt ["Apfel", "Orange", "Kirsche", "Erdbeere"] aus
```

Entfernen von Elementen: Am Ende des Array kann ein Element mit `pop()` Funktion entfernt werden.

```
fruits.pop();
console.log(fruits); // Gibt ["Apfel", "Orange", "Kirsche"] aus
```

Objekte in JS

Objekte in JavaScript sind komplexe Datentypen, die verwendet werden, um Daten in Form von Schlüssel-Wert-Paaren zu organisieren. Ein Schlüssel ist eine Zeichenkette (String), die den Zugriff auf den zugehörigen Wert ermöglicht. Der Wert kann dabei jeder beliebige Datentyp (auch wieder ein Objekt) sein.

Objekte können im Code mit json Syntax initialisiert werden.

```
let person = {
  name: "John",
  age: 30,
  job: "Programmierer"
};
```

Da die Typisierung in JS dynamisch erfolgt können Objekte jederzeit um ein Attribut erweitert werden.

```
person.lastName = "Miller";
////person wird dynamisch um den lastName erweitert.
```

Um auf Eigenschaften der Objekte zugreifen zu können wird der Punkt-Operator(`.`) verwendet. Alternativ dazu kann mit `[]` auf ein Attribut zugegriffen werden.

```
console.log(person.name);  // Gibt "John" aus
console.log(person["age"]) // Gibt 30 aus
```

Darstellung von Daten in Datenaustauschformaten

Daten lassen sich zum Austausch mittels Datei, Webservice, serieller Schnittstelle oder anderen Möglichkeiten, strukturiert darstellen. Dazu gibt es grundsätzlich zwei Möglichkeiten.

- **Binäre Datenformate:** Dabei wird die Information nicht als lesbarer Text gespeichert, sondern als Abfolge von Bytes die dann jeweils eine spezielle Bedeutung haben. Der Vorteil dieser Formate ist, dass sie oft speicher-effizienter sind als Textformate. Beispiele dazu sind: doc, jpeg, avi, exe, ...
- **Textbasierte Datenformate:** Dabei werden die Daten als menschlich lesbarer Text gespeichert. In der Software werden diese zumeist als String geladen und dann in eine Datenstruktur umgewandelt. Der Vorteil ist, dass die Daten klar lesbar und auch veränderbar sind. Beispiele dazu sind: json, xml, csv, yaml

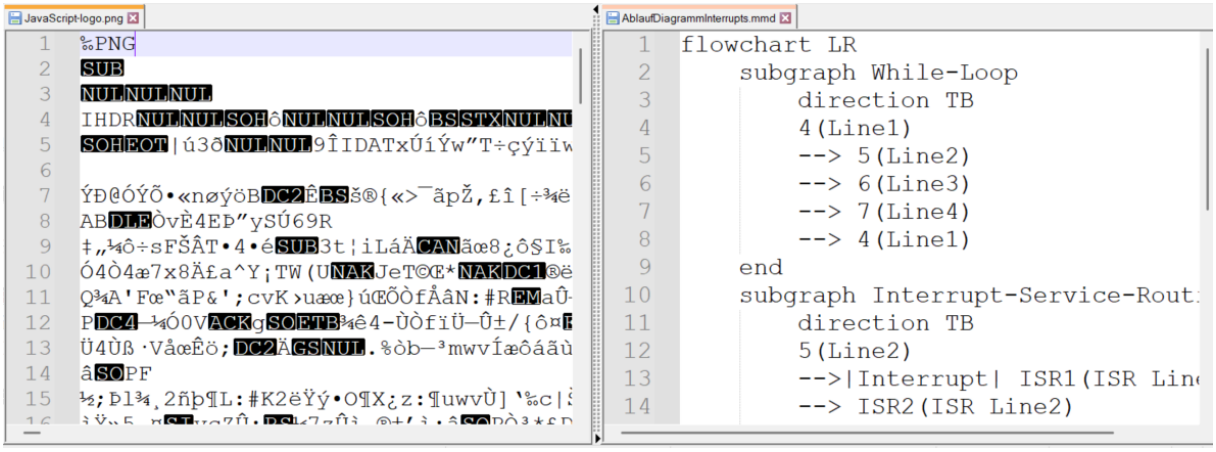

Abbildung 1: Binärformat vs. Textformat (geöffnet in einem Texteditor)

Für den Austausch von Daten zwischen mehreren Systeme werden meistens textbasierte Formate verwendet. Diese werden in diesem Skriptum behandelt.

Java Script Object Notation (JSON)

JSON steht für "JavaScript Object Notation" und ist ein einfaches Datenformat, das menschenlesbar ist und außerdem sehr einfach von Software interpretiert werden kann. Es wurde ursprünglich als Datenaustauschformat für JavaScript entwickelt, hat jedoch aufgrund seiner Einfachheit und Vielseitigkeit weit über JavaScript hinaus an Popularität gewonnen. JSON wird häufig für die Übertragung strukturierter Daten zwischen einem Server und einer Webanwendung sowie für die Speicherung von Konfigurationsdaten verwendet.

Ein json Objekt besteht immer aus einem oder mehreren Schlüssel-Werte-Paaren (Key-Value Pairs). Im einfachsten Fall:

```
{
   "Name":"John Doe"
}
```

In diesem Beispiel hat der Key `"Name"` den Value `"John"`. Wobei der Key immer ein String ist. Mehrere Key-Value Pairs werden durch einen "`,`" getrennt. Der Value kann auch ein anderer Datentyp sein. Z.B.

```
{
    "Name":"John Doe",
    "Alter": 30,
    "Stadt": "New York",
    "Verifiziert": true
}
```

auch ein Array oder eine weiteres json Objekt ist als Value möglich.

```
{
  "Name": "John Doe",
  "Alter": 30,
  "Stadt": "New York",
  "Verifiziert": true,
  "Freunde": ["Jane", "Bob", "Alice"],
  "Profil": {
    "Hobbys": ["Lesen", "Joggen", "Fotografie"],
    "MitgliedSeit": 2020,
    "PremiumStatus": false
  }
}
```

Die Objekte und Arrays können dabei beliebig tief verschachtelt werden. Für die meisten Programmiersprachen und Anwendung gibt es Bibliotheken die json Strings in Datenstrukturen umwandeln können.

Abbildung 2: JSON Node in Node-Red

Eine genaue Beschreibung der JSON Struktur findet sich in RFC8259

eXtensible Markup Language (XML)

XML steht für "eXtensible Markup Language" und ist eine Auszeichnungssprache, die zur Darstellung hierarchisch strukturierter Daten in Textform verwendet wird. Im Gegensatz zu JSON, das eher für den Datenaustausch zwischen Anwendungen verwendet wird, wurde XML entwickelt, um Daten zu strukturieren und zu speichern. XML ist erweiterbar und kann für eine Vielzahl von Anwendungsfällen eingesetzt werden, von der Konfigurationsdatei über Datenbankexporte bis hin zur Darstellung von Inhalten in Webseiten.

XML verwendet Tags, um Daten zu kennzeichnen und zu strukturieren. Ein Tag besteht aus einem Elementnamen, der von spitzen Klammern umschlossen ist, z.B., `<ElementName>`. Es gibt zwei Arten von Tags: Start-Tags und End-Tags. Zum Beispiel:

```
<Name>John Doe</Name>
```

Attribute von John Doe können bei XML auf zwei verschiedenen Arten hinzugefügt werden. Entweder als Einrückung mit einem eigen Tag oder als Attribut beim Start-Tag.

```
<Person Alter="30" Verifiziert="true">
  <Name>John Doe</Name>
  <Stadt>New York</Stadt>
</Person>
```

Ein Array kann in XML nicht direkt dargestellt werden. Stattdessen gibt es einen Container Knoten (`<Freunde>`) mit mehreren gleichen Einträgen (`<Freund>`). Die gesamte Datenstruktur aus dem json Kapitel sieht in XML wie folgt aus.

```
<Person Alter="30" Verifiziert="true">
  <Name>John Doe</Name>
  <Stadt>New York</Stadt>

  <Freunde>
    <Freund>Jane</Freund>
    <Freund>Bob</Freund>
    <Freund>Alice</Freund>
  </Freunde>

  <Profil PremiumStatus="false" MitgliedSeit="2020">
    <Hobbys>
      <Hobby>Lesen</Hobby>
      <Hobby>Joggen</Hobby>
      <Hobby>Fotografie</Hobby>
    </Hobbys>
  </Profil>
</Person>
```

Für die meisten Programmiersprachen und Anwendung gibt es Bibliotheken die xml Strings in Datenstrukturen umwandeln können.

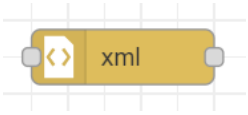

Abbildung 3: XML Node in Node-Red

Eine genaue Beschreibung der XML Struktur findet sich in RFC3470

Comma Separated Value (CSV)

CSV steht für "Comma-Separated Values" oder auf Deutsch "Kommagetrennte Werte". Es handelt sich um ein einfaches Dateiformat zur Darstellung von tabellarischen Daten in Textform. In CSV-Dateien werden Daten in Zeilen und Spalten organisiert, wobei die Werte durch ein bestimmtes Trennzeichen, typischerweise ein Komma (`,`) oder ein Strichpunkt (`;`), getrennt sind. Die erste Zeile kann optional als Kopfzeile verwendet werden.

```
Name;Alter;Stadt
John;25;New York
Jane;30;San Francisco
Bob;22;Los Angeles
```

In diesem Beispiel sind "Name", "Alter" und "Stadt" die Spaltenüberschriften, während jede nachfolgende Zeile einen Datensatz darstellt. Die Werte sind durch Strichpunkte getrennt.

Im Gegensatz zu json oder xml ist csv keine Verschachtelung in Klassen oder Arrays direkt möglich. Csv kann nur flache Datenstrukturen darstellen. Die Datenstruktur aus dem JSON Beispiel kann nur sehr umständlich und unflexibel dargestellt werden. Soll z.B. ein weiterer Freund hinzugefügt werden muss eine neue Spalte für alle Einträge hinzugefügt werden.

```
Name;Alter;Stadt;Verifiziert;Freund1;Freund2;Freund3
John Doe;New York;true;Jane;Bob;Alice
```

CSV-Dateien sind weit verbreitet und werden häufig für den Austausch von Daten zwischen verschiedenen Anwendungen, wie zum Beispiel zwischen Tabellenkalkulationsprogrammen und Datenbanken, verwendet. Sie sind einfach zu erstellen, zu lesen und zu interpretieren.

In Node Red können csv Daten mit dem csv Node interpretiert werden.

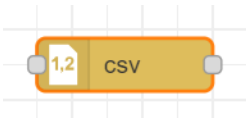

Abbildung 4: CSV Node in Node-Red

Eine genaue Beschreibung der CSV Struktur findet sich in RFC4180