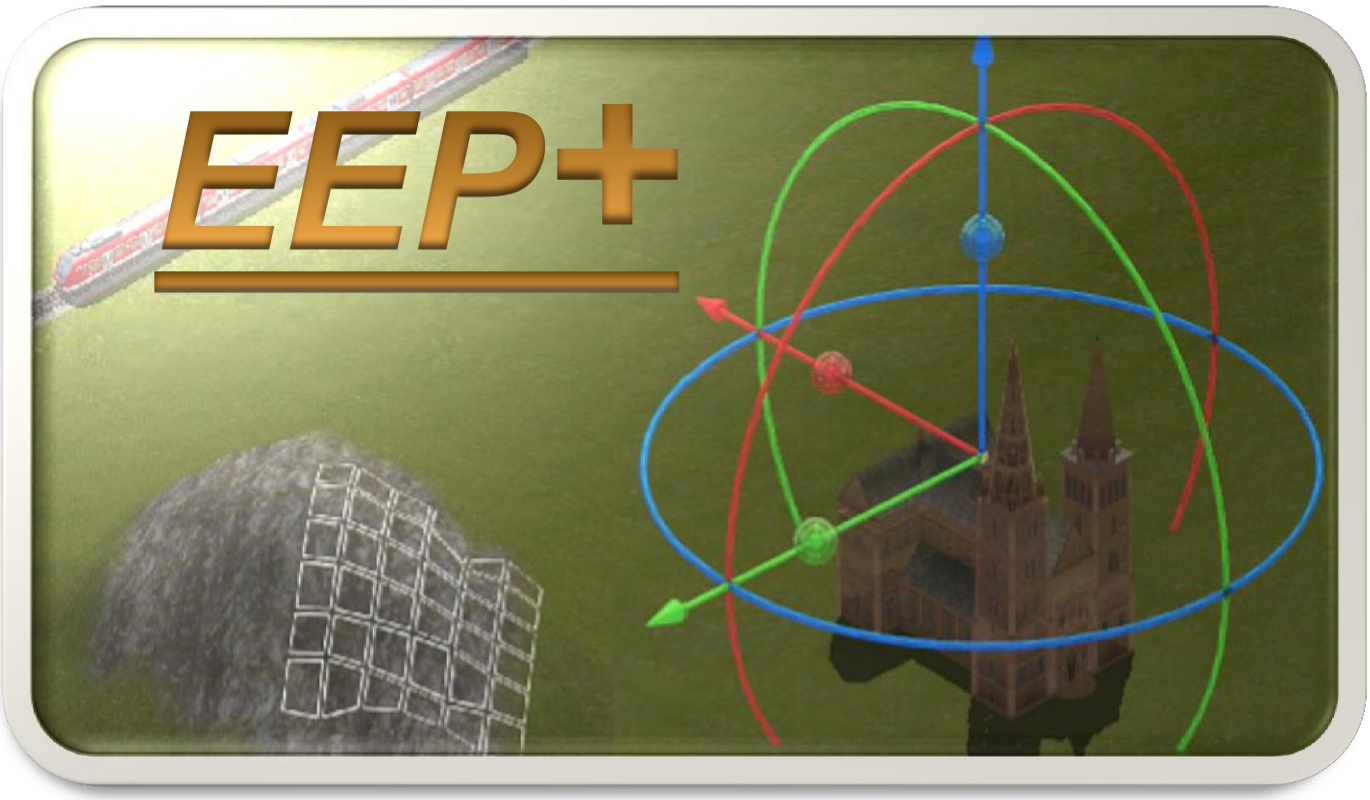

Kapitel:

Dateien speichern und laden mit LUA

Inhalt

Einführung

In diesem Kapitel möchte ich zwei LUA-Skripte vorstellen, welche es ermöglichen Daten in verschiedenen .txt-Dateien zu speichern und aus .txt-Dateien zu laden. Dies kann vielfach nützlich sein. So kann man z.B. Daten speichern und Laden, so dass diese beim Verlassen der Anlage nicht verloren gehen. Außerdem können bei einem automatischen Anlagenwechsel so Daten übergeben werden.

Anschließend werden noch einige hilfreiche Tipps zur Weiterverarbeitung gegeben. Da der Text aus den Dateien nur als Ganzes ausgelesen werden kann, muss dieser Gesamttext noch zerlegt werden.

Das LUA-Skript – Datei laden

```
datei = io.open("Anlagen\\Laden.txt","r")

ausgelesenerString = datei:read(10000)

datei:close()

print(ausgelesenerString)
```

Das LUA-Skript – Datei speichern

```
datei = io.open("Anlagen\\Speichern.txt","w")

MeinText = "hallo\nTest"

datei:write(MeinText)

datei:close()
```

Erklärung der Skripte

Beide Codeschnipsel sind innerhalb irgendeiner LUA-Funktion unterzubringen. Zunächst werden in der ersten Zeile der Speicherort der zu speichernden bzw. zu ladenden .txt-Datei und der Bearbeitungsmodus festgelegt. Dies geschieht durch Angabe des Dateipfades ausgehend vom Ressourcenordner Ihres EEPs. "Anlagen\\Laden.txt" bedeutet also, dass die Datei Resourcen\Anlagen\Laden.txt geladen wird. Beachten Sie die LUA-Schreibweise \\ statt \. “r“ definiert den Lese-Modus (r für read). “w“ definiert den Schreib-Modus (w für write).

Wenn Sie nun eine Datei auslesen, dann geschieht dies mit datei:read. Die 10000 ist die maximale Zeichenanzahl die ausgelesen wird und kann beliebig erhöht werden. Der gesamte ausgelesene Text wird in einem String „ausgelesenerString“ zwischengespeichert. Ein String ist eine Zeichenkette aus Buchstaben, Zahlen und anderen Zeichen.

Ich werde Ihnen später zeigen, wie Sie diesen String in kleinere Strings zerlegen können, denn es ist ja durchaus sinnvoll, viele verschiedene Dinge (Signalstellungen, Texte, etc.) in einer Datei zu speichern. Diese möchte man dann beim Laden wieder einzeln zugänglich haben.

Mit datei:close() wird die Datei wieder geschlossen. Dies ist wichtig, damit dort keine Rechenkraft vergeudet wird. Den Inhalt der Datei haben Sie ja sowieso im String zwischengespeichert.

Mit print(ausgelesenerString) wird der String im EEP-Ereignisfenster ausgegeben. Dies ist zur Kontrolle schön. Diesen Befehl können Sie natürlich weglassen, wenn unerwünscht.

Beim Speichern einer Datei definieren Sie zunächst den String „MeinText“, den Sie später in der definierten Datei speichern wollen. Innerhalb der Anführungszeichen, wo jetzt „hallo\nTest“ steht, können Sie eine beliebige Zeichenfolge aus Buchstaben, Zahlen, etc. hinschreiben, die gespeichert werden sollen.

Mit datei:write(MeinText) wird der String „MeinText“ dann in die Datei geschrieben. Danach wird mit datei:close() die Datei gespeichert und geschlossen.

Nützliches zu Strings

Strings sind grundsätzlich immer unformatierte Zeichenketten. Manchmal ist es allerdings schön oder nützlich, z.B. Absätze einzufügen. Ein Absatz innerhalb eines Strings wird mit \n eingefügt. Der String

```
hallo\nTest
```

sieht also formatiert (z.B. im EEP-Ereignisfenster) so aus:

```
hallo

Test
```

Manchmal ist es sinnvoll zwei Strings zu einem neuen String zusammenzufügen. Dies geschieht durch zwei Punkte:

```
NeuerString=String1..String2
```

.

Zum Beispiel ergibt

```
“Neuer“..“String“
```

den neuen String

```
„NeuerString“
```

.

Wenn Sie z.B. eine Signalstellung im String speichern wollen, so muss integer (Ganzzahl) zunächst in String (allgemeine Zeichenkette) übersetzt werden:

```
MeinInteger = tonumber(MeinString)
```

Oder andersherum

```
MeinString = tostring(MeinInteger),
```

um eine Ganzzahl in einen String umzuwandeln.

Diese Befehle sollten genügen, um einen beliebigen String aufzubauen und diesen dann mit obigem Code zu speichern.

Wenn Sie einen String aus einer Datei laden, dann laden Sie quasi den gesamten Textblock aus der Datei. Wir wollen den Textblock zerlegen bzw. gezielt Informationen herausziehen. Eine nützliche Methode ist substring:

```
String = "Hello Lua user"

SubString = string.sub(String, 7, 9)
```

Ergebnis wäre: SubString = „Lua“. Es werden alle Zeichen zwischen dem siebten und neunten Zeichen (inklusive) als Substring gespeichert.

Eine andere Methode ist gmatch. Nehmen wir an Ihr String ist „a b c d“. Das könnten z.B. Signalstellungen sein, die mit einem Leerzeichen getrennt gespeichert wurden.

```
Table = {}
index = 1
for value in string.gmatch(String,"%w+") do
   Table [index] = value
   index = index + 1
end
```

Dieses Skript erzeugt ein Table, vergleichbar mit einem Vektor. Die einzelnen Signalstellungen können danach bequem abgefragt werden:

Table[1] ist der String „a“, Table[2] der String „b“, usw.

Das Leerzeichen dient hier als sogenannter Separator.