

## ALEXA PRESENTATION LANGUAGE

# Multimodale Skills für Alexa

Nach der Card und die standardisierten aber stark limitierten Templates, stellt Amazon nun die APL mit viel Freiheit zur grafischen Gestaltung bereit.

Während Google zur Erweiterung seiner Actions um grafische Elemente auf die bewährten Techniken HTML, CSS und JavaScript zurückgreift, hat Amazon mit der APL (Amazon Presentation Language) eine eigene Technik entwickelt. Diese basiert, wie die bisherige Alexa-Welt, auf JSON - sogar für die logischen Bedingungen und Steuerungscodes.

Bereits in der in der ersten Ankündigung zur APL wurde in Aussicht gestellt, dass auch HTML5 Einzug halten wird. Zwar erweitert Amazon beständig die APL und insbesondere den dafür bereitgestellten Web Editor, aber es verfestigt sich der Eindruck, dass sich hiermit ein eigener Standard etabliert, der später nicht einfach auf standardisierte Webtechnologien zurückzuführen sein wird.

Auf dem September-Event hat Amazon die APL nun aus dem Beta-Status entlassen. Für Spiele wurde eine Preview

der Alexa-Web-API for Games angekündigt, mit der Spiele - basierend auf HTML und anderen Webtechnologien wie Canvas 2D, WebAudio, WebGL, JavaScript und CSS - auf einem Echo Show oder FireTV entwickelt werden können. Aber es wird dauern bis alle Entwickler darauf Zugriff bekommen. Insofern sollte sich jeder Alexa-Entwickler spätestens jetzt erst einmal mit der APL beschäftigen.

## APL Online-Editor

Für den einfachen Einstieg in die APL stellt Amazon einen grafischen Online-Editor bereit. Dieser kann zwar nur aus einer Skill-Konfiguration heraus aufgerufen werden (**Bild 1**), allerdings werden keine Informationen aus der bisherigen Skill-Konfiguration übernommen, noch übergibt der Editor Daten zurück. Man muss selbst den erzeugten Code aus dem

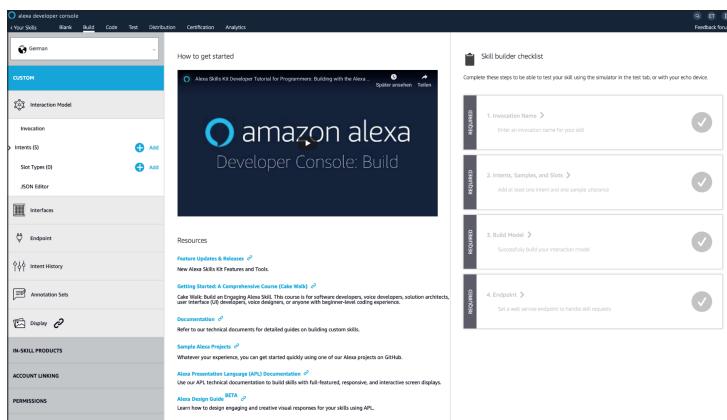
Editor exportieren und in seinem Skill implementieren. Der Klick auf den Menüeintrag öffnet ein neues Fenster (**Bild 2**) in dem sieben Vorlagen angeboten werden, welche die APL-Variante der bisherigen Template-Displays (siehe web & mobile DEVELOPER 5/2019) abbilden. Zusätzlich werden noch die Optionen *Start from Scratch* und *Upload Code* angeboten.

Der nachfolgende Screen enthält eine WYSIWYG-Ansicht. Allerdings funktionierte diese nicht immer zuverlässig auf einem Mac im Safari. Deshalb sollte auf den Firefox ausgewichen werden. Im Safari wird dann in der entsprechenden Spalte nur der Hinweis *Simulations may render differently on physical devices* angezeigt (**Bild 3**).

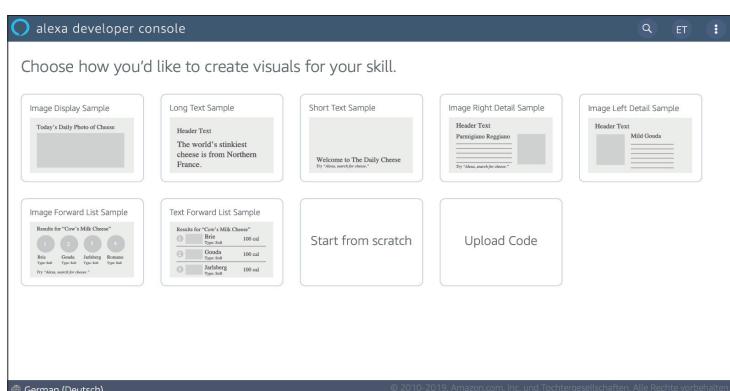
Unterhalb dieser Ansicht lassen sich die bisherigen Echo-Endgeräte als Simulationsobjekt auswählen. Über das +-Zeichen lässt sich ein weiterer Simulator hinzufügen, dessen Parameter *Shape* (rechteckig oder rund), Höhe und Weite in den Einheiten Pixel und die Pixeldichte konfigurierbar sind. Dieser zusätzliche Simulator steht dann unter der Bezeichnung *Custom* zur Verfügung und seine Parameter können auch wieder geändert werden.

## On Device Tests

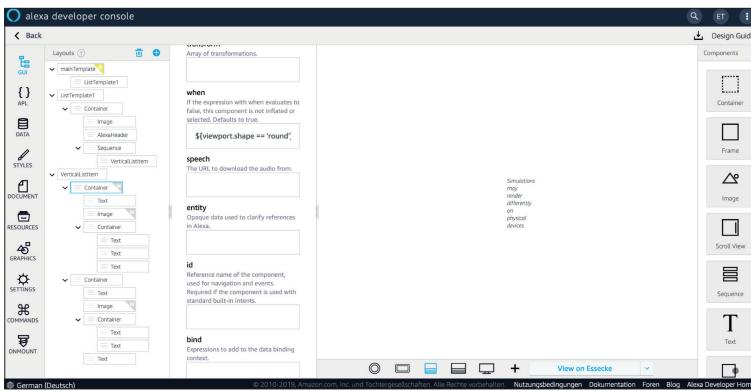
Die interessanteste Funktion verbirgt sich allerdings am Ende der Leiste. Hier verbirgt sich eine Dropdownliste aller Echos mit Bildschirm, welche mit dem Entwickler-Account angemeldet sind. In diesem Beispiel wurde mein Echo-Show mit dem Namen *Essecke* vorselektiert. Mit einem Klick auf diese Bezeichnung



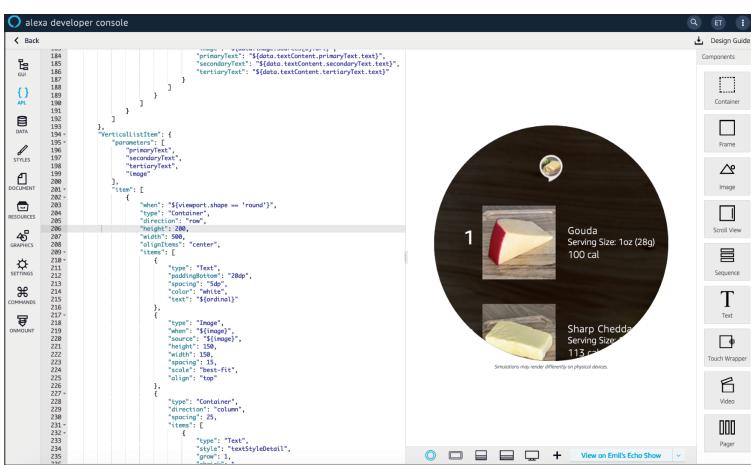
Die Alexa Developer Console mit dem neuen Menüpunkt Display (**Bild 1**)



Startauswahl des APL-Editors: Damit ist der Editor initial gefüllt (**Bild 2**)



**APL-Editor im GUI Modus:** Geöffnet im Safari Browser, der zurzeit keine Vorschau anzeigen kann (**Bild 3**)



**APL-Editor im Code Modus:** Geöffnet im Firefox Browser, inklusive WYSIWYG Vorschau (**Bild 4**)

wird der APL-Code direkt an das entsprechende Endgerät geschickt und dort angezeigt.

Sollte sich das Gerät im Nicht Stören-Modus befinden, muss noch der Bildschirm berührt werden, damit die Oberfläche angezeigt wird. Für diesen On Device-Test muss das Skill nicht gesondert konfiguriert werden. Wenn man von seinem Skill-Backend den APL-Code ausliefern lassen möchte, hingegen schon.

Wie bereits erwähnt kann der APL-Code aus dem Editor geladen werden. Hierzu muss das Download-Icon oben rechts in der Leiste genutzt werden. Auch im Editor kann man in eine Code-Ansicht wechseln, indem man in der linken Spalte von der Option *GUI* eine andere auswählt. Dann werden die nächsten beiden Spalten (in der ersten werden die Elemente in Ihrer hierarchischen Beziehung angezeigt und in der zweiten die Attribute des selektierten Elementes aufgelistet) zu einem großen Textfeld für den JSON-Code zusammengefasst (**Bild 4**).

Hierzu muss man allerdings wissen, dass jede der Optionen nur einen bestimmten JSON-Anteil des gesamten APL-Codes anzeigt. Insbesondere die Option *APL* ist hier missverständlich. Denn diese enthält neben dem eigentlichen Element innerhalb des *mainTemplate* auch die JSON-Anteile aller and-

## Listing 1: Rumpf APL

```

{
  "document": {
    "type": "APL",
    "version": "1.2",
    "settings": {},
    "theme": "dark",
    "import": [],
    "resources": [],
    "styles": {},
    "onMount": [],
    "graphics": {},
    "commands": {},
    "layouts": {},
    "mainTemplate": {
      "parameters": [
        "payload"
      ],
      "items": []
    },
    "datasources": {}
  }
}

```

ren Optionen - mit Ausnahme von *Data*. Im gesamten APL-Code fallen diese beiden Optionen unter die Elemente *datasources* (Option *Data*) und *document* (Option *APL*). Die Namensgleichheit des APL-Elements mit der Option *Document* ist verwirrend, da die Option nur die Attribute *type* (*APL*), *version* (der Editor erstellt zur Zeit noch Code mit der Version 1.1, dabei ist bereits die dritte Version mit der Bezeichnung 1.2 veröffentlicht worden) und *theme* (default *dark*) sowie das Array *import* mit den zu importierenden Zusatzinformationen, enthält, obwohl das Element *document* alle Informationen der Option *APL* beinhaltet ([Listing 1](#)).

## Live Edit

Unabhängig ob man nun in der Elementansicht (GUI) oder in einer der Code-Ansichten Änderungen vornimmt, werden diese sofort in der Simulator-Ansicht umgesetzt. Für eine Aktualisierung der Darstellung auf Endgeräten muss der zuvor erwähnte Button jeweils erneut gedrückt werden.

Als dritter Modus zum Editieren steht noch *drag & drop* zur Verfügung. Damit lassen sich APL-Komponenten von der rechten Spalte als Elemente in die Simulator-Ansicht ziehen und dort auch wieder verschieben. Die Komponenten lassen sich zwar nicht direkt in der GUI-Hierarchie-Ansicht ablegen, aber die dort vorhandenen Elemente sind innerhalb der Hierarchie-Ansicht dieser Spalte auch per Drag & Drop verschiebbar.

Das erste Element in der GUI Ansicht heißt *mainTemplate* und es fungiert als Wurzel für die anzuseigende Darstellung der Elemente, die in der Baumdarstellung darunter aufge- ►

spannt werden. Der Basis-Code (erzeugt mit *Start von Scratch*) sieht für dieses Element wie folgt aus:

```
"mainTemplate": {
  "parameters": [
    "payload"
  ],
  "items": []
}
```

Dabei entspricht der generierte Code nicht der Festlegung, denn der Plural vom Element *items* und der Umstand dass *items* als Array definiert ist, deuten darauf hin, dass hier mehrere Elemente angelegt werden könnten. Allerdings wird in der entsprechenden Definition der Dokumentation nur der Singular benutzt (an anderer Stelle verwendet aber auch die Dokumentation den Plural). Beim Hinzufügen einer Text-Komponente zum Beispiel per Drag & Drop, erzeugt der Editor automatisch erst eine Container-Element und legt dann in dessen *items*-Attribut das entsprechende Element an:

```
"mainTemplate": {
  "parameters": [
    "payload"
  ],
  "items": [
    {
      "type": "Container",
      "items": [
        {
          "type": "Text",
          "text": "Type in the text for your
layout...",
          "fontSize": "16dp",
          "paddingTop": "12dp",
          "paddingBottom": "12dp",
          "height": "32dp",
          "width": "300dp"
        }
      ],
      "height": "100%",
      "width": "100%"
    }
  ]
}
```

Man kann das Container-Element manuell entfernen und das *items*-Element (im Beispiel ein Text) direkt hinzufügen. Fügt man weitere Text-Elemente hinzu, wird nur das Erste verarbeitet und angezeigt. Der Editor funktioniert weiterhin einwandfrei, wenn man die Bezeichnung manuell von Plural auf Singular ändert.

## APL Element-Attribute

Im letzten Code-Beispiel sind neben den APL-Elementen auch ein paar Attribute gesetzt. Wie in anderen Spezifikationen (zum Beispiel HTML) gibt es dabei Attribute, die von je-

 Tabelle 1: Attribute

APL Version	Allgemeine Attribute
1.0	20 Zu den ursprünglichen Attributen gehören überwiegend Attribute zur Steuerung der Größe und Positionierung, sowie der Bindung an Daten und Style-Definitionen.
1.1	26 Die Bindung an Transformationen war das wichtigste neue Attribute.
1.2	32 Vier der sechs neuen Attribute beziehen sich auf den Schatten-Effekt.

der APL-Komponente, und welche die nur von bestimmten Komponenten unterstützt werden. Dabei sind alle Attribute optional. Wenn Attribute nicht explizit angegeben werden, wird der in der Dokumentation definierte jeweilige Default-Wert verwendet ([Tabelle 1](#)).

Die Größe eines Elements wird über die Attribute *width* und *height* definiert. Da diese Angabe auch relativ sein und durch andere Elemente beeinflusst werden kann, gibt es die Möglichkeit eine Mindestgröße (*minWidth* beziehungsweise *minHeight*) und eine Maximalgröße (*maxWidth* beziehungsweise *maxHeight*) mit anzugeben.

Vom Rand eines Elementes kann dessen Inhalt über die entsprechende Padding-Angabe (*paddingBottom*, *paddingLeft*, *paddingRight* und *paddingTop*) auf Abstand gehalten werden. Obwohl beide Angaben auf Maßeinheiten beruhen, hat Amazon hier unterschiedliche Datentypen vergeben. Beide können pixelbezogene Angaben enthalten - *px* für physikalische Pixel und *dp* für abstrakte Pixel (160 pro Inch).

Die Größenangaben haben den Typ *relative dimension* und können demnach auch Werte in % enthalten, welche sich auf die jeweilige passende Achse beziehen. Auch wenn bei der Padding-Angabe als *absolute dimension* die Prozentwerte nicht verwendet werden dürfen, enthält die APL noch zwei weitere Einheiten, die für beide Dimensions-Datentypen verwendet werden können, die sich auf die anteilige Bildschirmgröße beziehen: *vh* (für *viewHeight*) und *vw* (für *viewWidth*). So bezieht sich 50vh auf 50 Prozent der Höhe der View (des sichtbaren Bildschirms). Dabei ist eine solche Angabe nicht an die Achse gebunden. So kann auch die Breite eines Elements mittels einer *vh*-Angabe definiert werden.

Besonders hervorzuheben ist noch das *when*-Attribut. In diesem wird ein logischer Ausdruck hinterlegt, der beim Rendering der Anzeige ausgewertet wird. Nur wenn dieser Ausdruck wahr ist, wird das Element (und seine Unterelemente) angezeigt. In [Bild 4](#) ist der folgende Wert für dieses Attribut zu sehen:

```
 ${viewport.shape == 'round'}
```

In diesem beispielhaften Ausdruck wird vom globalen Objekt *viewport* das Attribut *shape* mit dem Wert *round* verglichen -

**Tabelle 2: Globale Objekte**

Name	Funktion
environment	Informationen über die aktuelle APL Engine wie zum Beispiel <code>agentName</code> und <code>agentVersion</code> aber auch <code>allowOpenURL</code> und <code>disallowVideo</code>
viewport	Informationen über den Bildschirm wie zum Beispiel <code>pixelWidth</code> und <code>pixelHeight</code> aber auch <code>video</code> welches die unterstützen Video-Codes enthält
math	Eine Sammlung von mathematischen Funktionen wie zum Beispiel <code>max</code> und <code>min</code> aber auch <code>random</code> und <code>sin</code> (für den Sinus) etc.
string	Eine (kleine) Sammlung zum Bearbeiten von Text: <code>toLowerCase</code> , <code>toUpperCase</code> und <code>slice</code>

sprich das Element und seine Unterelemente werden nur auf einem Echo Spot angezeigt. Die APL definiert die folgenden globalen Objekte (**Tabelle 2**).

Neben den globalen Objekten kann auch auf die Eigenschaft von Elementen, die im APL Code definiert werden, zugegriffen werden. Dies macht insbesondere mit dem noch folgenden Konzept der *datasource* Sinn.

## APL-Elemente

Neben den generischen Attributen hat jedes Element entsprechend seiner zugrundeliegenden APL-Komponente spezifische Attribute für die vorgesehene Funktion. Entsprechend der Anordnung im Editor werden die Komponenten in **Tabelle 3** kurz vorgestellt. Alternativ können auch mehrere Element direkt enthalten sein, die dann überlappt werden.

Im Editor nicht enthalten findet sich in der Dokumentation noch die Komponente *VectorGraphic*, die eine Grafik anzeigt, welche mittels Alexa-Vector-Graphic-Format definiert wird. Ein Element dieses Komponenten-Typs kann Parameter setzen, die innerhalb der AVG benutzt werden.

## Container

Wenn dem `mainTemplate` eine erste Komponente per Drag & Drop hinzugefügt wird, welche selbst keine weiteren Elemente enthalten kann (Text, Image oder Video), fügt der APL-Editor automatisch ein Container-Element vorher ein.

Denn der Container ist die Allzweckwaffe der APL, dessen Bedeutung nicht überschätzt werden kann. Er stellt diverse Attribute bereit, um zu definieren, wie die gruppierten Elemente angezeigt werden sollen:

- **direction:** Dieses Attribut kann die Werte `column` oder `row` annehmen und definiert, ob die Elemente untereinander oder nebeneinander angeordnet werden.
- **alignItems:** Definiert die Ausrichtung der Elemente auf der entgegengesetzten Achse zu `direction`. Hierfür können die Werte `stretch`, `center`, `start`, `end` und `baseline` verwendet werden.
- **justifyContent:** Entsprechend der Ausrichtung durch `direction` legt dieses Attribut fest, wie nicht genutzter Platz um die Elemente verteilt wird. Hierfür stehen die Werte `start`, `end`, `center`, `spaceBetween` und `spaceAround` zur Verfügung. Die-

**Tabelle 3: APL-Komponenten**

Name	Funktion
Container	Dient der Gruppierung weiterer Elemente.
Frame	Enthält nur ein Element und definiert für dieses das Aussehen, wie zum Beispiel die Hintergrundfarbe und den Rahmen.
Image	Dient der Anzeige eines Bildes.
Scroll View	Enthält wie der Frame nur ein Element, welches aber größer sein kann, als die Scroll View und dann innerhalb dieser sichtbar ist verschoben werden kann.
Sequence	Wie die Scroll View dient dieses Komponente zum verschieben. Das enthaltene Elemente an sich ist aber nicht zwangsläufig größer. Es wird aber über eine Verbindung zu Daten für jeden Datensatz erzeugt.
Text	Stellt einen Text über eine oder mehrere Zeilen dar.
Touch Wrapper	Enthält wie der Frame nur ein Element und macht dieses touchbar – sprich, wenn es angetippt wird, schickt wird dem Skill ein <code>touchEvent</code> mit den im Touch Wrapper definierten Werten geschickt.
Video	Dient der Anzeige eines Videos oder auch einer Serie von Videos
Pager	Vergleichbar mit der Sequence kann der Pager ein Element enthalten, dass über Daten vervielfältigt wird. Allerdings werden diese nicht nebeneinander angezeigt, sondern überlappend und nur eines ist sichtbar.

se Angabe bleibt aber von Bedeutung, auch wenn die Elemente mehr Platz brauchen, als zur Verfügung steht. Da der Container keine Scroll-Funktionalität beinhaltet, wird die Darstellung in diesem Fall einfach beschränkt. Auf Basis des konfigurierten Attributwertes wird dann entschieden, wie die Ansicht beschnitten wird bzw. was als Fixpunkt bei der Platzierung der Elemente angesehen wird.

Abschließend sei noch erwähnt, dass der Container die Attribute für das Verarbeiten von Daten bereitstellt. In diesem Fall wird aber nur das erste Kind-Element genutzt und entsprechend der Anzahl der Datensätze dupliziert. Vergleichbar mit einer Sequence, allerdings bleiben die Elemente fixiert (nicht scrollbar) und werden bei über-schreiten der Container-Größe abgeschnitten.

## Frame

Warum Amazon eine dedizierte Komponente zur Definition der Hintergrundfarbe und der Border eingeführt hat und dies nicht inhärenter Bestandteil aller Komponenten oder zumindest der Container-Komponente ist, bleibt ein Geheimnis. Ein Element dieser Komponente kann nur jeweils ein weiteres Element enthalten und bildet so nur eine Zwischenschicht, um die Attribute `backgroundColor` und `borderColor` zu definieren.

Diese Attribute setzen die Hintergrundfarbe innerhalb der Border des Elements beziehungsweise die Farbe der Border. Als Wert können alle HTML-Standardfarben (zum Beispiel ►

*red*) und die Hexadezimalschreibweise von RGB und RGBA (inklusive deren Kurzform) verwendet werden. Darüber hinaus können auch Funktionen zur Farbberechnung übergeben werden. Allerdings werden diese nicht als Databinding (\${}-Notation) sondern direkt übergeben: *borderColor* : *rgba(0,0,255,20%)*. Abschließend sei noch der Sonderwert *transparent* als Kurzform für *rgba(0,0,0,0)* erwähnt.

Per absoluter Dimensionsangabe kann die Breite der Umliegenden Begrenzung definiert werden. Nur wenn *borderWidth* größer 0 ist, hat die *borderColor* einen sichtbaren Effekt. Die Elemente sind Rechtecke, können über das Attribut *borderRadius* aber abgerundete Ecken erhalten. Die Konfiguration erfolgt über eine absolute Dimensionsangabe und kann alternativ zu diesem allgemeingültigen Attribut auch für jede Ecke einzeln gesetzt werden (*borderBottomLeftRadius*, *borderBottomRightRadius*, *borderTopLeftRadius* und *borderTopRightRadius*).

## Runde Ecken

In Listing 2 werden zwei Frames ineinander verschachtelt. Der äußere setzt die Hintergrundfarbe auf rot und der innere auf grün. Dabei hat der innere auch noch eine gelbe Umrandung (Border) und bis auf die rechte untere Ecke hat jede Ecke einen eigenen *BorderRadius*. Das Ergebnis ist in Bild 5 zu sehen. Da der Radius der rechten oberen Ecke der Höhe des Frames entspricht, stellt sich die Frage, was passiert, wenn die Radien der Ecken einer Kante zusammen die Kantenlänge übersteigen.

In diesem Fall bleibt die Länge der Kante unverändert. Die Radien an einer Kante, welche zusammen (prozentual zur Kante) am größten sind, definieren eine virtuelle Kante. Das prozentuale Verhältnis dieser virtuellen Kante zur eigentlichen Kante wird nun auf alle Radien angewandt.

Wenn man im Beispiel noch eine Ecke rechts unten mit ebenfalls 50vh definiert, dann sind die Radien an der rechten Kante zusammen 100vh lang, während die rechte Kante selbst nur 50vh lang ist. Damit ist die Kante 50% kleiner. Demnach werden alle Radien nun um 50% verkleinert. Das Ergebnis ist in Bild 6 zusehen.

Auf den Bildern ist ebenfalls zu erkennen, dass der Inhalt an dem vollständigem Ausmaß des Rechteck eines Frames ausgerichtet (zum Beispiel Zeilenumbruch bei Text) wird. Abgerundeten Ecken überdeckt dann einen dort platzierten Inhalt, sodass dieser nicht mehr sichtbar ist.



So sehen die runden Ecken des APL-Frame-Elementes aus (Bild 5)



Runde Ecken des APL-Frame-Elementes, wenn die Radien größer sind als die Kante (Bild 6)

## Listing 2: APL Frame und borderRadius

```
"items": [
  "type": "Frame",
  "width": "100%",
  "height": "100%",
  "item": [
    {
      "type": "Frame",
      "width": "50%",
      "height": "50%",
      "backgroundColor": "green",
      "borderColor": "yellow",
      "borderWidth": "20",
      "borderTopLeftRadius": "10vh",
      "borderTopRightRadius": "50vh",
      "borderBottomLeftRadius": "25vh",
      "item": [
        {
          "type": "Text",
          "fontSize": "16dp",
          "text": "Dies ist ein sehr langer Text, welcher über den Rand hinaus geht und dann umgebrochen wird."
        }
      ]
    },
    "backgroundColor": "red"
  ]
}
```

Ähnlich wie der Frame nimmt auch die *ScrollView* nur ein Element auf (auch wenn anstelle von *item* der Plural *items* valide ist und mehrere Elemente übergeben werden, wird nur das erste gültige Element angezeigt). Das einzige spezifische Attribut ist *onScroll*, welches eine Liste von sogenannten *commands* enthält, die ausgeführt werden, wenn gescrollt wird. Abschließend seien noch die Einschränkungen fürs Scrollen erwähnt.

Nur wenn die Höhe des Inhaltes der *ScrollView* die Höhe der View überschreitet, kann gescrollt werden. Da die Breite irrelevant ist, bedeutet dies im Umkehrschluss, dass nur vertikal und nicht horizontal gescrollt werden kann.

Die Sequence ist im einfachsten Fall eine erweiterte *ScrollView*, die mit dem Attribut *scrollDirection* von *vertical* auf *horizontal* umgestellt werden kann. Dann gilt natürlich, dass die Breite des Inhaltes die Breite der Sequence übersteigen muss, damit Scrollen aktiv wird.

Der eigentliche Mehrwert einer Sequence ist die Unterstützung des Data-Binding, welches vorab bei der *Container*-

Komponente angerissen wurde. Hierbei wird im Attribut *data* eine sich wiederholende JSON-Datenstruktur abgelegt. In den Attributen der Elemente kann auf diese Datenstruktur mit der *\$(/data.)-*-Notation zugegriffen werden. Sobald mehrere Array-Elemente in der Datenstruktur vorliegen, wird für jedes das Kind-Element der Sequenz erzeugt und entsprechend der *scrollDirection* unter beziehungsweise neben dem vorherigen angeordnet. Idealerweise ist das Kind-Element erst einmal ein Container, damit mehrere APL-Elemente pro Datensatz genutzt werden können. Um vor beziehungsweise nach den Daten auch noch Elemente anzeigen zu können, gibt es das Attribut *firstItem* (welches vor) und *lastItem* (welches nach) den Elementen für die Daten noch ein APL-Element darstellt. Auch hier wird der Einsatz eines Containers empfohlen, um ein Layout aus mehreren Elementen zu erzeugen. Diese können nicht auf die Daten (auch nicht allgemein zum Beispiel auf die Anzahl der Elemente) zugreifen.

Ein vereinfachtes Beispiel ohne Container, nur mit einem Text Element, ist in [Listing 3](#) dargestellt. Im Text wird mit der *\$(.)-*-Notation auf das Feld *Buchstabe* im Datensatz zugegriffen und mit *\$(ordinal)* die Nummer des Datensatzes angezeigt - wobei dieser bei 1 beginnt und nur vorhanden ist,

**Alphabet Anfang**  
Der 1. Buchstabe ist: A  
Der 2. Buchstabe ist: B  
Der 3. Buchstabe ist: C  
Der 4. Buchstabe ist: D  
Der 5. Buchstabe ist: E  
Der 6. Buchstabe ist: F  
Der 7. Buchstabe ist: G  
Der 8. Buchstabe ist: H  
Der 9. Buchstabe ist:  
Simulations may render differently on physical devices.

**Vereinfachtes Beispiel** ohne Container, nur mit einem Text Element ([Bild 7](#))

**Alphabet Anfang**



Simulations may render differently on physical devices.

**Ein APL-Pager** funktioniert ähnlich wie eine Sequence ([Bild 8](#))

wenn das Attribut *numbered* der Sequence auf *true* gesetzt wurde. Das Ergebnis ist in [Bild 7](#) zu sehen.

## Pager

Ein Pager funktioniert ähnlich wie eine Sequence, nur dass die generierten Elemente (inklusive des *firstItem* und *lastItem*) nicht gleichzeitig auf dem Bildschirm erscheinen. Vielmehr handelt es sich jeweils um eigene Bildschirmseiten, welche durch Wischen vor und zurück geblättert werden können. Wenn der Code aus [Listing 3](#) einfach von *Sequence* auf *Pager* geändert wird, so ändert sich die Anzeige entsprechend [Bild 8](#) und nur das *firstItem* wird angezeigt. Mit einem Wischen nach links erscheint dann das erste Item mit dem Text *Der . Buchstabe ist: A*. Die Nummerierung erscheint nicht, weil das Attribut *numbered* bei einem Pager nicht vorgesehen ist.

Dafür erhält er das Attribut *initialPage*. Dieses enthält einen Index auf das Element, mit dem die Anzeige nach dem Laden beginnt. Der Index fängt bei 0 an und bezeichnet entweder das erste Data-Element oder wenn vorhanden das *firstItem*.

Darüber hinaus wird mit dem weiteren Attribut *navigation* festgelegt, welches Verhalten beim Wischen angewendet wird:

- **none:** Der Nutzer kann nicht wischen, nur über Skripte wird zwischen den Seiten gewechselt.
- **forward-only:** Der Nutzer kann nach links wischen (Elemente kommen von rechts - mit dem nächst höheren Index), aber nicht zurück.
- **normal:** Der Nutzer kann per Wischen vor und zurück zwischen den Seiten wechseln.
- **wrap:** Wie bei normal, aber zusätzlich verhält der index sich wie ein Karussell, sodass nach der letzten Seiten wieder die erste Seite folgt und umgekehrt.

Wie bereits häufiger in den Beispielen verwendet, dienen die Text-Komponenten dem einfachen Anzeigen von Text. Dieser Text kann sowohl normale ASCII-Zeichen, als auch Unicode inklusive Emojis enthalten. Bei den Emojis ist zu beachten, dass deren Darstellung plattformspezifisch ist. Im Simulator werden die kleinen Bildchen im Stil des zugrundeliegenden Betriebssystems angezeigt, während ►

### Listing 3: APL Sequence

```
"items": [
  {
    "type": "Sequence",
    "height": "100%",
    "width": "100%",
    "numbered": true,
    "data": [
      {"Zeichen": "A"}, {"Zeichen": "B"}, {"Zeichen": "C"}, {"Zeichen": "D"}, {"Zeichen": "E"}, {"Zeichen": "F"}, {"Zeichen": "G"}, {"Zeichen": "H"}, {"Zeichen": "I"}, {"Zeichen": "J"}, {"Zeichen": "K"}, {"Zeichen": "L"}, {"Zeichen": "M"}, {"Zeichen": "N"}, {"Zeichen": "O"}, {"Zeichen": "P"}, {"Zeichen": "Q"}, {"Zeichen": "R"}],
    "firstItem": {
      "type": "Text",
      "text": "Alphabet Anfang"
    },
    "item": [
      {
        "type": "Text",
        "text": "Der ${ordinal}. Buchstabe ist: ${data.Zeichen}"
      }
    ],
    "lastItem": {
      "type": "Text",
      "text": "Alphabet Ende"
    }
  }
]
```

auf den Echo-Endgeräten ein eigener Stiel vorhanden ist ([Bild 9](#)). Auf dem Bild ist auch zu erkennen, dass selbst bei den klassischen Smiley-Emojis nicht alle auf der Echo-Plattform verfügbar sind. Zurzeit sind im Unicode-Standard 3019 Emojis definiert, von denen 410 nicht unterstützt werden. Dabei handelt es sich zum einen um die 2019 neu eingeführten Emojis zum Beispiel rund um körperliche Einschränkungen, als auch Pärchen von Menschen unterschiedlicher Hautfarbe. Da letztere als eine Kombination von Unicodes der beiden Partner und des Händchen-Halten-Symbols gebildet werden, werden auf einem Echo diese drei Bilder lediglich nacheinander angezeigt ([Bild 10](#)).

Besonders ärgerlich wird die fehlende Unterstützung bei Flaggen. Eine Zusammenfassung der nicht unterstützten Emojis ist in [Bild 11](#) zu sehen.

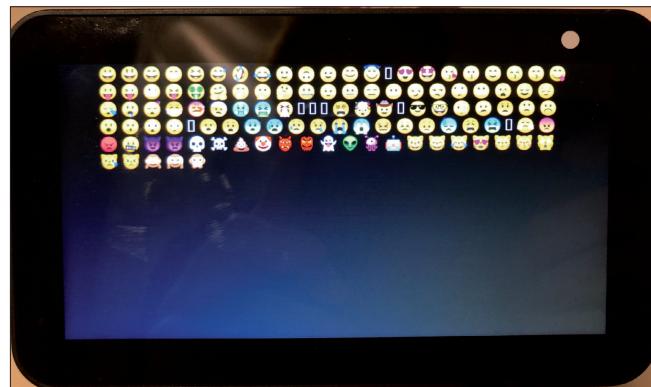
## Textformatierung

Während bei den Display-Templates noch explizit angegeben werden musste, ob der Text *PlainText* oder *RichText* ist, wird der Text nun immer als *RichText* behandelt. Zum Formatieren innerhalb des Textes funktioniert der Tag `<font size>` nicht mehr, sondern nur noch die Tags `<b>` `<u>` und `<i>` (fett, unterstrichen und kursiv). Auch die `<action>`- und `<image>`-Tags können nicht wie bisher genutzt werden, denn dafür sieht die APL den Touch-Wrapper beziehungsweise das Image vor. Dafür kann mit dem `<sup>`- und `<sub>`-Tag ein Textbereich hoch- bzw. tiefgestellt werden. Mit dem `<strike>`-Tag kann Text durchgestrichen und mit dem `<code>`-Tag als Monospace ausgegeben werden.

Zusätzlich zum Markup innerhalb des Textes, können an dem Element für den gesamten Text die folgenden Attribute gesetzt werden:

- `color`: Schriftfarbe.
- `fontFamily`: als String eine Auflistung der Schriftarten. Die erste in der Liste, welche auf dem Gerät verfügbar ist, wird dann verwendet.
- `fontSize`: Schriftgröße.
- `fontWeight`: ist *normal* oder *italic* (kursiv).
- `fontStyle`: neben *normal* und *bold* (fett) können auch Abstufungen gewählt werden, die aber nicht von allen Schriftarten unterstützt werden.
- `letterSpacing`: zusätzlicher Abstand zwischen den Buchstaben.
- `lineHeight`: Höhe der Textzeile, wird relativ zur `fontSize` angegeben. Wenn 100% unterschritten werden, kann dies zu Überlappungen der Textzeilen führen.
- `textAlign`: horizontale Ausrichtung anhand *left*, *center* und *right*. Zusätzlich kann mit dem Wert *auto* die Ausrichtung anhand der im Gerät genutzten Sprache (im Deutschen links).
- `textAlignVertical`: vertikale Ausrichtung wird nur genutzt, wenn das Element größer ist als der zu präsentierende Text.

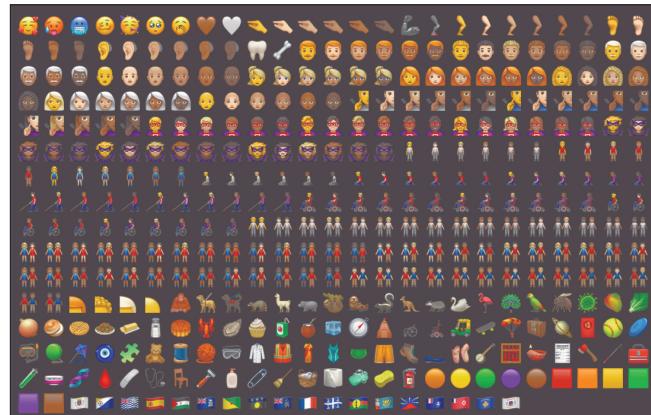
Wichtig für die Verarbeitung des Textes ist noch `maxLines`. Solange dieser Wert 0 ist, wird der Text am Zeilenende umgebrochen und eine neue Zeile gefüllt, bis die maximale Größe des Text-Elementes gefüllt ist.



Klassische Emojis, angezeigt auf einem Echo Show 5 ([Bild 9](#))



**Emojis:** Unterschiedliche Darstellung auf macOS und Echo Show 5 ([Bild 10](#))



**Keine Unterstützung:** Diese 410 Emojis werden nicht unterstützt ([Bild 11](#))

Die letzte sichtbare Zeile kann dabei sowohl horizontal als auch vertikal abgeschnitten werden. Wenn dieser Wert  $> 0$  gesetzt ist, werden maximal so viele Zeilen dargestellt wie angegeben und beim Abschneiden werden am Ende drei Punkte angezeigt. Dabei wird eventuell das letzte Wort in der Zeile aus Platzgründen entfernt.

## Image

Die Image-Komponente wird zur Anzeige eines Bildes verwendet, welches über eine URL im Attribut `source` referenziert wird.

Für die Ausrichtung des Bildes können zwei Attribute genutzt werden. In `align` wird die Ausrichtung an die Mitte (*center*) eine Kante (zum Beispiel *top*) oder eine Ecke (zum Beispiel *bottom-left*) vorgenommen. Mit dem Attribut `scale` wird

festgelegt, wie bei einem Größenunterschied zwischen dem Element und dem enthalten Bild verfahren wird. Mit dem Wert *fill* wird das Bild auf die gleichen Dimensionen gesetzt und damit auch das Attribut *align* ignoriert. Demgegenüber sorgt *best-fit* für ein Ausfüllen des Elements und *best-fit* dafür, dass das Bild vollständig im Element angezeigt wird, bei proportionaler Vergrößerung/Verkleinerung. Die dadurch entstehenden Bereiche, die zu groß (abgeschnitten) beziehungsweise zu klein (nicht gefüllt) sind, werden anhand des *align*-Attributs ermittelt. Mit *best-fit-down* gibt es eine *best-fit*-Variante, die nur verkleinert, um kein Qualitätsproblem bei einem eventuellen Vergrößern zu erzeugen.

Mit den Attributen *filters*, *overlayColor* und *overlayGradient* existieren Möglichkeiten, das Bild allgemein zu manipulieren, zum Beispiel einen Grauschleier darüber zu legen. Abschließend können mit dem Attribut *borderRadius* die Ecken gleichmäßig abgerundet werden.

## Vector Graphic

Als besondere Bild-Komponente hat Amazon mit der APL Version 1.1 die VectorGraphic eingefügt. Sie verfügt nur über die Image-Attribute *align*, *scale* und *source*. Da aber in der Source eine Vectorgraphic im Format AVG (Amazon Vector Graphic) referenziert wird, können in deren Code auch Eigenschaften definiert werden, die im APL-Dokument in dem Element der Vector-Graphik als Parameter gesetzt und damit übergeben werden.

Diese Parameter können auch als Styles übergeben werden und so Bedingungen enthalten beziehungsweise andere Einstellungen abfragen.

### Listing 4: APL UserEvent

```
{
  "version": "1.0",
  "session": {...},
  "context": {...},
  "request": {
    "type": "Alexa.Presentation.APL.UserEvent",
    "requestId": "amzn1.echo-api.request.788030fa-322e-4a5f-8aa2-bbf59791aeb7",
    "timestamp": "2019-11-21T18:55:35Z",
    "locale": "de-DE",
    "arguments": [
      "WasAuchImmer"
    ],
    "components": {},
    "source": {
      "type": "TouchWrapper",
      "handler": "Press",
      "id": ""
    },
    "token": "anydocument"
  }
}
```

Die Video-Komponente dient dem Definieren eines Bereiches, in welchem Videos angezeigt werden können. Diese Videos werden über das Attribut *source* referenziert. Wenn mehrere URLs enthalten sind, werden diese nacheinander abgespielt. Jede Quelle kann dabei zusätzlich mit folgenden Attributen weiter spezifiziert werden:

- *offset*: Der Zeitpunkt in Millisekunden ab dem das Video abgespielt wird (quasi vorgespult).
- *duration*: Die Länge in Millisekunden die das Video wiedergeben wird (damit kann die Spieldauer nur gekürzt und nicht künstlich verlängert werden).
- *repeatCount*: Wie oft das Video gezeigt wird. Bei einem *offset* und oder *duration*, wird dabei nur der definierte Abschnitt des Videos wiederholt.

Das Video-Element selbst kann für alle Videos mit dem Attribut *scale* festlegen ob diese per *best-fit* oder *best-fill* mittig im Element angezeigt werden. Eine Ausrichtung an Kanten oder Ecken ist bisher nicht möglich.

Mit dem Attribut *autoplay* kann festgelegt werden, dass das Abspielen der Videos ohne weiteres Zutun des Nutzers beziehungsweise durch Skripte startet. Zusätzlich können Kommandos zu diversen Events wie hinzugefügt werden.

## Touch Wrapper

Die letzte APL-Komponente ist auf dem Bildschirm nicht direkt sichtbar, sondern definiert Bereiche mit denen der Nutzer per Touch interagieren kann. Hierzu erhält ein Touch-Wrapper Element ein sichtbares *item* und *commands* für das *onPress*-Event:

```
{
  "type": "TouchWrapper",
  "item": {},
  "onPress": {
    "type": "SendEvent",
    "arguments": [
      "WasAuchImmer"
    ]
  }
}
```

Mit dem Command *SendEvent* wird dem Skill-Backend ein Request vom Typ *Alexa.Presentation.APL.UserEvent* geschickt, der die definierten Parametern enthält ([Listing 4](#)).

## APL-Interface

Mit den bisher vermittelten Informationen ist es nun möglich, eigene APL-Dokumente zu erzeugen. Bevor diese im Response zurückgeliefert werden können, gibt es allerdings noch einige Vorbereitungen zu erledigen.

Zuerst muss in der Alexa-Developer-Console die APL-Unterstützung des Skills aktiviert werden. Diese ist im Custom-Menü unter Interfaces zu finden. Mit dem Aktivieren des Schalters erscheint ein neues Menü zum Auswählen der unterstützten Bildschirmvarianten ([Bild 12](#)). Hub Round (Echo Spot), Hub Landscape (Echo Show) und TV Fullscreen (Fi- ►

re TV) sind dabei vorausgewählt und können auch nicht deaktiviert werden. Dies liegt daran, dass zu Beginn der APL-Entwicklung diese Auswahl nicht angeboten wurde und diese Varianten automatisch initialer Bestandteil waren. Zurzeit ist nur der Hub Landscape Small (Echo Show 5) selektierbar. Er ist aber nicht nur bei neu erstellten Skills nicht aktiviert, sondern auch bei alten Skills, die bereits die APL aktiviert hatten, bleibt diese Einstellung deaktiviert.

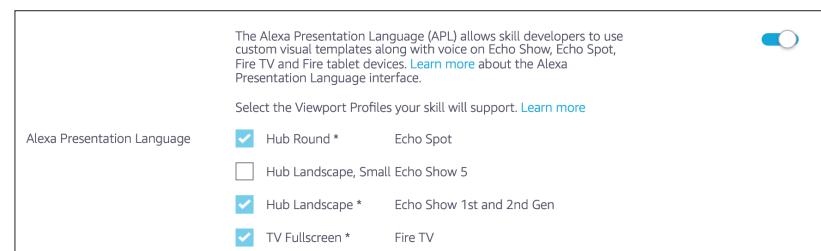
Wenn diese Einstellung deaktiviert bleibt, wird eine Anzeige zwar auf dem entsprechenden Echo Show 5 Modell gerendert, aber zu Grunde gelegt wird ein Hub Landscape-Bildschirm. Dies führt dazu, dass selbst eine 100%-Nutzung des ViewPorts nicht 100% des Bildschirms ausnutzt, sondern links und rechts schwarze Streifen zu sehen sind - wie bei einem 4:3 Film auf einem 16:9 Fernseher ([Bild 13](#)).

## APL-Directive

Nach dem Aktivieren des Interfaces kann der APL-Code in die Antwort des Skills als Directive eingefügt werden. Wie bei anderen Directives sollte aber vorher geprüft werden, ob das anfragende Endgerät die Unterstützung dieser Directive auch in seiner Anfrage zugesichert hat. Andernfalls wird das Endgerät auf die Rückmeldung mit einer nicht unterstützten Directive mit einem Abbruch reagieren.

In [Listing 5](#) ist ein minimaler PHP-Code dargestellt, welcher den Skill-Request in ein JSON-Objekt umwandelt. Dann beginnt die Ausgabe des Skill-Response, wobei die Direktive nur ausgegeben wird, wenn im Request-Objekt das Attribut zur maximal unterstützten APL-Version enthalten ist.

Innerhalb der Directive ist der grundsätzliche Aufbau des APL-Codes nur angedeutet. Hervorzuheben ist, dass die In-



**Die Custom-Interfaces-APL-Einstellung** in der Alexa Developer Console ([Bild 12](#))



**Rotes Rechteck auf Echo Show 5:** Ohne entsprechendes Interface und deshalb erfolgt die Anzeige mit einem schwarzen Rand links und rechts ([Bild 13](#))

halte der beiden Arrays `document` und `datasources` aus dem APL-Editor übernommen werden können. Wie eingangs erwähnt, ist dies aber nicht über die gleichlau-tenden Menü-Einträge auf der lin-ken Seite mög-lich. Am einfachsten lädt man den gesamten APL-Code mit dem Download-Icon rechts oben herunter. Dabei gilt es allerdings zu beachten, dass hier noch geschweifte Klammern um die beiden Arrays gesetzt sind, da-mit es sich um eine gültige JSON-

Datei handelt. Diese Klammern dürfen nicht in die Directive übernommen werden, da sonst deren Definition nicht einge-halten wird.

## Erweiterte Konzepte

Mit dem bisherigen Wissen können die eigenen Skills nun durch die APL erweitert werden. Aber gerade anspruchsvolle Layouts profitieren von weiteren Konzepten, die im We-sentlichen die Wiederverwendung von Code-Fragmenten oder eine semantische Gruppierung ermöglichen.

Bereits im [Listing 5](#) kam neben dem `document`-Array auch das `datasource`-Array vor. Dieses dient der freien Definition von Datenstrukturen. Im `document` kann auf die Datenstruk-

### Listing 5: APL Ausgabe im PHP Code

```
<?php
$postPlain = file_get_contents( 'php://input' );
$alexa_request = json_decode( $postPlain );
?>
{
    "version": "1.0",
    "response": {
        "outputSpeech": {
            "type": "PlainText",
            "text": "Mein erster APL Test."
        },
        <?php
        if (isset($post->context->System->device
->supportedInterfaces->['Alexa.Presentation.APL']
->runtime->maxVersion)) {
?>
        "directives": [
            {
                "type": "Alexa.Presentation.APL.RenderDocument",
                "token": "anydocument",
                "document": {...},
                "datasources": {...}
            },
            <?php } ?>
            "shouldEndSession": true
        },
        "sessionAttributes": {}
    }
}
```

tur mit dem Data-Binding zugegriffen werden – zum Beispiel per `$(payload.)`-Notation. Dazu muss allerdings im `mainTemplate`-Element der Begriff `payload` als Parameter definiert wurde:

```
"document": {...  
"mainTemplate": {  
  "parameters": [  
    "payload"  
  ],  
  ...  
  "text" : "${payload.MeineDaten.MeinText}"  
  ...  
}  
},  
"datasources": {  
  "MeineDaten" : {  
    "MeinText" : "Hallo Welt",  
    "MeineFarbe" : "yellow"  
  }  
}
```

In diesem Beispiel wird dem Text-Attribut eines APL-Elements im Dokument der Wert `Hallo Welt` zugewiesen, der im Array `datasource` abgelegt ist.

Auf diese Weise ist es möglich, sämtlichen Content vom Layout zu trennen. Denn auch ein Data-Attribut eines APL-Elements kann per Data-Binding auf ein Array innerhalb von `datasource` verweisen.

## Resources

So wie inhaltliche Daten sich in die Datasource auslagern lassen, so lassen sich Werte für Attribute als Konstante auslagern. Hierfür wird innerhalb von `document` das Array `resources` genutzt. Hierzu werden Gruppen innerhalb des `resources`-Array angelegt. Diese können eine Beschreibung (`description`) und eine `when` Klausel enthalten, wann die jeweilige Gruppe gültig ist. Damit die ausgelagerten Attribute in dem richtigen Datenformat interpretiert werden, müssen diese jeweils noch in eine Untergruppe für den jeweiligen Datentyp (`boolean`, `colors`, `dimensions` und `strings` wobei bei den letzten drei auch der Singular gültig ist, auch wenn dann der Editor eine Warnung ausgibt) gruppiert werden:

```
"resources": [  
  {  
    "description": "Allgemeine Werte",  
    "colors": {  
      "wichtig": "red",  
      "unwichtig": "white"  
    },  
    "dimensions": {  
      "TextGroesse": 48,  
    },  
    {  
      "description": "Werte fuer das Dark Theme",  
      "when": "${viewport.theme == 'dark'}",  
      "color": {  
        "wichtig": "green"  
      }  
    }  
]
```

```
"when": "${viewport.theme == 'dark'}",  
"color": {  
  "wichtig": "green"  
}  
}  
]  
]
```

In diesem Beispiel werden in der ersten Gruppe allgemeingültig Werte gesetzt. In der zweiten Gruppe wird nur der Wert der Farbe `wichtig` durch grün überschrieben, wenn das Gerät im Dark-Modus betrieben wird. Hierbei ist die Reihenfolge wichtig. Wären die beiden Gruppen vertauscht, würde die Farbe `wichtig` im Dark-Modus zwar eingangs auf grün gesetzt werden, durch die dann aber folgenden allgemeingültigen Gruppen jedoch wieder durch rot überschrieben werden.

## Data-Binding

Zwar ist in der `when`-Klausel das Data-Binding möglich, aber nur für die allgemeinen Datenobjekte. Auf die Daten des `datasource`-Elements kann hier nicht zugegriffen werden. In den Attributen der APL-Elemente werden dann die Ressourcen wie folgt verwendet:

```
...  
"color": "@wichtig"  
...
```

Neben der Definition von Attributen zur einzelnen Nutzung, kann man auch Gruppen bilden, die zusammen referenziert werden. Diese werden innerhalb von `document` durch das `array-styles` definiert. Im Gegensatz zu den Ressourcen werden diese Gruppen benannt, sodass alle Attribute einer Gruppe einem APL-Element gleichzeitig zugewiesen werden können. Darüber hinaus kann hier für die Werten beim Data-Binding auch auf Daten des `datasource`-Elements zugegriffen werden. Zusätzlich kann ein Style sich auf ein oder mehrere andere Beziehen, um dessen Definition zu erben. Diese können dann ergänzt oder auch ersetzt werden:

```
"styles": {  
  "normalerText": {  
    "description": "Grundeinstellung",  
    "values": [{  
      "fontSize" : "@TextGroesse",  
      "color": "black"  
    }, {  
      "when": "${viewport.theme == 'dark'}",  
      "color": "white"  
    }, {  
      "when": "${state.checked}",  
      "color": "blue"  
    }]  
  },  
  "wichtigerText" {  
    "extend": "normalerText",  
    "values" : {  
      "color" : "${payload.MeineDaten.MeineFarbe}"  
    }  
  }  
}
```

```

}
}
}
}
```

In diesem Beispiel werden zwei Styles definiert, wobei der Zweite (*wichtigerText*) durch den *extended*-Verweis auf den Ersten (*normalerText*), dessen Einstellungen erbt. Das Attribut *color* wird dabei mit einem Wert überschrieben, der aus den Daten des *datasource*-Elements bezogen wird. Hierzu gilt zu beachten, dass der Name *payload* vorher (wie eingangs beschrieben) selbst definiert wurde.

Im ersten Style werden gleich drei Attribut-Blöcke definiert. Der erste Block ist allgemeingültig und setzt für alle APL-Elemente, die diesen Style verwenden, die Schriftgröße und -farbe. Dabei wird in diesem Beispiel für die Schriftgröße direkt eine Ressource wiederverwendet, die wir vorher bereits definiert hatten.

Der zweite Block ist nur im Dark-Modus gültig und überschreibt dann die Schriftfarbe. Während der dritte Block gegebenenfalls erneut die Schriftfarbe überschreibt, wenn sich das betreffende Element im Zustand *checked* befindet. Verwendet wird ein Style, indem dessen Name einem APL-Element im Attribut *style* zugewiesen wird:

```
"style": "wichtigerText"
```

Das im letzten Block genutzte Data-Binding-Objekt *state* ermöglicht das Abfragen von verschiedenen Statusinformationen von einem APL-Element. Dieses Objekt lässt sich auch in der *when* Klausel eines APL-Elementes nutzen, sorgt dann aber nur dafür dass dieses Element entweder angezeigt wird, oder nicht. Innerhalb eines Styles können damit spezifische Attribute für das Erscheinungsbild anpassen werden.

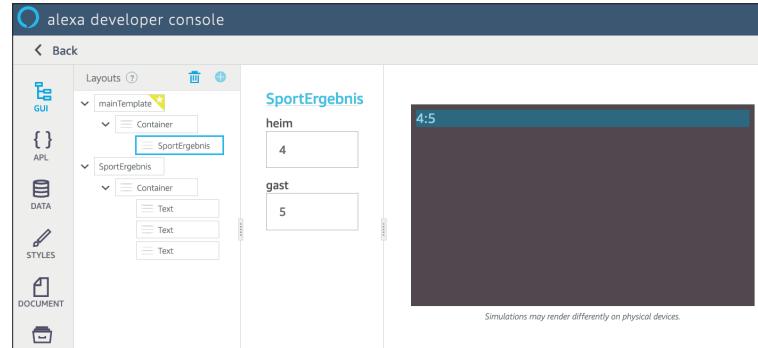
Allgemein können die Informationen *checked* und *disabled* (sperrt das Element zusätzlich für Interaktionen) abgefragt werden. Aber auch elementspezifische (*focused*, *karaoke*, *karaokeTarget* und *pressed*) beziehungsweise gerätespezifische seien an dieser Stelle noch erwähnt. Auf eine Vertiefung wird in diesem Artikel verzichtet, da die Eigenschaften ihre Stärken erst ausspielen, wenn man statische APL-Dokumente um Dynamik während der Ausführung erweitert, indem auch Skript-Logik integriert wird. Diese Thematik würde einen Einstiegsartikel aber bei weitem sprengen.

## Layouts

Im Rahmen eines strukturierten Aufbaus verbleiben als letzte Organisationstechnik die Layouts. In diesen werden Gruppen von APL-Elementen zusammengefasst. Damit bei einer Wiederverwendung der Gruppen nicht nur eins zu eins die gleichen Inhalte angezeigt werden, kann man Parameter definieren, welche per Data-Binding in der Gruppe verwendet werden können:

```

"layouts": {
  "SportErgebnis": {
    "parameters": [
```



**APL-Editor** mit eigenen Layout Elementen, deren Parameter im Editor angezeigt werden (Bild 14)

```

      "heim",
      "gast"
    ],
    "item": [
      "type": "Container",
      "direction": "row",
      "items": [
        {
          "type": "Text",
          "text": "${heim}"
        },
        {
          "type": "Text",
          "text": ":"},
        {
          "type": "Text",
          "text": "${gast}"
        }
      ]
    ]
  }
}
```

In diesem Beispiel werden drei Textfelder nebeneinander definiert, die ein Sport-Ergebnis 4:5 darstellen, wobei diesem Layout nur die beiden Werte als *heim* und *gast* Parameter übergeben werden:

```
{
  "type": "SportErgebnis",
  "heim": "4",
  "gast": "5"
}
```

Die Parameter-Logik ist uns bereits am *MainTemplate* begegnet, um auf die Datenstruktur von *datasource* zugreifen zu können. Wenn diese dort definiert ist, kann auch innerhalb der Layouts darauf zugegriffen werden. Für eine bessere Wiederverwendung sollten diese Daten aber jeweils als Parameter übergeben werden.

Die Gruppen erscheinen aber auch in der GUI-Hierarchie-Darstellung und die selbst definierten Parameter lassen sich auch direkt im Editor verändern (Bild 14). All diese Techniken zur Wiederverwendung entfalten ihre Wirkung erst, wenn sie übergreifend genutzt werden können. Zwar kann bei der dy-

namischen Generierung der Skill-Antwort der APL-Code aus verschiedenen Quellen zusammengesetzt werden, aber auch die APL selbst bietet einen Weg um Ressourcen, Styles und Layouts in eine externe Datei auszulagern und diese dann aus dem APL-Code heraus zu referenzieren. Diese Datei wird APL-Package genannt und nutzt ein JSON-Format basierend auf dem APL *document*-Element. Zwar kann in diesem auch ein *MainTemplate*-Element enthalten sein, dieses wird allerdings ignoriert und sollte zur Unterscheidung von vollwertigen APL-Dokumenten nicht enthalten sein.

Im eigentlichen APL-Dokument werden die Packages dann mit dem *import*-Statement im gleichnamigen Array referenziert:

```
"import": [
  {
    "name": "my-own-package",
    "source": "https://www.woauchimmer.de/general.json"
  },
  {
    "name": "alexa-layouts",
    "version": "1.1.0"
  }
]
```

In dem Beispiel sind die zwei Arten des *import*-Statements zu sehen. Im Ersten wird die JSON-Datei per eindeutiger URL referenziert. Im Zweiten wird nur der Name und die Versionsnummer eines Package angegeben, welches dann aus dem zentralen Amazon-Repository bezogen wird. Zurzeit stehen dort drei zur Verfügung, die zum Beispiel aus den Standard-Elementen einen vollumfänglichen Button erstellen.

Selbst das eingangs erwähnte lästige Erzeugen der Hintergrundfarbe durch das Zwischenschalten eines Frames ist Amazon damit angegangen. So ist in den Alexa-Layouts das



**Mighty Cards:** Ein Skill, das durch seinen Einsatz der APL beeindruckt (Bild 15)

Layout *AlexaBackground* definiert, welches man als erstes Element einem Container hinzufügt und schon kann in diesem eine Hintergrundfarbe festgelegt werden.

## Ausblick

Mit den vorgestellten Grundlagen lassen sich bereits anspruchsvolle Oberflächen gestalten. Wie teilweise angesprochen, gibt es darüber hinaus Möglichkeiten für die Erstellung interaktiver Oberflächen, die sich sogar während der Sprachausgabe von Alexa an das jeweils Gesprochene anpassen. Oder automatisierte Animationen bis hin zu vollwertigen Spielen, wofür Amazon gerade in internen Test die Alexa Web API for Games Developer testet.

Wer einmal ausprobieren möchte wie weit man mit der APL jetzt schon kommt, sollte seinen Account auf Englisch umstellen und das Skill Mighty Cards auf einem Echo Show starten. Neben einem gut animierten Intro werden die Karten beim Vergleich eingeflogen und animiert, wenn diese angesprochen werden (Bild 15).

## Fazit

Da die APL nun offiziell freigegeben ist, kann diese Technologie ohne weitere Bedenken in produktiven Skills eingesetzt werden. Der Mehrwert im Vergleich zu den eingeschränkten Display-Templates oder den ursprünglichen Cards (welche durchaus noch Ihre Berechtigung für die Aktivitäten-Übersicht in der Alexa-App hat), macht kleinere Unzulänglichkeiten der Editoren oder Endgeräte in der Verarbeitung mehr als wett. Durch die Ausweitung der Echo Show-Produktpalette mit immer neuen Bildschirmgrößen, wird die Erwartungshaltung an Skills zur Unterstützung der APL stetig steigen. ■

### Links zum Thema

- Introducing the Alexa Presentation Language (Preview)  
<https://developer.amazon.com/de/blogs/alexa/post/1dee3fa0-8c5f-4179-ab7a-74545ead24ce/introducing-the-alexa-presentation-language-preview>
- Der APL Editor (APL Authoring Tool)  
<https://developer.amazon.com/alexa/console/ask/displays>
- Komplette Emoji-Liste 2019 auf Deutsch samt Unicode Codepoints  
<https://emojiterra.com/de/codepoints/>
- Announcing the General Availability of Alexa Smart Screen SDK for TVs, Smart Displays, and More  
<https://developer.amazon.com/en-US/blogs/alexa/device-makers/2019/11/announcing-general-availability-of-alexa-smart-screen-sdk-for-tvs-smart-displays-and-more>



**Emil Thies**

ist in der IT Abteilung der Deutschen Telekom angestellt und probiert auch nebenberuflich neue Technologien aus, indem er eigene Apps, Skills und Actions programmiert und veröffentlicht.