ENISCO FORCAM
smart factory experts

FORCE EDGE CONNECT

Grafische Komposition

Einstieg in die Signal-Komposition

Version 230406

Handbuch

Dokument: Handbuch- Grafische Komposition: Einstieg in die Signal-Komposition

Freigabedatum: 06.04.2023

Dokumentversion: 1

Autor: FORCAM GmbH

FB_8-41 ERSTELLT AM: 18.11.22 GEÄNDERT AM: 13.04.23 VERSION: V1
FREIGEGEBEN DURCH: HEAD OF TTE AUTOR: TECHNICAL WRITER VERTRAULICHKEITSSTUFE: ÖFFENTLICH

Inhaltsverzeichnis

1 Über das Dokument

Dieses Dokument beschreibt die Verwendung des Editors zur Grafischen Komposition zur einfachen Interpretation von Assetsignalen.

ⓘ Aus Gründen der besseren Lesbarkeit wird im Text verallgemeinernd das generische Maskulinum verwendet. Diese Formulierungen umfassen jedoch gleichermaßen alle Geschlechter und sprechen alle gleichberechtigt an.

1.1 Zielgruppe

Dieses Handbuch setzt Kenntnisse im Umgang mit FORCE EDGE CONNECT (im Folgenden nur noch EDGE CONNECT) voraus. Sollten Sie dazu keine oder wenige Kenntnisse haben, nehmen Sie sich die Zeit, sich mit den Grundlagen vertraut zu machen.

Detaillierte Information finden Sie im **Handbuch –FORCE EDGE CONNECT.** (Mindestens ab Version 230406.)

ⓘ Wir empfehlen Ihnen die Nutzung unserer Academy: **https://forcam.com/academie/**
Die FORCAM Academy bietet das Wissen zum effektiven Einsatz der Methoden für die digitale Transformation und der Technologien für die Smart Factory.
Unser Institutsteam begleitet Sie auf Basis von Lean Manufacturing und TPM-Methoden, Veränderungen im Unternehmen einzuleiten und die Technologien richtig einzusetzen.

2 Konzept

Um Signale, die aus der Maschine, Sensoren usw. (kurz: Assets) ausgelesen werden, auszuwerten, müssen diese zuerst interpretiert werden. Mithilfe der Grafischen Komposition bestimmen Sie einfach und ohne Programmierkenntnisse, welche Signale wie verarbeitet werden. Sie können hier auch definieren, wann Daten an ein MES, ERP oder ein Dritt-System versendet werden.
Die bisherige Signalinterpretation in der EDGE CONNECT hat sich für viele Einsteiger als komplex und verwirrend herausgestellt. Die Grafische Komposition stellt eine einfache Alternative dazu dar. Das Ziel dieses grafischen Editors ist es, gängige Maschinenanweisungen durch die einsteigerfreundliche Lösung für jeden zugänglich zu machen. So wird das Senden und Interpretieren von Daten allen Nutzern ermöglicht.
In diesem Handbuch werden die unterschiedlichen Funktionen und Möglichkeiten nach ihren Themengebieten im Detail und an einem Praxisbeispiel erläutert.

2.1 FORCE EDGE CONNECT & Grafische Komposition

In den Maschinenparks finden sich viele Maschinen aus unterschiedlichen Jahrgängen und von verschiedenen Herstellern. Diese Vielfalt führt zu unterschiedlichen Signalformaten und Ausgaben der Maschinen, wodurch eine Auswertung dieser Daten sehr aufwändig werden kann. Doch die EDGE CONNECT bereitet die rohen Signale so auf, dass sie standardisiert und vergleichbar werden und damit eine optimale Grundlage für weitere Analysen bieten. Dafür ist die richtige Interpretation der Assetdaten essenziell. Zuständig ist dafür die Komponente Signal Composition. Mit den aus dem Soutbound Link erhaltenen Daten können hier die Signale interpretiert und aufbereitet werden. Auch wird hier festgelegt, wann die Daten versendet werden sollen. Diese Datenpakete, werden anschließend über den Northbound Link versendet. So können Dritt-Systeme durch die EDGE CONNECT mit Daten aus der Produktion versorgt werden.

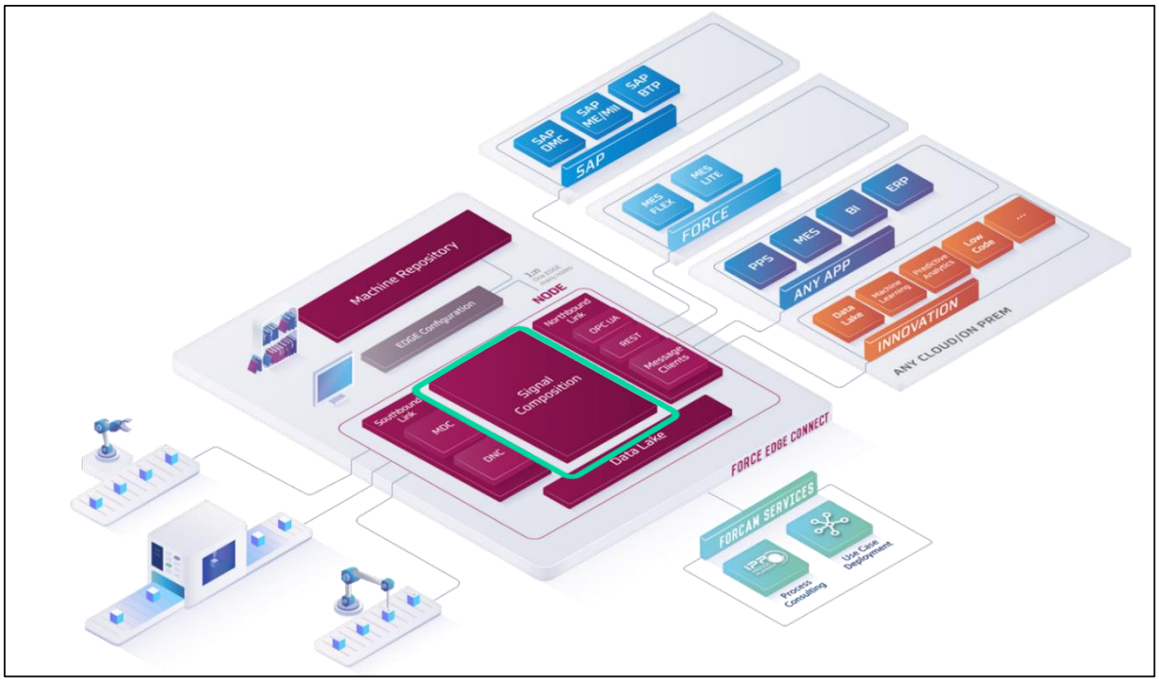


Bild 1: FORCE EDGE CONNECT Übersicht der Architektur

2.2 Kundennutzen der Grafischen Komposition

Der grundlegende Vorteil der Grafischen Komposition ist die einsteigerfreundliche Herangehensweise zum Interpretieren von Assetsignalen. Mithilfe von farbigen Puzzleteilen werden ausgewählte Befehle grafisch dargestellt. Somit können Programmiereinsteiger einfach und ohne Vorkenntnisse grundlegende Befehle ausführen. Eine zuvor erlernte Programmiersprache ist nicht erforderlich. Syntaxfehler können ausgeschlossen werden. Verschiedene Mechanismen erleichtern das Erkennen von anderen Fehlern. Eine übersichtliche Darstellung des Funktionsumfangs erleichtert den Umgang mit dem Editor. Auch die Kombination eines Templates wie beispielsweise dem Machine Repository mit der Grafischen Komposition ist möglich.

3 Vom Maschinensignal zum Event

Die Grafische Komposition ist eine Alternative zur klassischen skriptbasierten **Komposition** im EDGE CONNECT Asset Wizard. Ein Skript ist die kurze Abfolge von Anweisungen, die vom gewünschten Programm ausgeführt werden. In der Komposition werden Signalen Bedeutungen zugeordnet. So wird beispielsweise aus einem reinen Zahlenwert (wie 0 und 1) eine für Menschen lesbare Information, beispielsweise Produktion oder Stillstand. Für diese Zuordnung wird die Grafische Komposition wie ein Baukastensystem eingesetzt.

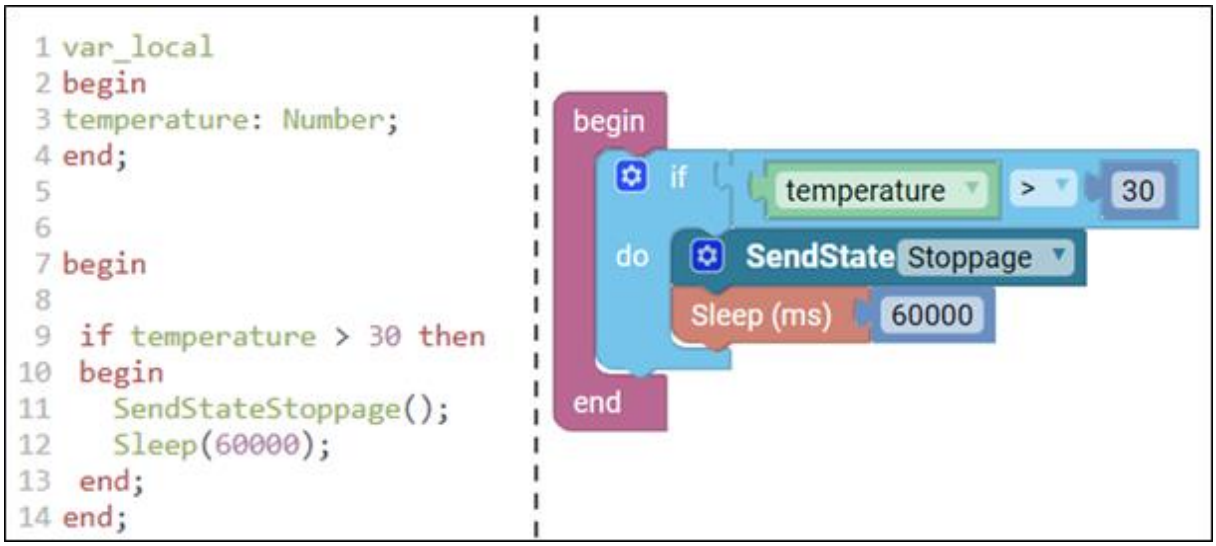


Bild 2: Skriptbasierter vs. grafischer Editor

3.1 Ablauf der Signal-Interpretation

Bild 3: Configuration Wizard

Es müssen mehrere Schritte durchlaufen werden, um ein Signal interpretieren zu können:

Asset anlegen

Zu Beginn müssen die benötigten Informationen zu einem Asset (Maschine oder Sensor) neu angelegt werden. Dies erfolgt in den Schritten 2 bis 5 im Configuration Wizard. Neben den automatisch angelegten Signalen können neue Signale hinzugefügt werden. Das geschieht in Schritt 5.

Kollektoren vorbereiten

In Schritt 6 **Komposition** werden zunächst Variablen angelegt. Diesen Variablen müssen Bedeutungen zugewiesen werden, damit Sie auswertbar werden.

Signale interpretieren

Signale werden nach festgelegten Bedingungen auswertet. Je nach Bedingung werden Events (Ereignisse oder Aktionen) ausgeführt.

Events senden

Die Signale werden in der Komposition mithilfe von Events (deutsch: Ereignissen) an Dritt-Systeme versendet.

Signale aufzeichnen

Alle Signale können dokumentiert werden.

3.2 Wichtige Grundinformationen

3.2.1 Generelle Erklärung der Benutzeroberfläche

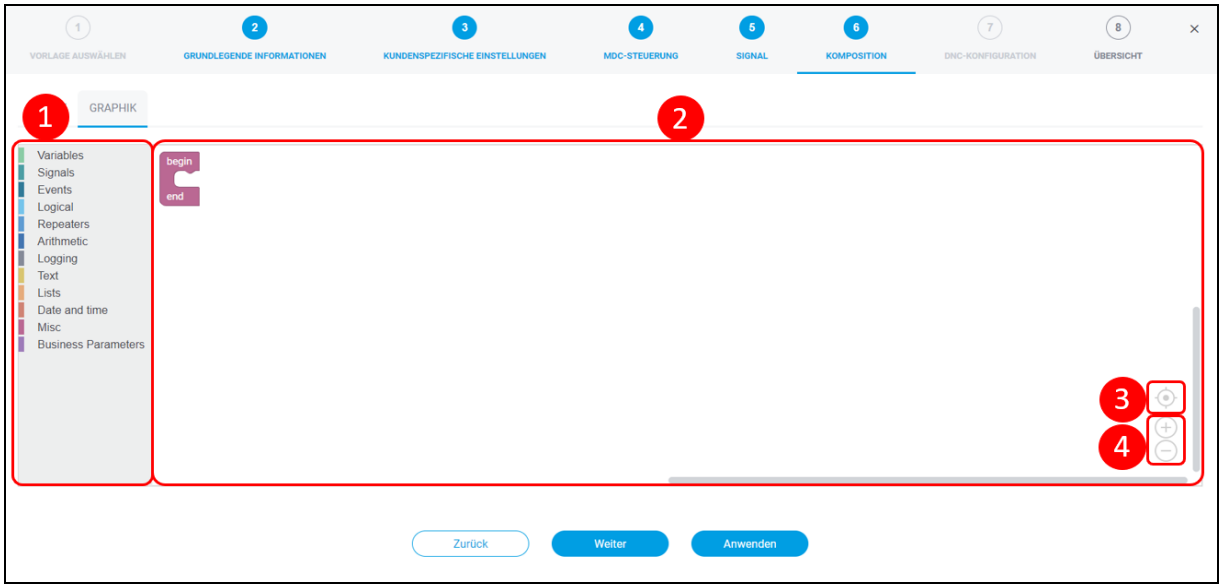


Bild 4: Grafischer Editor

Bedingungen für die Interpretation von Signalen werden im Bereich des Configuration Wizards in Schritt 6 **Komposition** geschrieben. Sie werden auf zwei Arten angezeigt: Zum einen im grafischen Editor (unter **GRAFIK**) als grafische Blöcke. Das sind die Programmier-Bausteine, die zusammengesetzt werden können. Zum anderen wird im Skripteditor (unter **SKRIPT**) der text-basierte Code angezeigt.

Das Bearbeitungsfeld unter **GRAFIK** (siehe Bild 4) ist folgendermaßen untergliedert:

(1) Auswahlfeld für Funktionskategorien und zur Wahl der Blöcke
(2) Feld zum Zusammensetzen der Blöcke
(3) Navigation: Zentrieren
(4) Navigation: Ansicht vergrößern/verkleinern

Einsteiger werden vor allem mit diesem Feld und den grafischen Blöcken arbeiten.

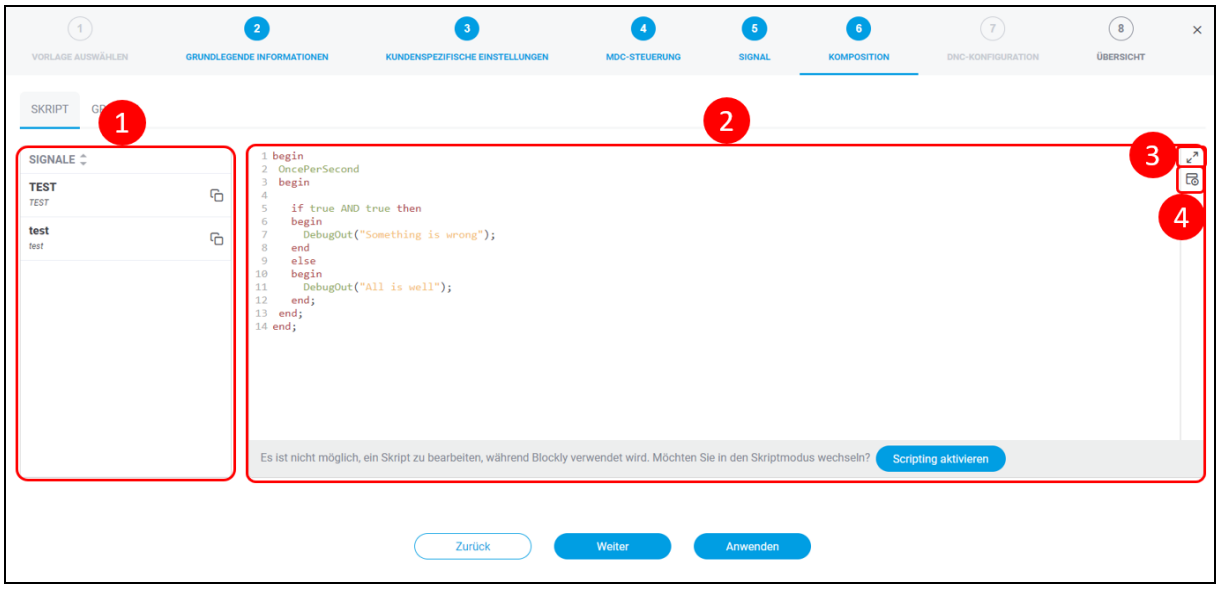


Bild 5: Skript-Editor

Unter dem Reiter **SKRIPT** können Fortgeschrittene den Programmiercode passend zu den Blöcken lesen. Die Anzeige wird automatisch synchronisiert.

⚠ Eine gleichzeitige Bearbeitung in **SKRIPT** und **GRAFIK** ist nicht möglich.
Wurde die GRAFIK zu einem SKRIPT konvertiert, kann man diese dort bearbeiten, aber nicht zurück in die Blockform setzten.

In **SKRIPT** sollte nur arbeiten, wer Vorkenntnisse im Programmieren hat.

Das linke Feld (1) zeigt die im Schritt 5 im Configuration Wizard hinzugefügten Signale an. Das rechte Feld (2) zeigt das aktuelle Skript. Um das Feld zu vergrößern, werden die Pfeile (3) verwendet. Um das Skript zu validieren, also auf Gültigkeit zu prüfen, dient das Icon bei (4).

3.2.2 Funktionskategorien

Bild 6: Kategorien der Blöcke

Die Funktionen der Komposition sind je nach Themengebiet in die Kategorien Variables, Signals, Events, Logical, Repeaters, Arithmetic, Logging, Text, Lists, Date and time, Misc (engl. Miscellaneous) und Business Parameters unterteilt. Jeder Kategorie wurde eine Farbe zugeordnet. Blöcke einer Kategorie besitzen immer dieselbe Farbe. So ist die Aufgabe der Blöcke einfach zu unterscheiden. Die Arbeitssprache ist Englisch. Im Folgenden werden die Kategorien kurz erläutert:

(1) Variables

Variables (deutsch: Variablen) sind wie ein „Eimer“ zum Speichern von Daten. Sie werden verwendet, um Dinge (normalerweise Berechnungen oder Eingaben) sicher aufzubewahren. Diesen Eimern können bestimmte Typen zugewiesen werden. Die Typen haben Beschränkungen, was ihren Inhalt (Zahlen, Wörter usw.) und ihre Größe (wie groß/klein usw.) betrifft. Sie enthalten entweder statische Werte (Assetnamen, Status usw.) oder die Variablen enthalten Berechnungen (Temperaturen, Druck, Zeiteinheiten usw.).
Ändert sich die Variable, wird die Veränderung im System registriert. Alle Blöcke dieser Variable übernehmen diesen neuen Wert.
Es existieren drei Arten von Variablen, die nicht immer miteinander kombinierbar sind. Daher im Folgenden die Unterscheidung:

- **Boolean:**
  Ein Boolean stellt einen Wahrheitswert dar. Ein Wahrheitswert sagt aus, ob ein Ereignis True (wahr/1) oder False (falsch/0) ist. Oder er gibt an, ob das Ereignis eingetreten (True/wahr/1) ist, oder nicht (False/falsch/0). Man nennt diese Werte auch boolesche Werte.
  Bsp.: Der Höchstwert einer Temperatur wurde erreicht oder eine Zeitdauer wurde überschritten.
- **String:**
  Ein String ist eine beliebig lange Zeichenkette. Zeichen können Zahlen, Buchstaben und verschiedene Symbole sein. Strings werden automatisch in Anführungszeichen gesetzt.
  Bsp.: Identifikationsnummer, Assetname
- **Number:**
  Zahlen werden hier Number genannt.
  Bsp.: Temperatur in °C, Anzahl an Minuten

(2) Signals
Signals (deutsch: Signale) sind Funktionen, die meist mit Sensoren gemessen werden und Informationen übertragen. Typische Signale im produzierenden Gewerbe sind Intervalle, Temperaturen, Maschinenzustände oder Druckmessungen.

(3) Events
Zum Senden von Impulsen, Ereignissen, Produktionszuständen oder Werten werden sogenannte Event-Blöcke verwendet.

(4) Logical
Diese Kategorie dient dazu, Werte in einen Zusammenhang zu bringen. Dadurch können Entscheidungen über ihren Wahrheitswert oder Status getroffen werden.
Nach einer zuvor festgelegten Regel wird entschieden, ob und welche Aktion folgen soll.
Z. B: Events, Lists, Date and Time, Business Parameters, Logging und viele mehr.

(5) Repeaters
Häufig werden Aktionen in regelmäßigen Abständen wiederholt. Repeaters (deutsch: Schleifen/Wiederholungen) führen diese in einem zuvor bestimmten Rhythmus aus.

(6) Arithmetic
Rechenfunktionen wie das Addieren, Subtrahieren oder Multiplizieren von Werten werden mit diesen Blöcken ausgeführt. Außerdem werden Strings in Zahlen (Numbers) umgewandelt.

(7) Logging
Um Werte für die Analyse zu nutzen, werden diese gezielt protokolliert und ausgegeben (debug out). Hierbei wird zwischen verschiedenen Warnstufen unterschieden.

(8) Text
Auch im Baukastenprinzip werden Wörter und Sätze benötigt, um die Werte verständlich zu machen. In dieser Kategorie können diese Texte erstellt und gezählt werden.

(9) **Lists**
Üblicherweise wird eine Liste zum Sammeln von verschiedenen Produktionszuständen genutzt. Hier wird bestimmt wie die Listen erstellt, gefüllt, geleert oder gelöscht werden.

(10) **Date and Time**
Um eine Aktion zu einem bestimmten Zeitpunkt ausführen zu können, muss die Uhrzeit und das Datum festgelegt werden. Auch die aktuelle Uhrzeit eines Ereignisses oder eine Pause werden gespeichert.

(11) **Misc**
Diese Kategorie ist eine Sammlung an weiteren Befehlen und stellt die Verbindung zu anderen Systemen her. Dazu gehören: Daten aus dem Internet integrieren, der Assetstatus abrufen, einen Asset als Offline definieren oder die IP-Adresse und den Host Namen ausgeben

(12) **Business Parameters**
Business Parameter sind Eigenschaften der Maschine. Zum Beispiel Description, Manufacturer, Model Number, Serial Number, Inventory Number und Location. Sie wurden in früheren Konfigurationsschritten angelegt.

3.2.3 Schreibweise der Zahlen

⚠ In der Grafischen Komposition wird die englische Schreibweise der Zahlen verwendet. Dabei ändert sich die Verwendung von Punkt und Komma. Zum besseren Verständnis sind Beispiele in Tabelle 1 aufgelistet.

Tabelle 1: Schreibweise der Zahlen

Deutsch	In Worten	Englisch
0,5	Ein halb	0.5
1.000	Eintausend	1,000
-1.750.000	Minus Einemillionensiebenhundertfünfzigtausend	-1,750,000

3.3 Generelle Handhabung

Jeder Baustein steht für sich und hat seine Aufgabe. Um dieses ausführen zu können, werden weitere Informationen von anderen Blöcken benötigt. Die Blöcke werden mit der Maus per Drag-and-drop verschoben. Zur einfachen Handhabung können die Blöcke mit den Tastaturkombinationen Strg+C und Strg+V kopiert und eingefügt oder über entf gelöscht werden.

Jeder Komposition beginnt und endet mit dem Rahmen **begin&end.** Dieser Block erscheint immer automatisch im Feld. Ansonsten besteht keine verpflichtende Reihenfolge der Bausteine.

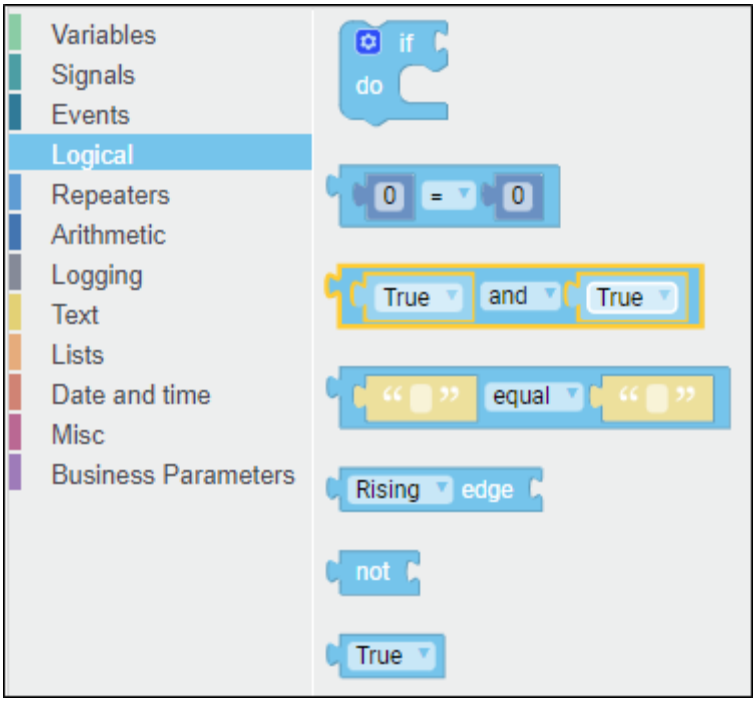


Bild 7: Auswahl der Funktionen

Beim Klick auf eine Funktionskategorie öffnet sich ein Fenster mit einer Auswahl an Funktionen.
Es gibt Blöcke, die als Klammern funktionieren. Sie benötigen weitere Blöcke, um vollständig zu sein. Außerdem gibt es Blöcke, die mit anderen Blöcken verbunden werden müssen. Sie enthalten Platzhalter für Variablen oder Eingaben.

Jeder Block hat bestimmte Vorgaben für den Input aus Variablen oder Eingaben. Es sind zum Teil nur Booleans, Strings oder Numbers als Input möglich.
Ein nicht passender Block kann nicht verbunden werden. Er springt weg und wird grau hinterlegt.
Eine Übersicht aller Vorgaben für Inputs und Outputs der einzelnen Blöcke befindet sich in Kapitel 17.1 „Übersicht der Parameter“.

Gelesen werden die Blöcke in der Reihenfolge von oben nach unten und von links nach rechts.
Die Inputs (deutsch: Eingaben) von Informationen in die Grafische Komposition sind die Inhalte, die das Programm zur Ausführung benötigt. Der Output (deutsch: Ausgabe) ist das Ergebnis oder die Befehlsausgabe.

Von oben nach unten

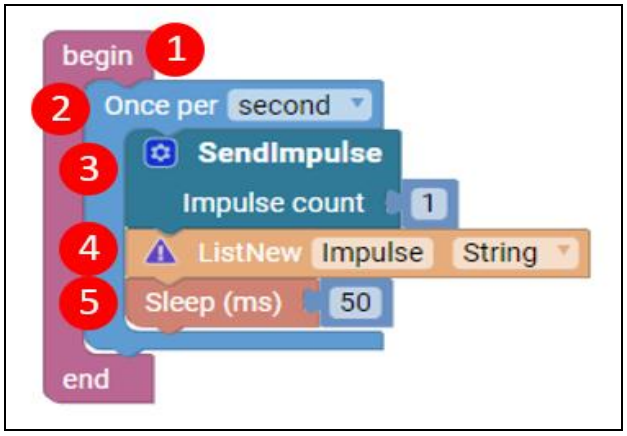


Bild 8: Beispiel für die Reihenfolge von oben nach unten

Der Block **begin/end** (1) bildet den äußeren Rahmen jeder Konstruktion. Zuerst führt das Programm den hellblauen Block **Once per second** (2) aus. Danach folgt der dunkelblaue Block **SendImpulse** (3). Anschließend geht es mit dem organgen Block **ListNew** (4) weiter. Zuletzt wird der roten Block **Sleep** (5) ausgeführt.

Von links nach rechts

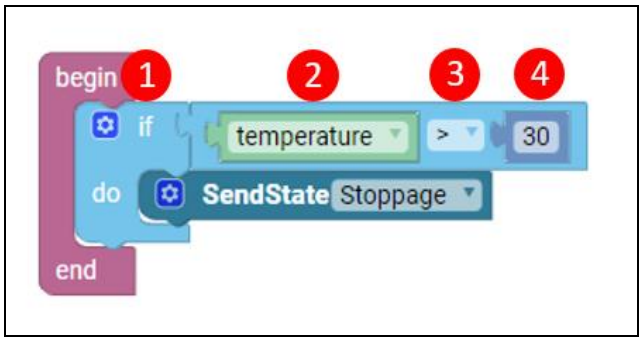


Bild 9: Beispiel für die Reihenfolge von links nach rechts

Wie in einem Buch werden die Zeilen in der Grafischen Komposition von links nach rechts gelesen. Es startet mit **if** (1), danach folgt der grüne Block **temperature** (2), dann das mathematische Zeichen **>** (3) und anschließend die Zahl **30** (4).
Danach springt man in die nächste Zeile. Diese fängt mit **do** an, danach wird innerhalb des dunkelblauen Block ebenfalls von links nach rechts gelesen. Folglich ist es irrelevant, dass ein Block zwei Zeilen einleitet, die Reihenfolge des Lesens bleibt dieselbe.

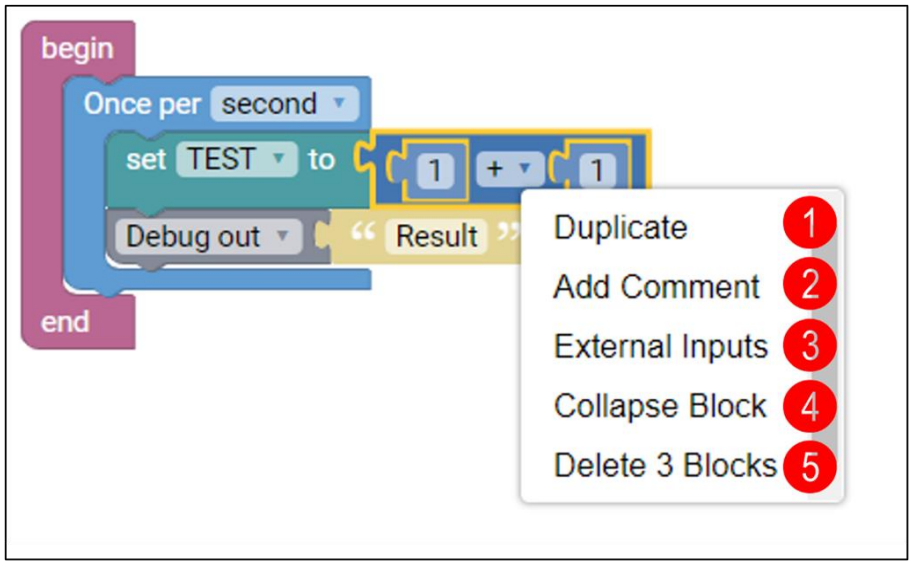


Bild 10: Möglichkeiten an Aktionen eines Blocks

Über einen Rechtsklick auf den Block sind weitere Aktionen möglich:

(1) Block duplizieren
(2) Kommentar hinzufügen
(3) Eternal Inputs oder Inline Inputs: Ändert die Darstellungsform
(4) Block einklappen
Um Platz zu sparen und das Gesamtbild übersichtlich zu halten, kann man zusammenhängende Blöcke einklappen.
(5) Blockgruppen löschen

3.3.1 Anatomie der Blöcke

Jeder Block steht für eine Aktion. Sie werden wie in einem Baukastenprinzip zusammengesetzt. Passende Bausteine können untereinander gestapelt werden und ergeben ein Gesamtbild. Das Gesamtbild in der Grafischen Komposition ist die Befehlsausgabe an ein Asset oder ein Dritt-System. Das generelle Aussehen und die Besonderheiten der Blöcke werden im Folgenden anhand von grafischen Beispielen erläutert.

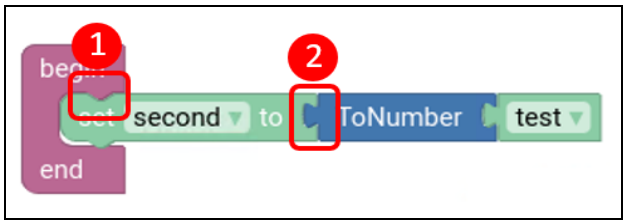


Bild 11: Verbindungspunkte

Jeder Baustein besitzt Verbindungpunkte zu anderen Blöcken. Ähnlich wie bei einem Puzzlespiel können nur die passenden Verbindungen miteinander kombiniert werden.

Zum einen gibt es Blöcke mit einen abgerundeten Pfeil nach unten (1). Diese Bausteine bilden Rahmenbedingungen und geben den Output an einen anderen Block weiter. Wenn an der rechten Seite eines Blocks ein Verbindungspunkt offen ist, wird Output weitergeben. Die Blöcke müsse an den rechten Seiten vollständig abgeschlossen sein.
Zum anderen gibt es die klassischen Puzzleelemente (2), die von links nach rechts aneinandergereiht werden. Sie geben die Eingabe der Daten weiter.

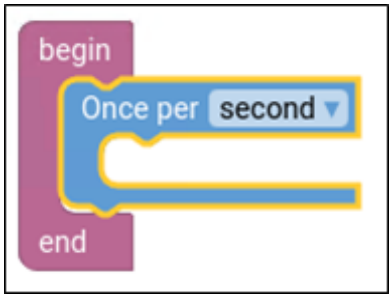


Bild 12: Gelbe Umrandung

Die gelbe Umrandung zeigt an, welcher Block gerade ausgewählt ist.

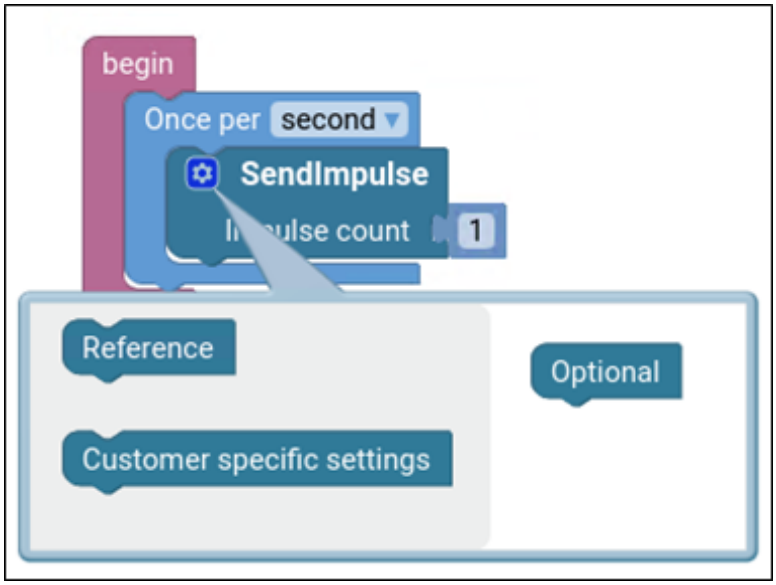


Bild 13: Beispiel für weitere Erweiterungsmöglichkeiten

An manchen Blöcken befindet sich ein blaues Einstellungszeichen. Bei einem Klick darauf öffnet sich ein Feld mit weiteren Blöcken. Diese sind Möglichkeiten zur Erweiterung. Dazu können die Blöcke von der linken grauen Seite zur rechten unter **`Optional`** gezogen werden. **`Customer specific settings`** dient zur Übertragung individueller Einstellungen, je nach Wunsch.

ⓘ Die unteren optionalen Blöcke können nur verwendet werden, wenn auch die oberen eingefügt wurden. Im Beispiel kann also **`Cusomter specific setting`** nur benutzt werden, wenn zuvor eine **`Reference`** geschrieben wurde.

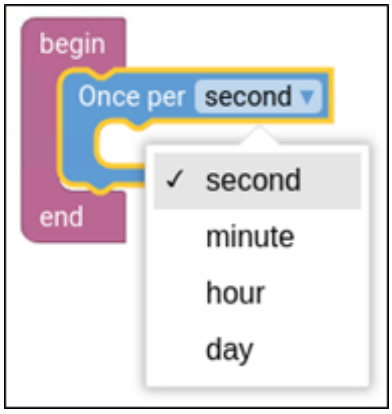


Bild 14: Drop-down-Menü

Einige Blöcke besitzen vorgegebene Input-Parameter. Bei einem Klick auf das kleine Dreieck am rechten Rand werden die verfügbaren Möglichkeiten angezeigt.

3.3.2 Schattenblöcke

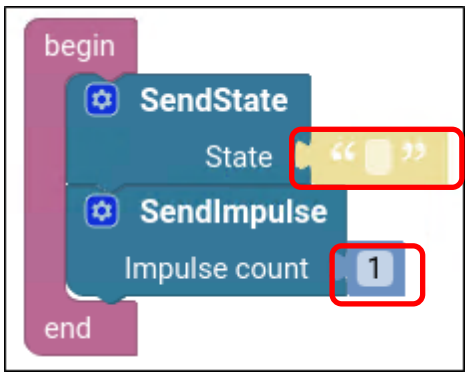


Bild 15: Schattenblöcke

Schattenblöcke dienen als Platzhalter für Inputs. Das bedeutet, dass eine Eingabe zwingend benötigt wird, damit der Block seinen Sinn erfüllen kann. Ein Schattenblock hängt immer schon an einem übergeordneten Funktionsblock, wenn dieser in den Funktionskategorien ausgewählt wird.

Ein Schattenblock ist gekennzeichnet durch eine hellere Farbgebung und zeigt an, dass dieser Parameter eines Blocks nicht leer sein darf. Entweder wird ein Wert manuell eingegeben, wie im gelben und blauen Feld in Bild 15, oder ein anderer Block hineingezogen.

3.4 Fehlererkennung

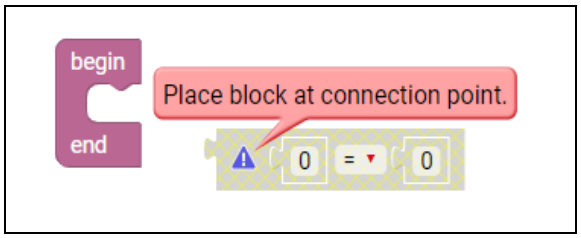


Bild 16: Hinweis: ungültiger Block

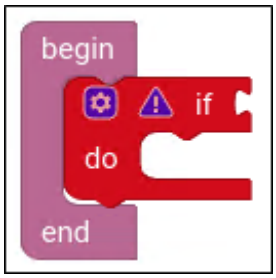


Bild 17: Hinweis: noch nicht vollständiger Block

Ein Fehler im Baukastensystem wird auf zwei Arten deutlich gemacht.

Zum einen nimmt das System einen Block nicht an, wie in Bild 16. Der Block wird grau hinterlegt. Das ist der Fall, wenn die Funktionskategorie oder die Art der Variable an dieser Stelle nicht passen. Ein typischer Fehler ist, dass ein String benötigt wird, jedoch Numbers einfügt wurden.

Zum anderen bleibt ein Block rot, wenn noch nicht alle notwendigen Informationen für den Block in der Komposition eingetragen sind. Wenn alle notwendigen Blöcke vorhanden sind, erhält er wieder seine ursprüngliche Farbe.

Nur ein korrekter Block wird zugelassen. Mit einem Klick auf das Ausrufezeichen wird der Grund für die Meldung angezeigt. Somit ist ein Fehler schnell erkennbar.

4 Variables

Variablen speichern Daten. Es gibt drei Arten von Variablen: Boolean, String oder Number. Je nach Variablentyp haben sie Einschränkungen bezüglich ihres Inhalts und ihrer Größe. Weitere Informationen hierzu sind in Kapitel 3.2.2 „Funktionskategorien“ zu finden.

Umgang mit Variablen

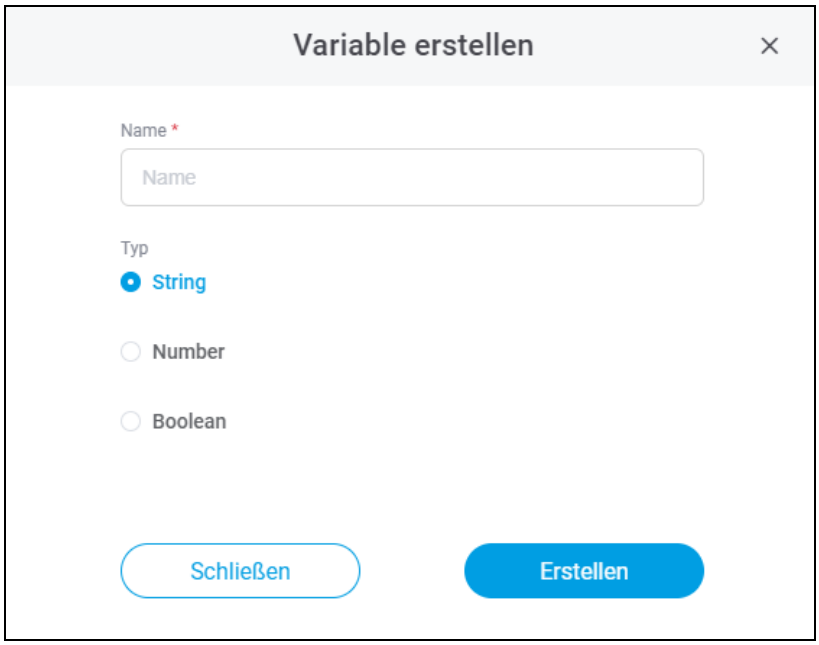


Bild 18: Neue Variable erstellen

Bevor man Variablen in der Grafischen Komposition verwenden kann, müssen die nötigen Variablen angelegt werden.
Klickt man innerhalb der Funktionskategorien die **Variabls** an, erscheint die Auswahl der Bausteine. Ganz oben im Feld gibt es die Opiton **Create new variable**. Hier wird der Name und die Art der Variable (Number, String oder Boolean) festgelegt.

Variablen können beliebig oft hinzugefügt werden.

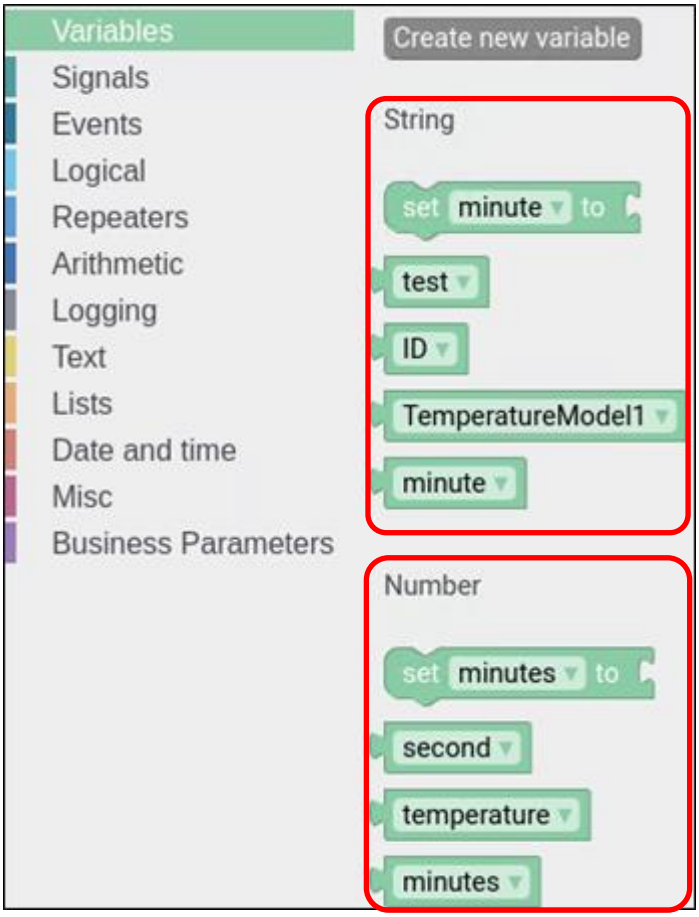


Bild 19: Übersicht der Variablenarten

Um die Übersicht zu wahren, werden sie nach der Erstellung unter den Unterüberschriften String, Number und Boolean angezeigt.

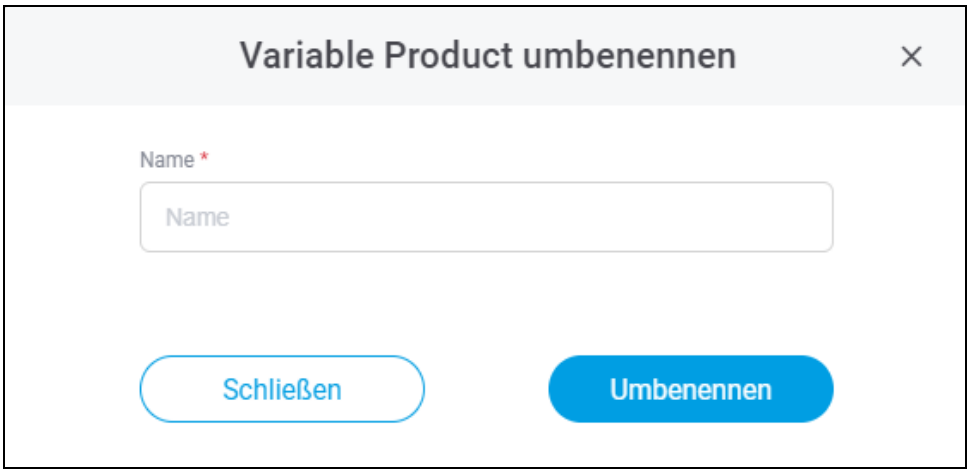


Bild 20: Variable umbenennen

Variablen können auch umbenannt werden. Dazu wird mit der linken Maustaste auf die Variable geklickt. Im Drop-down-Menü wird die Option **Rename** gewählt.

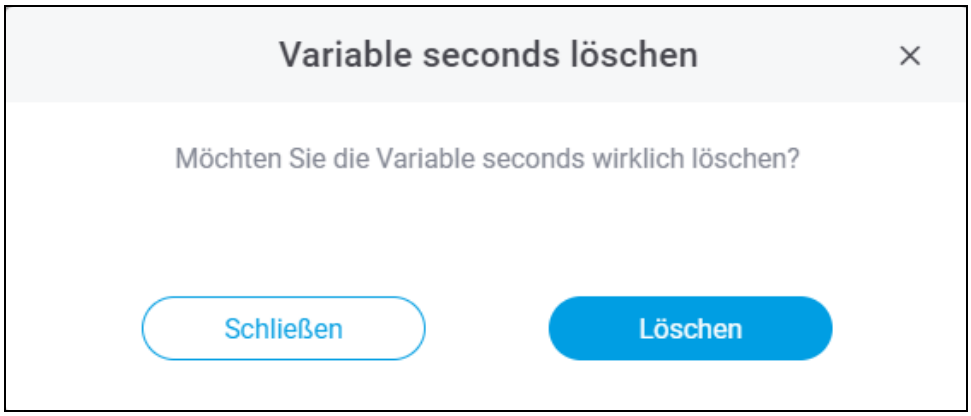


Bild 21: Löschen einer Variable

Im Drop-down-Menü kann auch die Option **Delete** gewählt werden, dadurch wird die Variable gelöscht.

⚠ Jede gelöschte Variable wird aus dem kompletten Baukastensystem gelöscht. Das gilt auch für verwendete Variablen.

4.1 Get [Variable]

Um eine Variable im Baukastensystem zu verwenden, wird dieser Block benötigt. Mit jedem Block können über das Drop-down-Menü alle erstellten Variablen der Arten Strings, Numbers und Boolean ausgewählt werden.
Einschränkungen für den Input gibt es nicht. Der Output entspricht der Art der erstellten Variable.

Beispiel:

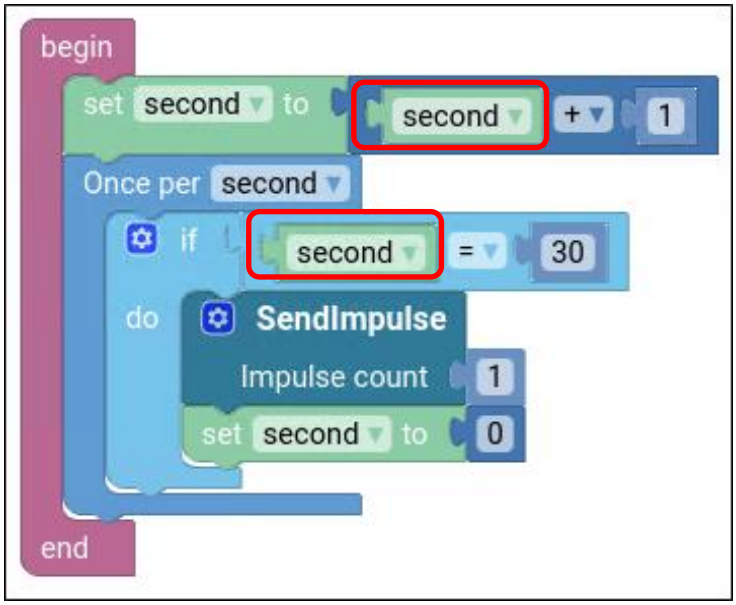


Bild 22: Beispiel zu Get [Variable]

Einmal pro Sekunde soll hier die Sekundenanzahl um 1 erhöht werden. Dazu wird die zuvor angelegte Variable **second** verwendet. Wenn die Variable **second** 30 erreicht, dann soll ein Impuls versendet und die **second** auf 0 zurückgesetzt werden.

4.2 Set [Variable] to

`Set [Variable] to` ist ein Bindeglied. Der Block wird benutzt, um einer Variable einen Wert zu geben. Abhängig vom Typ der Variable ist dies ein String, eine Number oder ein Boolean. Über das Drop-down-Menü kann ausgewählt werden, welche Variable verwendet werden soll.
Input und Output entsprechen der Art der erstellten Variable.

Beispiel

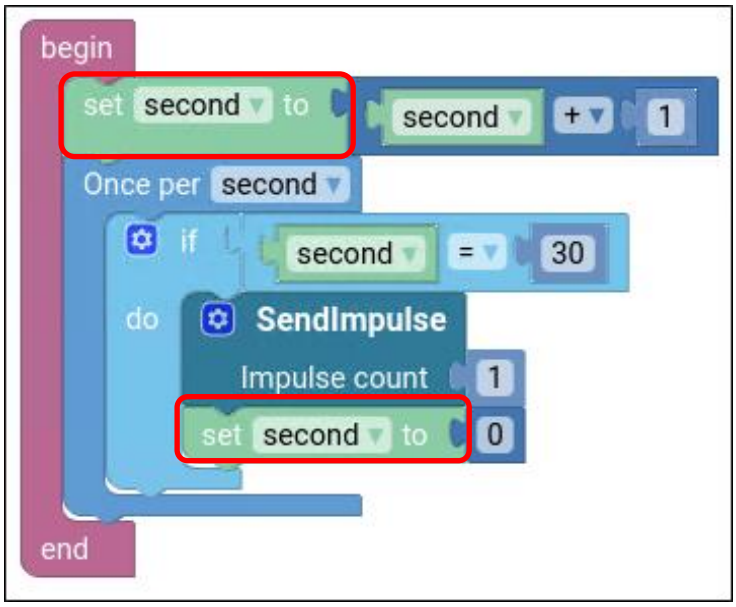


Bild 23: Beispiel zu set [Variable] to

Einmal pro Sekunde soll hier die Sekundenanzahl um 1 erhöht werden. Um diese Anweisung umzusetzen, wird mit dem Block **`set second to`** definiert, dass die Variable second neu berechnet werden soll. Wenn die Variable second 30 erreicht, dann soll ein Impuls versendet werden. Am Ende wird erneut die Sekundenanzahl auf 0 zurückgesetzt.

5 Signals

Signale werden meist mit Sensoren an den Assets gemessen und übertragen die gewonnenen Informationen an ein EDGE CONNECT.

Umgang mit Signalen

Die verwendeten Signale kommen im Unterschied zu Variablen von den Assets. Signale können im Schritt 5 konfiguriert und später im Skript verwendet werden.

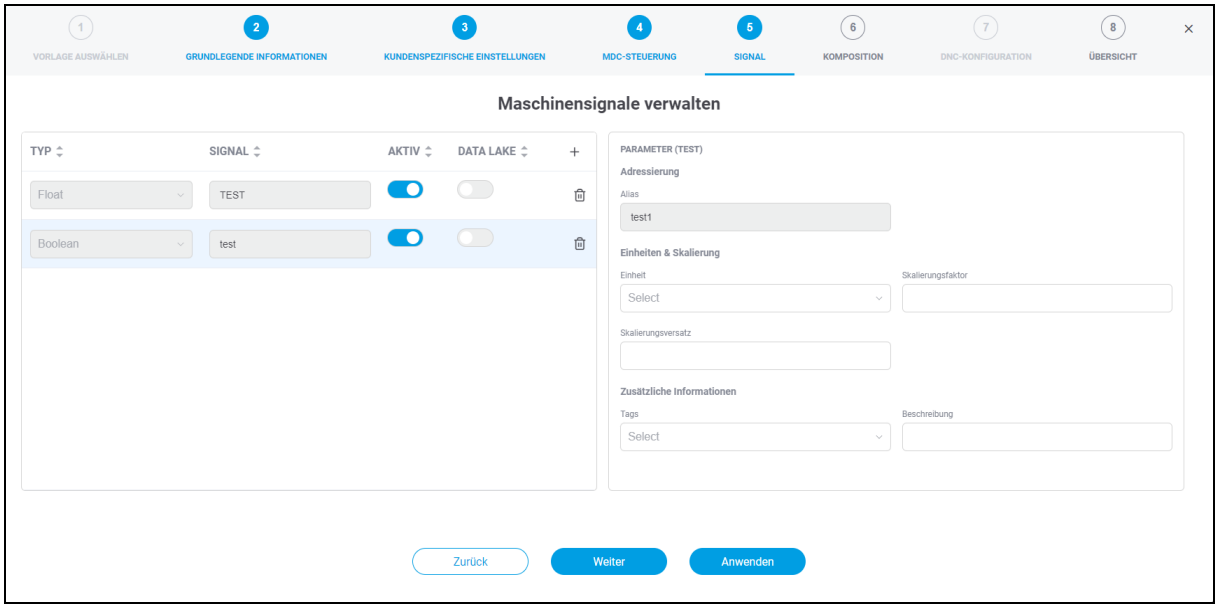


Bild 24: Hinzufügen eines neuen Signals

5.1 Set [Signal] to

`Set [Signal] to` ist ein Bindeglied. Der Block wird verwendet, um einem Signal eine Number, ein Boolean oder einen String zuzuordnen. Über das Drop-down-Menü kann gewählt werden, welches Signal verwendet werden soll.
Einschränkungen für Input und Output gibt es nicht.

Beispiel

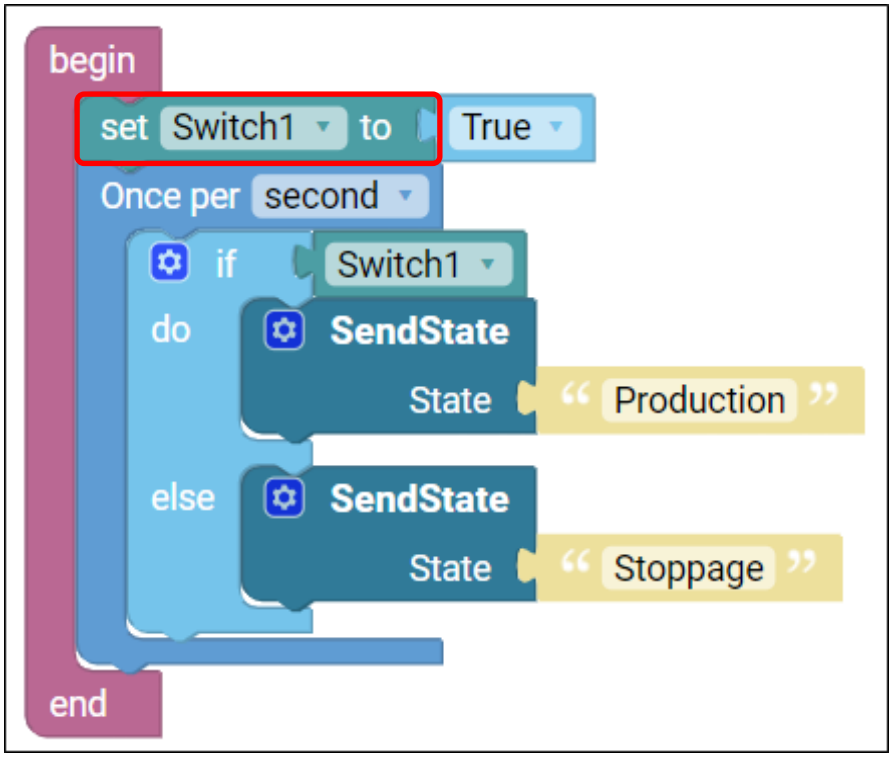


Bild 25: Beispiel für Set [Signal] to

Zu Beginn ist der **`Switch1`** (deutsch: Schalter) auf **`True`** (wahr/1) geschalte. Daher läuft ein Mal pro Sekunde ein Repeater ab. Wenn der **`Switch1`** umgelegt wurde, wird der Produktionsstatus auf **`Production`** gesetzt. Andernfalls wird der Status **`Stoppage`** ausgegeben.

5.2 Get Signal

Um Signale im Baukastensystem zu verwenden, wird dieser Block benötigt. Er liest den Wert des Signals ab. Über das Drop-down-Menü kann das gewünschte Signal ausgewählt werden. Einschränkungen für Input und Output gibt es nicht.

Beispiel

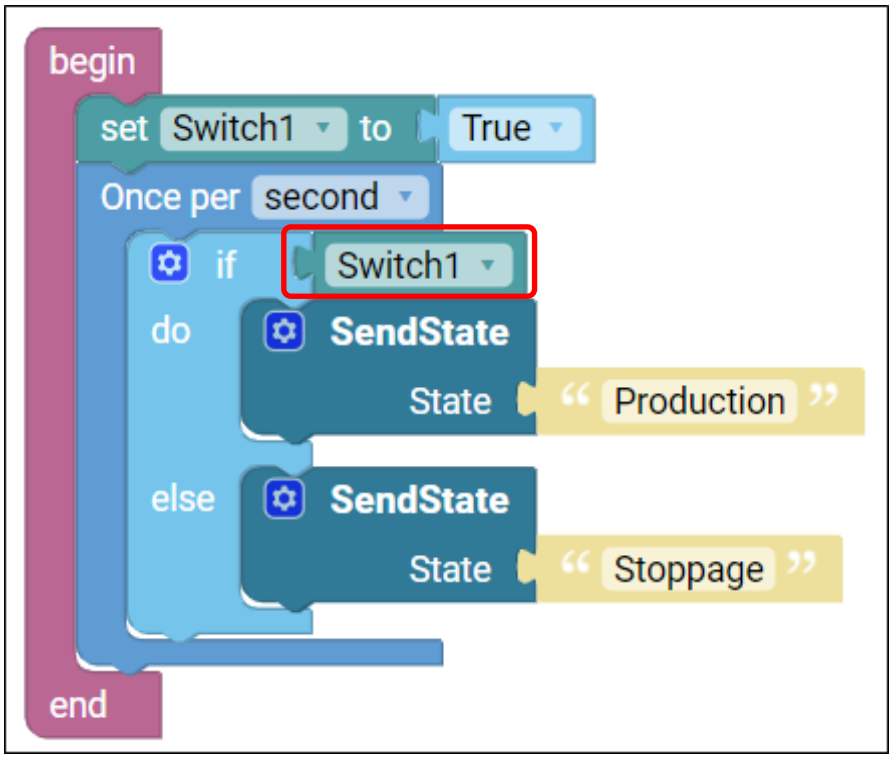


Bild 26: **Beispiel für Get Signal**

Wird der **`Switch1`** (deutsch: Schalter) auf **`True`** (wahr/1) gesetzt, läuft ein Mal pro Sekunde läuft ein Repeater ab. Dieser prüft, ob der **`Switch1`** umgelegt wurde. Wenn das so ist, wird der Produktionsstatus auf **`Production`** gesetzt. Andernfalls wird der Status **`Stoppage`** ausgegeben.

5.3 Get base / scaled value for

Der Block **`Get base Value`** wandelt den Wert eines Signals in eine andere Einheit um und gibt sie aus.
Der Block **`Get scaled value`** gibt den durch Skalierung und Offset umgerechneten Wert aus.

Im Configuration Wizard wurden unter Schritt 5 numerische Signale mit ihrer Einheit, dem Skalierungsfaktor und dem Skalierungsoffset eingegeben.

Der **`base value`** gibt den Wert in der vorgegebenen SI-Basiseinheit an.
Während der Signalkonfiguration werden Skalierungsfaktor und Skalierungsoffset für ein Signal festgelegt.
Zum Beispiel wird 0 °C bei einem Skalierungsfaktor und einem Skalierungsoffset von 0 in 273,15 °Kelvin ausgegeben.
Der **`sclaed value`** ist der Eingangswert multipliziert mit dem **`scaled factor`** (deutsch: Skalierungsfaktor) und dem **`scaled offset`**. Der **`scaled offset`** wird zum Ausgleich des Versatzes zum Realwert ausgerechnet.
Einschränkungen für Input und Output gibt es nicht.

Beispiel

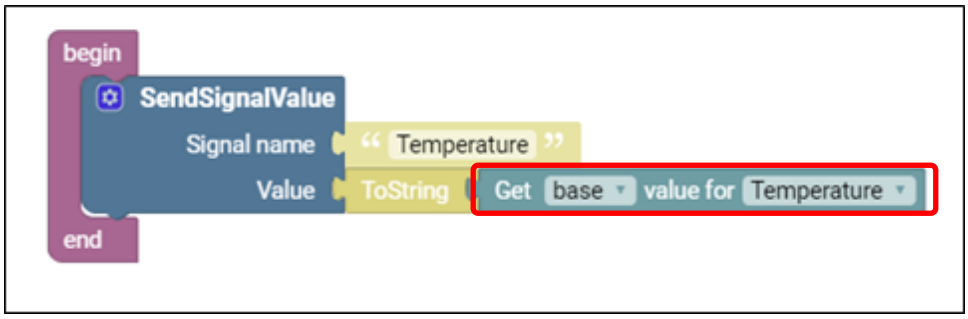


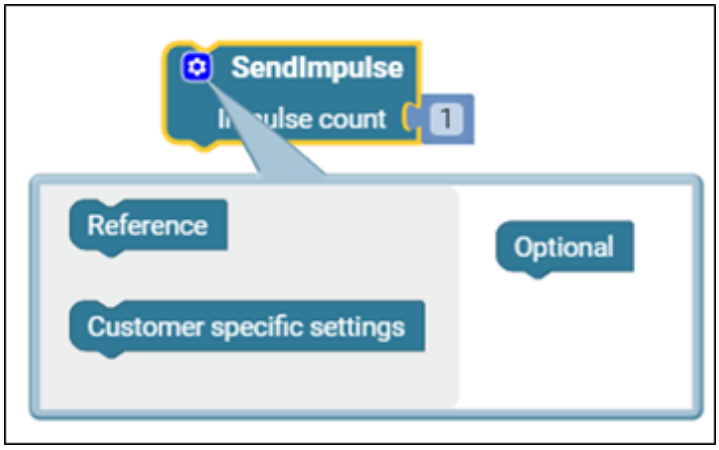
Bild 27: Beispiel für Get base value for

Der Temperaturwert wird in diesem Beispiel in eine andere Einheit umgewandelt und an ein Drittsystem weitergegeben. Der **`Signal name`** ist **`Temperature`** und wird in einen Text-Block eingegeben. Der Wert wird vom Block **`Get base value for Temperature`** in die Meldung eingefügt. Da das Event **`SendSignalValue`** nur Strings als Input zulässt, muss er dazu umgewandelt werden. Genauere Details folgen in den entsprechenden Kapiteln.

6 Events

Events senden Informationspakete an Drittsysteme. In der Grafischen Komposition wird bestimmt, welche Informationen in diesen Paketen enthalten sind.

6.1 SendImpulse



Wenn ein bestimmter Impuls gesendet werden soll, wird der Block **`SendImpuls`** eingesetzt. Die Zahl bei **`Impulse Count`** gibt an, wie viele Impulse gesendet werden sollen. Optional können weitere Blöcke (**`Reference`** und **`Customer specific settings`**) angegeben werden.
Der Input für **`Impulse count`** sind ausschließlich Numbers.
Input für alle weiteren Angaben sind ausschließlich Strings. Einschränkungen für den Output gibt es nicht.

Beispiel

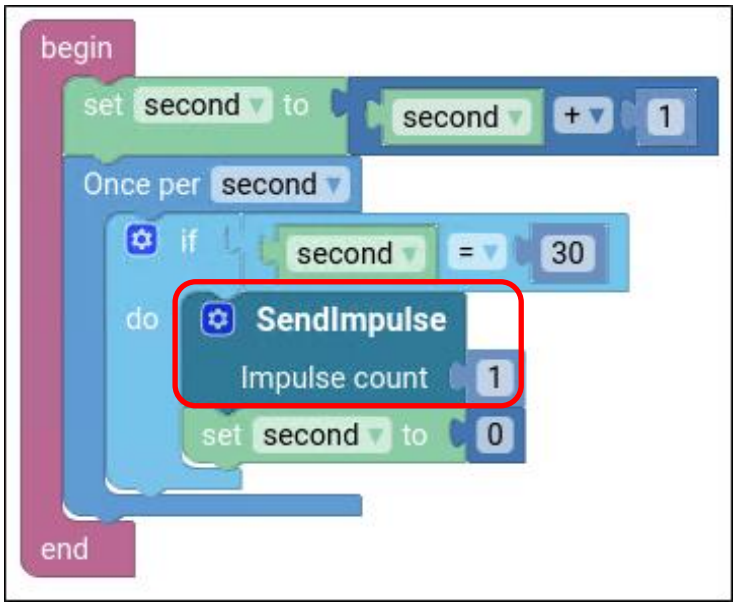
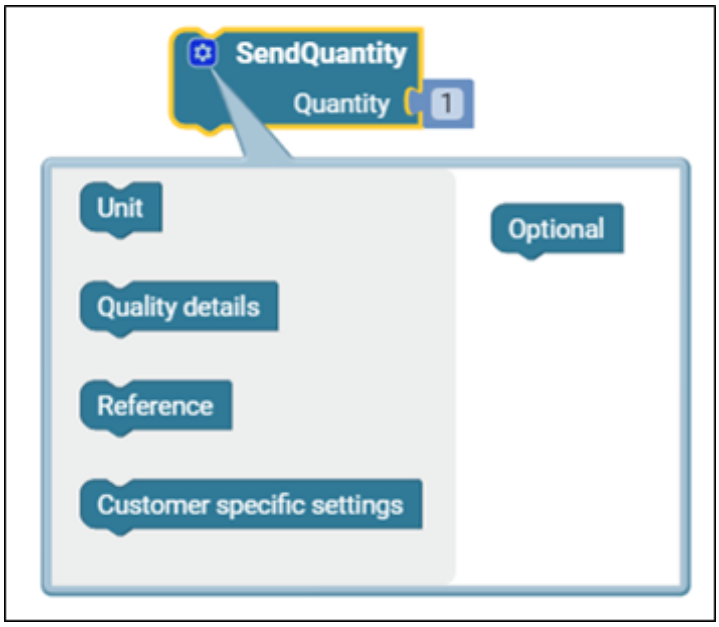


Bild 28: **Beispiel zu SendImpulse**

Einmal pro Sekunde soll hier die Sekundenanzahl um 1 erhöht werden. Wenn die Sekundenanzahl 30 erreicht, dann sendet der Block **`SendImpuls`** eine Meldung und die Variable **`second`** wird auf 0 zurückgesetzt.

6.2 SendQuantity



Der Block **`SendQuantity`** versendet eine bestimmte Menge an ein Drittsystem. Die gewünschte Menge wird als Number bei **`Quantity`** eingegeben. Optional können **`Unit`** (Einheit), **`Quality details`**, **`Reference`** und **`Customer specific settings`** mit versendet werden.
`Units` müssen vorher als Variable angelegt werden.
Input für **`Quantity`** sind ausschließlich Numbers.
Input für alle weiteren Angaben sind ausschließlich Strings. Einschränkungen für Input und Output gibt es nicht.

Beispiel

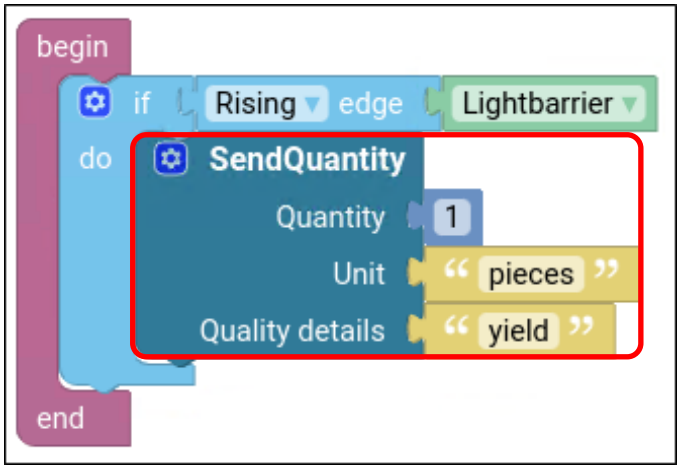
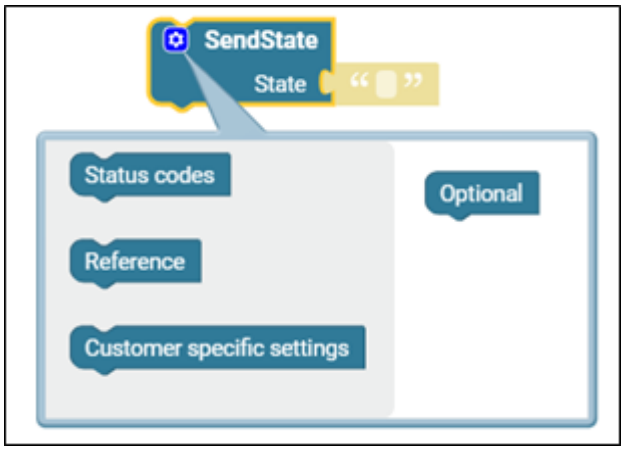


Bild 29: Beispiel für SendQuantitiy

Wenn eine Lichtschranke ausgelöst wird, dann soll der Block **`SendQuanity`** eine Meldung versenden. Diese Meldung enthält die Information, dass eine Menge (**`Quantity`**) von 1 mit der Einheit (**`Unit`**)

Stück produziert wurde und dass, diese Menge das Qualitätsmerkmal (**`Quality detail`**) gut (**`yield`**) besitzt.

6.3 SendState



Der Block **`SendState`** versendet den im Feld **`State`** definierte Asset-Zustand. Die Zustände können hier frei eingetragen werden.
Zudem gibt es noch optional die Möglichkeit die Liste der **`Status codes`**, **`Reference`** und **`Customer specific settings`** mit zu versenden. Die Angaben dazu wurden in 3 im Configuration Wizard vorgenommen. Genauere Informationen dazu sind in Kapitel 3.1 zu finden.

ⓘ Für das Versenden von **`Status codes`** muss eine Liste angelegt werden.
Weitere Informationen dazu sind in Kapitel 12 zu finden.

Input sind ausschließlich Strings. Einschränkungen für den Output gibt es nicht.

Beispiel

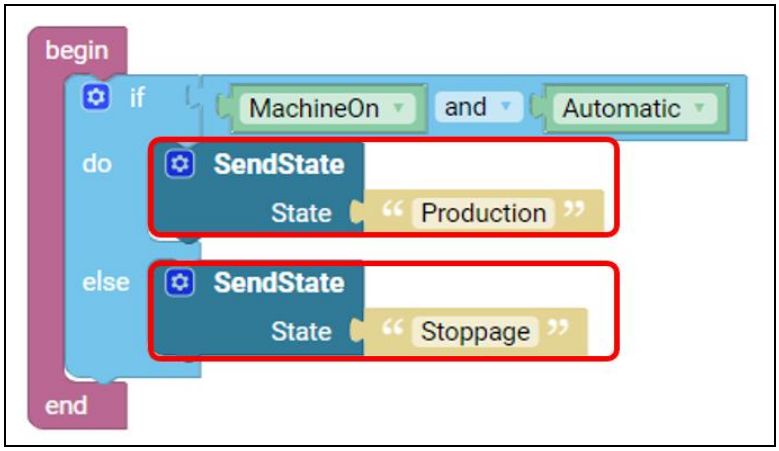
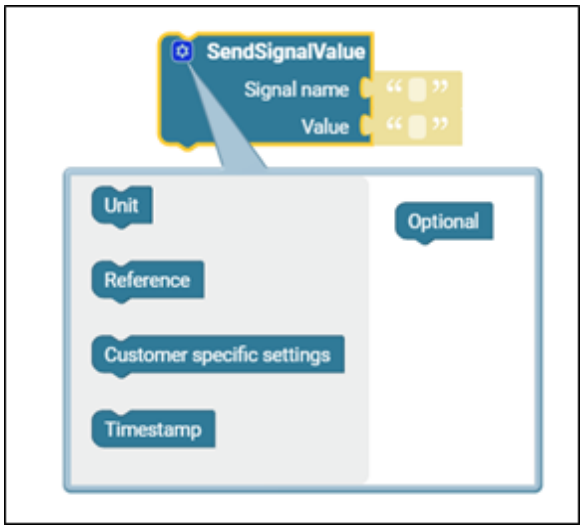


Bild 30: Beispiel zu SendState

In diesem Beispiel wird einer von zwei Status gemeldet. Wenn die Maschine an (**`MachineOn`**) ist und sich im Automatic-Modus (**`Automatic`**) befindet, dann sendet der Block **`SendState`** den Status **`Production`**. Andernfalls wird der Status **`Stoppage`** gesendet.

6.4 SendSignalValue



Mit dem Block **`SendSignalValue`** werden Signalwerte versendet.
Unter **`Signal name`** wird der Name des Signals eingetragen.
Unter **`Value`** wird der dazugehörige Wert eingegeben, die **`Unit`** ist die hierbei verwendete Einheit der Signale.
Wenn einer der unteren optionalen Blöcke verwendet werden soll, müssen trotzdem alle oberen Blöcke eingefügt werden. Diese können allerdings leer bleiben.
Input sind ausschließlich Strings. Einschränkungen für den Output gibt es nicht.

Beispiel

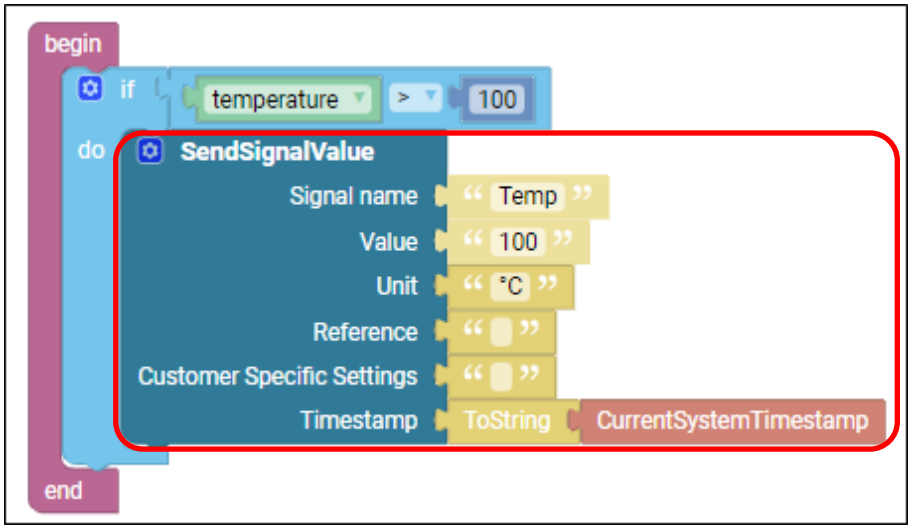
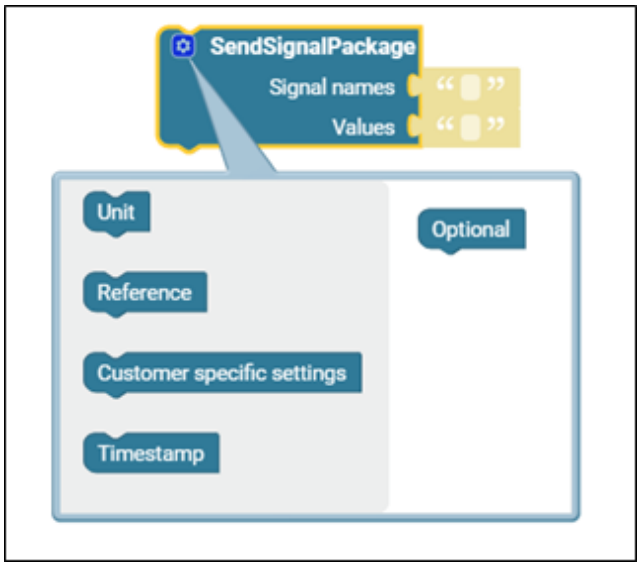


Bild 31: Beispiel für SendSignalValue

In diesem Beispiel soll eine Warnmeldung weitergegeben werden, wenn die Temperatur zu hoch wird.
Wenn das Signal **`temperature`** größer als **`100`** ist, wird ein Signalwert versendet.
Das gesendete Signal enthält als Informationen den Signalnamen (**`Signal name`**: **`Temp`**), den Wert (**`Value`**: **`100`**), die Einheit des Werts (**`Unit`**: **`°C`**) und den Zeitpunkt, an dem der Grenzwert überschritten wurde (**`Timestamp: CurrentSystemTimestamp`**). Informationen zu **`Reference`** oder **`Customer specific settings`** sind optional und können leer bleiben.

6.5 SendSignalPackage



`SendSignalPackage` sendet Listen mit Signalen. Die Inhalte stammen aus zuvor definierten Listen. Weitere Signale können in der Namensliste mit den passenden Signalwerten hinzugefügt werden. Input sind ausschließlich Strings. Einschränkungen für den Output gibt es nicht.

ⓘ Hier muss die Reihenfolge beachtet werden:
Der erste Eintrag in der Signalliste muss dem ersten Eintrag in der Werteliste entsprechen.

Beispiel

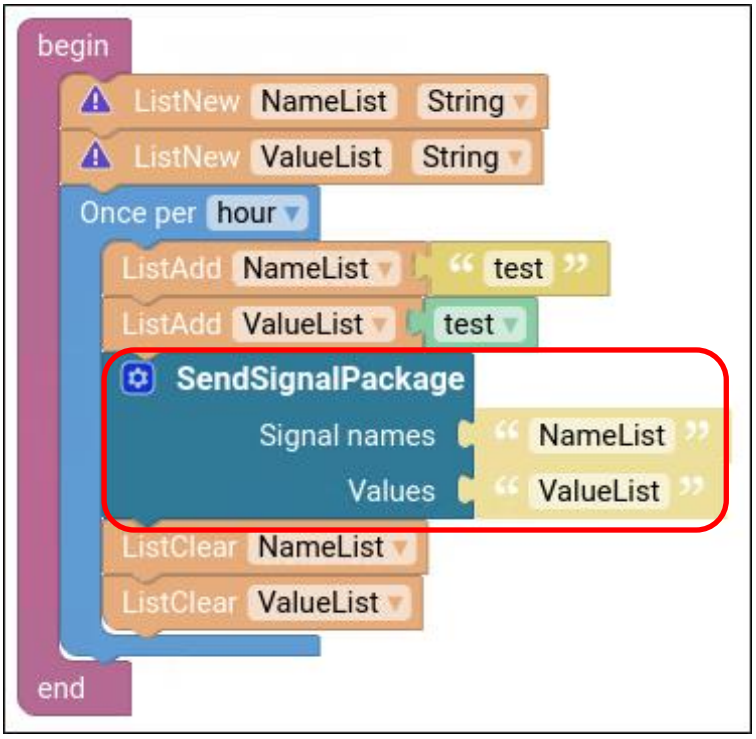
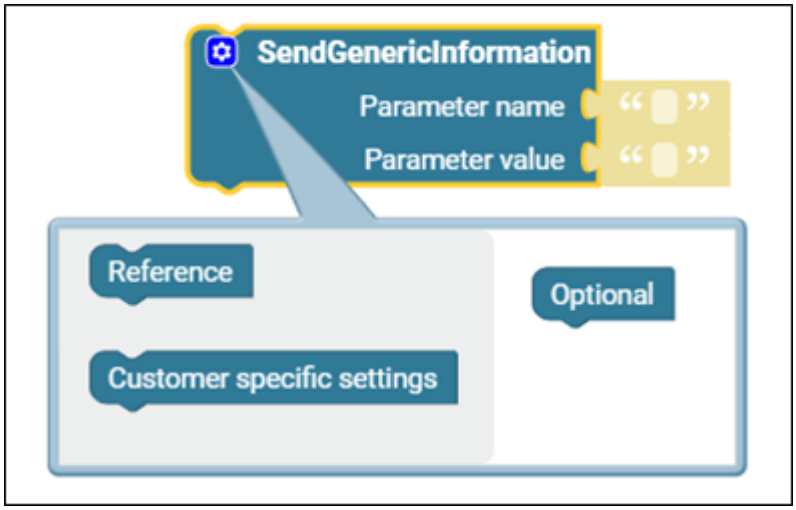


Bild 32: Beispiel für SendSignalPackage

Zuerst wurden zwei neue Listen erstellt, eine Namensliste (**`NameList`**) und eine Werteliste (**`ValueList`**).
In einem Repeater **`Once per hour`** wird das Signal **`test`** in der **`NameList`** definiert. Der dazugehörige Wert **`test`** wird in der **`ValueList`** eingetragen.
Der Block **`SendSignalPackage`** dient hier dazu, die Listen stündlich zu versenden.
Anschließend werden die Listen geleert.

6.6 SendGenericInformation



Der Block **`SendGenericInformation`** sendet ein Event mit aktuellen Maschinen-/(Asset-) Informationen. Die Einträge **`Parameter name`** und **`Parameter value`** werden immer versendet.
Optional können Reference und **`Customer specific settings`** versendet werden.
Input sind ausschließlich Strings. Einschränkungen für den Output gibt es nicht.

Beispiel

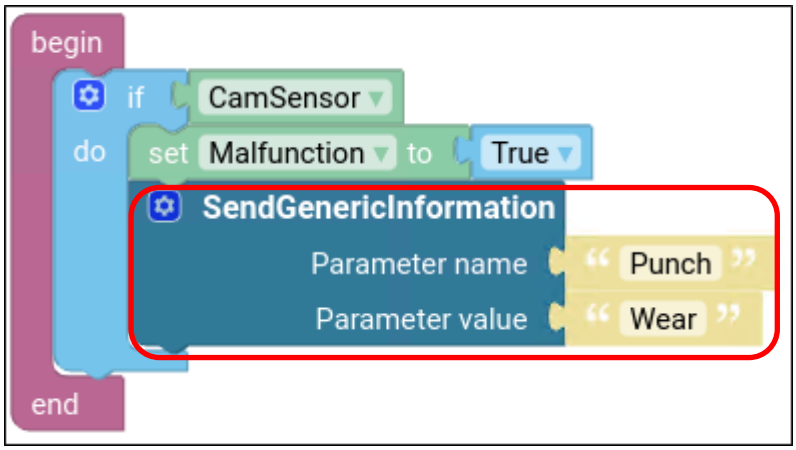
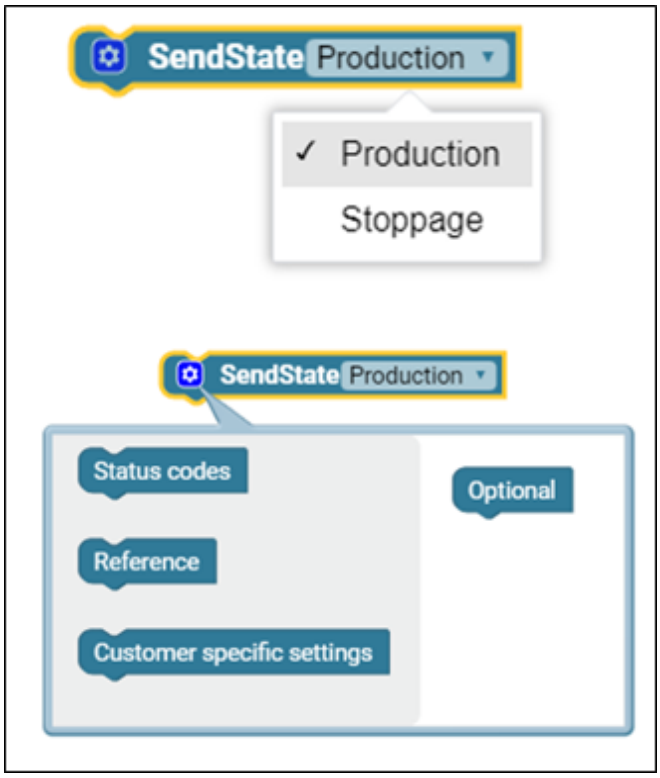


Bild 33: Beispiel für SendGenericInformation

Wenn der Kamerasensor anschlägt, dann wird der Status **`Malfunction`** auf **`True`** (wahr/1) gesetzt. Es ist also eine Fehlfunktion eingetreten.
Der Block **`SendGenericInformation`** sendet die Fehlermeldung der Stanze (englisch: **`punch`**) als Verschleiß (englisch: **`wear`**).

6.7 SendState [Auswahl]



Der Block **`SendState [Auswahl]`** versendet einen Asset-Zustand. Es gibt zwei Möglichkeiten: **`Production`** oder **`Stoppage`**. Zudem gibt es noch optional die Möglichkeit die Liste der **`Status codes`**, **`Reference`** und **`Customer specific settings`** mit zu versenden. Die Angaben dazu wurden in 3 im Configuration Wizard vorgenommen. Genauere Informationen dazu sind in Kapitel 3.1 zu finden.

ⓘ Für das Versenden von **`Status codes`** muss eine Liste angelegt werden.
Weitere Informationen dazu sind in Kapitel 12 zu finden.

Input für **`SendState [Auswahl]`** sind ausschließlich Strings. Einschränkungen für den Output gibt es nicht.

Beispiel

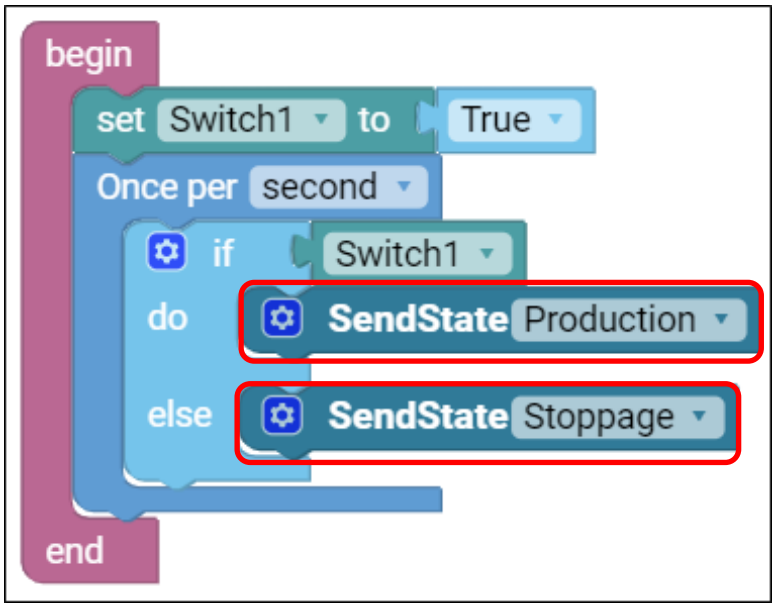
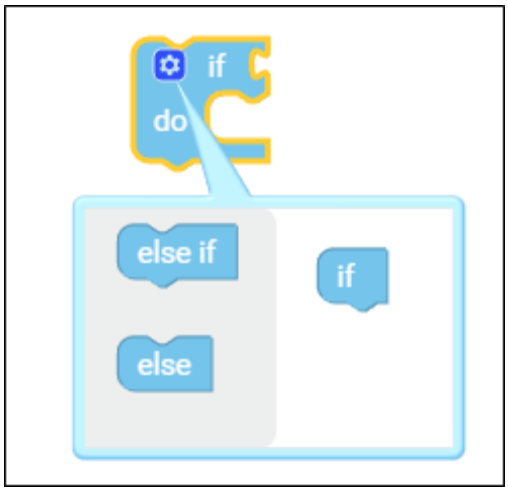


Bild 34: **Beispiel zu SendState [Auswahl]**

Zu Beginn wird das Signal **`Switch1`** von **`False`** (falsch/0) auf **`True`** (wahr/1) gesetzt. Dann beginnt ein Repeater. Wenn der Schalter **`Switch1`** umgelegt ist, dann wird der Produktionsstatus **`Production`** gesendet. Sonst wird der Status **`Stoppage`** gesendet.

7 Logical

7.1 If-do (wenn - dann)



Dieser Block bildet die übliche wenn-dann-Logik ab. **`If`** steht für eine Bedingung, die erfüllt sein muss, damit die folgende Anweisung (**`do`**) ausgeführt wird. Sollte eine Bedingung nicht erfüllt sein, kann eine andere Anweisung mit Hilfe von **`else`** übergeben werden.
Der Block **`else if`** ist optional. Die Anweisung wird dann ausgeführt, wenn die damit verbundene Bedingung als **`True`** (wahr/1) angesehen wird. Weitere Parameter können über die dunkelblaue Einstellungssymbol ausgesucht werden.
Input für **`if`**, **`else if`** und **`else`** sind Booleans.
Der Input für **`do`** hat keine Einschränkungen.
Einschränkungen für den Output gibt es nicht.

Beispiel

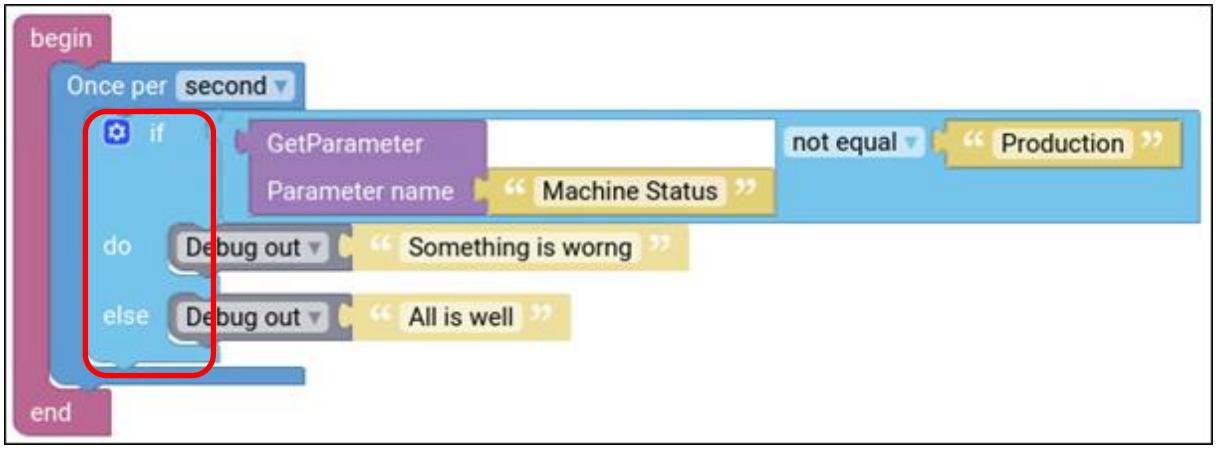


Bild 35: **Beispiel für den If-do-Block**

In dem Beispiel wird ein Mal pro Sekunde der Maschinenstatus abgefragt.
Wenn der Maschinenstatus nicht **`Produktion`** entspricht, dann soll **`Something is wrong`** ausgegeben werden. Sonst wird die Meldung **`All is well`** ausgegeben.

7.2 Mathematischer Vergleich „=/≠/</>/≤/≥“

Logische Verknüpfungen wie zum Beispiel „=“ verbinden zwei Variablen. Das mathematische Zeichen = verdeutlicht, dass in diesen Feldern nur Zahlwerte eingefügt werden können. Der Output ist ein Boolean-Wert, also der Wahrheitswert True (wahr/1) oder False (falsch/0).
Im Drop-Down-Menü kann das Symbol = durch andere Symbole ausgetauscht werden.

Die Bedeutungen sind der Tabelle zu entnehmen:

A = B	A gleich B
A ≠ B	A ist ungleich B
A > B	A größer als B
A ≥ B	A größer/gleich B
A < B	A kleiner als B
A ≤ B	A kleiner/gleich B

Input sind ausschließlich Zahlen (Numbers). Output sind ausschließlich Booleans.

Beispiel

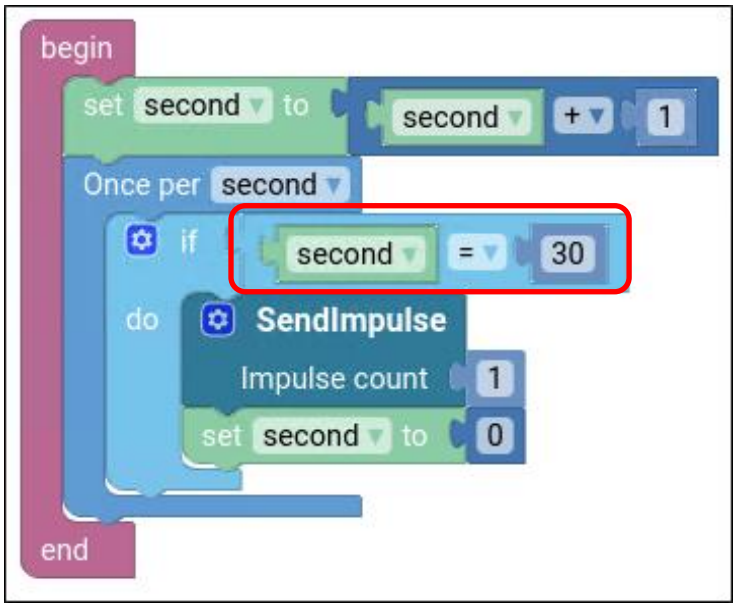
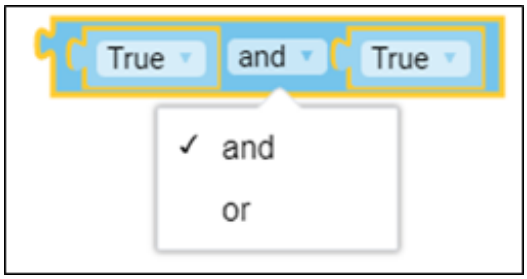


Bild 36: **Beispiel zu der Verknüpfung =**

In diesem Beispiel sollen 30 Sekunden abgezählt werden. Danach wird der Wert wieder auf Null gesetzt.
Die Variable **second** wird einmal pro Sekunde um eins erhöht.
In jeder Sekunde wird geprüft, wie viele Sekunden vergangen sind. Wenn die Anzahl der Sekunden gleich (=) 30 ist, dann wird ein Impuls versendet. Dieser Impuls setzt den Zähler wieder auf den Ausgangswert 0 zurück.

7.3 Logische Verknüpfung ‚and/or'



Die Verknüpfung **and** ist eine Grundverknüpfung. Treffen die Zustände oder Aussagen davor und danach zu, dann ist das Ergebnis **True** (wahr/1). Die Reihenfolge der Inputzustände ist dabei freigestellt. Der Output ist ein Boolean Wert, also der Wahrheitswert True (wahr/1) oder False (falsch/0).
Über das Drop-down-Menü kann **or** ausgewählt werden. Hier muss nur eine der beiden Aussagen zutreffen, damit das Ergebnis als True (wahr/1) gilt.
Input und Output sind ausschließlich Booleans.

Beispiel

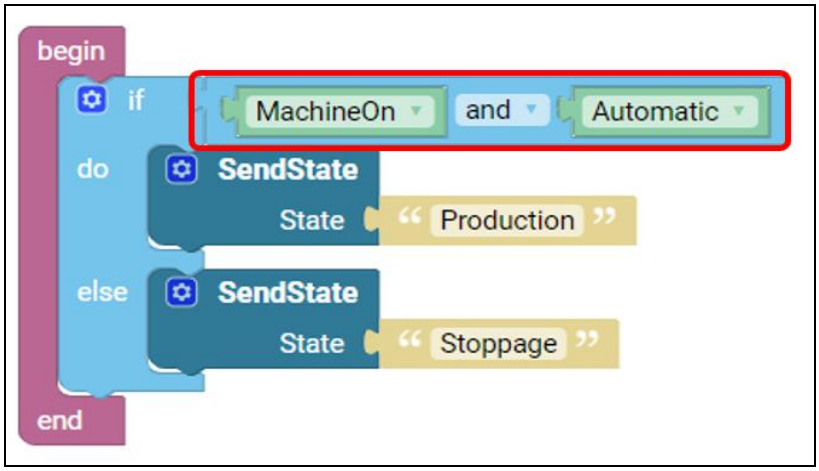
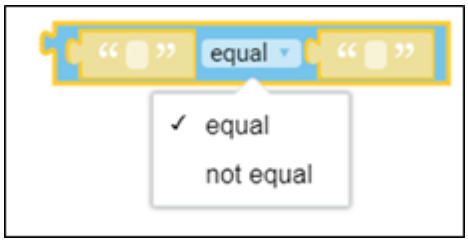


Bild 37: **Beispiel zu der Verknüpfung and**

In diesen Beispiel sendet der Block **`SendState`** den Status **`Production`** nur unter der Vorraussetzung, dass die Maschine an (**`MachineOn`**) ist und (**`and`**) sich im Autormatik-Modus (**`Automatic`**) befindet. Wenn nur eine der beiden Voraussetzungen zutrifft, dann wird der Status **`Stoppage`** verschickt.

7.4 Logische Verknüpfung ‚equal/not equal'



`Equal` ist eine Grundverknüpfung. Wenn zwei Zustände oder Aussagen gleich (englisch: equal) sind, dass ist das Ergebnis **`True`** (wahr/1).
Die Reihenfolge der Input-Zustände ist dabei freigestellt. Der Input hier sind String-Werte, der Output ist ein Boolean-Wert, also der Wahrheitswert **`True`** (wahr/1) oder **`False`** (falsch/0).
Über das Drop-down-Menü kann das Gegenteil **`not equal`** ausgewählt werden.
Der Unterschied zwischen „=" und **`equal`** besteht darin, dass bei **`equal`** Strings verglichen werden. Input sind ausschließlich Strings. Output sind ausschließlich Booleans.

Beispiel

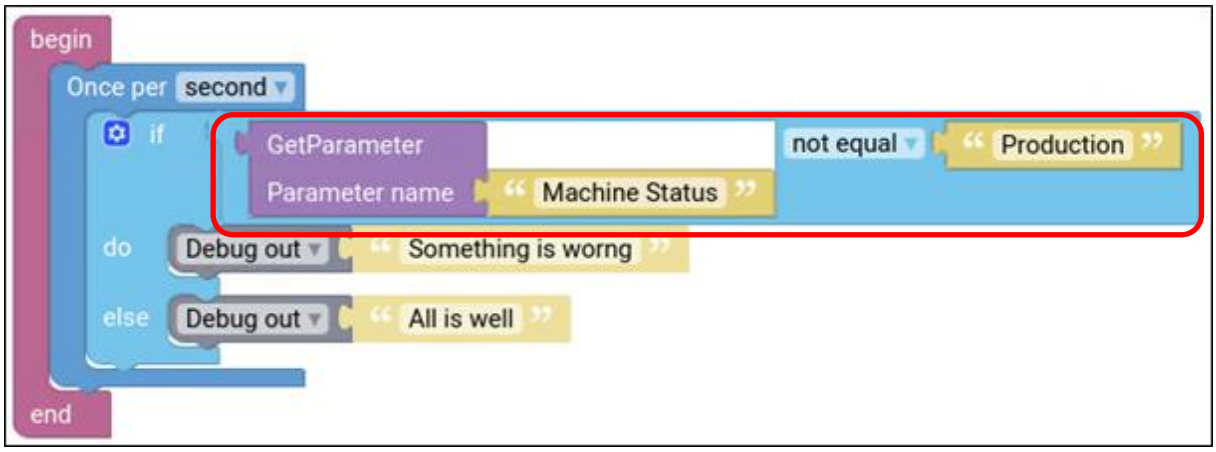
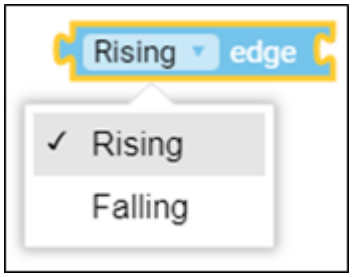


Bild 38: **Beispiel zu not equal**

In dem Beispiel wird ein Mal pro Sekunde der Maschinenstatus abgefragt.
Wenn der Maschinenstatus **`not equal`** (nicht gleich) **`Production`** ist, dann soll die Meldung **`Something is wrong`** ausgegeben werden. Andernfalls wird die Meldung **`All is well`** ausgegeben.

7.5 Rising/Falling edge



Dieser Block zeigt an, dass eine Variable oder Signal von True (wahr/1) auf False (falsch/0) gewechselt hat oder umgekehrt. Input sind ausschließlich boolesche Werte.

`Rising edge`: Zu Beginn ist der Boolean-Wert False (falsch/0). **`Rising edge`** prüft, ob dieser nun True (wahr/1) ist. Also, ob sich der Wert von 0 zu 1 verändert hat. Ist das der Fall, wird die dazugehörige Anweisung ausgeführt.

`Falling edge`: Zu Beginn ist der Boolean-Wert True (wahr/1). **`Falling edge`** prüft, ob dieser nun False (falsch/0) ist. Also ob sich der Wert von 1 zu 0 verändert hat. Ist das der Fall, wird die dazugehörige Anweisung ausgeführt.

Beispiel

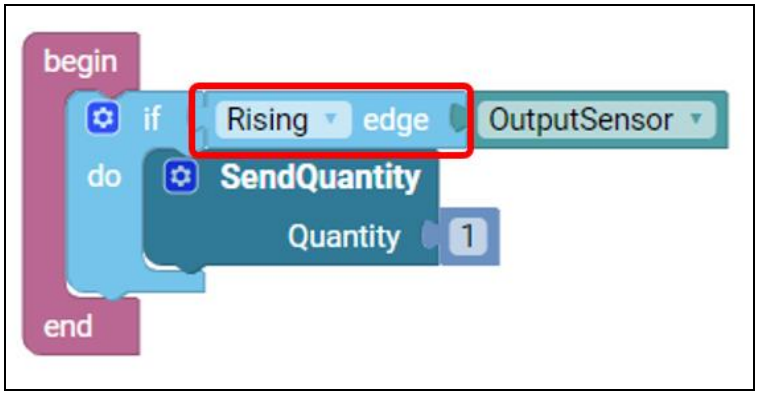


Bild 39: **Beispiel zu Rising edge**

In disem Beispiel gibt es einen **`OutputSensor`**. Dieser löst ein Signal aus, wenn ein Stück produziert wurde. Der Boolean-Wert des Signals wird also von False (falsch/0) auf True (wahr/1) verändert. Der Block **`Rising edge`** ist also True (wahr/1). Daher wird die folgende Anweisung ausgeführt und der Block **`SendQuantity`** meldet ein produziertes Stück.

7.6 Not-Statement

Das Ergebnis eines **`not`**-Statments (nicht-Aussage) wird wahr, wenn die Input-Variable falsch ist.
Der Ausgangszustand ist also entgegengesetzt zum Output-Zustand.
Input und Output sind ausschließlich Booleans.

Beispiel

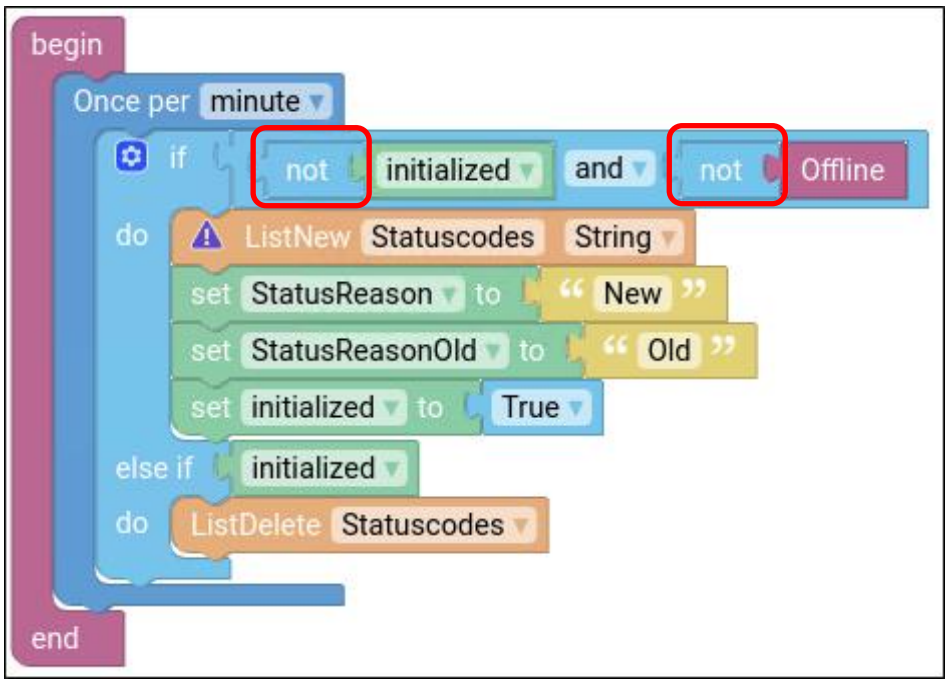


Bild 40: **Beispiel für ein not-Statement**

In dem Beispiel wird einmal pro Minute überprüft, ob die Maschine zum ersten Mal läuft.
Wenn das Programm nicht ausgeführt (**`not initialized`**) wurde und das Asset nicht offline (**`not offline`**) ist, dann läuft das Asset. Daher werden Listen erstellt mit den aktuellen und alten Statusgründen. Durch das Anlegen von Listen wird das Programm ausgeführt (**`initialized`**). Damit wird die Variable **`True`** (wahr/1).
Die Liste mit Statusgründen wird anschließend gelöscht.

7.7 Wahrheitsaussage

Dieser Block wird am Ende gesetzt, um zu definieren, ob das Ereignis **`True`** (wahr/1) oder **`False`** (falsch/0) ist. Dafür kann aus der Drop-down-Menü entsprechend **`True`** oder **`False`** ausgewählt werden.
Einschränkungen für den Input gibt es nicht. Output sind ausschließlich Booleans.

Beispiel

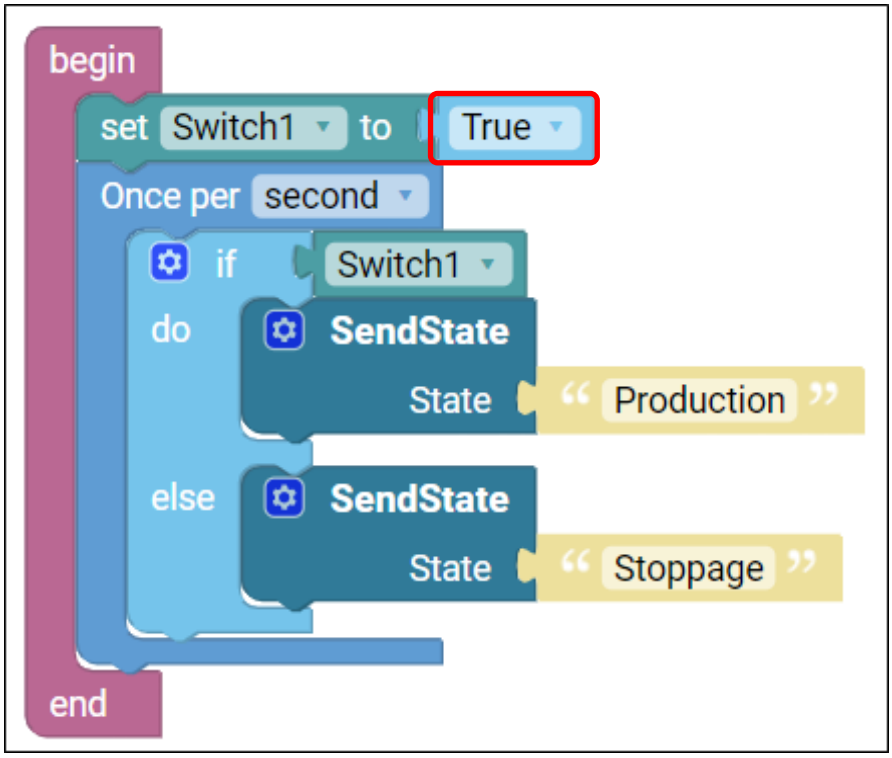
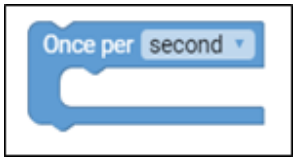


Bild 41: **Beispiel für True**

Zu Beginn ist der **`Switch1`** umgelegt, also wird das Signal ausgelöst und damit **`True`** (wahr/1). Dann läuft ein Mal pro Sekunde ein Repeater ab, der prüft ob **`Switch1`** umgelegt wurde. Wenn das ist ist, wird der Produktionsstatus auf **`Production`** gesendet. Andernfalls wird der Status **`Stoppage`** ausgegeben.

8 Repeaters

8.1 Once per



Repeaters (deutsch: Wiederholungen/Schleifen) werden benutzt, um eine Aktion regelmäßig zu wiederholen. Über das Drop-down-Menü kann diese Frequenz ausgewählt werden.

```
Once per:
        second
        minute
        hour
        day
```

Beispiel

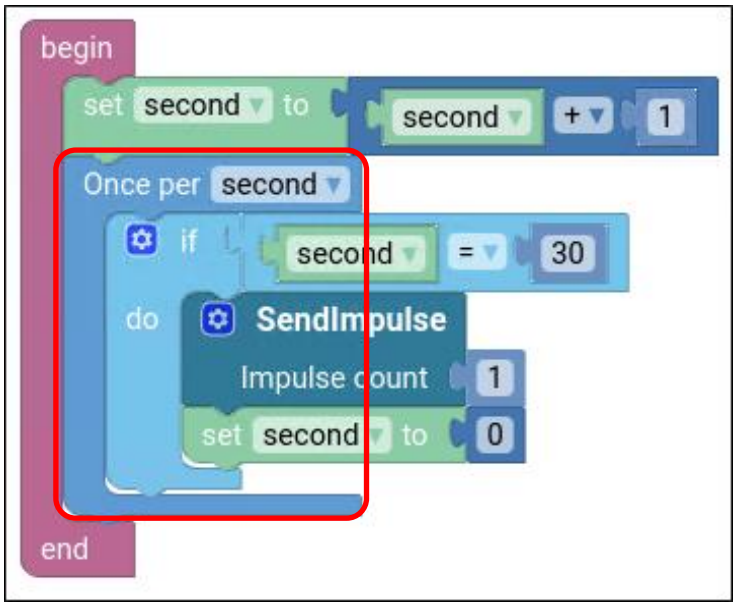


Bild 42: **Beispiel zu Once per**

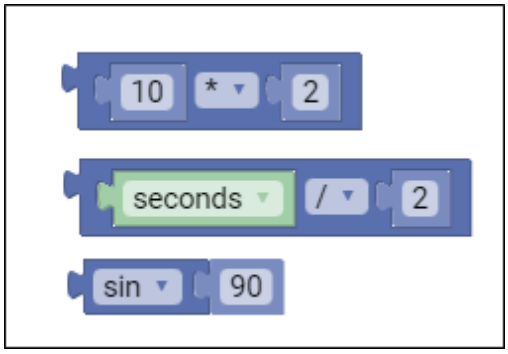
In diesem Beispiel sollen 30 Sekunden abgezählt werden. Danach soll der Wert wieder auf Null gesetzt werden.
Die Variable **second** wird einmal pro Sekunde um eins erhöht. In jeder Sekunde (**Once per second**) wird geprüft, wie viele Sekunden vergangen sind. Wenn die Anzahl der Sekunden gleich (=) 30 ist, dann wird ein Impuls versendet. Der Impuls setzt den Zähler für die Variable **second** wieder auf den Ausgangswert 0 zurück.

9 Arithmetic

9.1 Nummernfeld

In diesem Block werden numerische Werte eingefügt, um sie mit einer Aufgabe zu verbinden.
Input und Output sind ausschließlich Numbers.

9.2 Matheoperation



Verschiedene Matheoperationen wie das Addieren, Summieren, Multiplizieren, Dividieren, Hochstellen oder das Berechnen des Sinus, Cosinus und Tangens, sind mit diesen Blöcken möglich. Dabei können nicht nur Zahlen, sondern auch Variablen verwendet werden.
Input und Output sind ausschließlich Numbers.

Beispiel

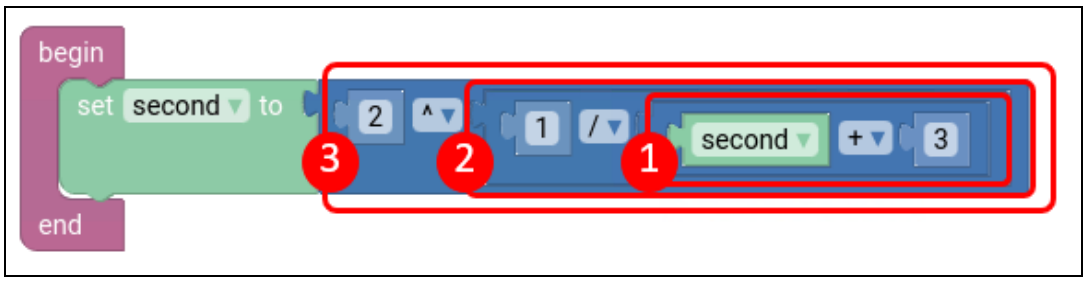


Bild 43: **Beispiel für mathematische Operationen**

Eine verschachtelte Rechnung gibt den Faktor an. Dabei ist es wichtig, sich zum Verständnis des Rechenwegs an die Regel „von innen nach außen“ zu halten. Nach diesem Prinzip werden die Klammern gesetzt und die Matheoperationen ausgerechnet.
In diesem Beispiel wird zuerst die Variable **second** mit drei addiert (1), diese Summe steht im Nenner des Bruch (2). Dieses Ergebnis ist wiederum der Exponent von 2 in der letzten Matheoperation (3).

9.3 ToNumber

Der Block **`ToNumber`** ändert den Datentyp eines Strings und wandeln ihn in einen numerischen Wert um. Der Inhalt des Strings darf ausschließlich aus Zahlen bestehen.
Der Input muss ein numerischer Wert im String Datentyp sein. Outputs sind ausschließlich Numbers.

Beispiel

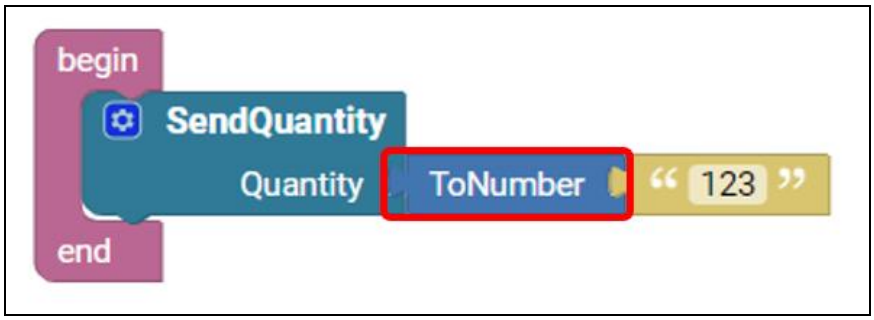
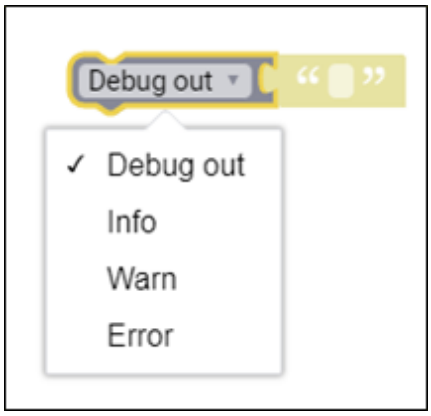


Bild 44: **Beispiel für ToNumber**

Der Block **`SendQuantity`** soll eine Menge melden. Der Input hat hier aber den Typ String. Auch wenn dieser String Zahlen enthält, ist das nicht der nötige Datentyp für den **`SendQuanity`**-Block. Mithilfe des Blocks **`ToNumber`** wird der Datentyp String in den Datentyp Number verändert. Nur so kann der Block **`SendQuantity`** ausgeführt werden.

10 Logging

10.1 Logging



Um die gewünschten Werte zu erhalten, werden Rohsignale oder Variablen ausgeloggt. Dabei können verschiedene Arten von Log-Einträgen ausgewählt werden.

`Debug out`: Informationen, die bei der Diagnose von Problemen hilfen
`Info`: Allgemeine Protokollebene, auf der alle Aktivitäten erfasst werden können
`Warn`: Probleme oder Störungen, der Code kann seine Arbeit dennoch fortsetzten
`Error`: Problem, das mehrere Funktionalitäten verhindert

Input sind ausschließlich Strings. Einschränkungen für den Output gibt es nicht.

Beispiel

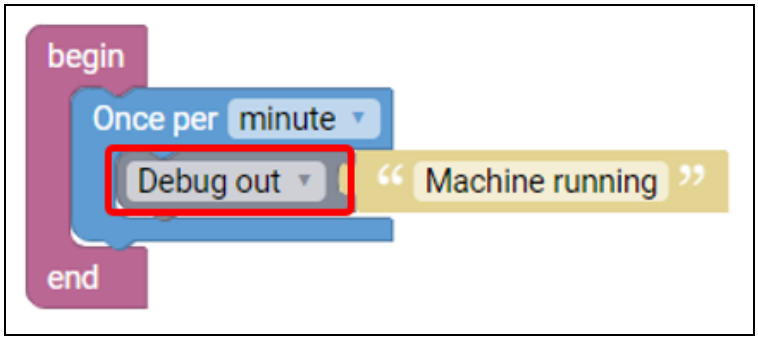


Bild 45: **Beispiel für Debug out**

In diesem Beispiel wird einmal pro Minute mit dem Block **`Debug out`** ein String mit dem Text Machine running ins Logfile geschrieben.

11 Text

In der Grafischen Komposition wird ein Text als ein String verstanden. Wie in einem String kann der Text aus Buchstaben, Zahlen und Zeichen bestehen.

11.1String

Mithilfe dieser Blöcke werden Strings durch Eingabe in die Anführungszeichen hinzugefügt. Einschränkungen für den Input gibt es nicht. Output sind ausschließlich Strings.

Beispiel

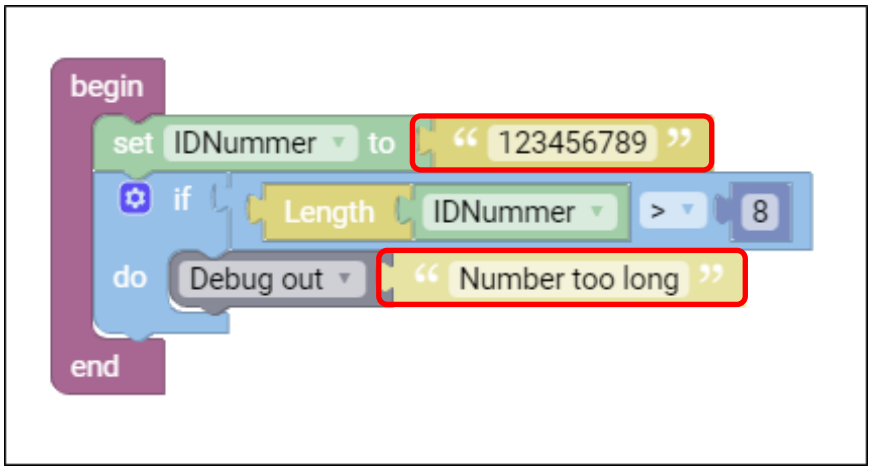


Bild 46: Beispiel für Length

Der Block **`set IDNummer to`** definiert die ID-Nummer eines Assets mit dem String „123456789“. Anschließend prüft der **`if-do`**-Block, ob die IDNummer länger ist als 8 Zeichen ist. Wenn das so ist, soll eine Meldung ins Logfile geschrieben werden. Diese Meldung wird im String eingetragen. In diesem Fall lautet die Meldung „Number too long“.

11.2Append String

Als Erweiterung zum einfachen String werden bei **`Append String`** mehrere Strings aneinandergereiht. Bei einem Klick auf die Plus- oder Minuszeichen, werden Strings hinzugefügt oder gelöscht.
Input und Output sind ausschließlich Strings.

Beispiel

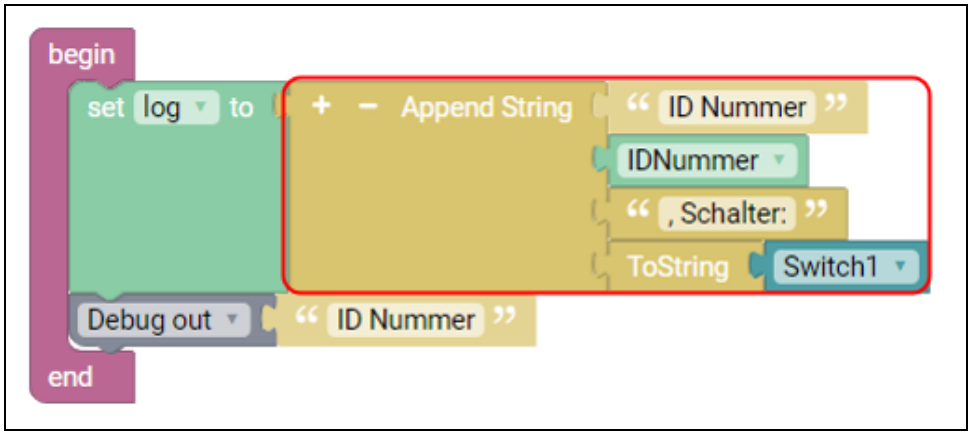


Bild 47: Beispiel zu Append String

In diesem Beispiel geht es darum die ID Nummer und den Schalter auszuloggen. Der Block **`set log to`** vereinfacht die Lesbarkeit. Gelesen wird der **`Append string`**-Block von oben nach unten. Somit wird zuerst der Text „ID Nummer“ angezeigt, danach wird der Wert der Variable **`IDNummer`** hinzugefügt. Anschließend kommt der Text „, Schalter:“ und das Signal des Schalters **`Switch1`** wird angefügt. Am Ende wird der gesamte String ins Logfile geschrieben.

11.3ToString

`ToString` wird verwendet, um Numbers oder Variablen, die Numbers darstellen, in einen String umzuwandeln.
Einschränkungen für den Input gibt es nicht. Output sind ausschließlich Strings.

Beispiel

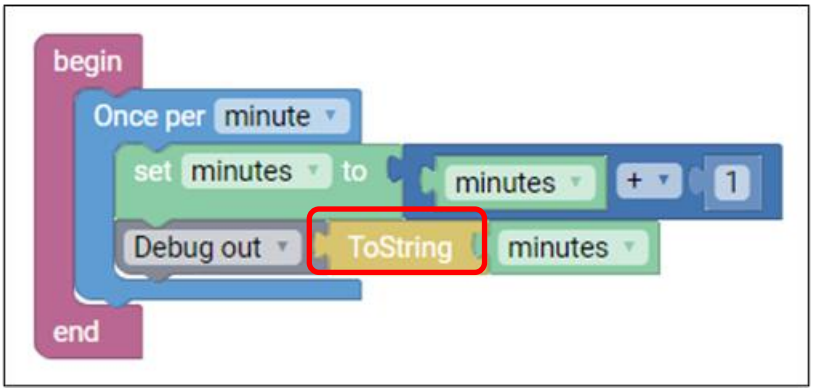
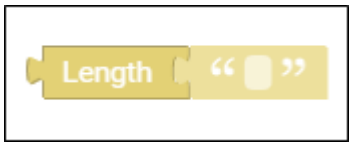


Bild 48: **Beispiel für ToString**

Ziel ist es, die Anzahl an Minuten auszugeben. **`Once per minute`** wird die Variable **`minutes`** um eines erhöht. Und der neue Wert mit **`Debug out`** ins Logfile geschrieben. **`Debug out`** arbeitet aber nur mit Strings als Input. Deswegen wird mit **`ToString`** die Variable **`minutes`** in einen Text umgewandelt.

11.4Length



Length zählt die Anzahl der Zeichen in einem String. Der gewünschte String wird in die Anführungszeichen eingegeben. Es ist auch möglich, eine Variable anzuhängen. Als Ergebnis werden die gezählten Zeichen des Strings ausgegeben. Das Ergebnis ist eine Zahl. Das Zählen startet mit 1. Input sind ausschließlich Strings. Output sind ausschließlich Numbers.

Beispiel

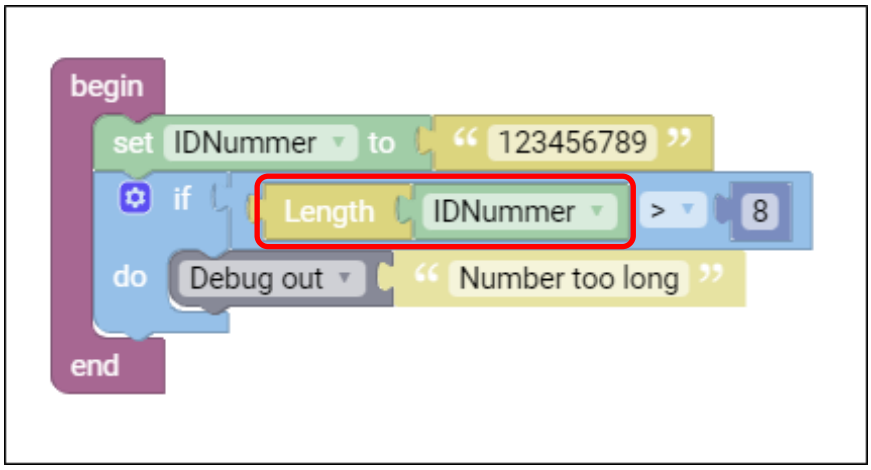
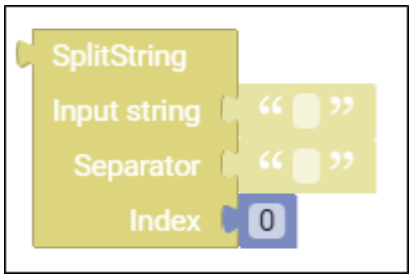


Bild 49: Beispiel für Length

Als Beispiel soll die Länge der Auftragsnummer gezählt werden, damit diese nicht den Schwellenwert von acht Zahlen überschreitet.
Wenn die **Length** (Länge) der **IDNummer** größer als 8 ist, soll der Block **Debug out** die Meldung „Number too long“ (Nummer zu lang) ins Logfile schreiben.

11.5SplitString



Im Block **SplitString** kann ein Wert aus einer selbst festgelegten Auswahl an Kategorien ausgegeben werden. Unter **Input string** werden die verschiedenen Kategorien definiert. Getrennt werden sie durch ein Zeichen. Dieses wird unter **Seperator** definiert, typischerweise ist es ein Komma oder ein Unterstrich.

Der **Index** zeigt an, welcher String der Auswahl im **Input string** ausgewählt werden soll. Es kann nur ein Wert ausgegeben werde. Gezählt werden die **Input strings** von links nach rechts. Begonnen wird mit 0.
Input und Output für **Input string** und **Separator** sind ausschließlich Strings.
Input und Output für den **Index** sind ausschließlich Numbers.

Beispiel

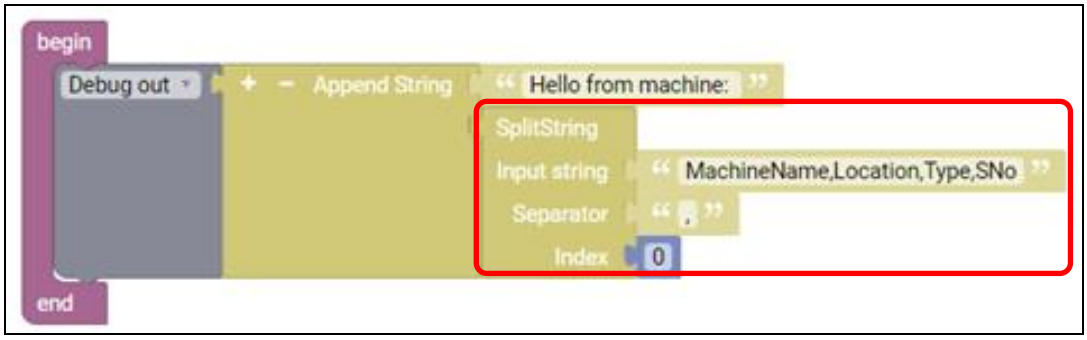


Bild 50: Beispiel für SplitString

In diesem Beispiel soll der Maschinenname ausgegeben werden. Zuerst kommt der Text „Hello from machine:“. Die möglichen Kategorien stehen unter **Input string** und werden durch Kommas (**Separator**) getrennt. Der ist **Index** mit 0 angegeben. Es wird also der MachineName ausgegeben. Wäre der Index 2, würde der Type ausgegeben werden.

11.6FromAscii

Der Block **FromAscii** bezieht sich auf eine festgelegte Wertetabelle mit Anweisungen und Zeichen. Der Block greift auf einen Wert aus dieser Tabelle zu. Die eingefügte Zahl gibt an, welcher Wert der Ascii-Tabelle ausgewählt werden soll.
Input sind ausschließlich Numbers. Output sind ausschließlich Strings.

ⓘ Die Ascii-Tabelle befindet sich in Kapitel 17.2 „Ascii-Tabelle“, Seite 76.

Beispiel

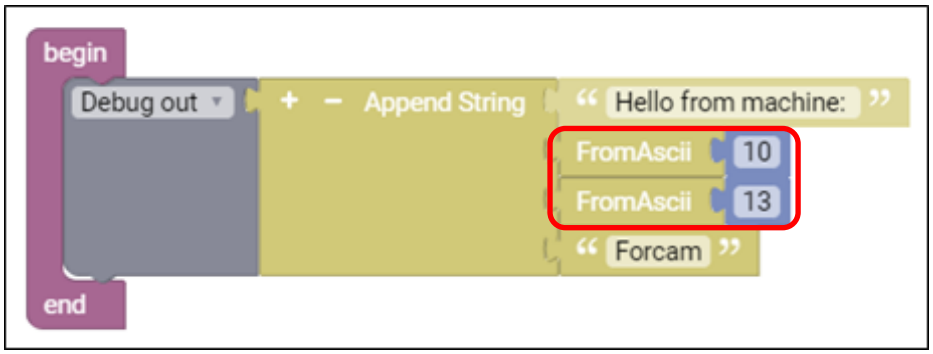
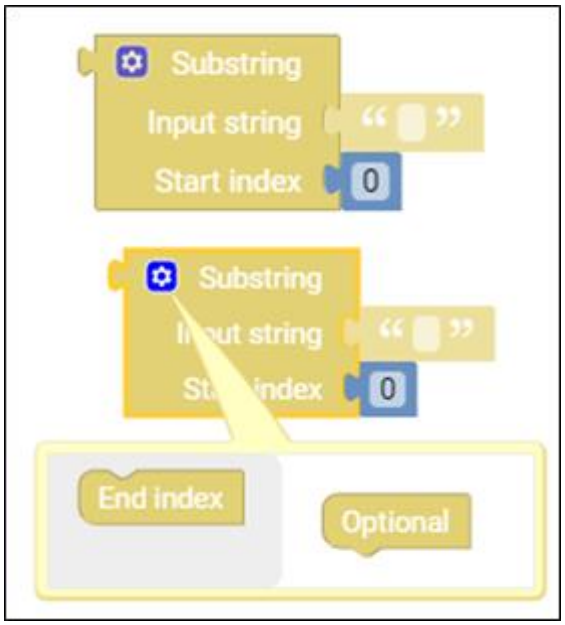


Bild 51: Beispiel zu FromAscii

In diesem Beispiel soll der Text „Hello from machine:“ und nach einem Absatz „Forcam“ ausgegeben werden.
Der Block **`Append String`** listet Strings nacheinander auf. Nachdem der erste Text-String „Hello from maschine:“ holt der Block **`FromAscii`** den zehnte Befehl aus der Ascii-Tabelle. Dieser ist LF für Line Feed (Zeilenumbruch). Danach holt ein zweiter **`FromAscii`**-Block den Befehl 13 aus der Ascii-Tabelle. Das ist CR, also Carriage Return (Enter-Taste drücken). Damit wird der Cursor wieder an den Anfang einer Zeile gestellt.
Das Ergebnis sieht also wie folgt aus:

Hallo from machine:
Forcam

11.7Substring



Der Block **`Substring`** gibt nur einen Teil eines Strings aus. Der gesamte String wird unter **`Input string`** angegeben. Der **`Start index`** und der **`End index`** werden als Numbers darunter eingetragen. Wie bei Indices üblich werden die Zeichen ab 0 gezählt. Der **`End Index`** ist ausgeschlossen.
Input und Output für den **`Input string`** sind Strings.
Input für **`Start index`** und **`End index`** sind ausschließlich Numbers. Output sind Strings.

Beispiel

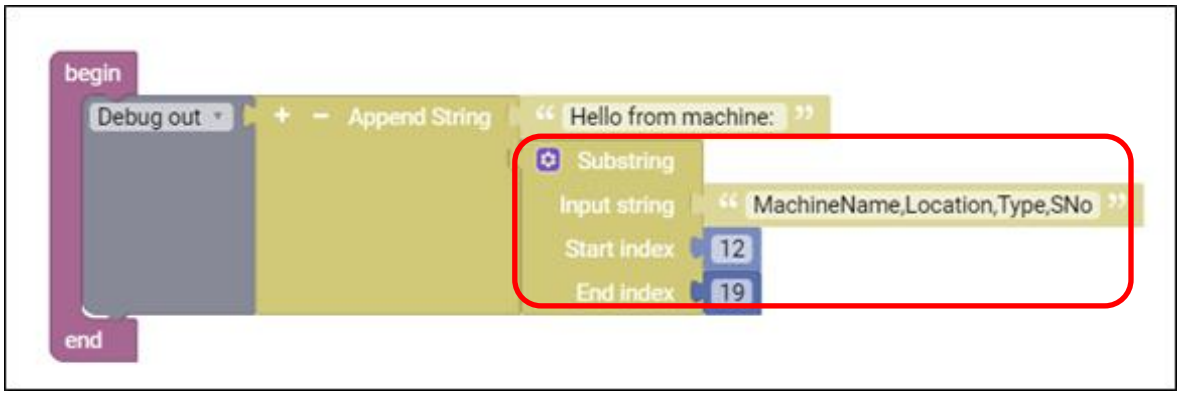


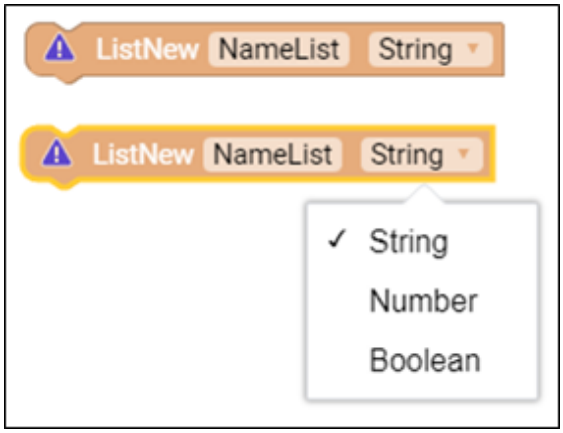
Bild 52: Beispiel zu Substring

In diesem Beispiel soll der Ort der Maschine ausgegeben werden.
Der Block **`Append String`** setzt zuerst der Text „Hello from machine:“. Der Block **`Input string`** listet eine Reihe von Asset-Eigenschaften auf. Der Block **`Start string`** gibt an, dass ab Zeichen 12 der Output erfolgen soll. Der Block **`End string`** gibt an, dass bis inklusive Zeichen 19 der Output erfolgen soll.
Weil mit 0 beginnend von links gezählt wird, wird die Eigenschaft **`Location`** ausgegeben.

12 Lists

ⓘ Es muss zuerst eine Listen angelegt werden.
Erst danach sind weitere Blöcke für die Listen verfügbar.

12.1 ListNew



Der Block **`ListNew`** erstellt eine neue Liste. Der Name der Liste kann im ersten Feld eingetragen werden. Die Art des Inputs der Liste (String, Number oder Boolean) wird im Drop-down-Menü gewählt.
Einschränkungen für den Input erfolgen über die Auswahl.

Beispiel

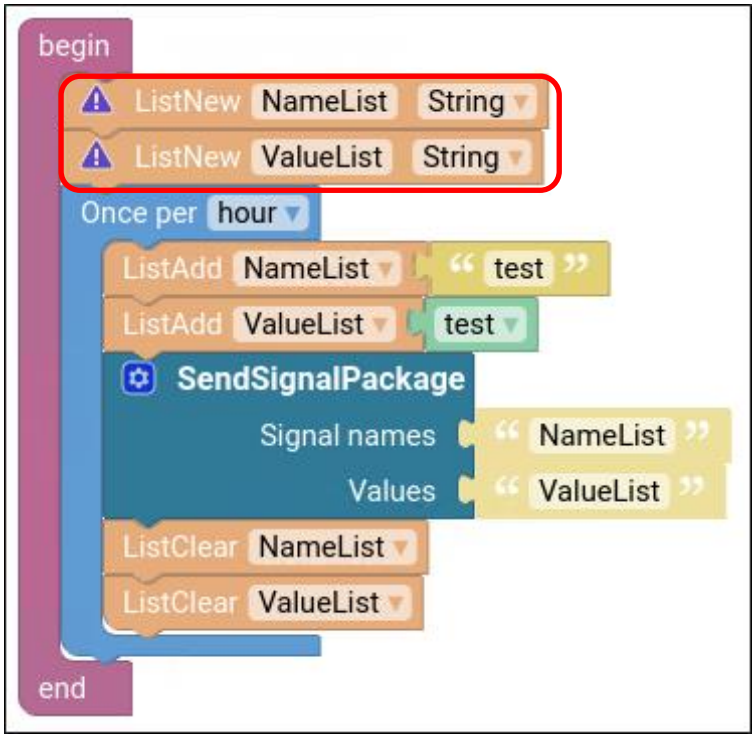
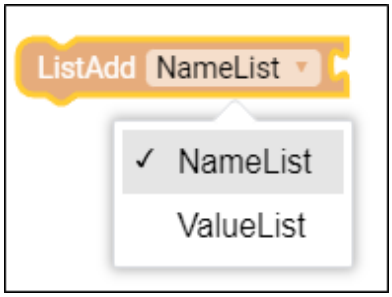


Bild 53: **Beispiel für ListNew**

Zuerst werden mit den Blöcken **ListNew** zwei neue Listen erstellt, eine Namensliste und eine Werteliste. Die Ausrufezeichen erinnern daran, am Ende die Liste zu leeren oder zu löschen. In einem Repeater wird der Signalname **test** in der Namensliste **NameList** eingefügt. Der dazugehörige Wert wird in der Werteliste **ValueList** eingefügt.
Anschließend versendet der Block **SendSignalPackage** beide Listen. Die Blöcke **ListClear** löschen die Inhalte der Listen.

12.2 ListAdd



Der Block **ListAdd** fügt Werte in eine List ein. Voraussetzung dafür ist, dass bereits eine Liste mit **ListNew** erstellt wurde. Die gewünschte Liste wird über das Drop-down-Menü ausgewählt.
Der Input des Blocks ist immer eine vorher erstellt Liste. Diese wird im Drop-down-Menü gewählt.
Einschränkungen für den Output gibt es nicht.

Beispiel

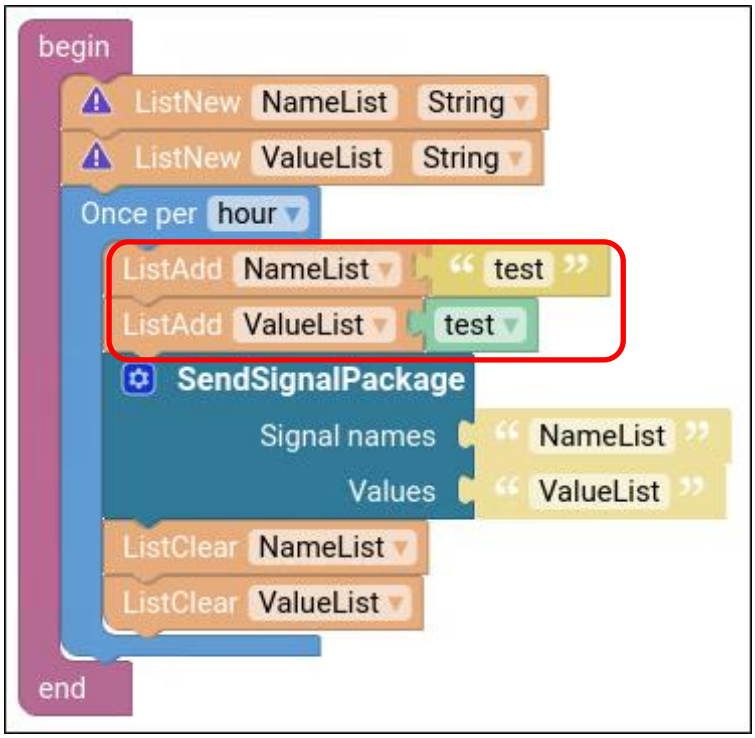
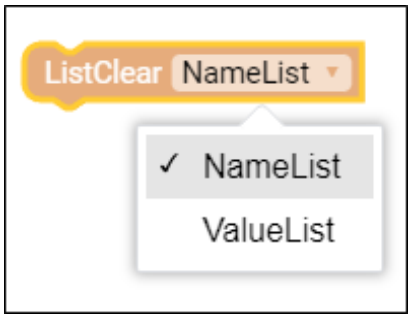


Bild 54: **Beispiel für ListAdd**

Zuerst werden zwei neue Listen erstellt, eine Namensliste und eine Werteliste. Die Blöcke **ListAdd** fügt ein Mal pro Stunde den Signalnamen **test** in die **NameList** und den dazugehörige Wert in die **ValueList** ein. Anschließend versendet der Block **SendSignalPackage** beide Listen. Die Blöcke **ListClear** löschen die Inhalte der Listen.

12.3 ListClear



`ListClear` löscht Inhalt einer Liste.
Der Input des Blocks ist immer eine vorher erstellt Liste. Diese wird im Drop-down-Menü gewählt.
Einschränkungen für den Output gibt es nicht.

ⓘ Es ist wichtig **`ListClear`** regelmäßig nach dem Erstellen einer neuen Liste auszuführen, damit genügend Speicherplatz frei bleibt.

⚠ **`ListClear`** löscht nur die Inhalte einer Liste.
`ListDelete` löscht eine zuvor erstellte Liste komplett.

Beispiel

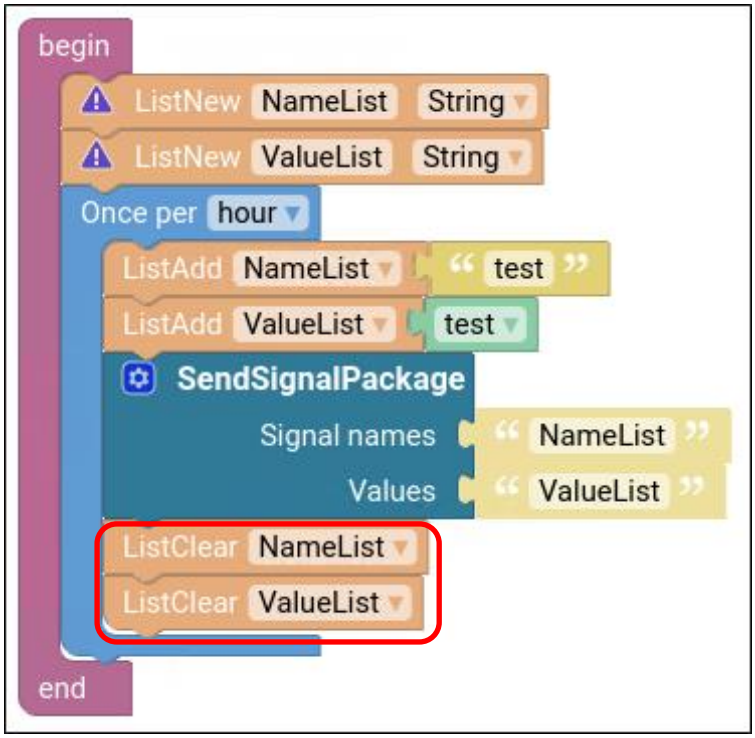
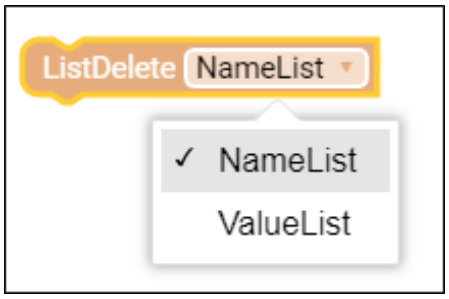


Bild 55: **Beispiel für ListClear**

Zuerst werden zwei neue Listen erstellt, eine Namensliste und eine Werteliste. Die Blöcke ListAdd fügt ein Mal pro Stunde den Signalnamen test in die NameList und den dazugehörige Wert in die ValueList ein. Anschließend versendet der Block SendSignalPackage beide Listen. Die Blöcke **`ListClear`** löschen die Inhalte der Listen.

12.4 ListDelete



Der Block **`ListDelete`** löscht eine erstellte Liste. Über das Drop-down-Menü wird die Liste ausgewählt, die gelöscht werden soll.
Der Input des Blocks ist immer eine vorher erstellt Liste. Diese wird im Drop-down-Menü gewählt. Einschränkungen für den Output gibt es nicht.

⚠ **`ListDelete`** löscht eine zuvor erstellte Liste komplett.
`ListClear` löscht nur die Inhalte einer Liste.

Beispiel

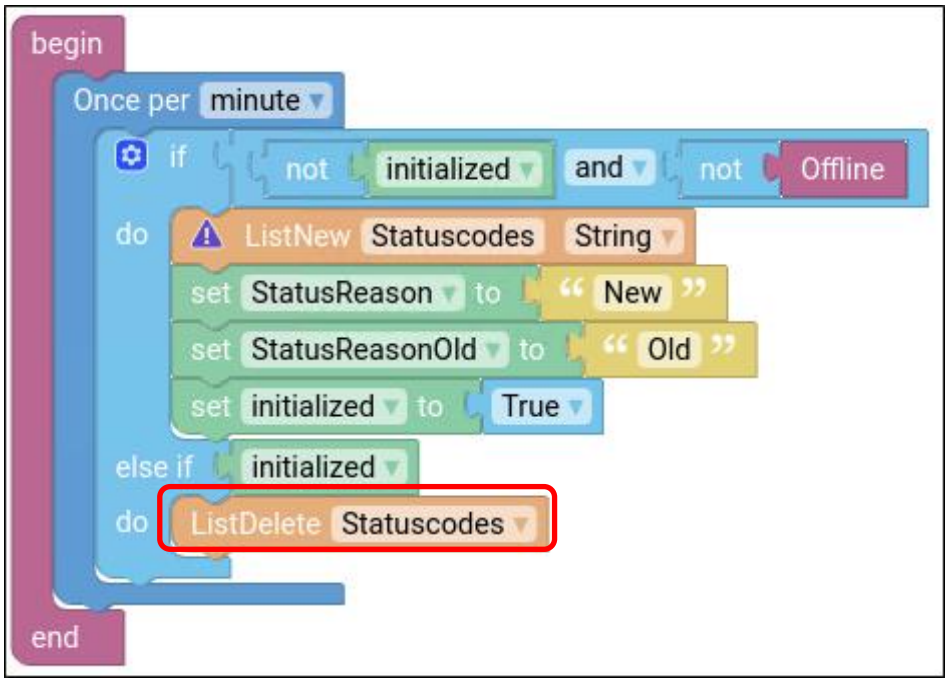
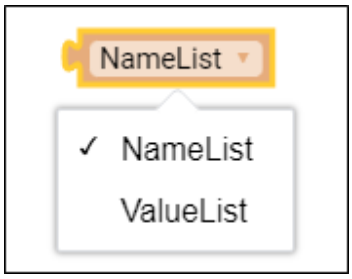


Bild 56: **Beispiel für ListDelete**

In dem Beispiel wird einmal pro Minute überprüft, ob die Maschine zum ersten Mal läuft.
Wenn das Programm nicht ausgeführt (**`not initialized`**) wurde und das Asset nicht offline (**`not offline`**) ist, dann läuft das Asset. Daher werden Listen erstellt mit den aktuellen und alten

Statusgründen. Durch das Anlegen von Listen wird das Programm ausgeführt (**`initialized`**). Damit wird die Variable **`True`** (wahr/1).
Die Liste mit Statusgründen wird mit dem Block **`ListDelete`** gelöscht.

12.5 GetList



Der Block **`GetList`** fügt eine Liste in das Baukastensystem ein. Im Drop-down-Menü wird die bereits erstellte Liste ausgewählt.
Der Input des Blocks ist immer eine vorher erstellt Liste. Diese wird im Drop-down-Menü gewählt.
Einschränkungen für den Output gibt es nicht.

Beispiel

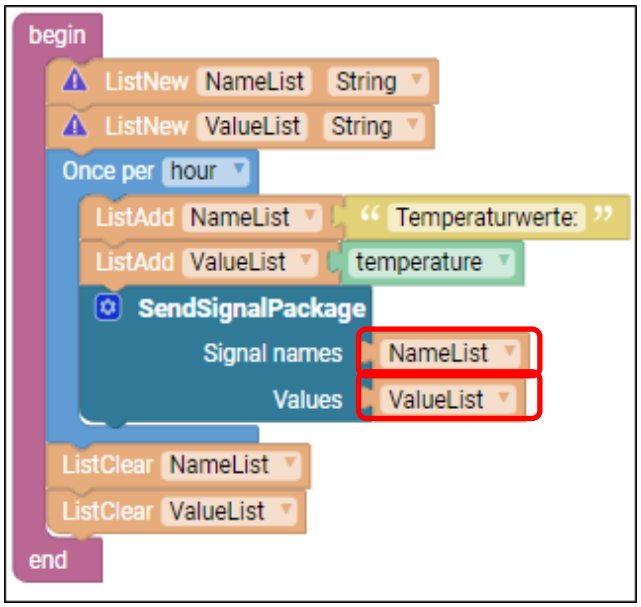


Bild 57: Beispiel für GetList

In dem Beispiel geht es darum zwei Listen zu Temperaturwerten zu erstellen, mit Werten zu füllen zu versenden und wieder zu leeren.
Nachdem die Listen erstellt und ein Mal pro Stunde mit den Temperaturwerten befüllt wurden, werden sie mit **`SendSignalPackage`** versendet. Der Signalname und die Werte werden aus der Namensliste und der Werteliste übernommen.

13 Date and time

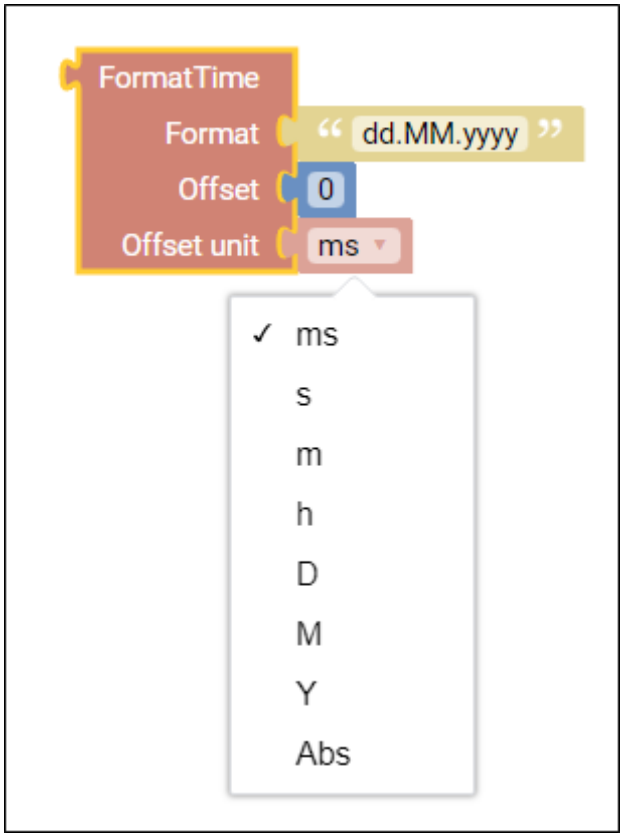
In dieser Funktionskategorie sind alle Aktionen zum Thema Zeit- oder Datumseinstellungen gesammelt. Dabei wird einheitlich die UTC-Zeit verwendet.
Da es verschiedene Abkürzungen der Zeiteinheiten gibt, werden die in der Grafischen Komposition verwendeten in Tabelle 2 aufgeführt.

Tabelle 2: Übersicht der Zeiteinheiten in der Grafischen Komposition

Buchstabe	Datum oder Zeitangabe	Beispiel
G	Ära im Kalender-System	AD
Y	Jahr	2018 (yyyy), 18 (yy)
M	Monat im Jahr	July (MMMM), Jul (MMM), 07 (MM)
w	Woche im Jahr	16
W	Woche im Monat	3
D	Tag im Jahr	266
d	Tag im Monat	4
F	Woche im Monat	4
E	Tag in der Woche	Tuesday, Tue
u	Nummer des Wochentags, wobei 1 für Montag, 2 für Dienstag usw. steht	2
a	AM oder PM	AM
h	Stunde des Tages mit am/pm (1-12)	12
H	Stunde des Tages (0-23)	12
k	Stunde des Tages (1-24)	23
K	Stunde des Tages mit am/pm (0-11)	2
m	Minute pro Stunde	59
s	Sekunde pro Minute	35
S	Millisekunde pro Minute	978
z	Zeitzone	GMT-08:00
Z	Zeitzone offset in Stunden (RFC Muster)	-0800

X	Zeitzone offset im ISO Format	-08;-08:00
E, dd MMM yyyy HH:mm:ss	Beispiel	Tue, 02 Jan 2023 11:22:35

13.1 FormatTime



Der Block **`FormatTime`** erstellt die gewünschte Zeiteinheit der aktuellen Zeit oder des Datums.
Das Format gibt die Einheit des **`Offset`**s an, bspw. dd.MM.yyyy oder MM.dd.yyyy.
Wenn im **`Offset`** die Zahl 0 eingegeben wird, ist die aktuelle Zeit gemeint.
Die **`Offset unit`** bestimmt die Zähleinheit. Mögliche Zähleinheiten sind Millisekunden, Sekunden, Minuten, Stunden, Tage, Monate oder Jahre.
`Abs` bedeutet Absolut time, also der **`Offset`**-Datum 01. Januar 1970 um 00.00Uhr.
Wenn beispielsweise im **`Offset`** 10 eingegeben wird und die Einheit ms ist, dann ist das Ergebnis die aktuelle Zeit plus 10 Millisekunden.
Input für **`Format`** sind Strings. Output sind Strings.
Input für **`Offset`** sind Numbers. Output sind Strings.
Input für **`Offset unit`** bildet ein Dorp-down-Menü. Output sind Strings.

Beispiel

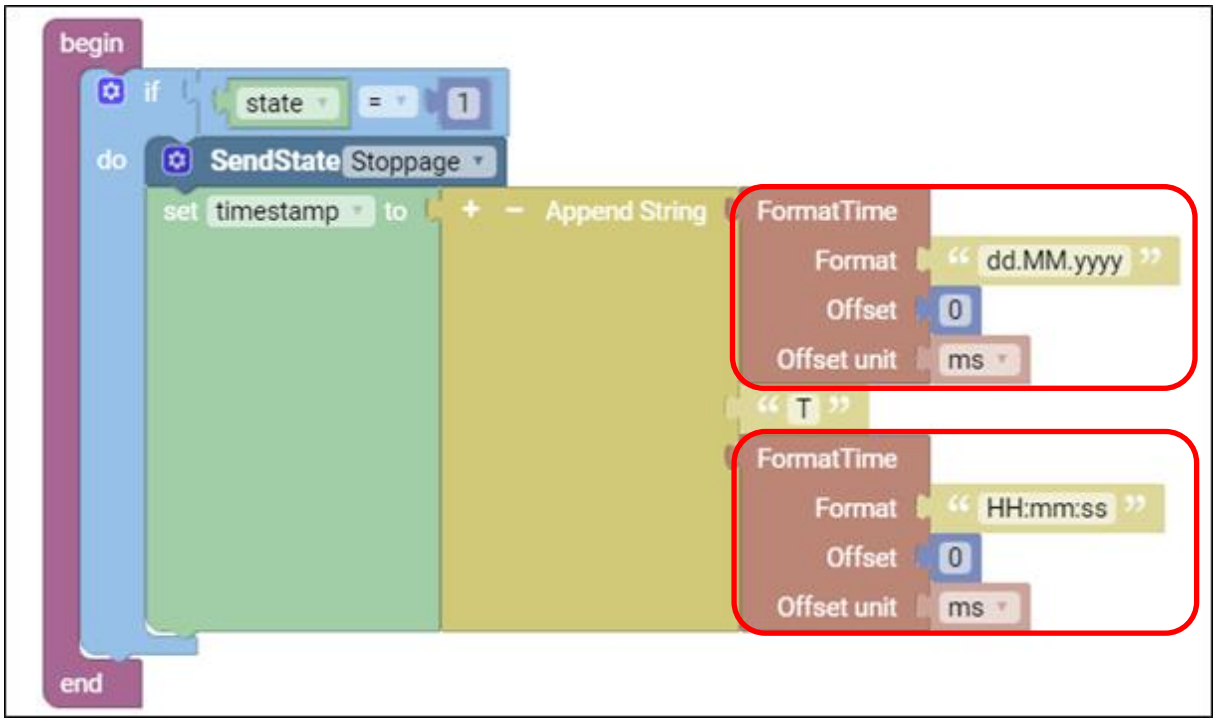
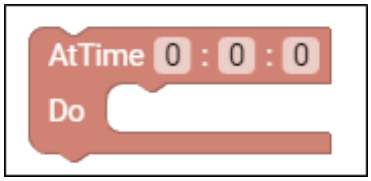


Bild 58: Grafisches Beispiel für FormatTime

In diesem Beispiel soll ein Zeitstempel bei einem Stillstand festgehalten werden.
Wenn der Status gleich eins ist, dann soll **`SendState`** den Status **`Stoppage`** versenden. In diesem Moment soll die Variable **`timestamp`** folgenden String enthalten: Zuerst das Datum in der Reihenfolge: Tag.Monat.Jahr. Danach folgt der Text-String **`T`** für Time. Anschließend wird die Uhrzeit in der Reihenfolge: Stunde:Minute:Sekunde gesetzt.

13.2AtTime Do



Der Block **`AtTime Do`** führt zu einem gewünschten Zeitpunkt eine bestimmte Aktion aus.
Der Zeitpunkt wird in folgender Reihenfolge angegeben: HH : mm: ss. Der Zahlenbereich der Stunden geht von 0 bis 23, der von Minuten und Sekunden von 0 bis 59.
Input sind ausschließlich Numbers.

Beispiel

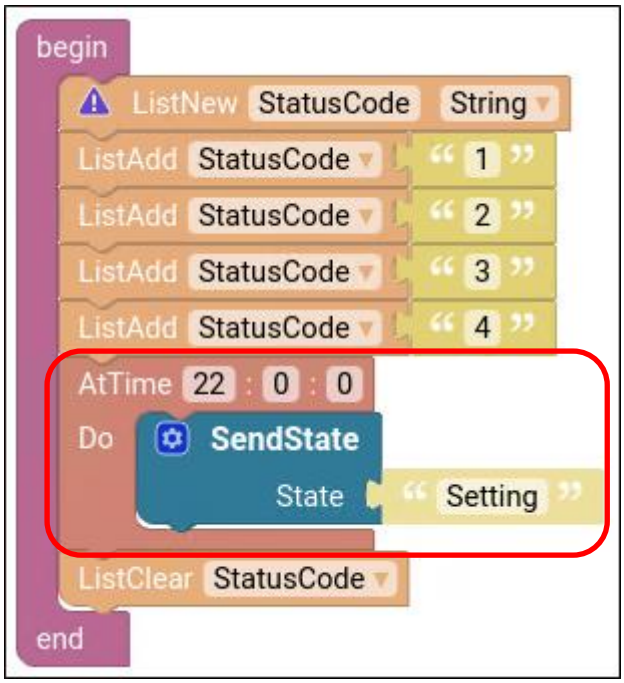


Bild 59: **Beispiel für AtTime Do**

In diesem Beispiel wird gezeigt, dass immer um dieselbe Uhrzeit ein Status gesendet werden soll. Dafür wird mit **`ListNew`** die Liste **`StatusCode`** erstellt. Diese Liste enthält Strings. Im Block **`AtTime Do`** wird die Uhrzeit 22:0:0 Uhr definiert. Zu diesem Zeitpunkt soll die Aktion **`SendState`** ausgeführt werden. Unter **`State`** wird ein String hinzugefügt, der die Aktion bestimmt.
Anschließend wird die Liste wieder geleert.

13.3Sleep

Der Block **`Sleep`** wartet eine gewisse Zeitspanne ab. Der numerische Wert gibt an, wie viele Millisekunden keine Aktion durchgeführt werden soll. Danach beginnt der nächste Block.
Dies ist dann hilfreich, wenn die Durchführung einer Aktion längere Zeit benötigt. So wird sie nicht von den folgenden Aufgaben überholt.
Input sind ausschließlich Numbers. Einschränkungen für den Output gibt es nicht.

Beispiel

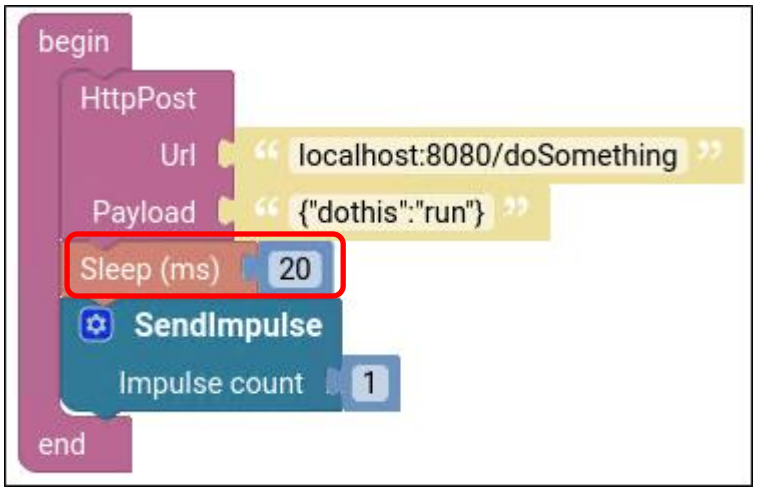
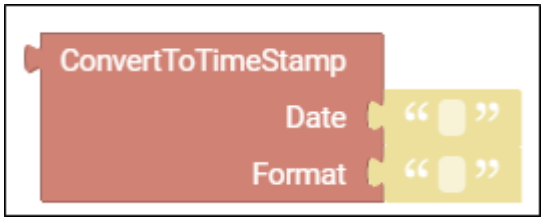


Bild 60: **Beispiel für Sleep**

In diesem Beispiel wird **`Sleep`** als Zeitpuffer verwendet. Ohne die 20 Millisekunden Ruhepause wäre das Senden eines Impulses (**`SendImpulse`**)schneller, als der Endpunkt auf einem Server aufgerufen wäre. Dies würde einen Fehler auslösen.

13.4ConvertToTimeStamp



Der Block **ConvertToTimeStamp** gibt einen Zeitstempel aus. **`Date`** gibt das Datum an, die Systematik des Datums wird darunter in dem String **`Format`** definiert. Output ist ein unix-Wert, also sie Zeit in Millisekunden nach dem 01.01.1970 um 0.00 Uhr.
Input und Output sind ausschließlich Strings.

Beispiel

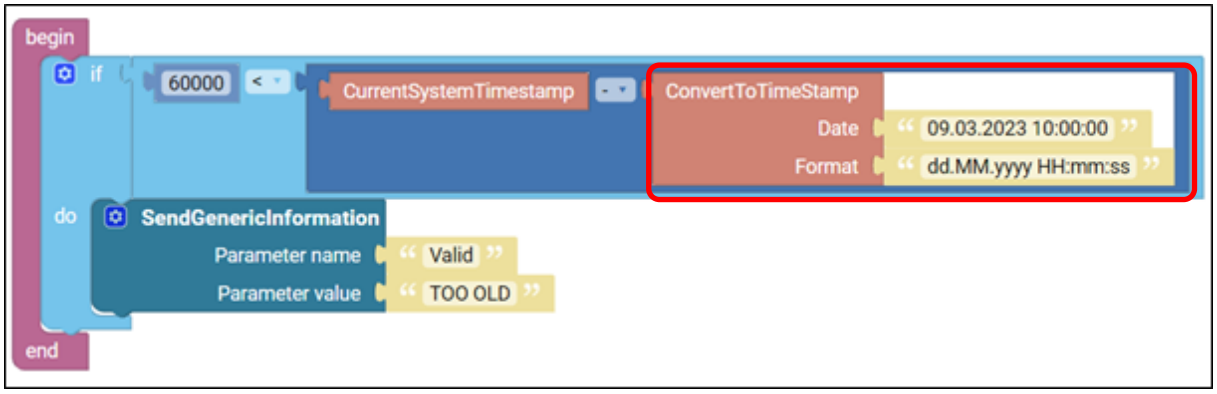
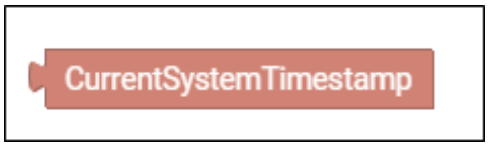


Bild 61: Beispiel für ConvertToTimeStamp

In diesem Beispiel geht es darum, zwei verschiedene Zeitpunkte zu vergleichen.

Wenn der empfangene Zeitpunkt (**`ConvertToTimeStamp`**) mehr als 60.000 ms (also eine Stunde) vom aktuellen Zeitpunkt (**`CurrentSystemTimestamp`**) abweicht, wird eine Meldung mit **`SendGernericInformation`** versendet. Diese sagt, dass der empfangene Zeitpunkt zu alt ist.

13.5 CurrentSystemTimestamp



`CurrentSystemTimestamp` trägt immer die aktuelle Uhrzeit nach den unix-Werten ein. Sie gibt an, wie viele Sekunden seit dem 01.01.1970 vergangen sind.
Einschränkungen für den Input gibt es nicht. Output sind Strings.

Beispiel

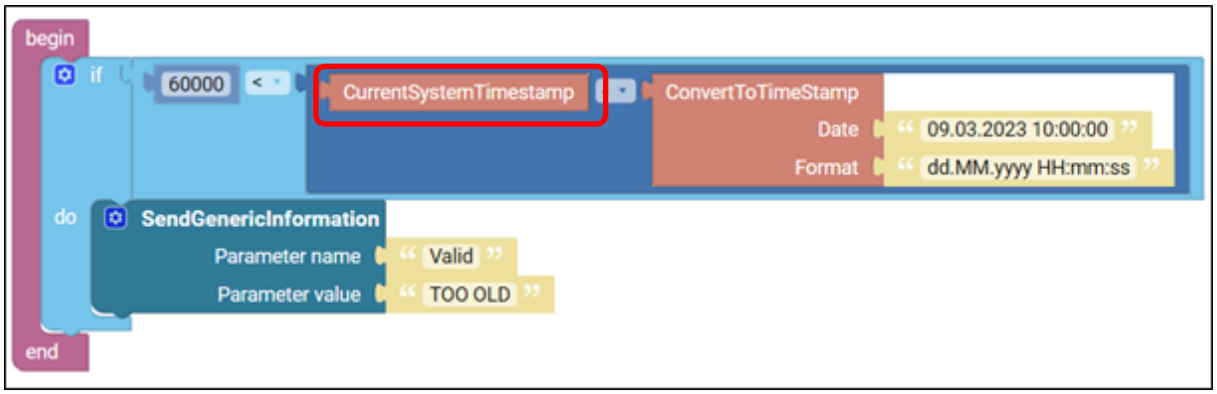


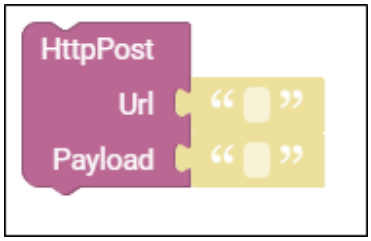
Bild 62: Beispiel für CurrentSystemTimestamp

In diesem Beispiel geht es darum, zwei verschiedene Zeitpunkte zu vergleichen.
Wenn der empfangene Zeitpunkt (**`ConvertToTimeStamp`**) mehr als 60.000 ms (also eine Stunde) vom aktuellen Zeitpunkt (**`CurrentSystemTimestamp`**) abweicht, wird eine Meldung mit **`SendGernericInformation`** versendet. Diese sagt, dass der empfangene Zeitpunkt zu alt ist.

14 Misc

Misc (vom englischen: miscellaneous) und bedeutet so viel wie Sonstiges. In diesem Kapitel sind wichtige Blöcke mit unterschiedlichen Zielen zusammengefasst.

14.1 HttpPost



Der Block **`HttpPost`** sendet eine Nachricht an ein Drittsystem. Unter **`Url`** wird die Internetadresse (das Ziel) eingetragen. **`Payload`** bezeichnet den Teil der übermittelten Daten, das die eigentlich beabsichtigte Nachricht darstellt.
Wir empfehlen die Schreibweise mit den zwei Primes (hochgestellten Anführungsstrichen, z. B.: "k").
Inputs sind Strings. Einschränkungen für den Output gibt es nicht.

Beispiel

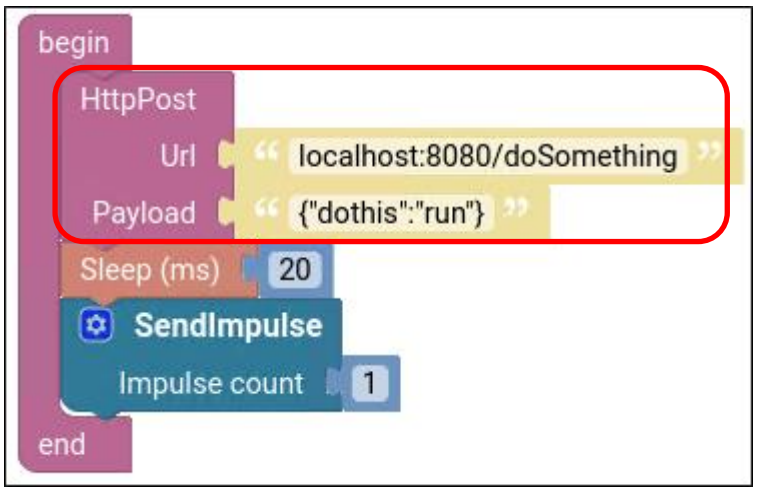
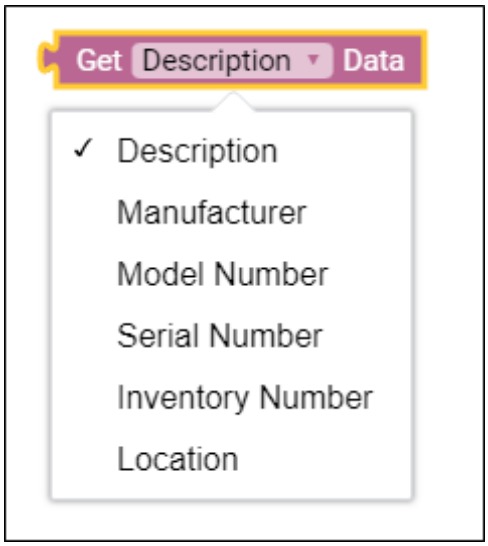


Bild 63: Beispiel für HttpPost

In diesem Beispiel soll ein Kommunikationsendpunkt auf einem Server abgerufen werden. Die dafür vorgesehene **`Url`** und **`Payload`** werden eingegeben.
Dann wartet das Programm 20 ms (**`Sleep`**). Damit die Seite aufgerufen werden kann. Anschließend sendet **`SendImpulse`** den Wert 1.

14.2 Get [specific] Data



`Get [specific] Data` gibt spezielle Informationen aus. Die vordefinierten Daten sind **`Description`**, **`Manufacturer`**, **`Model Number`**, **`Serial Number`**, **`Inventory Number`** und **`Location`**.
Im Configuration Wizard wurden unter Schritt 2 und Schritt 3 bereits Parameter bestimmt. Genauere Informationen dazu sind im Kapitel 3.1 zu finden.
Diese Parameter werden automatisch in das Drop-down-Menü hinzugefügt.
Der Input wird im Drop-down-Menü gewählt. Output sind Strings.

Beispiel

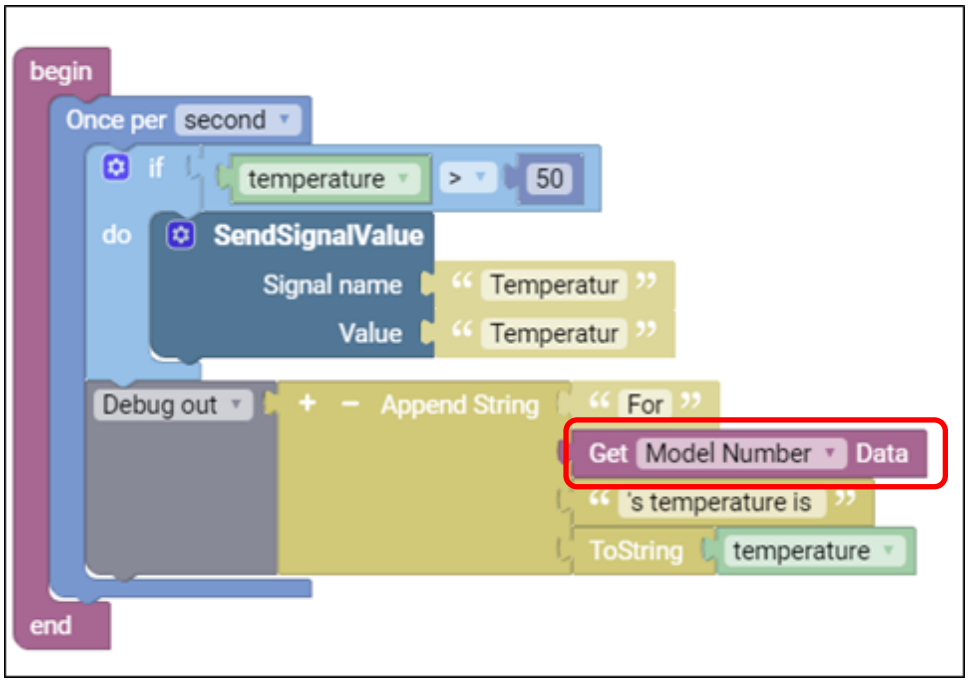


Bild 64: Beispiel für Get [specific] Data

Wenn die Temperatur höher als 50°C ist, dann versendet **`SendSignalValue`** den **`Signal name`** Temperatur mit dem dazugehörenden Temperaturwert (**`Value`**).

Danach erfoglt ein Eintrag in das Logfile. Der Eintrag enthält die Nummer des Assets (**`Get [Model Number] Data`**), den Text „'s temperatur is" und den aktuellen Wert der Variable temperature.

14.3GetMachineStatus

`GetMachineStatus` gibt den aktuellen Maschinenstatus aus.
Einschränkungen für den Input gibt es nicht. Output sind Strings.

Beispiel

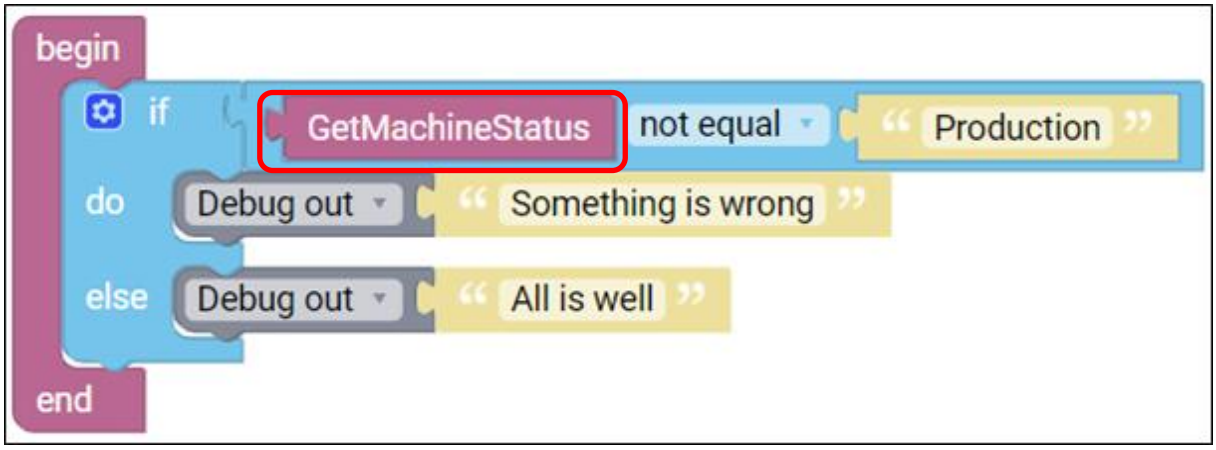


Bild 65: **Beispiel von GetMachineStatus**

Im Beispiel wird mit **`GetMachineStatus`** der Maschinenstatus abgefragt. Wenn dieser nicht gleich (**`not equal`**) **`Production`** ist, wird **`Something is wrong`** ins Logfile geschrieben (**`Debug out`**). Sonst wird die Meldung **`All is well`** ins Logfile geschrieben.

14.4Offline

Wenn ein System oder eine Maschine nicht in Betrieb ist, kann die Statusabfrage **`Offline`** verwendet werden.
Einschränkungen für den Input gibt es nicht. Output sind Booleans.

Beispiel

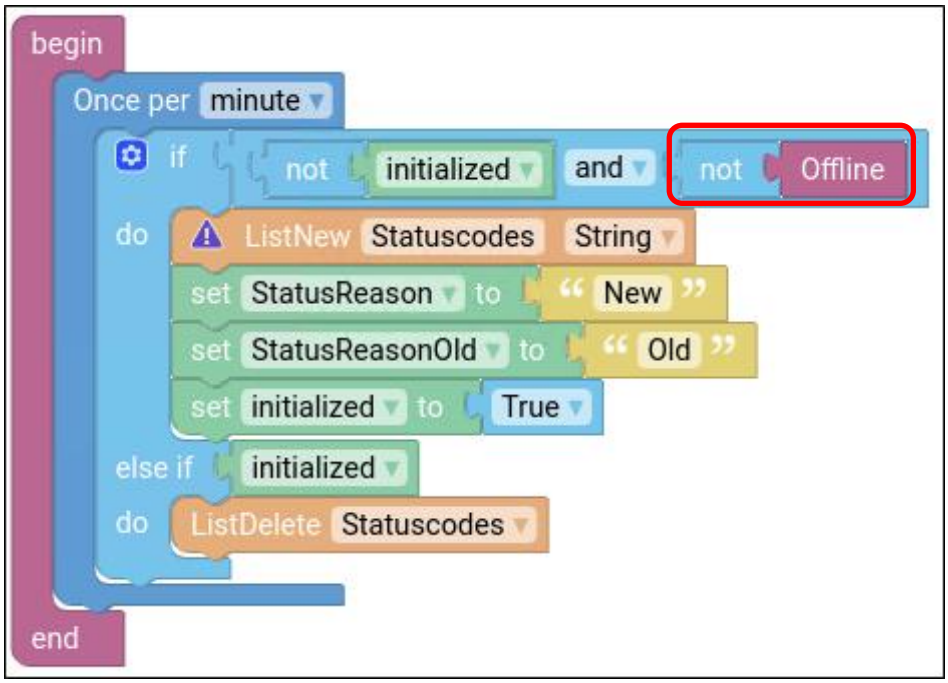


Bild 66: **Beispiel für Offline**

In dem Beispiel wird zu ein Mal pro Minute überprüft, ob das Programm nicht gerade gestartet (**`not initialized`**) wurde und das Asset nicht offline (**`not Offline`**) ist. Wenn das Programm nicht gerade gestartet und das Asset nicht offline ist, dann läuft das Asset. Dann werden Listen mit den aktuellen und alten Statusgründen erstellt. Anschließend wird mit **`True`** bestätigt, dass das Programm gereade gestartet (**`initialized`**) wurde.Dadurch wird der obere Teil der Liste nicht nochmal durchlaufen. Der Block **`ListDelete`** löscht anschließend die Liste der Statuscodes.

14.5IpAddress

`IPAdresse` gibt die IP-Adresse eines Assets aus. Bei der IP-Adresse handelt es sich um eine individuelle Adresse, die ein Gerät im Interent oder auf einem lokalen Netzwerk identifiziert. Einschränkungen für den Input gibt es nicht. Output sind Strings.

Beispiel

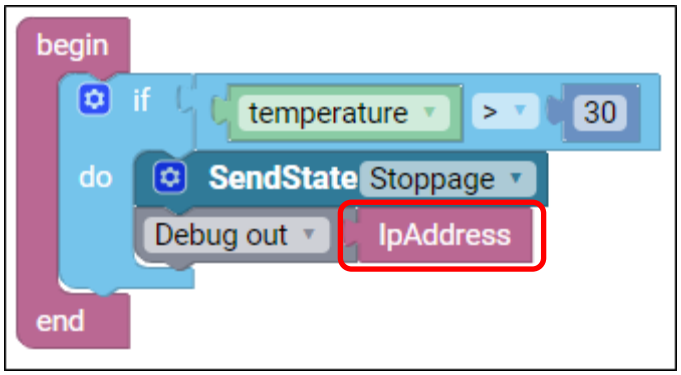


Bild 67: Beispiel für IpAddress

Wenn die Temperatur größer als 30 ist, dann versendet der Block **`SendState`** den Maschinenstatus **`Stoppage`**. Außerdem wird die IP-Adresse (**`IPAdress`**) ins Logfile geschrieben.

14.6HostName

`HostName` trägt den Namen des Hosts eines Assets ein.
Als Host (deutsch: Wirt) wird ein Rechner mit einem dazugehörigen Betriebssystem bezeichnet, der Teil eines Netzwerks ist und seine Leistungen anderen Netzwerkstationen zur Verfügung stellt.
Einschränkungen beim Input gibt es nicht. Output sind Stings.

Beispiel

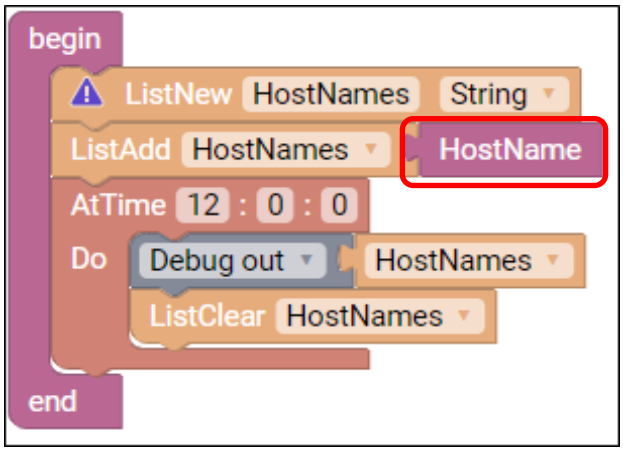
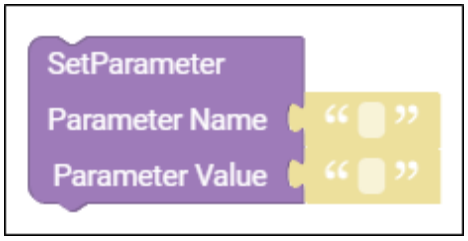


Bild 68: Beispiel für HostName

In diesem Beispiel wird eine neue Liste (**`ListNew`**) erstellt. Dieser werden alle **`HostNames`** hinzugefügt (**`ListAdd`**). Um 12 Uhr wird diese Liste ins Logfile geschrieben und danach geleert.

15 Business Parameters

15.1SetParameter



Der Block **`SetParameter`** gibt einen neuen Parameter an und ordnet diesem einen Wert zu. Der Name und der Wert werden in einem String eingetragen.
Im Configuration Wizard wurden unter Schritt 2 und Schritt 3 bereits Parameter bestimmt. Genauere Informationen dazu sind im Kapitel 3.1 zu finden.
Soll ein bereits definierter Parameter verwendet werden, wird **`GetParameter`** (siehe Kapitel 15.2) genutzt.
Input sind ausschließlich Strings. Einschränkungen für den Output gibt es nicht.

Beispiel

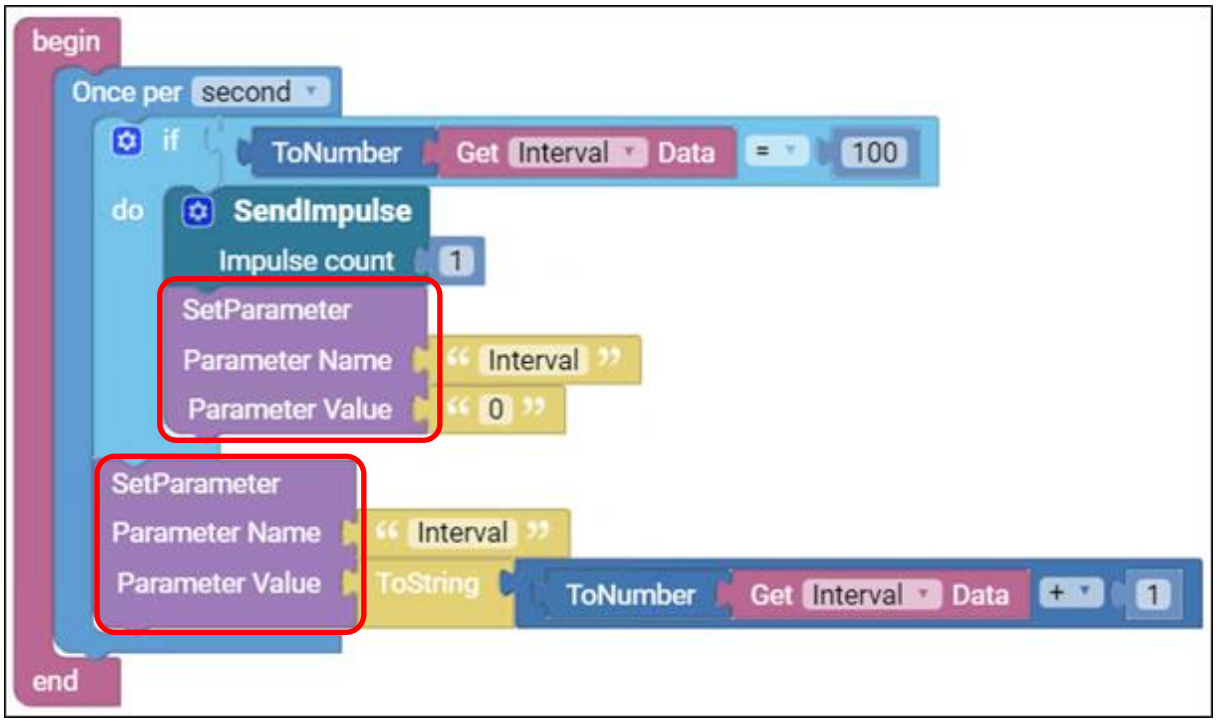
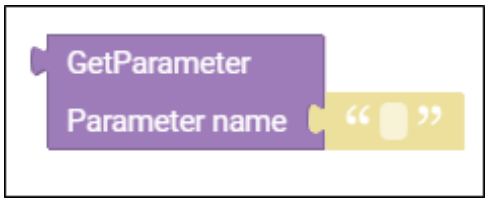


Bild 69: **Beispiel für Set Parameter**

Einmal in der Sekunde wird in dem Beispiel geprüft, ob das Intervall gleich 100 ist. Ist dies der Fall, wird ein Impuls gesendet. Danach wird der **`Parameter Name`** Interval zurück auf den Wert (**`Paramenter Value`**) 0 gesetzt (**`SetParameter`**). Andernfalls wird das Intervall weiter mit 1 hochgezählt.

15.2 GetParameter



Der Block **`GetParameter`** zieht den Wert eines Parameters.
Input und Output sind ausschließlich Strings.

Beispiel

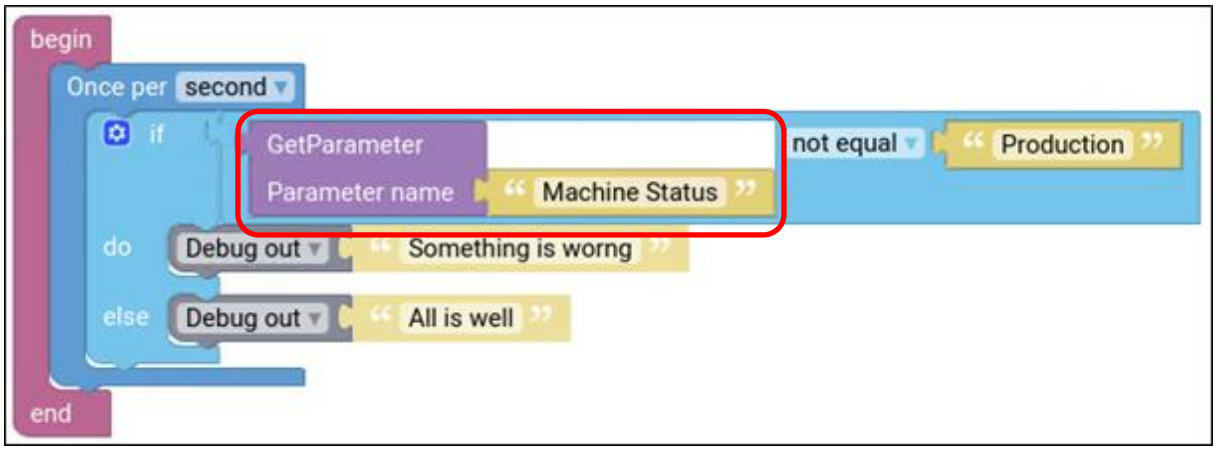
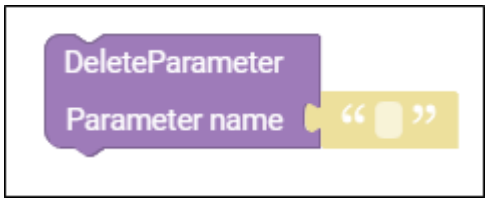


Bild 70: Beispiel GetParameter

In dem Beispiel wird ein Mal pro Sekunde der Maschinenstatus abgefragt. Die Abfrage erfolgt über den Block **`GetParameter`**. Der Name des Parameters (**`Parameter name`**) ist **`Machine Status`**. Wenn der Maschinenstatus nicht gleich (**`not equal`**) **`Production`** ist, dann soll die Meldung **`Something is wrong`** ins Logfile geschrieben werden (**`Debug out`**). Andernfalls wird die Meldung **`All is well`** ins Logfile geschrieben (**`Debug out`**).

15.3 DeleteParameter



Der Block **`DeleteParameter`** setzt den Parameterwert in der Datenbank auf 0 zurück.
Input sind ausschließlich Strings. Einschränkungen beim Output gibt es nicht.

Beispiel

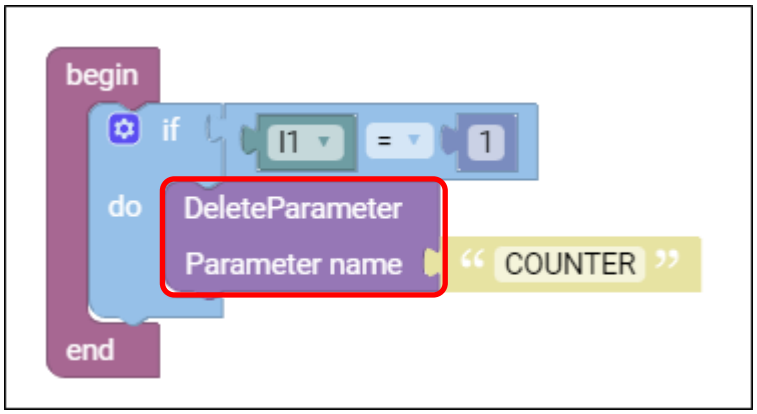


Bild 71**: Beispiel für Delete Parameter**

Wenn das Signal I1 gleich (=) 1 ist, wird die Aussage des Blocks Mathematischer Vergleich = **`True`** (wahr/1). Dann setzt der Block **`DeleteParameter`** der **`Parameter name`** COUNTER auf 0 zurückgesetzt.

15.4 DeleteAllParameter

Der Block **`DeleteAllParameter`** löscht aller Parameter. Verwendet wird der Block wie **`DeleteParameter`**.
Einschränkungen für Input und Output gibt es nicht.

⚠ Alle bereits verwendeten Parameter werden auch auf null zurückgesetzt.

16 Glossar

Abkürzungen und Begriffe	Erklärung
Bit	Die kleinste Speichereinheit für einen Computer: 0 oder 1
ERP	Enterprise Resource Planning (Eine Softwarelösung zur Ressourcenplanung in einem Unternehmen)
Hexa-Dezimalzahl	Ein Zahlensystem, das aus 16 mögliche Ziffersymbolen besteht und zur erleichterten Lesbarkeit von großen Zahlen oder langen Bitfolgen z. B. in der ASCII-Tabelle verwendet wird
IoT	Internet of Things
MES	Manufacturing Execution System
SFT	Shopfloor Terminal
UTC	Coordinated Universal Time
°C	Grad Celsius

17 Anhang

17.1 Übersicht der Parameter

Blöcke	Sonstiges	Input	Output
Variables			
Get [Variable]		N/A	Abhängig von der Auswahl String, Number oder Boolean
Set [Variable] to		Abhängig von der Auswahl String, Number oder Boolean	N/A
Signals			
Set [Signal] to		N/A	N/A
Get Signal		N/A	N/A
Get base / scaled value for		N/A	Number
Events			
SendImpulse			
Impulse count		Number	N/A
Reference	Optional	String	N/A
Cusotmer specific settings	Optional	String	N/A
SendQuantity			
Quantity		Number	N/A
Unit	Optional	String	N/A
Quality details	Optional	String	N/A
Reference	Optional	String	N/A
Customer specific settings	Optional	String	N/A
SendState			
State		String	N/A
Status codes	Optional	String	N/A
Reference	Optional	String	N/A
Customer specific settings	Optional	String	N/A
SendSignalValue			
Signal name		String	N/A
Value		String	N/A
Unit	Optional	String	N/A
Reference	Optional	String	N/A
Customer specific settings	Optional	String	N/A
Timestamp	Optional	String	N/A
SendSignalPackage		String	N/A
Signal name		String	N/A
Value		String	N/A
Unit	Optional	String	N/A
Reference	Optional	String	N/A
Customer specific settings	Optional	String	N/A
SendGenericInformation			
Parameter name		String	N/A
Parameter value		String	N/A
Reference	Optional	String	N/A

Customer specfic settings	Optional	String	N/A
SendState Status codes Reference Customer specific settings	 Optional Optional Optional	 String String String	 N/A N/A N/A
Logical			
If-do If Else if Else Do	 Optional Optional	 Boolean Boolean Boolean Any	 N/A N/A N/A N/A
Mathematischer Vergleich =/≠/</>/≤/≥		Number	Boolean
Logische Verknüpfung and/or		Boolean	Boolean
Logische Verknüpfung equal/not equal		String	Boolean
Rising/Falling edge		Boolean	Boolean
Not-Statement		Boolean	Boolean
Wahrheitsaussage		N/A	Boolean
Repeaters			
Once per		Drop-down-Menü	N/A
Arithmetic			
Nummernfeld		Number	Number
Matheoperation +/- /*/:/sin/cos/tan/sqrt		Number	Number
ToNumber		N/A	Number
Logging			
Logging		String	N/A
Text			
String		N/A	String
Append String		String	String
ToString		N/A	String
Length		String	Number
SplitString Input string Separator Index		 String String Number	 String String Number
FromAscii		Number	String
Substring Input string Start index End index	 Optional	 String Number Number	String N/A N/A N/A
Lists			
ListNew		String	Drop-down-Menü
ListAdd		String	N/A
ListClear		Drop-down-Menü	N/A
ListDelete		Drop-down-Menü	N/A
GetList		N/A	String

Date and time			
FormatTime Format Offset Offset unit		 String Number Drop-down-Menü	String String String String
AtTime Do		Number	N/A
Sleep		Number	N/A
ConvertToTimeStamp Date Format		 String String	Long String String
CurrentSystemTimestamp		N/A	Long
Misc			
HttpPost Url Payload		 String String	 N/A N/A
Get [specific] Data		Drop-down-Menü	String
GetMachineStatus		N/A	String
Offline		N/A	Boolean
IpAddress		N/A	String
Host		N/A	String
Business Parameters			
SetParameter Paramter name Parameter value		 String String	 N/A N/A
GetParameter		String	String
DeleteParameter		String	N/A
DeleteAllParameter		N/A	N/A

17.2 Ascii-Tabelle

Dec	Char	Beschreibung
0	NUL	Keine Eingabe
1	SOH Start of heading	Start der Überschrift
2	STX Start of text	Start eines Textes
3	ETX End of text	Ende eines Textes
4	EOT End of transmission	Abschluss einer Übertragung
5	ENQ Enquiry	Eine Anfrage, die eine Antwort von der Empfangsstation anfordert
6	ACK Acknowledge	Bestätigung
7	BEL Bell	Erzeugt ein hörbares Signal
8	BS Backspace	Verschiebt den Cursor eine Position nach links und entfernt das Zeichen dort
9	TAB Horizontal tab	Tabulator zum horizontalen Einschieben des nächsten Textzeichen
10	LF Line feed	Zeilenumbruch
11	VT Vertical tab	Tabulator zum vertikalen Einschieben des nächsten Textzeichen
12	FF Form feed	Seitenumsprung
13	CR Carriage return	Stellt den Cursor an den Anfang einer Zeile
14	Shift out	Verschiebt den Cursor hinaus
15	SI Shift in	Verschiebt den Cursor hinein
16	DLE Data link escape	Umschaltzeichen
17	DC1 Device control 1	Gerätespezifische Funktion - oft als XON (Übertragung fortsetzen) eingesetzt
18	DC2 Device control 2	Gerätespezifische Funktion
19	DC3 Device control 3	Gerätespezifische Funktion -

Dec	Char	Beschreibung
		oft als XOFF (Übertragung pausieren) eingesetzt
20	DC4 Device control 4	Gerätespezifische Funktion
21	NAK Negative acknowledge	Negative Bestätigung
22	SYN Synchronous idle	Ermöglicht bei synchronen Daten-übertragungen die Synchronisierung auch bei Abwesenheit von zu übertragenden Signalen.
23	ETB End of trans. Block	Kennzeichnet das Ende eines Datenblocks
24	CAN Cancel	Abbruch
25	EM End of medium	Kennzeichnet das Ende eines Mediums.
26	SUB Substitute	Ersetzen
27	ESC Escape	Abbruch einer Tätigkeit
28	FS File separator	Trennung von Hauptgruppen
29	GS Group separator	Trennung von Gruppen
30	RS Record separator	Trennung von Untergruppen
31	US Unit separator	Trennung von Teilgruppen
32	Space	Leerzeichen
33	!	
34	”	
35	#	
36	$	
37	%	
38	&	
39	‘	
40	(	
41	)	
42	*	
43	+	
44	,	
45	-	
46	.	
47	/	

Dec	Char	Beschreibung
48	0	
49	1	
50	2	
51	3	
52	4	
53	5	
54	6	
55	7	
56	8	
57	9	
58	:	
59	;	
60	<	
61	=	
62	>	
63	?	
64	@	
65	A	
66	B	
67	C	
68	D	
69	E	
70	F	
71	G	
72	H	
73	I	
74	J	
75	K	
76	L	
77	M	
78	N	
79	O	
80	P	
81	Q	
82	R	
83	S	
84	T	
85	U	
86	V	
87	W	
88	X	
89	Y	
90	Z	
91	[	
92	\	
93	]	
94	^	
95	_	
96	`	
97	a	
98	b	
99	c	
100	d	

Dec	Char	Beschreibung
101	e	
102	f	
103	g	
104	h	
105	i	
106	j	
107	k	
108	l	
109	m	
110	n	
111	o	
112	p	
113	q	
114	r	
115	s	
116	t	
117	u	
118	v	
119	w	
120	x	
121	y	
122	z	
123	{	
124	|	
125	}	
126	~	
127	DEL Delete	Löschen des letzten Zeichens