ENISCO FORCAM
smart factory experts

FORCE EDGE CONNECT

Grafische Komposition

Einstieg in die Signal-Komposition

Version 240112

Handbuch

	Dokument: Handbuch- Grafische Komposition - Einstieg in die Signal-Komposition
	Freigabedatum: 24.01.2024
	Dokumentversion: 1.0
	Autor: FORCAM GmbH

Inhaltsverzeichnis

1 Über dieses Dokument 5

2 Grafische Komposition in EDGE CONNECT 6

3 Arbeiten mit der grafischen Komposition 8

3.1 Die Benutzungsoberfläche 8

3.2 Blöcke und Funktionskategorien 8

3.2.1 Funktionskategorien 9

3.2.2 Aufbau und Eigenschaften der Blöcke 10

3.2.3 Schattenblöcke 12

3.2.4 Optionale Blöcke zur Erweiterung 13

3.3 Arbeiten im grafischen Editor 13

3.3.1 Leserichtung der Blöcke 14

3.3.2 Blöcke hinzufügen und bearbeiten 15

3.3.3 Schreibweise der Zahlen 16

3.3.4 Fehlererkennung 16

4 Variablen verarbeiten (Variables) 18

4.1 [Variable] - Variable auslesen 20

4.2 Set [Variable] to 21

5 Signalinterpretation (Signals) 23

5.1 Set [Signal] to 24

5.2 [Signal] - Signalwert auslesen 25

5.3 Get base / scaled value for 26

6 Ereignisse definieren (Events) 27

6.1 SendImpulse 27

6.2 SendQuantity 28

6.3 SendState 29

6.4 SendSignalValue 31

6.5 SendSignalPackage 32

6.6 SendGenericInformation 33

6.7 SendState [Auswahl] 35

7 Logische Verknüpfungen herstellen (Logical) 37

7.1 if-do (wenn - dann) 37

7.2 Mathematischer Vergleich (=/≠/<//>/≤/≥) 38
7.3 and/or (Logische Verknüpfung) 39
7.4 equal/not equal (Logische Verknüpfung) 40
7.5 Rising/Falling edge (Flanken erkennen) 41
7.6 Not-Statement (Negation) 42
7.7 True/False (Wahrheitsaussage) 43
8 Schleifen einfügen (Repeaters) 45
8.1 Once per 45
9 Mathematische Operationen (Arithmetic) 46
9.1 Nummernfeld 46
9.2 Matheoperation 46
9.3 ToNumber 47
10 Werte protokollieren (Logging) 48
10.1 Debug out 48
11 Texte erstellen und verarbeiten (Text) 49
11.1 String 49
11.2 Append String 49
11.3 ToString 50
11.4 Length 51
11.5 SplitString 52
11.6 FromAscii 52
11.7 Substring 54
12 Listen erstellen und verwalten (Lists) 55
12.1 ListNew 55
12.2 ListAdd 56
12.3 ListClear 57
12.4 ListDelete 58
12.5 [Liste] - Liste einfügen 59
13 Zeitangaben verwalten (Date and time) 61
13.1 FormatTime 62
13.2 AtTime Do 63
13.3 Sleep 64
13.4 ConvertToTimeStamp 65

13.5 CurrentSystemTimestamp 66
14 Weitere Aktionen (Misc) 67
14.1 HttpPost 67
14.2 Get [specific] Data 68
14.3 GetMachineStatus 69
14.4 Offline 69
14.5 IpAddress 70
14.6 HostName 71
15 Asset-Eigenschaften verarbeiten (Business Parameters) 72
15.1 SetParameter 72
15.2 GetParameter 73
15.3 DeleteParameter 74
15.4 DeleteAllParameter 74
16 Glossar 75
17 Anhang 76
17.1 Übersicht der Parameter 76
17.2 ASCII-Tabelle 79

1 Über dieses Dokument

Dieses Dokument beschreibt die Verwendung des Editors zur grafischen Komposition, mit dem Sie einfach Asset-Signale interpretieren können.

ⓘ Die grafische Komposition ist Teil von FORCE EDGE CONNECT (im Folgenden nur noch EDGE CONNECT genannt und kann daher nur innerhalb dieser Anwendung zur Signalinterpretation verwendet werden.

Das Handbuch erläutert die unterschiedlichen Funktionen, Elemente und Möglichkeiten der grafischen Signalinterpretation anhand der verschiedenen Themengebiete (Funktionsbereiche). Jede Funktion wird anhand eines Praxisbeispiels erläutert.

ⓘ 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.

Zielgruppe

Dieses Handbuch setzt folgende Kenntnisse voraus:

- Kenntnisse in der Maschinenanbindung und -konfiguration mit EDGE CONNECT
  Information dazu finden Sie im Handbuch zur FORCE EDGE CONNECT.
- Grundkenntnisse in der Signalverarbeitung / elektronischen Datenverarbeitung

Sollten Sie dazu keine oder wenige Kenntnisse haben, nehmen Sie sich die Zeit, sich mit den Grundlagen vertraut zu machen.

ⓘ 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 und unterstützt Sie dabei, Veränderungen im Unternehmen einzuleiten und die Technologien richtig einzusetzen.

Weitere Informationen

In unserem **Kundenportal** finden Sie alle Handbücher, Produktbeschreibungen und weitere Informationen zu Ihrem Release.

2 Grafische Komposition in EDGE CONNECT

Damit Signale, die aus Assets (also Maschinen, Sensoren, usw.) ausgelesen wurden, auch ausgewertet werden können, müssen diese Signal zunächst interpretiert werden.
Mithilfe der grafischen Komposition bestimmen Sie einfach und ohne Programmierkenntnisse, welche Signale auf welche Art verarbeitet werden. Sie können so auch definieren, wann Daten an ein MES-, ERP- oder Drittsystem versendet werden. Die grafische Komposition ist damit Teil der Signal Composition, sie kann bei der Signalkomposition anstelle des Skript-Editors verwendet werden.

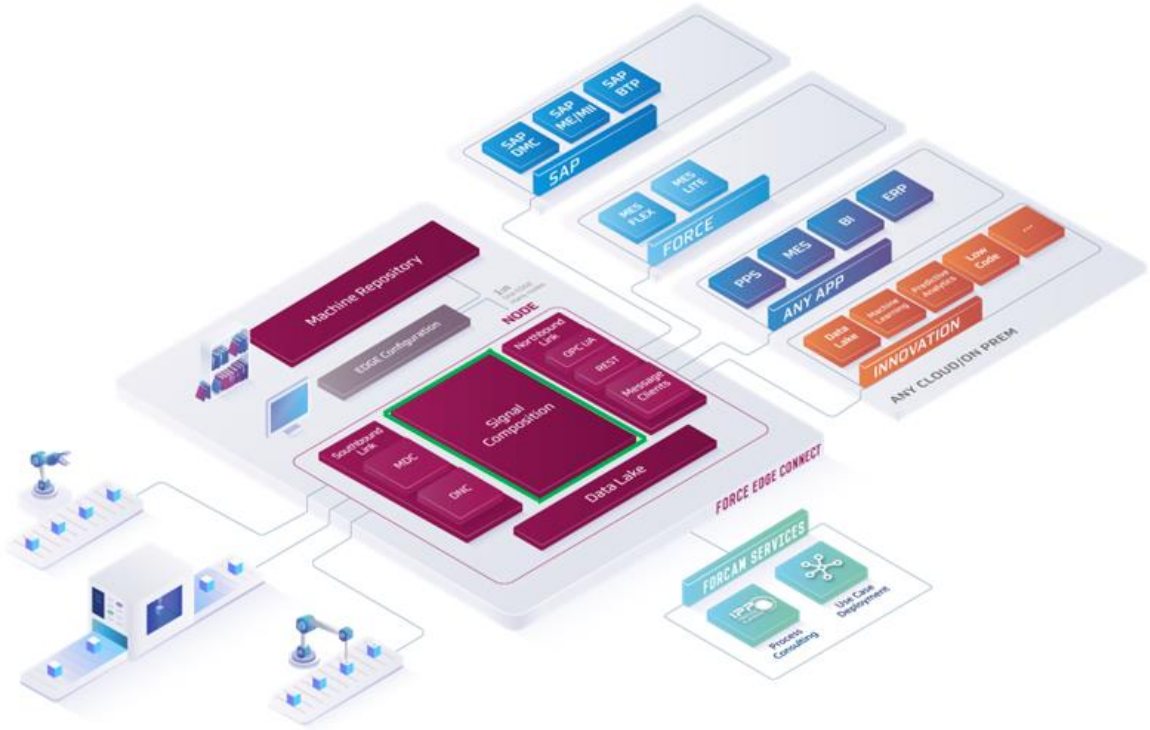


Bild 1: Positionierung der grafischen Komposition in EDGE CONNECT

Grafische vs. skript-basierte Signalinterpretation

In der Komponente Signal Composition werden den Signalen Bedeutungen zugeordnet. So wird beispielsweise aus einem reinen Zahlenwert (wie 0 und 1) eine für Menschen lesbare Information wie „Produktion“ oder „Stillstand“.
Ein Skript ist eine kurze Abfolge von Anweisungen, die vom Programm ausgeführt werden. Die grafische Komposition stellt eine einfache und einsteigerfreundliche Alternative zum klassischen Scripting dar und macht die Signalinterpretation für jedermann zugänglich. Die grafische Komposition wird dabei wie ein Baukastensystem eingesetzt.

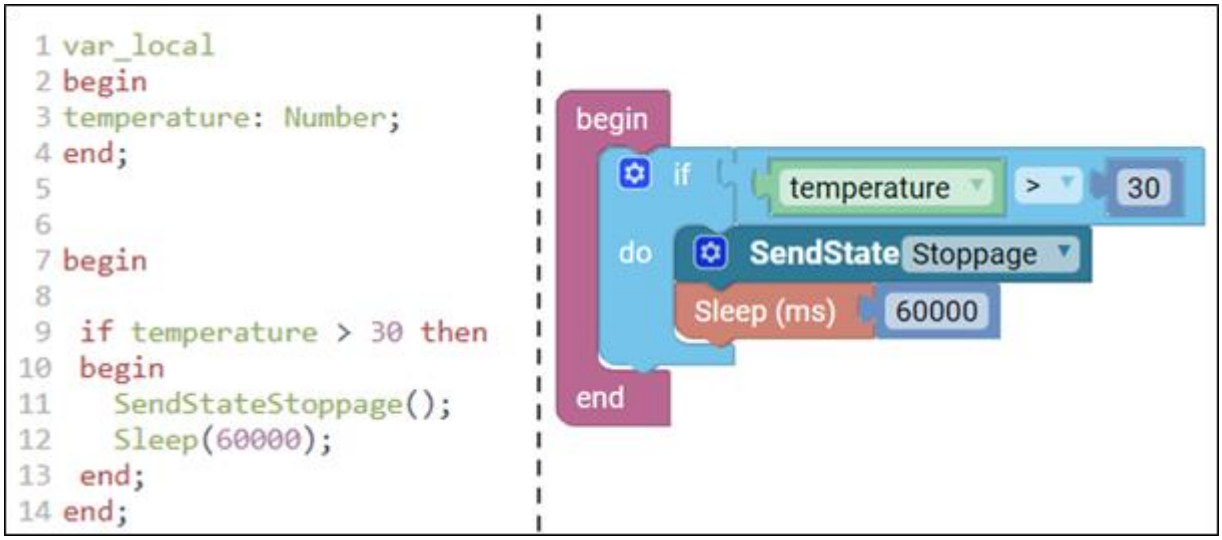


Bild 2: Skriptbasierter vs. grafischer Editor

- Mithilfe von farbigen Puzzleteilen werden die Befehle grafisch dargestellt. Programmiereinsteiger können so ohne Vorkenntnisse in einer Programmiersprache grundlegende Befehle ausführen.
- Anwendungsunterstützte Fehlervermeidung von Anfang an.
  Syntaxfehler können durch die grafischen Elemente ausgeschlossen werden.
  Verschiedene Mechanismen erleichtern das Erkennen von anderen Fehlern.
- Die übersichtliche Darstellung der Kategorien und Funktionen zeigt den Funktionsumfang und erleichtert den Umgang mit dem Editor.
- Die grafische Komposition kann auch mit einem MR-Template kombiniert werden.

ⓣ Allgemeine Informationen zur Signalinterpretation bei der Asset-Konfiguration finden Sie im Handbuch zu FORCE EDGE CONNECT.

3 Arbeiten mit der grafischen Komposition

3.1 Die Benutzungsoberfläche

Die grafische Komposition ist Teil des Configuration Wizard. Hier werden im Schritt **Komposition** die Bedingungen für die Interpretation der Signale definiert.
Den grafischen Editor finden Sie im Reiter **GRAFIK**. Die verschiedenen Programmierbausteine sind hier als grafische Blöcke dargestellt, die baukastenähnlich zusammengesetzt werden können.

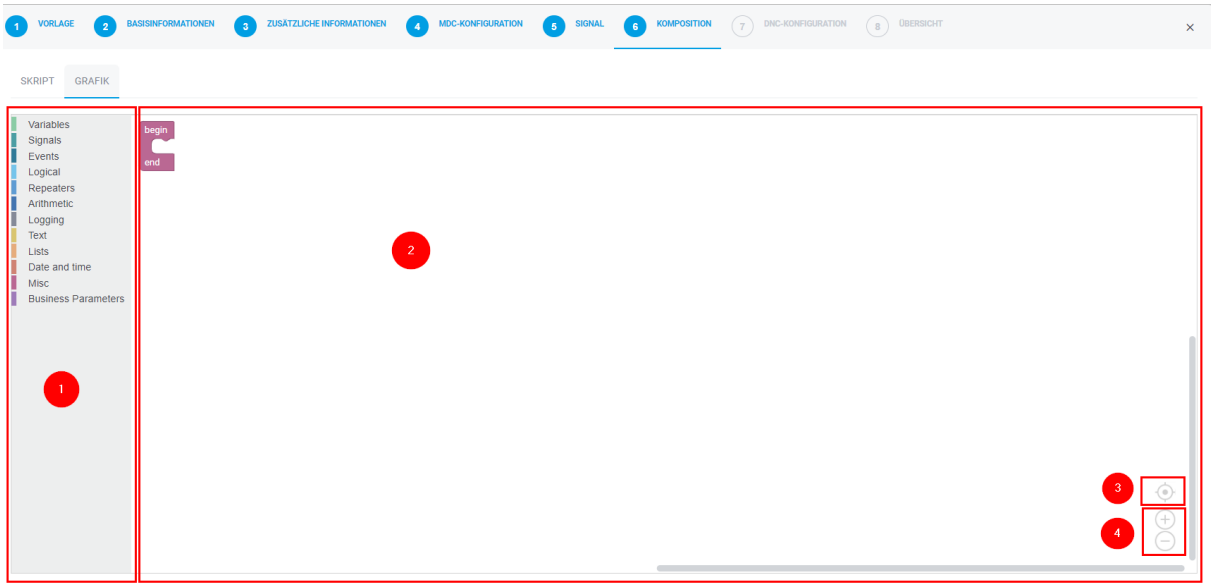


Bild 3: Grafischer Editor

(1) Blöcke über Funktionskategorien auswählen
(2) Blöcke im Bearbeitungsfenster zusammensetzen („Scripting“)
(3) Ansicht zentrieren
(4) Ansicht vergrößern/verkleinern

Allgemeine Informationen zu den verschiedenen Arten von Blöcken und Kategorien finden Sie in den folgenden Abschnitten.

3.2 Blöcke und Funktionskategorien

Jedes „Puzzleteil“ in der grafischen Komposition ist ein Block und entspricht in der Regel einer Funktion bzw. Aktion (Funktionsblöcke). Die Blöcke werden wie in einem Baukastenprinzip zusammengesetzt. Passende „Bausteine“ können untereinander kombiniert werden und ergeben so eine Gesamtstruktur mit allen benötigten Befehlen zur Signalverarbeitung.

Die Blöcke sind entsprechend ihrer Funktion in Funktionskategorien zusammengefasst, z.B. alle Blöcke für die Erstellung und Verwaltung von Listen in der Kategorie *Lists*).

Neben den klassischen Funktionsblöcken gibt es obligatorische Blöcke, über die Werte mitgegeben werden müssen (Schattenblöcke) sowie optionale Blöcke, mit denen zusätzliche Informationen weitergegeben werden können.

3.2.1 Funktionskategorien

Die Funktionsblöcke sind entsprechend ihrer Funktionen/des Themengebiets in verschiedene Kategorien zusammengefasst. Jeder Kategorie ist eine Farbe zugeordnet. Blöcke einer Kategorie besitzen immer dieselbe Farbe. So ist die Aufgabe der Blöcke einfach zu unterscheiden.

Wenn Sie eine Kategorie auswählen, werden rechts davon die zugehörigen Funktionsblöcke angezeigt (siehe “Arbeiten im grafischen Editor“).

Bild 4: Funktionskategorien

Die folgende Tabelle gibt Ihnen eine Übersicht über alle verfügbaren Kategorien und deren Verwendung sowie Links zu den entsprechenden Kapiteln in diesem Handbuch.

Kategorie	Verwendung	Siehe Kapitel
Variables	Variablen anlegen, auslesen und Werte in Variablen schreiben. (Variablen dienen als Container zum Speichern von Daten.)	4 „Variablen verarbeiten (Variables)“
Signals	Signalwerte auslesen, typisieren und umrechnen. (Signale enthalten und übermitteln Informationen, die meist mit einem Sensor gemessen werden.)	5 „Signalinterpretation (Signals)“
Events	Daten (Impulse, Produktionszustände oder Werte) an Drittsysteme versenden	6 „Ereignisse definieren (Events)“
Logical	Werte in einen Zusammenhang bringen Dadurch können Entscheidungen über ihren Wahrheitswert oder Status getroffen werden	7 Logische Verknüpfungen herstellen (Logical)
Repeaters	Aktionen in einem definierten Rhythmus ausführen	8 Schleifen einfügen (Repeaters)
Arithmetic	Rechenfunktionen ausführen (Werte addieren, subtrahieren, multiplizieren) Datenformate umwandeln	9 Mathematische Operationen (Arithmetic)
Logging	Werte gezielt protokollieren und ausgegeben	10 Werte protokollieren (Logging)
Text	Texte erstellen und weiter verarbeiten	11 Texte erstellen und verarbeiten (Text)
Lists	Listen erstellen, füllen, leeren und löschen	12 Listen erstellen und verwalten (Lists)
Date and Time	Alle Aktionen zum Thema Zeit- oder Datumseinstellungen	13 Zeitangaben verwalten (Date and time)
Misc	Sammlung mit weiteren Befehlen und Funktionen zur Kommunikation mit anderen Systemen	13Zeitangaben verwalten (Date and time)
Business Parameters	Spezifische Eigenschaften der Maschine auslesen und weiterverarbeiten (Die zugehörigen Daten werden im Configuration Wizard in den vorherigen Konfigurationsschritten hinterlegt.)	14 Weitere Aktionen (Misc)

3.2.2 Aufbau und Eigenschaften der Blöcke

Alle Arten von Blöcken sind grundsätzlich gleich aufgebaut. Sie unterscheiden sich je nach Funktion in ihrer Farbe und den Möglichkeiten, an andere Blöcke anzudocken.

Verbindungspunkte

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

Verbindungspunkte haben je nach Art des Bausteins verschiedene Funktionen:

Bild 5: Verbindungspunkte

(1) Anbindung weiterer Blöcke möglich
(2) Weitergabe von Eingabedaten (von rechts nach links)

⚠ Eine Zeile muss an der rechten Seite vollständig abgeschlossen sein (keine offenen Verbindungspunkte).

Input und Output

Inputs (Eingaben) sind die Inhalte, die das Programm zur Ausführung benötigt. Der Output (Ausgabe) ist das Ergebnis der Verarbeitung dieser Eingaben und/oder die Befehlsausgabe.

Eine Übersicht aller Vorgaben für Inputs und Outputs der einzelnen Blöcke befindet sich in Kapitel 17.1 „Übersicht der Parameter“.

Datentypen

Jeder Block/jede Variable hat bestimmte Vorgaben zu den erlaubten Formaten (Datentypen) für Ein- und Ausgabewerte.

Folgende Datentypen sind je nach Block / Parameter möglich:

Datentyp	Erklärung
Boolean	Ein Boolean (Boolescher Wert) stellt einen Wahrheitswert dar. Ein Wahrheitswert sagt entweder aus, ob ein Ereignis wahr (True/1) oder falsch (False/0) ist. Oder er gibt an, ob das Ereignis eingetreten (True/1) ist oder nicht (False/0).
String	Zeichenkette aus Zahlen, Buchstaben oder Symbolen. Dieser Datentyp wird zur Darstellung von Texten verwendet.
Number	Zahlen

Vordefinierte Eingabewerte

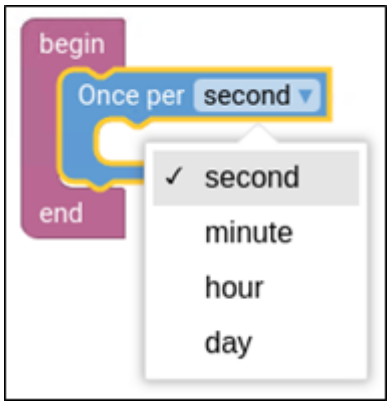


Bild 6: Drop-down-Menü

Einige Blöcke besitzen vorgegebene Eingabewerte (Inputs). Bei einem Klick auf das kleine Dreieck am rechten Rand werden die verfügbaren Möglichkeiten angezeigt.

3.2.3 Schattenblöcke

Schattenblöcke dienen als Platzhalter für Eingaben, die zwingend benötigt werden, damit ein Block seine Funktion/die gewünschte Aktion erfüllen kann. Ein Schattenblock hängt immer schon an einem übergeordneten Funktionsblock, wenn dieser in den Funktionskategorien ausgewählt wird.

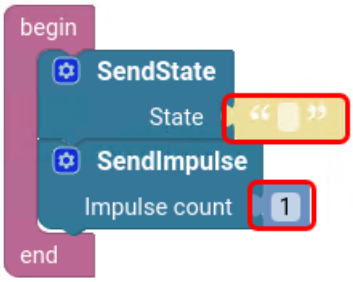


Bild 7: Schattenblöcke

Ein Schattenblock hat eine hellere Farbgebung und zeigt an, dass dieser Parameter eines Blocks nicht leer sein darf. Sie müssen dann entweder manuell einen Wert eingeben (Bild 7) oder den Input durch einen weiteren Block sicherstellen.

ⓣ Ein vorgegebener Schattenblock kann durch einen anderen Block mit gleichem Datentyp ersetzt werden.

3.2.4 Optionale Blöcke zur Erweiterung

Einige Funktionsblöcke können durch weitere Blöcke mit optionalen Parametern erweitert werden. Diese Blöcke haben zusätzlich ein blaues Zahnrad für weitere Einstellungen.

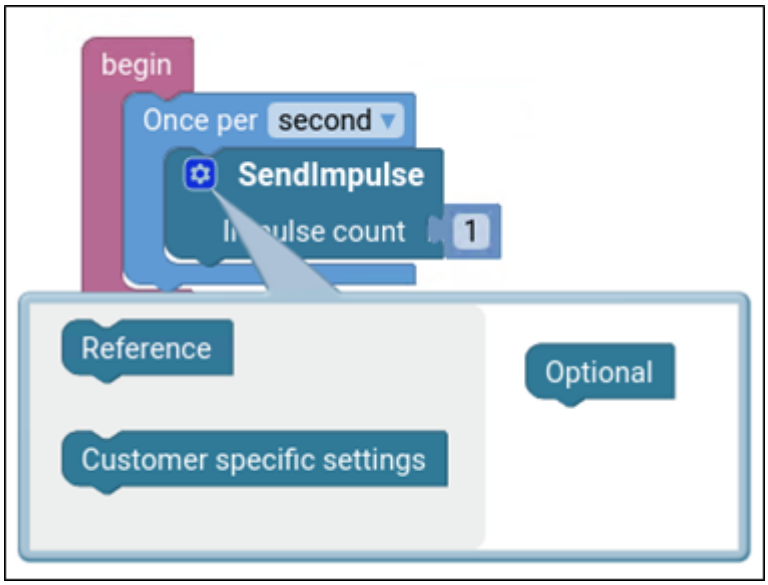


Bild 8: Optionale Parameter an einem Block

Reference

Über diesen Block kann ein frei wählbarer Wert mitgegeben werden, z.B. Text.

Customer-specific settings

Hier können Sie die im Schritt „Kundenspezifische Einstellungen“ des Configuration Wizard angelegten Parameter einfügen. Weitere Informationen zu diesen Parametern finden Sie im allgemeinen Handbuch.

ⓘ Die unteren optionalen Blöcke können nur verwendet werden, wenn auch die übergeordneten Blöcke eingefügt wurden. Im Bild oben kann **`Customer specific settings`** also nur verwendet werden, wenn zuvor der Block **`Reference`** eingefügt wurde.

3.3 Arbeiten im grafischen Editor

Jede Komposition beginnt und endet mit dem Rahmen **begin…end.**

Dieser Block erscheint immer automatisch im Feld. Die weitere Reihenfolge der Bausteine innerhalb der Struktur können Sie im Rahmen der Funktionen selbst bestimmen. Jeder Baustein steht für sich und hat seine Aufgabe.
Um die zugehörige Aktion ausführen zu können, benötigt ein Block ggf. weitere Informationen (Input) von anderen Blöcken, die Sie dazu anfügen müssen. Das Ergebnis ist eine in sich konsistente Struktur von Befehlen zur Signalverarbeitung.

3.3.1 Leserichtung der Blöcke

Gelesen werden die Blöcke in der Reihenfolge von oben nach unten und von links nach rechts.

Beispiel: Leserichtung von oben nach unten

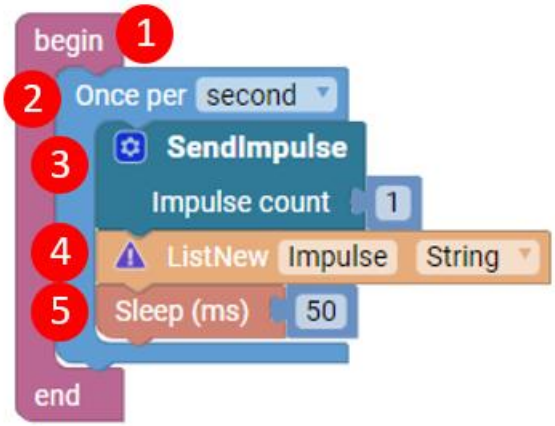


Bild 9: Beispiel für die Abarbeitung von oben nach unten

Der Block begin…end (1) bildet den äußeren Rahmen jeder Konstruktion. Danach werden die Funktionsblöcke entsprechend der Nummerierung abgearbeitet.

Beispiel: Leserichtung von links nach rechts

Die einzelnen Zeilen werden wie in einem Buch von links nach rechts gelesen und abgearbeitet:

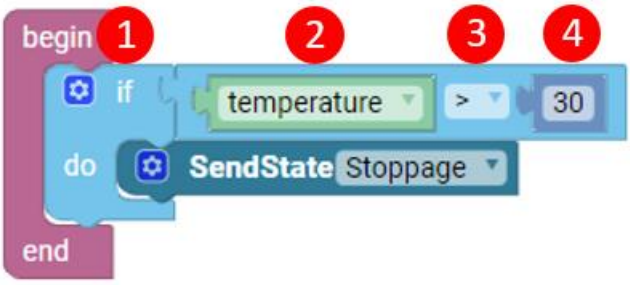


Bild 10: Beispiel für die Abarbeitung von links nach rechts

Es startet mit **if** (1), danach folgt der grüne Block **temperature** (2), dann das mathematische Zeichen **>** (3) und anschließend die Zahl **30** (4).

Danach wird die nächste Zeile abgearbeitet: Diese fängt mit **do** an, danach wird innerhalb des dunkelblauen Blocks ebenfalls von links nach rechts gelesen. Folglich ist es irrelevant, dass ein Block zwei Zeilen einleitet, die Reihenfolge des Lesens bleibt dieselbe.

3.3.2 Blöcke hinzufügen und bearbeiten

Block hinzufügen

Um einen Block hinzuzufügen, wählen Sie im linken Bereich die gewünschte Kategorie und ziehen den Block dann per Drag & Drop in den Bearbeitungsbereich.
Der gerade ausgewählte Block ist immer gelb umrandet.

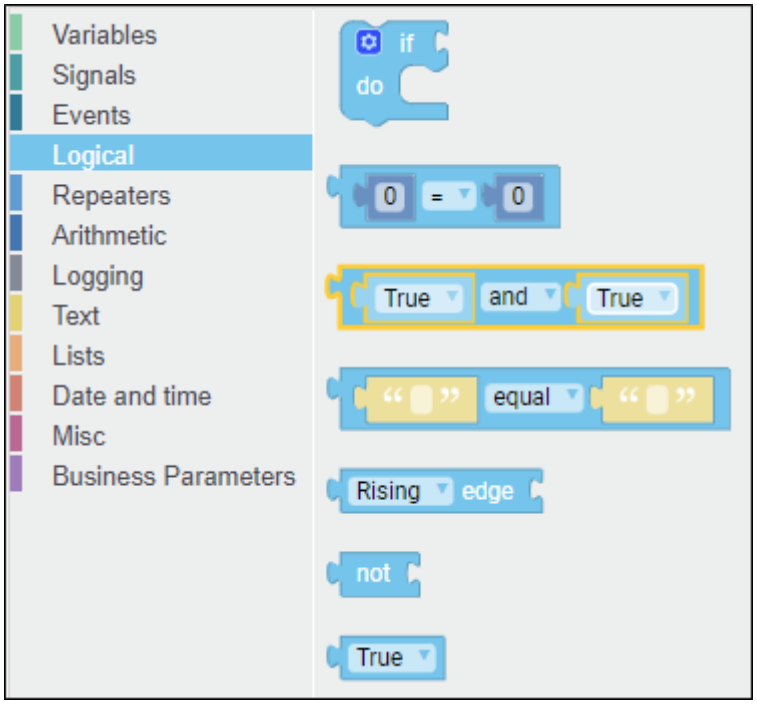


Bild 11: Block über Funktionskategorie auswählen

Im Bearbeitungsbereich können Sie die Blöcke ebenfalls per Drag & Drop an die gewünschte Position verschieben.

ⓣ Zur einfachen Handhabung können die Blöcke mit den Tastaturkombinationen STRG+C und STRG+V kopiert und eingefügt werden.

Blöcke löschen

Blöcke können über die ENTF-Taste aus der Struktur bzw. dem Bearbeitungsfenster entfernt werden oder über das Kontextmenü des Blocks (siehe unten).

Weitere Aktionen für einen Block

Über einen Rechtsklick auf den Block sind weitere Aktionen möglich:

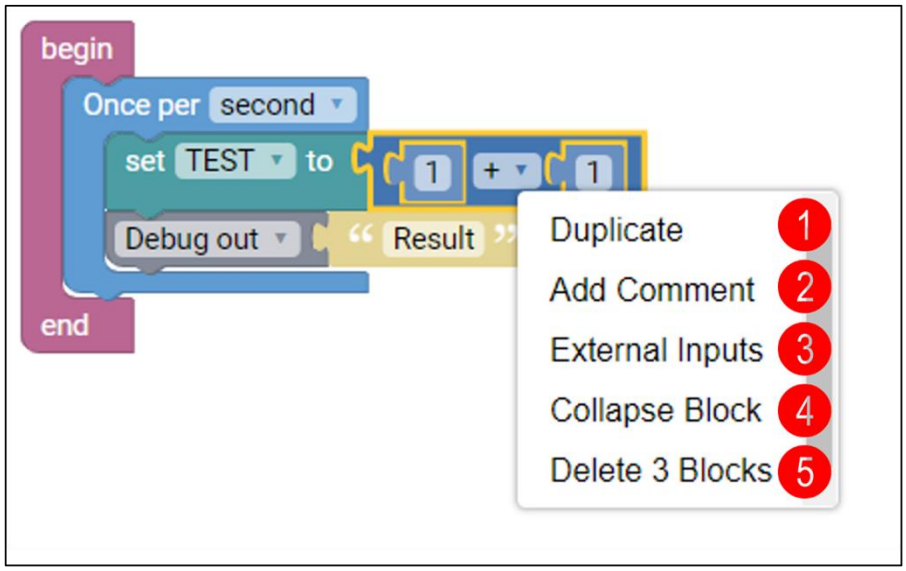


Bild 12: Möglichkeiten an Aktionen eines Blocks

(1) Block duplizieren
(2) Kommentar hinzufügen
(3) External Inputs/Inline Inputs: Ändert die Darstellungsform
(4) Block einklappen
Um Platz zu sparen und das Gesamtbild übersichtlich zu halten, können Sie die Ansicht für einen Block (und ggf. die untergeordneten Blöcke) reduzieren.
(5) Blockgruppen löschen

3.3.3 Schreibweise der Zahlen

In der grafischen Komposition wird die englische Schreibweise der Zahlen verwendet. Beachten Sie dies bei der Verwendung von Punkt und Komma:

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

3.3.4 Fehlererkennung

Ein Fehler in der Struktur wird auf zwei Arten deutlich gemacht:

Block kann nicht eingefügt werden

Wenn ein Block in seiner Funktionskategorie oder dem verwendeten Datentyp nicht in die Struktur passt, kann er an dieser Stelle nicht eingefügt werden. Ein typischer Fehler wäre zum Beispiel, dass als Eingabeformat ein String benötigt wird, der Block aber Daten vom Datentyp Number enthält.

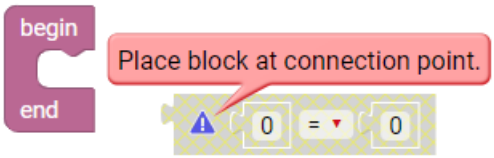


Bild 13: Fehler - ungültiger Block

Block ist unvollständig

Ein eingefügter Block bleibt rot, solange noch notwendige Informationen (Input) fehlen.
Sobald der Input vollständig ist (Werte gefüllt oder Blöcke eingefügt), erhält der Block wieder seine ursprüngliche Farbe.

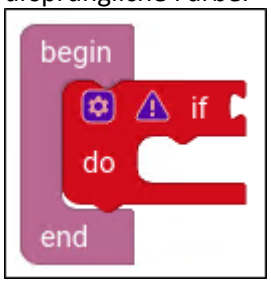


Bild 14: Fehler - unvollständiger Block

Nur ein korrekter Block wird zugelassen.

⚠ Mit einem Klick auf das Ausrufezeichen wird der Fehlergrund angezeigt und ist somit schnell erkennbar (siehe Bild 13).

4 Variablen verarbeiten (Variables)

Variablen dienen als Container zum Speichern von Daten. Dies können statische Werte (Assetname, Status, usw.) oder Ergebnisse von Berechnungen (Temperaturwert, Druck, Zeiteinheit, usw.) sein.

Eine Variable kann immer nur eine zugewiesene Art von Inhalten speichern. So können z.B. reine Zahlen nicht in einer Variable für Texte gespeichert werden. Die Art des Inhalts wird über den Datentyp der Variablen festgelegt. Diese Typen haben Beschränkungen, was ihren Inhalt (Zahlen, Wörter usw.) und ihre Größe (wie groß/klein usw.) betrifft. (Weitere Informationen zu Datentypen unter „Aufbau und Eigenschaften der Blöcke“).

ⓘ Ändert sich der Inhalt einer Variablen, wird die Veränderung im System registriert. Alle Blöcke, die mit dieser Variable arbeiten, übernehmen automatisch den neuen Wert.

Umgang mit Variablen

Variablen müssen zunächst in der Kategorie angelegt werden, bevor sie in der grafischen Komposition verwendet werden können. Die Kategorie ist daher zunächst leer.
Klickt man innerhalb der Funktionskategorien die Kategorie **Variables** an, erscheint die Auswahl der Bausteine und/oder der Button zum Anlegen einer neuen Variablen.

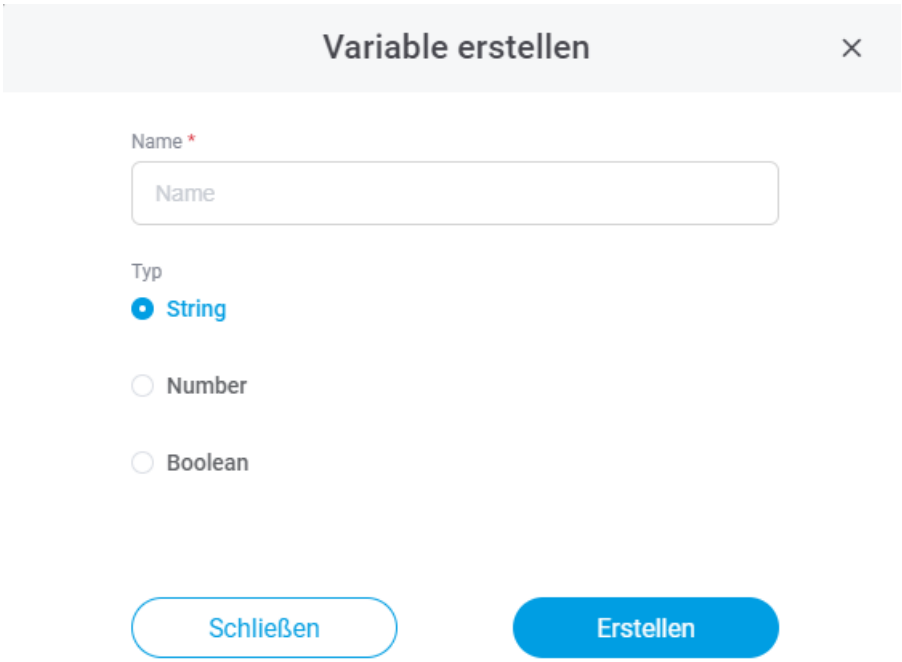


Bild 15: Neue Variable erstellen

Hier wird neben dem Namen auch der Datentyp der Variable (Number, String oder Boolean) festgelegt.

ⓘ Angelegte Variablen gelten immer nur für die aktuelle Konfiguration, können innerhalb der Struktur aber beliebig oft verwendet werden. Der Name der Variable muss innerhalb der Konfiguration eindeutig sein.

Bereits angelegte Variablen werden zur besseren Übersicht nach Datentyp sortiert angezeigt:

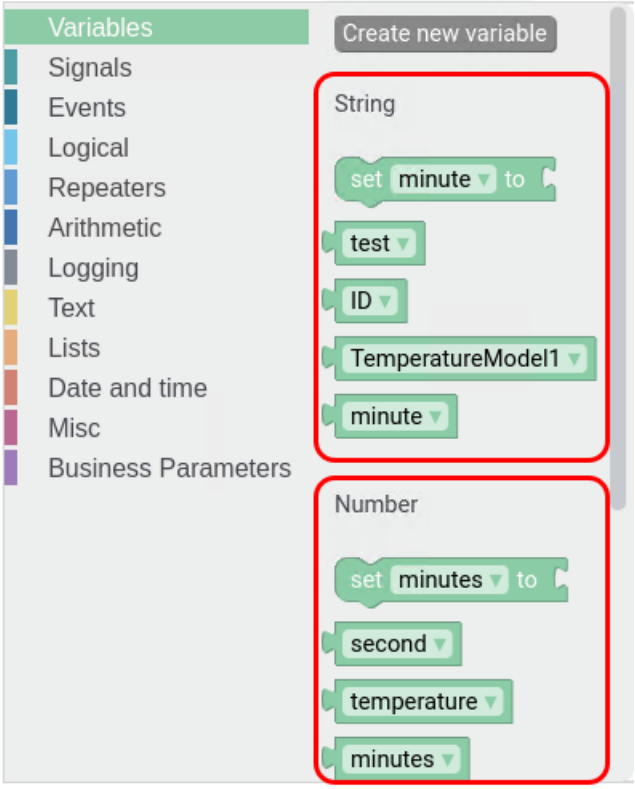


Bild 16: Übersicht der Variablenarten

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

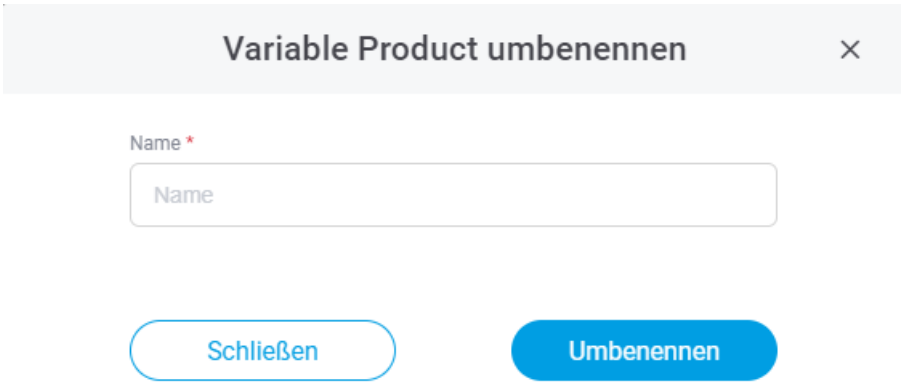


Bild 17: Variable umbenennen

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

⚠ Jede gelöschte Variable wird aus der kompletten Struktur und aus der Funktionskategorie gelöscht. Dies kann zu einer ungültigen Struktur führen.

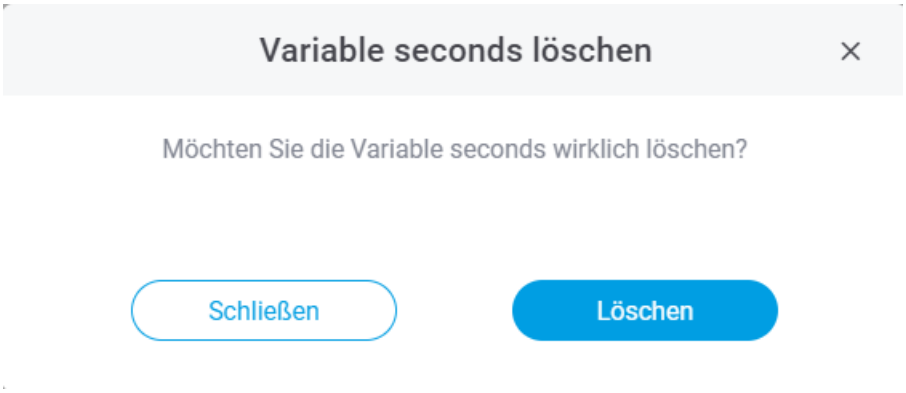


Bild 18: Löschen einer Variable

4.1 [Variable] - Variable auslesen

Verwendung

Um eine Variable in der Struktur 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.

Input/Output

Erlaubte Input-Typen	Output
Keine Einschränkungen	entspricht dem Typ der erstellten Variable

Beispiel

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

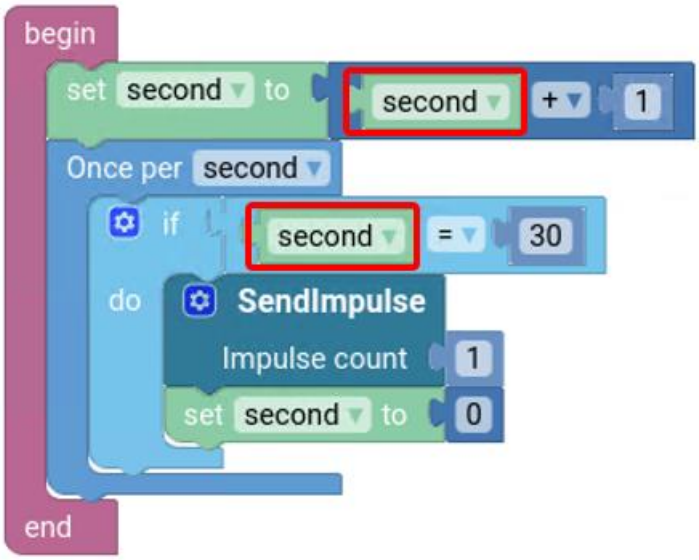


Bild 19: Beispiel zu [Variable]

4.2 Set [Variable] to

Verwendung

`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.

Datentypen

Erlaubte Input-Typen	Output
entspricht dem Typ der erstellten Variable	entspricht dem Typ der erstellten Variable

Beispiel

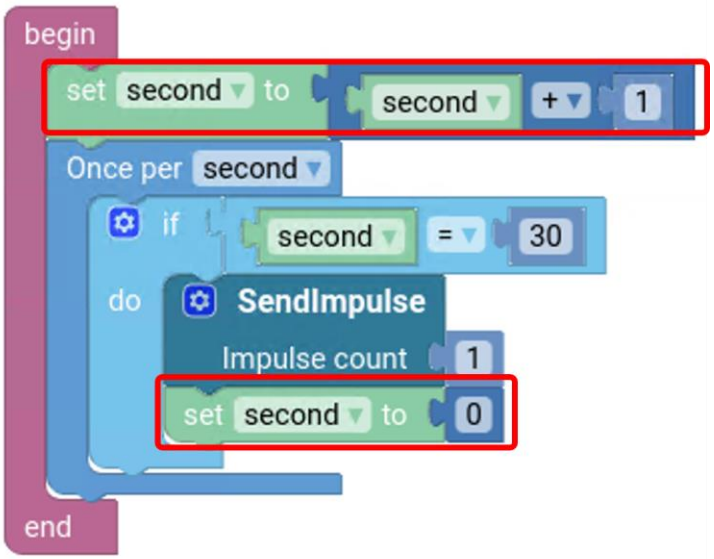


Bild 20: Beispiel zu set [Variable] to

Einmal pro Sekunde soll 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** den Wert 30 erreicht, soll ein Impuls versendet werden. Am Ende wird erneut die Sekundenanzahl auf 0 zurückgesetzt.

5 Signalinterpretation (Signals)

Signale werden meist mit Sensoren an den Assets gemessen und übertragen die gewonnenen Informationen an EDGE CONNECT. Typische Signale im produzierenden Gewerbe sind Intervalle, Temperaturen, Maschinenzustände oder Ergebnisse von Druckmessungen.

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.

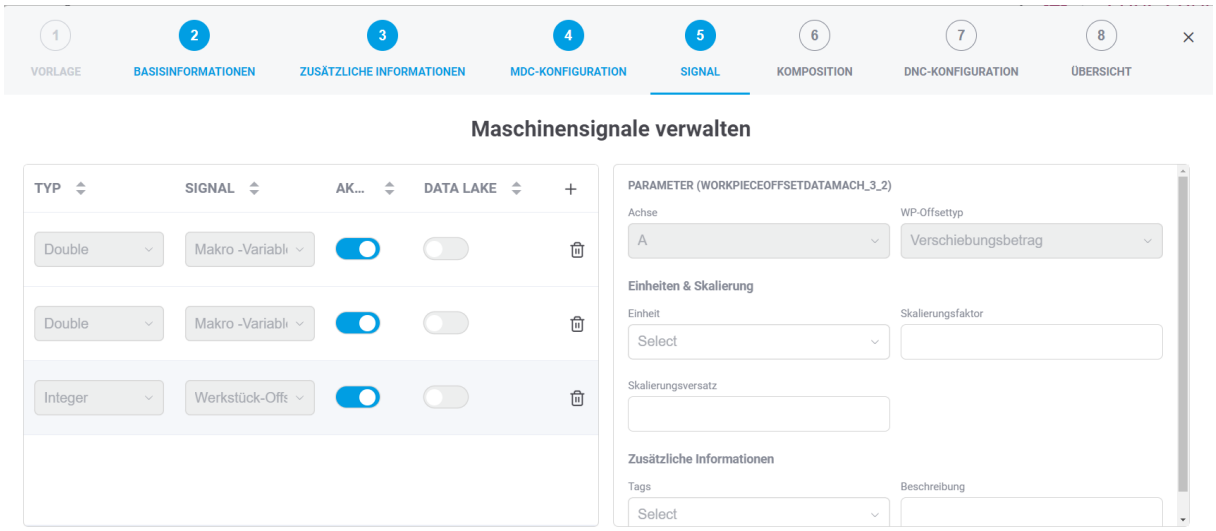


Bild 21: Hinzufügen eines neuen Signals

5.1 Set [Signal] to

Verwendung

`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.

Datentypen

Erlaubte Input-Typen	Output
Keine Einschränkungen	Keine Einschränkungen

Beispiel

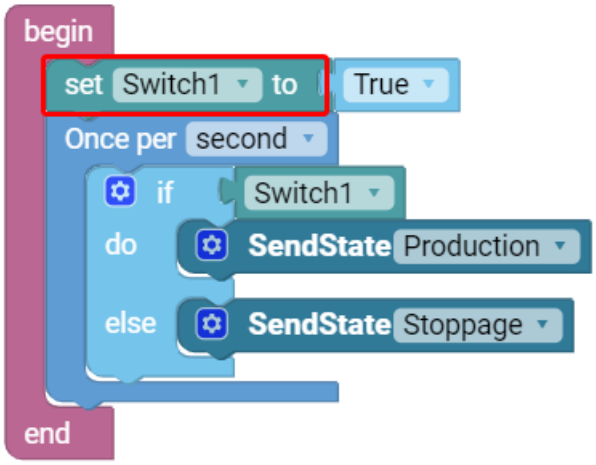
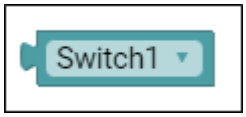


Bild 22: Beispiel für Set [Signal] to

Zu Beginn ist **`Switch1`** (deutsch: Schalter) auf **`True`** (wahr/1) geschaltet. 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 [Signal] - Signalwert auslesen



Verwendung

Um Signale in der Struktur zu verwenden, wird dieser Block benötigt. Er liest den Wert des Signals ab. Über das Drop-down-Menü kann das gewünschte Signal ausgewählt werden.

Datentypen

Erlaubte Input-Typen	Output
Keine Einschränkungen	Keine Einschränkungen

Beispiel

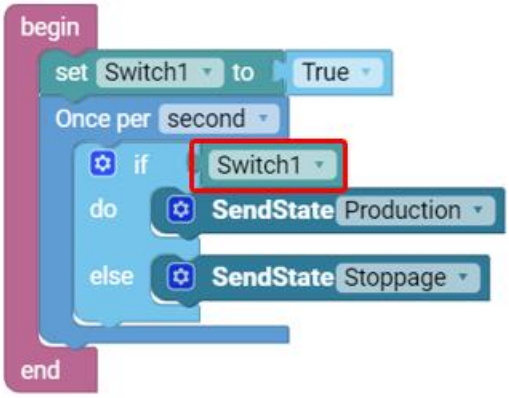


Bild 23: **Beispiel für [Signal]**

Wird der **`Switch1`** (deutsch: Schalter) auf **`True`** (wahr/1) gesetzt, läuft ein Mal pro Sekunde 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

Verwendung

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 **`scaled value`** ist der Eingangswert multipliziert mit dem scale factor (deutsch: Skalierungsfaktor) und dem scale offset (deutsch: Skalierungsversatz).

Datentypen

Erlaubte Input-Typen	Output
Keine Einschränkungen	Keine Einschränkungen

Beispiel

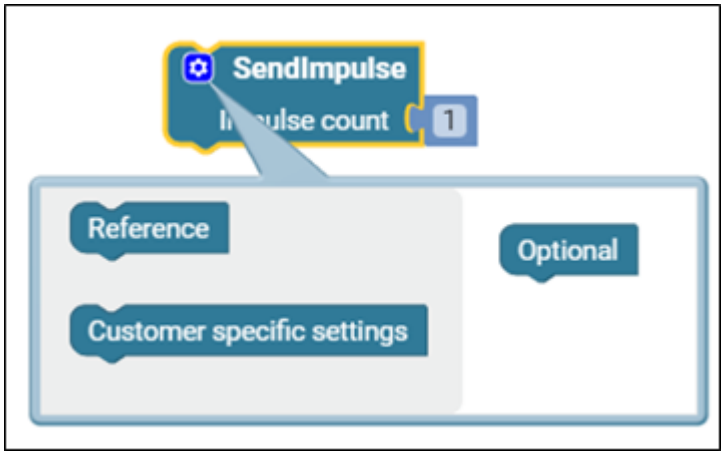
Bild 24: Beispiel für Get base value for

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 Ereignisse definieren (Events)

Events (deutsch: Ereignisse) senden Informationspakete an Drittsysteme. In der grafischen Komposition wird bestimmt, welche Informationen in diesen Paketen enthalten sind (Impulse, Produktionszustände oder Werte).

6.1 SendImpulse



Verwendung

Wenn ein bestimmter Impuls gesendet werden soll, wird der Block **`SendImpulse`** 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.

Input/Output

Der Input für **`Impulse count`** sind ausschließlich Numbers.
Input für alle weiteren Angaben sind ausschließlich Strings. Einschränkungen für den Output gibt es nicht.

Beispiel

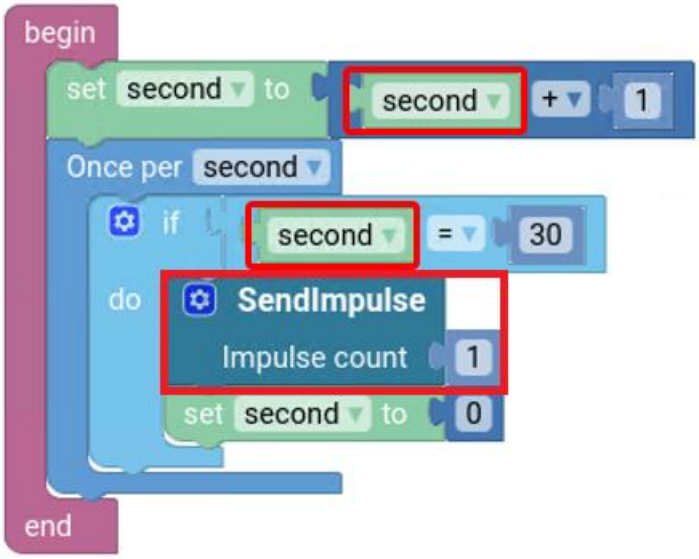
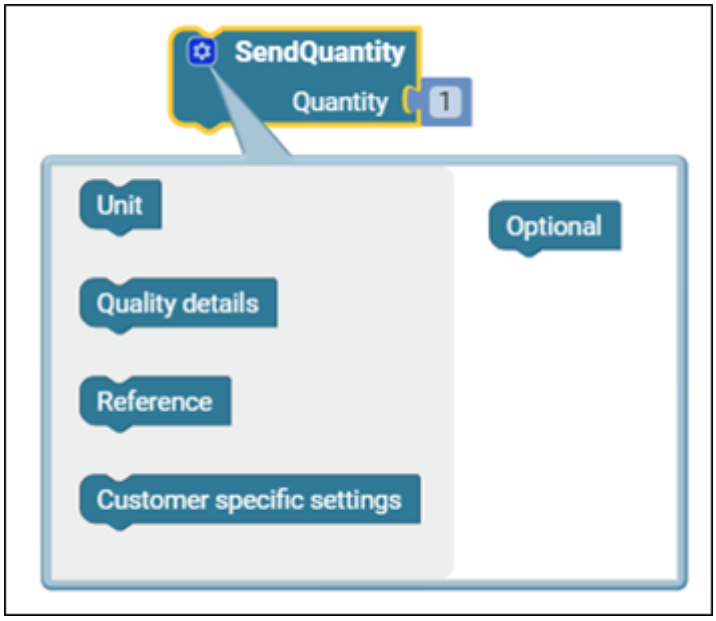


Bild 25: **Beispiel zu SendImpulse**

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



Verwendung

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/Output

Input für **`Quantity`** sind ausschließlich Numbers.
Input für alle weiteren Angaben sind ausschließlich Strings.

Beispiel

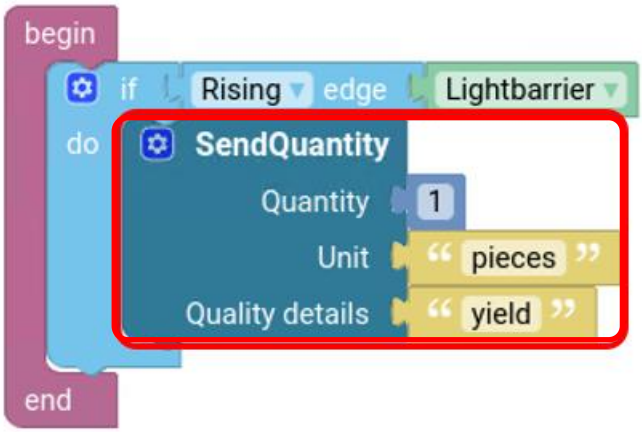
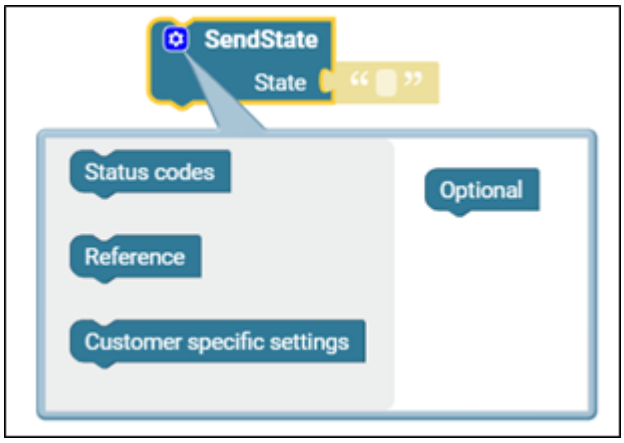


Bild 26: Beispiel für SendQuantity

Wenn eine Lichtschranke ausgelöst wird, dann soll der Block **`SendQuantity`** 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 details`**) gut (**`yield`**) besitzt.

6.3 SendState



Verwendung

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, auch die Liste der **`Status codes`**, **`Reference`** und **`Customer specific settings`** zu versenden. Die Angaben dazu wurden im Configuration Wizard vorgenommen (siehe „

Optionale Blöcke zur Erweiterung“).

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

Input/Output

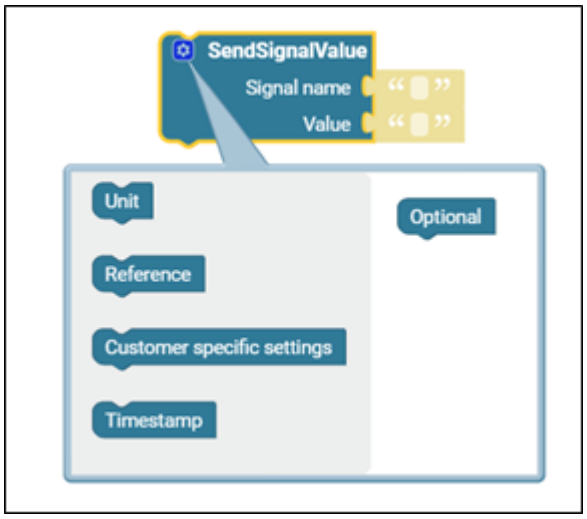
Input sind ausschließlich Strings. Einschränkungen für den Output gibt es nicht.

Beispiel

Bild 27: 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



Verwendung

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/Output

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

Beispiel

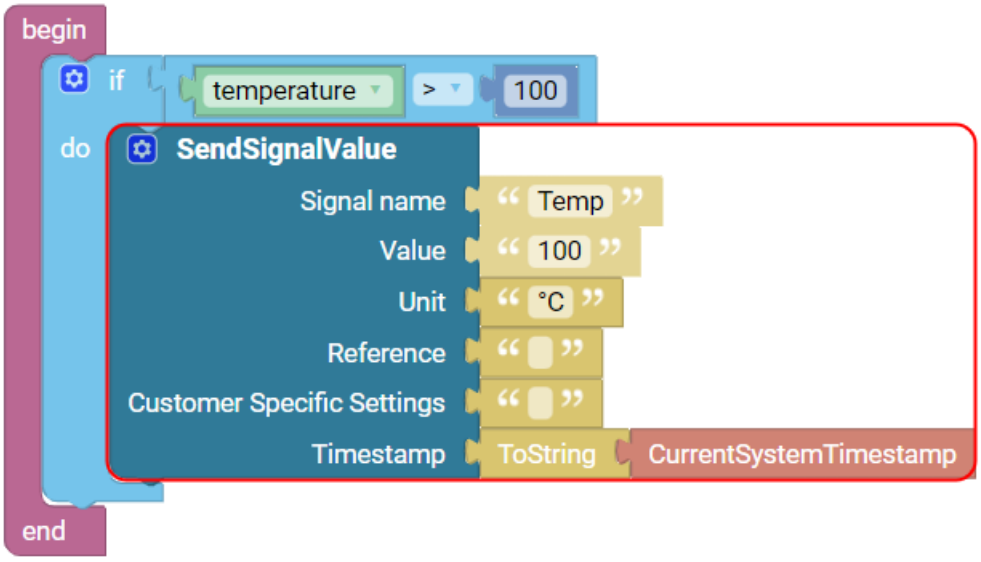
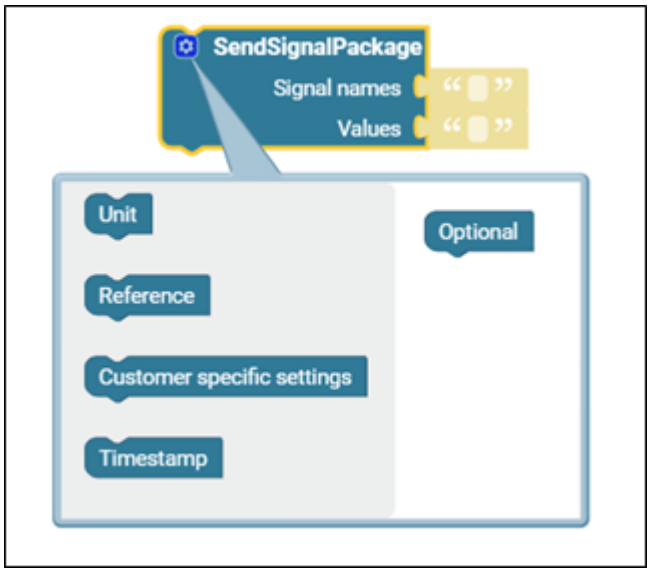


Bild 28: Beispiel für SendSignalValue

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 (**`Temp`**), den Wert (**`100`**), die Einheit des Werts (**`°C`**) und den Zeitpunkt, an dem der Grenzwert überschritten wurde (**`CurrentSystemTimestamp`** – aktuelle Systemzeit). Informationen zu **`Reference`** oder **`Customer specific settings`** sind optional, diese können leer bleiben.

6.5 SendSignalPackage



Verwendung

`SendSignalPackage` sendet Listen mit Signalen. Die Inhalte stammen aus zuvor definierten Listen (siehe „Listen erstellen und verwalten (Lists)“).
Weitere Signale können in der Namensliste mit den passenden Signalwerten hinzugefügt werden.

ⓘ Reihenfolge beachten:
Der erste Eintrag in der Signalliste muss dem ersten Eintrag in der Werteliste entsprechen.

Input/Output

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

Beispiel

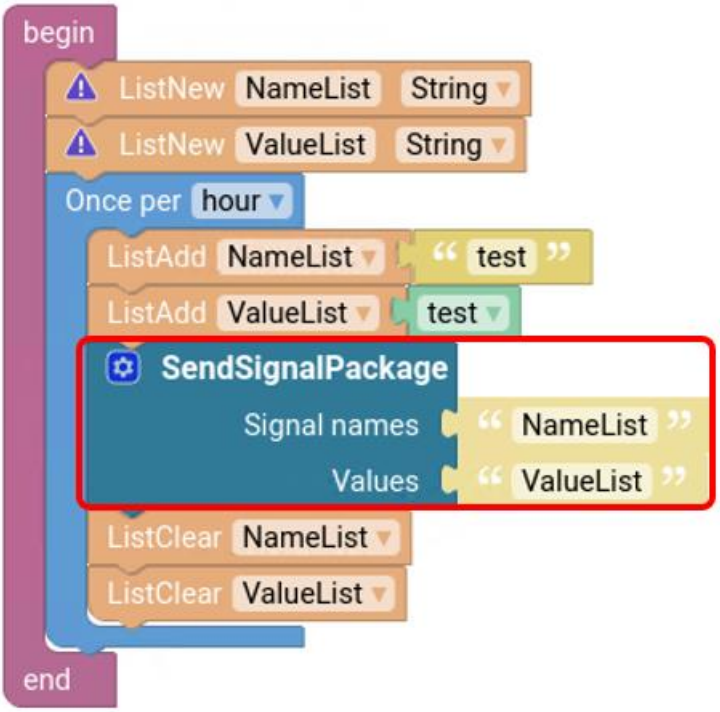
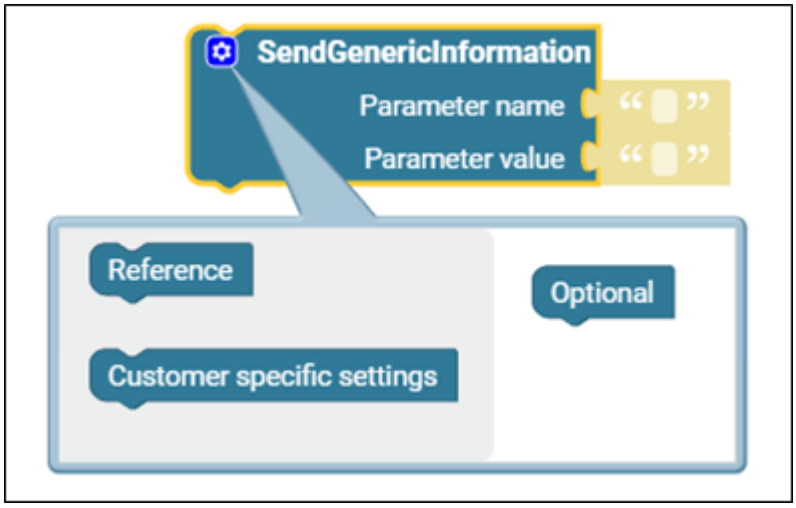


Bild 29: Beispiel für SendSignalPackage

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



Verwendung

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/Output

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

Beispiel

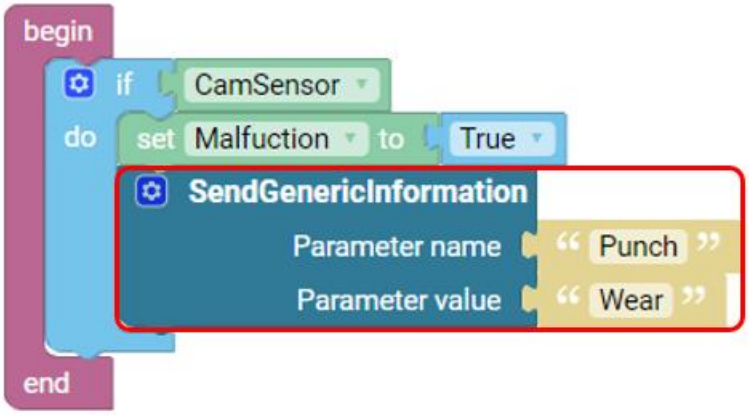
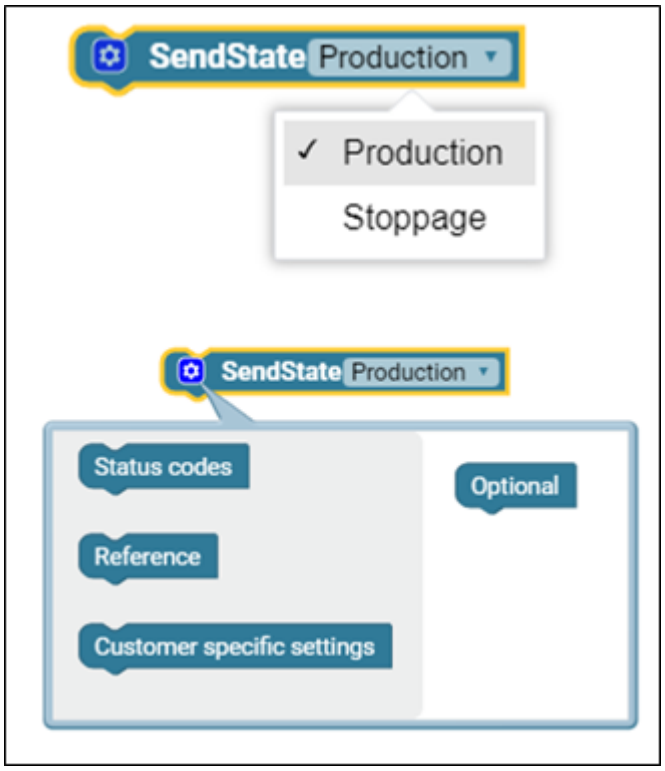


Bild 30: Beispiel für SendGenericInformation

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]



Verwendung

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.

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

Input/Output

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

Beispiel

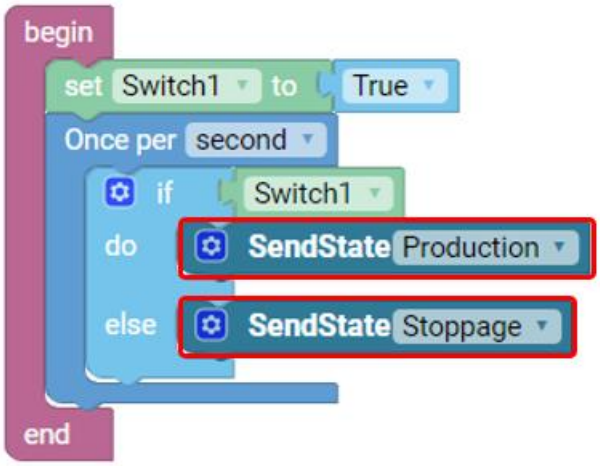
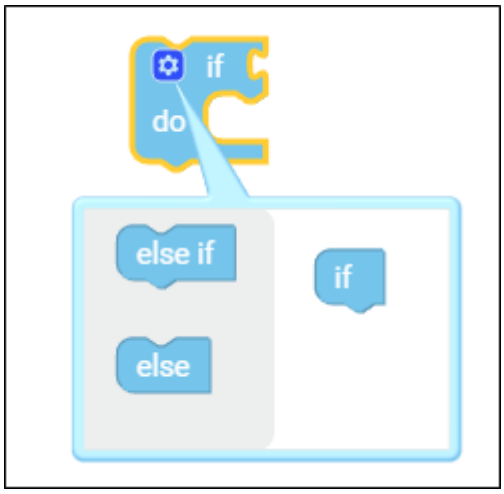


Bild 31: Beispiel zu SendState [Auswahl]

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 Logische Verknüpfungen herstellen (Logical)

7.1 if-do (wenn - dann)



Verwendung

Dieser Block bildet die 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 das dunkelblaue Einstellungssymbol ausgesucht werden.

Input/Output

Input für **`if`**, **`else if`** und **`else`** sind Bool'sche Werte.
Der Input für **`do`** hat keine Einschränkungen.
Einschränkungen für den Output gibt es nicht.

Beispiel

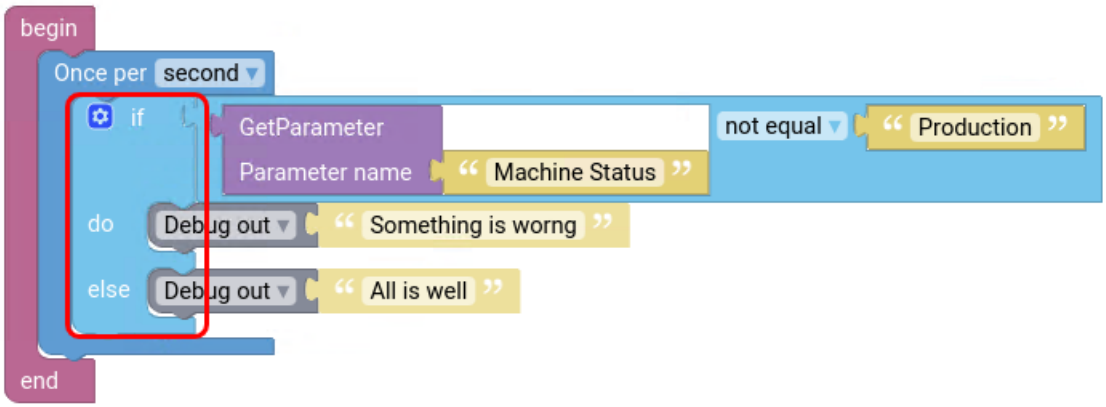


Bild 32: **Beispiel für den If-do-Block**

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 (=/≠/<>/≤/≥)

Verwendung

Logische Verknüpfungen wie zum Beispiel „=“ verbinden zwei Variablen vom Datentyp *Number*.
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:

V1 = V2	V1 gleich V2
V1 ≠ V2	V1 ist ungleich V2
V1 > V2	V1 größer als V2
V1 ≥ V2	V1 größer/gleich V2
V1 < V2	V1 kleiner als V2
V1 ≤ V2	V1 kleiner/gleich V2

Input/Output

Input sind ausschließlich Zahlen (Numbers). Output sind ausschließlich Bool’sche Werte.

Beispiel

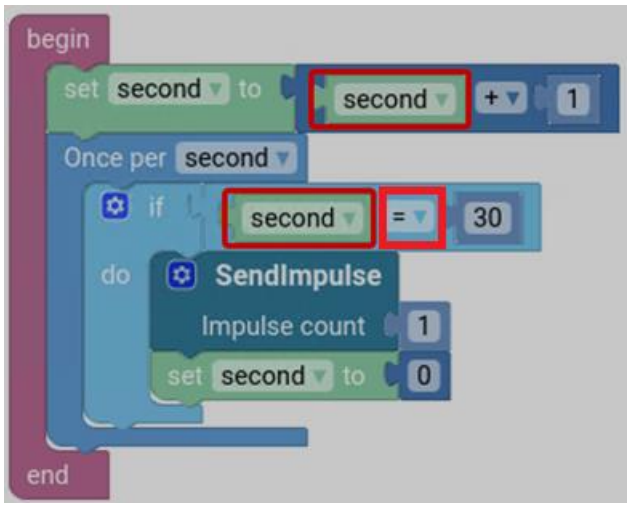
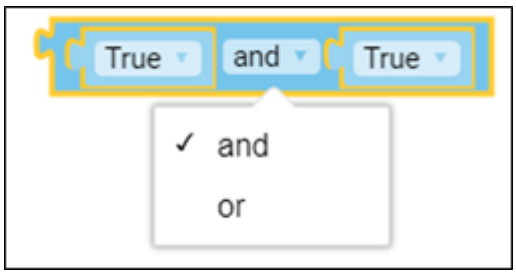


Bild 33: **Beispiel zu der Verknüpfung =**

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 and/or (Logische Verknüpfung)



Verwendung

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/Output

Input und Output sind ausschließlich Booleans.

Beispiel

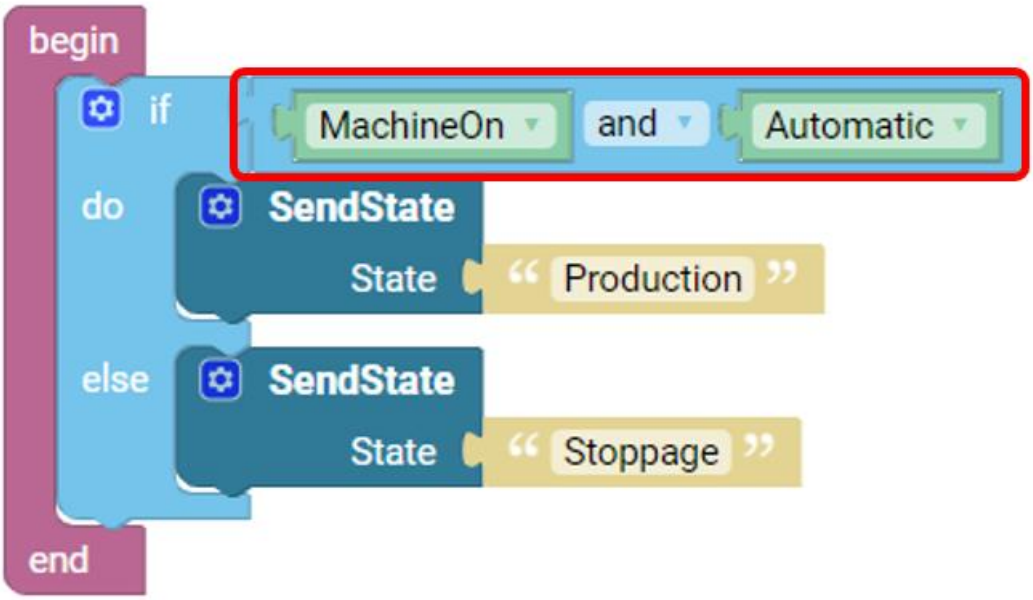
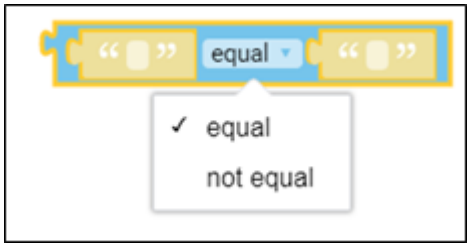


Bild 34: **Beispiel zu der Verknüpfung and**

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

7.4 equal/not equal (Logische Verknüpfung)



Verwendung

`Equal` ist eine Grundverknüpfung. Wenn zwei Zustände oder Aussagen gleich (englisch: equal) sind, ist das Ergebnis (Output) = **`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/Output

Input sind ausschließlich Strings. Output sind ausschließlich Booleans.

Beispiel

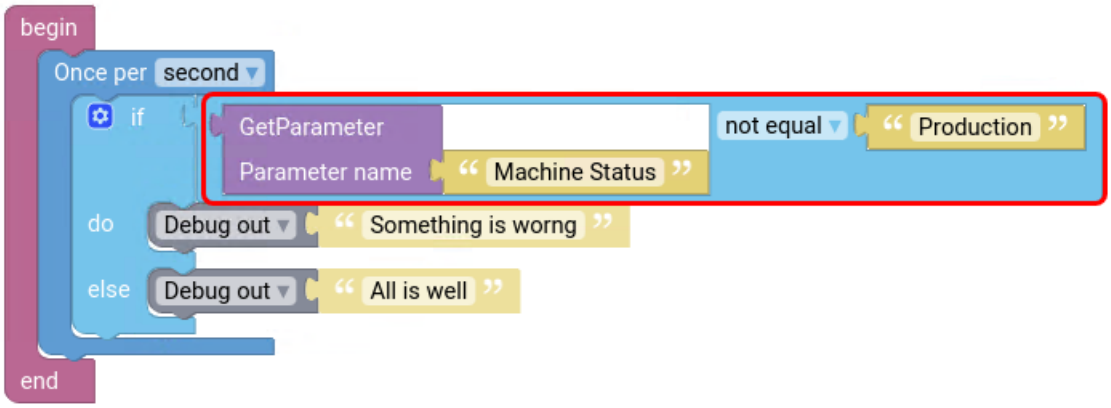
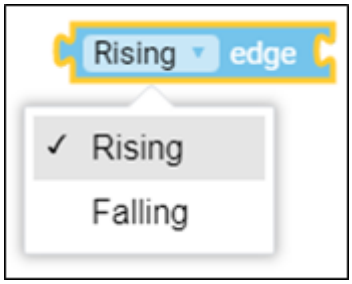


Bild 35: Beispiel zu not equal

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 (Flanken erkennen)



Verwendung

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

`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.

Input/Output

Input und Output sind ausschließlich boolesche Werte.

Beispiel

Bild 36: **Beispiel zu Rising edge**

In diesem 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 (Negation)

Verwendung

Das Ergebnis eines **`not`**-Statements (nicht-Aussage) wird wahr, wenn der Input-Wert falsch ist. Der Ausgangszustand ist also entgegengesetzt zum Output-Zustand.

Input/Output

Input und Output sind ausschließlich Booleans.

Beispiel

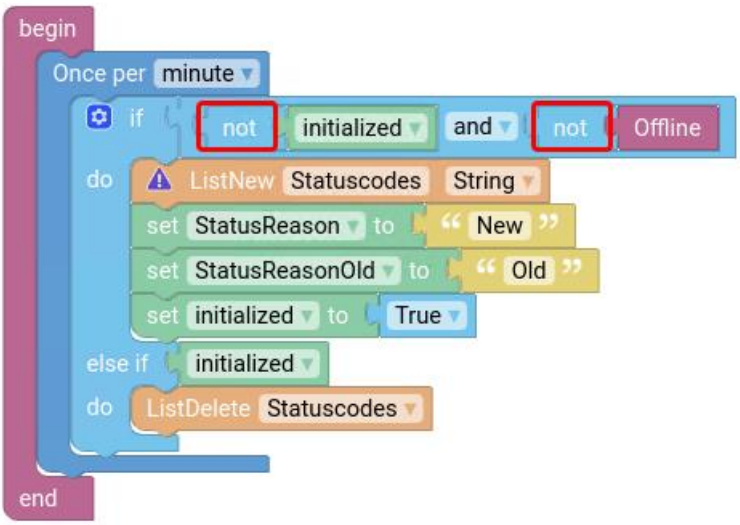


Bild 37: Beispiel für ein not-Statement

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 True/False (Wahrheitsaussage)

Verwendung

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.

Input/Output

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

Beispiel

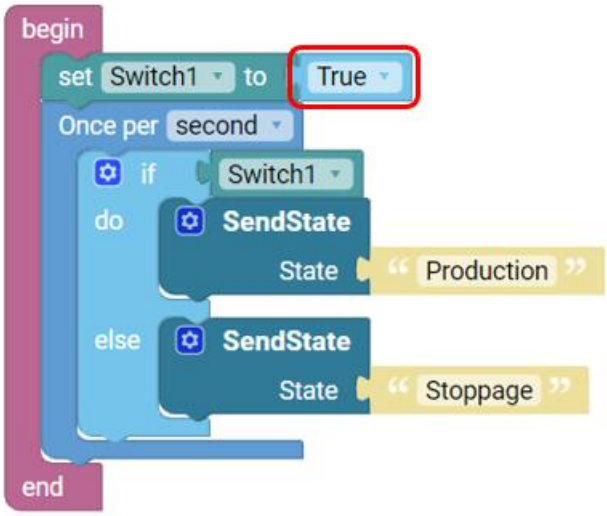


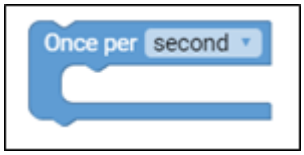
Bild 38: **Beispiel für True**

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 der Fall ist, wird der Produktionsstatus auf **`Production`** gesendet. Andernfalls wird der Status **`Stoppage`** ausgegeben.

8 Schleifen einfügen (Repeaters)

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

8.1 Once per



Verwendung

Repeaters 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

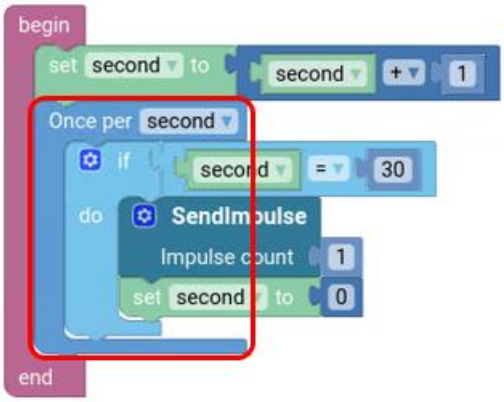


Bild 39: **Beispiel zu Once per**

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 Mathematische Operationen (Arithmetic)

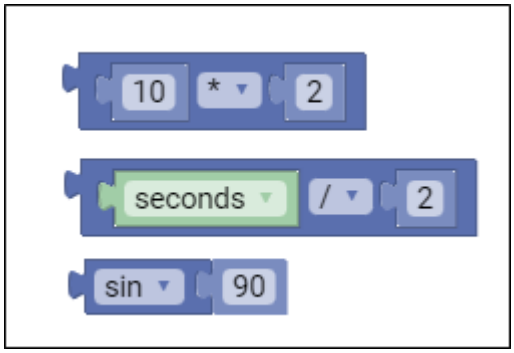
Mit diesen Blöcken werden Rechenfunktionen wie das Addieren, Subtrahieren oder Multiplizieren von Werten ausgeführt und Datenformate umgewandelt.

9.1 Nummernfeld

Verwendung

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

9.2 Matheoperation



Verwendung

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/Output

Input und Output sind ausschließlich Numbers.

Beispiel

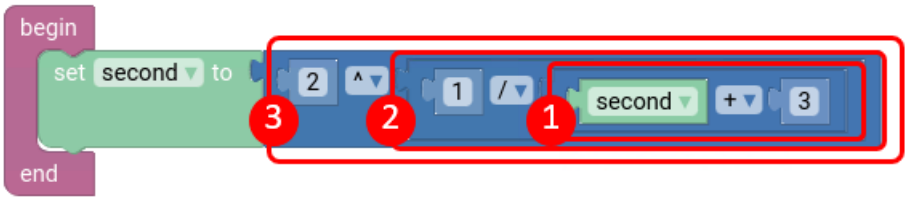


Bild 40: **Beispiel für mathematische Operationen**

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 Bruchs (2). Dieses Ergebnis ist wiederum der Exponent von 2 in der letzten Matheoperation (3).

9.3 ToNumber

Verwendung

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.

Input/Output

Der Input muss ein numerischer Wert im String Datentyp sein. Outputs sind ausschließlich Numbers.

Beispiel

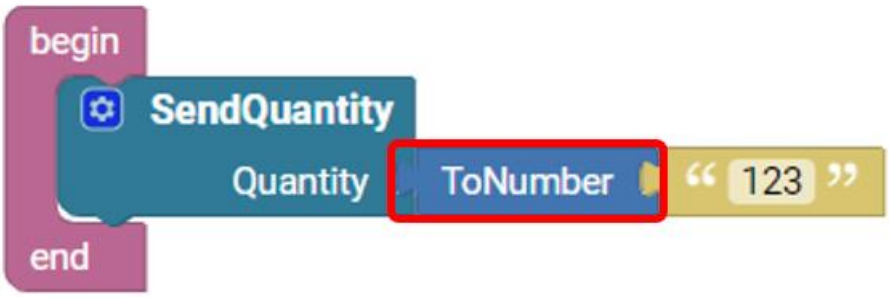


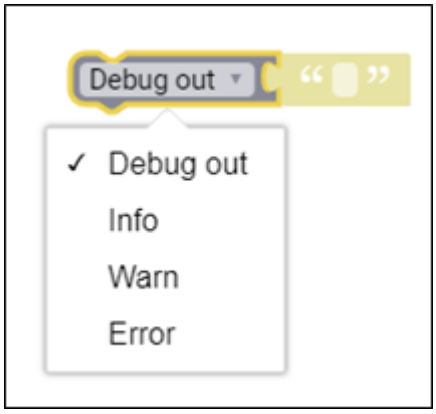
Bild 41: Beispiel für ToNumber

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 Werte protokollieren (Logging)

Die Blöcke dieser Kategorie können dazu genutzt werden, Werte für die Analyse gezielt zu protokollieren und auszugegeben. Dabei wird zwischen verschiedenen Warnstufen unterschieden.

10.1 Debug out



Verwendung

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 helfen
`Info`: Allgemeine Protokollebene, auf der alle Aktivitäten erfasst werden können
`Warn`: Probleme oder Störungen, der Code kann seine Arbeit dennoch fortsetzen
`Error`: Problem, das mehrere Funktionalitäten verhindert

Input/Output

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

Beispiel

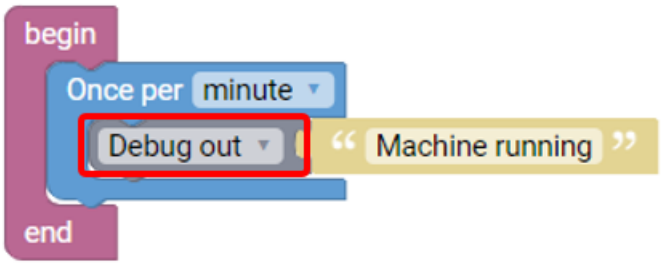


Bild 42: Beispiel für Debug out

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

11 Texte erstellen und verarbeiten (Text)

Auch im Baukastenprinzip werden Wörter und Sätze benötigt, um die Werte verständlich zu machen. In dieser Kategorie können diese Texte erstellt und mit weiteren Funktionen verarbeitet werden.

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

Verwendung

Mithilfe dieser Blöcke werden Strings durch Eingabe in die Anführungszeichen hinzugefügt.

Input/Output

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

Beispiel

Bild 43: Beispiel für String

Der Block **`set IDNummer to`** definiert die ID-Nummer eines Assets mit dem String „123456789“. Anschließend prüft der **`if-do`**-Block, ob die ID-Nummer 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

Verwendung

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/Output

Input und Output sind ausschließlich Strings.

Beispiel

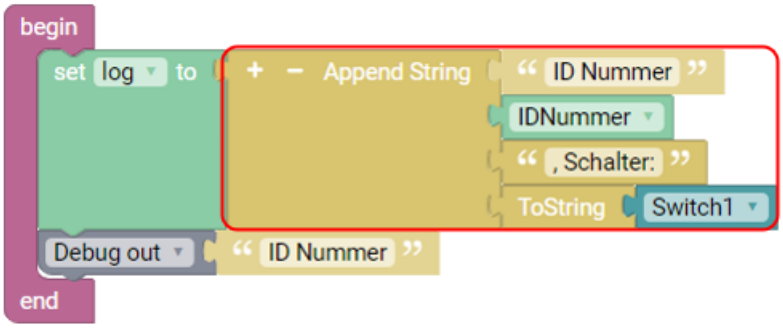


Bild 44: Beispiel zu Append String

In diesem Beispiel geht es darum, die ID-Nummer und den Zustand des Schalters zu protokollieren. Der Block **`set log to`** vereinfacht die Lesbarkeit. Gelesen wird **`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

Verwendung

`ToString` wird verwendet, um Numbers oder Variablen, die Numbers darstellen, in einen String umzuwandeln.

Input/Output

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

Beispiel

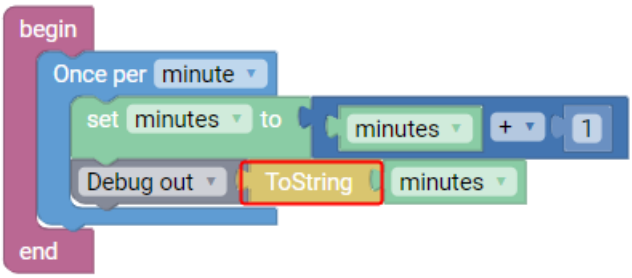


Bild 45: **Beispiel für ToString**

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

Verwendung

`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/Output

Input sind ausschließlich Strings. Output sind ausschließlich Numbers.

Beispiel

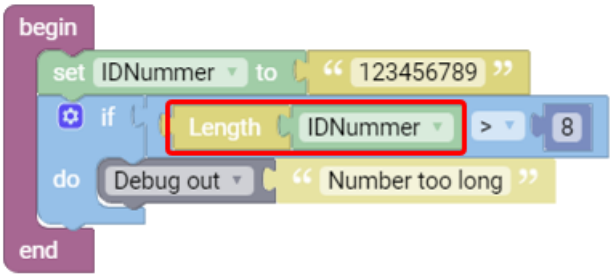
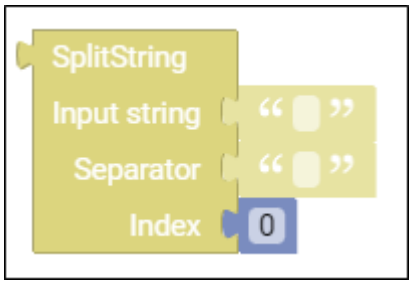


Bild 46: Beispiel für Length

Als Beispiel soll die Länge der Auftragsnummer gezählt werden, damit diese nicht den Schwellenwert von acht Zeichen 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.

11.5 SplitString



Verwendung

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 vordefiniertes Zeichen. Dieses wird unter **`Separator`** 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/Output

Input und Output für **`Input string`** und **`Separator`** sind ausschließlich Strings.
Input und Output für den **`Index`** sind ausschließlich Numbers.

Beispiel

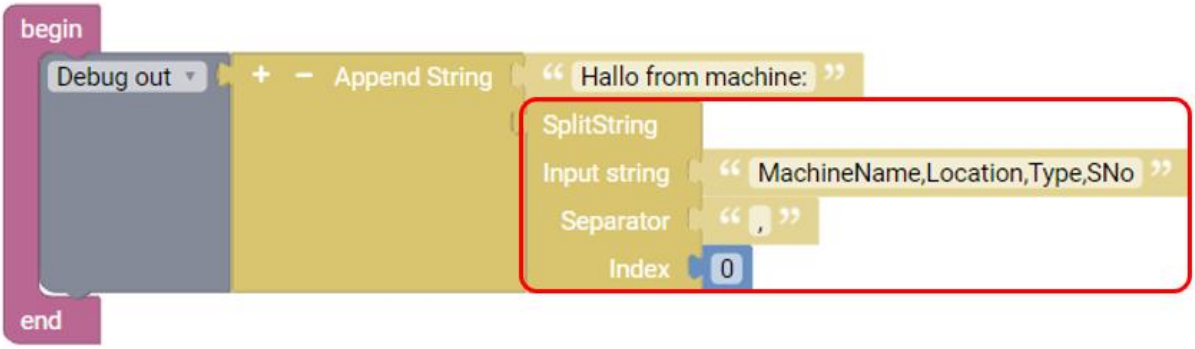


Bild 47: Beispiel für SplitString

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 **`Index`** ist mit 0 angegeben. Es wird also der MachineName ausgegeben. Wäre der Index 2, würde der Type ausgegeben werden.

11.6FromAscii

Verwendung

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/Output

Input sind ausschließlich Numbers. Output sind ausschließlich Strings.

ⓘ Die Ascii-Tabelle befindet sich in Kapitel 17.2 „ASCII-Tabelle“, Seite 79.

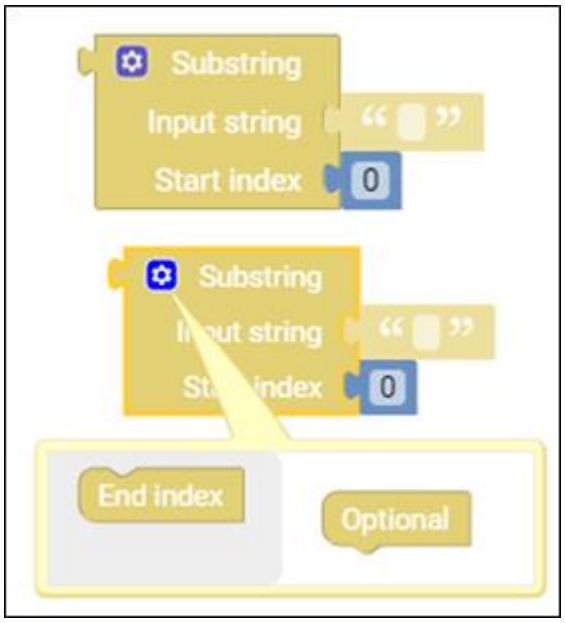
Beispiel

Bild 48: 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 machine:“ eingefügt wurde, holt der Block **FromAscii** den zehnten 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



Verwendung

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 Indizes üblich werden die Zeichen ab 0 gezählt. Der **`End Index`** ist ausgeschlossen.

Input/Output

Input und Output für den **`Input string`** sind Strings.
Input für **`Start index`** und **`End index`** sind ausschließlich Numbers. Output sind Strings.

Beispiel

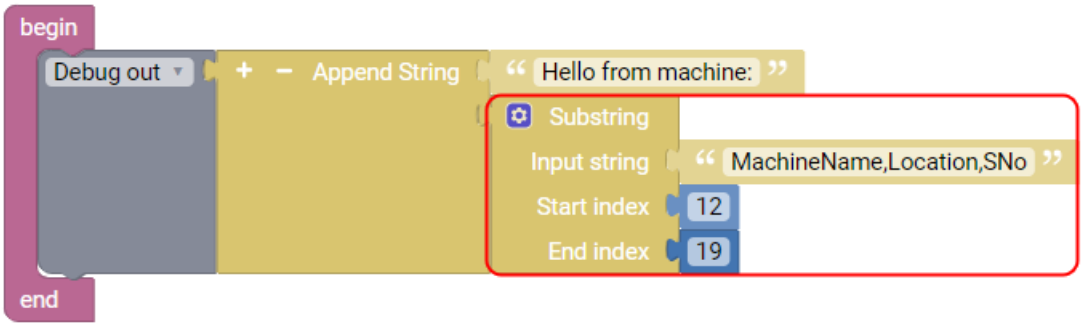


Bild 49: Beispiel zu Substring

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 index`** gibt an, dass ab Zeichen 12 der Output erfolgen soll. Der Block **`End index`** gibt an, dass bis inklusive Zeichen 19 der Output erfolgen soll.
Weil mit 0 beginnend von links gezählt wird, wird die Eigenschaft **`Location`** ausgegeben.

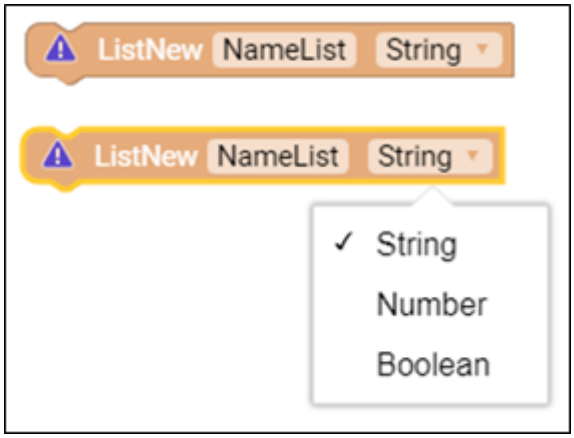
12 Listen erstellen und verwalten (Lists)

Üblicherweise wird eine Liste zum Sammeln von verschiedenen Produktionszuständen genutzt. Mit den Blöcken dieser Kategorie werden Listen erstellt, gefüllt, geleert und gelöscht.

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

⚠ Nach der Verwendung muss eine Liste immer geleert werden (siehe Funktion ListClear).

12.1 ListNew



Verwendung

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.

Input/Output

Einschränkungen für den Input erfolgen über die Auswahl.

Beispiel

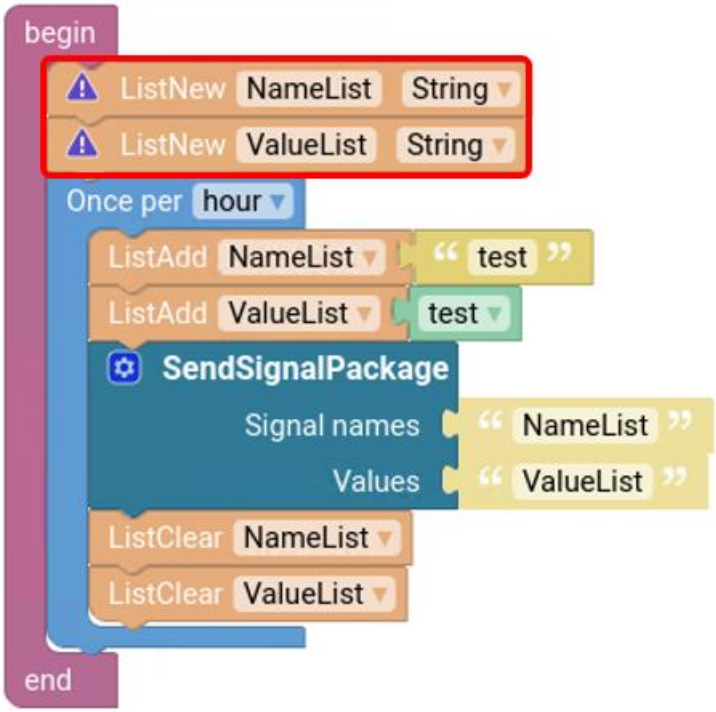
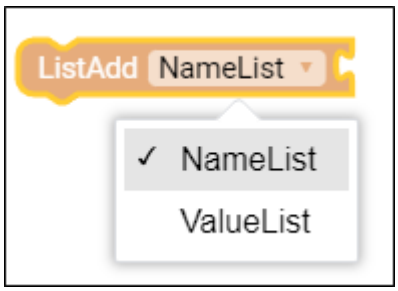


Bild 50: **Beispiel für ListNew**

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



Verwendung

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.

Input/Output

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

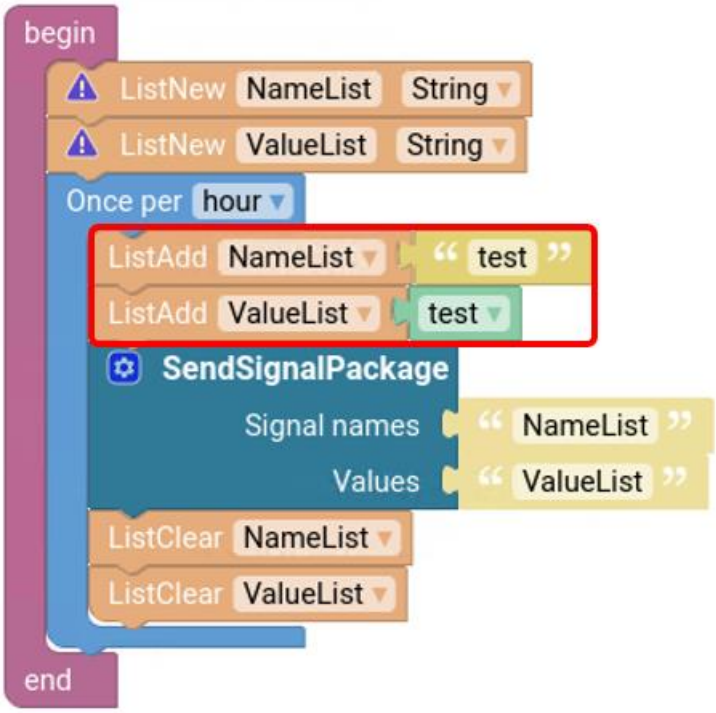
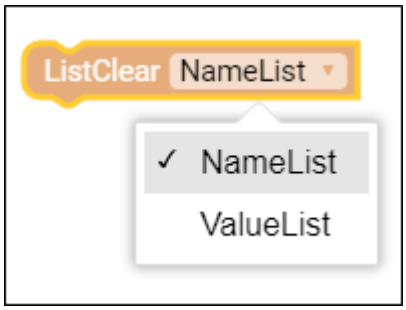


Bild 51: Beispiel für ListAdd

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

12.3 ListClear



Verwendung

`ListClear` löscht Inhalt einer Liste.

ⓘ 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.

Input/Output

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

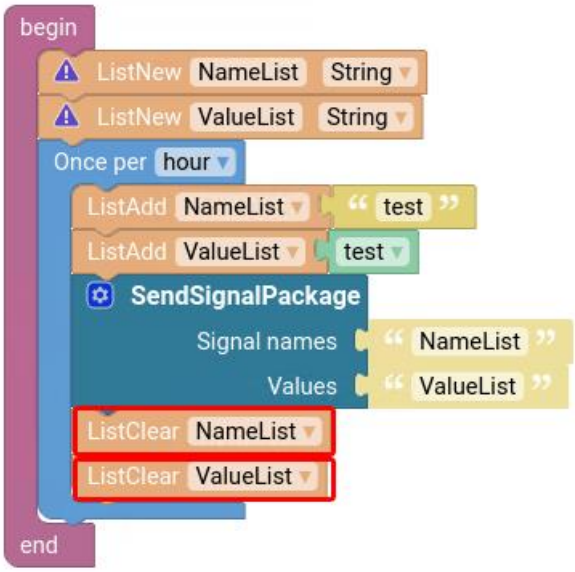
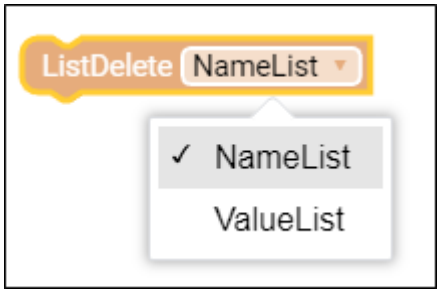


Bild 52: **Beispiel für ListClear**

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

12.4 ListDelete



Verwendung

Der Block **`ListDelete`** löscht eine erstellte Liste. Über das Drop-down-Menü wird die Liste ausgewählt, die gelöscht werden soll.

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

Input/Output

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

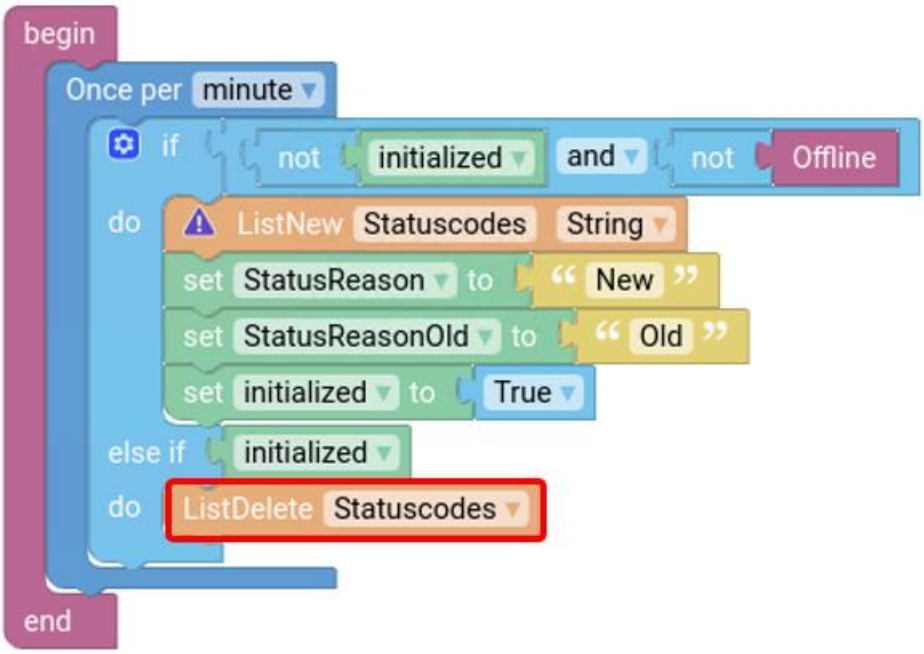
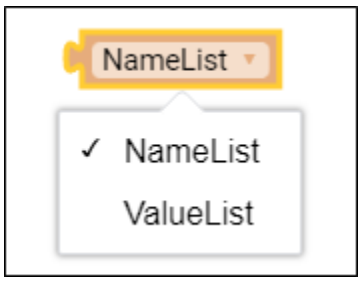


Bild 53: **Beispiel für ListDelete**

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 [Liste] - Liste einfügen



Verwendung

Der Block fügt eine vorhandene Liste in die Struktur ein. Im Drop-down-Menü wird die bereits erstellte Liste ausgewählt.

Input/Output

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

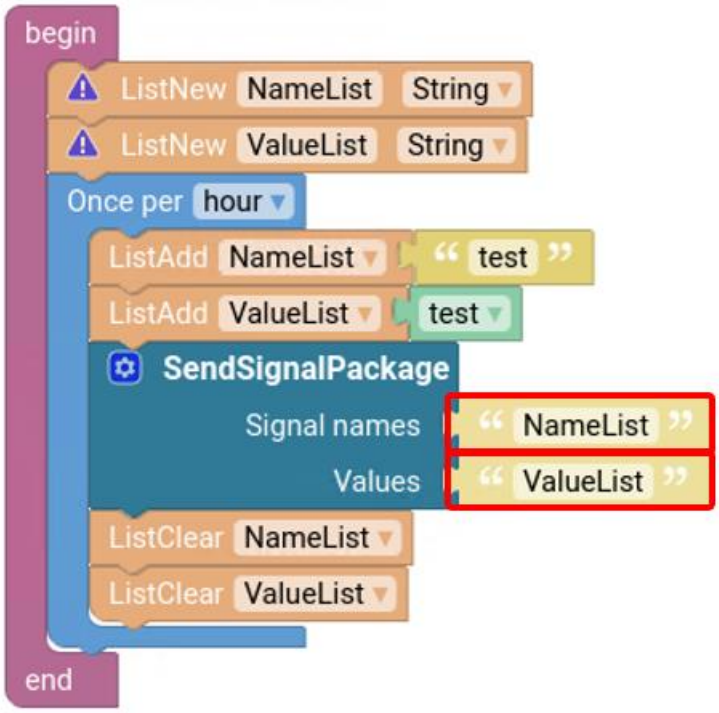


Bild 54: Beispiel für [Liste]

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 Zeitangaben verwalten (Date and time)

Um eine Aktion zu einem bestimmten Zeitpunkt ausführen zu können, müssen Uhrzeit und Datum festgelegt werden. Auch die aktuelle Uhrzeit eines Ereignisses oder eine Pause werden gespeichert.

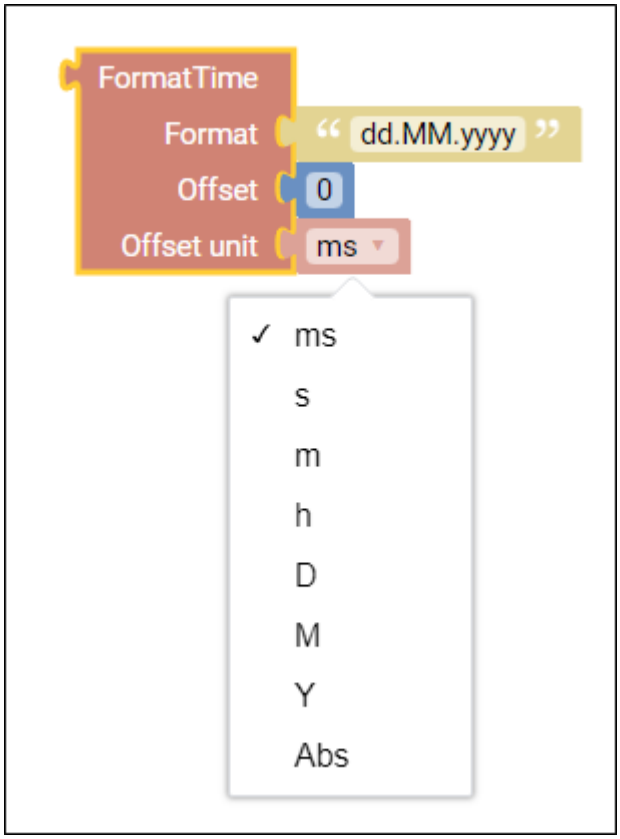
In dieser Funktionskategorie sind alle Aktionen zum Thema Zeit- oder Datumseinstellungen gesammelt. Dabei wird einheitlich die UTC-Zeit verwendet.

Die folgende Tabelle enthält die in der grafischen Komposition verwendeten Abkürzungen für die jeweiligen Zeiteinheiten.

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



Verwendung

Der Block **`FormatTime`** erstellt die gewünschte Zeiteinheit der aktuellen Zeit/eines darauf basierenden Datums.
Das Format gibt die Einheit des **`Offsets`** 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. Wenn beispielsweise im **`Offset`** 10 eingegeben wird und die Einheit ms ist, dann ist das Ergebnis die aktuelle Zeit plus 10 Millisekunden.

ⓣ **`Abs`** wird für die Umrechnung von Unix-Zeitstempeln verwendet (z. B. für Zeitstempel, die direkt von der Maschine empfangen werden). Das Referenzdatum (Offset = 0) ist dabei nicht das aktuelle Datum, sondern der 01.01.1970 00:00 Uhr. Wenn **`Abs`** ausgewählt ist, entspricht der

Offset-Wert also der seit diesem Datum vergangenen Zeit (in ms). Dieser Wert wird in das gewünschte Format übertragen.

Input/Output

Input für **`Format`** sind Strings. Output sind Strings.
Input für **`Offset`** sind Numbers. Output sind Strings.
Input für **`Offset unit`** bildet ein Drop-down-Menü. Output sind Strings.

Beispiel

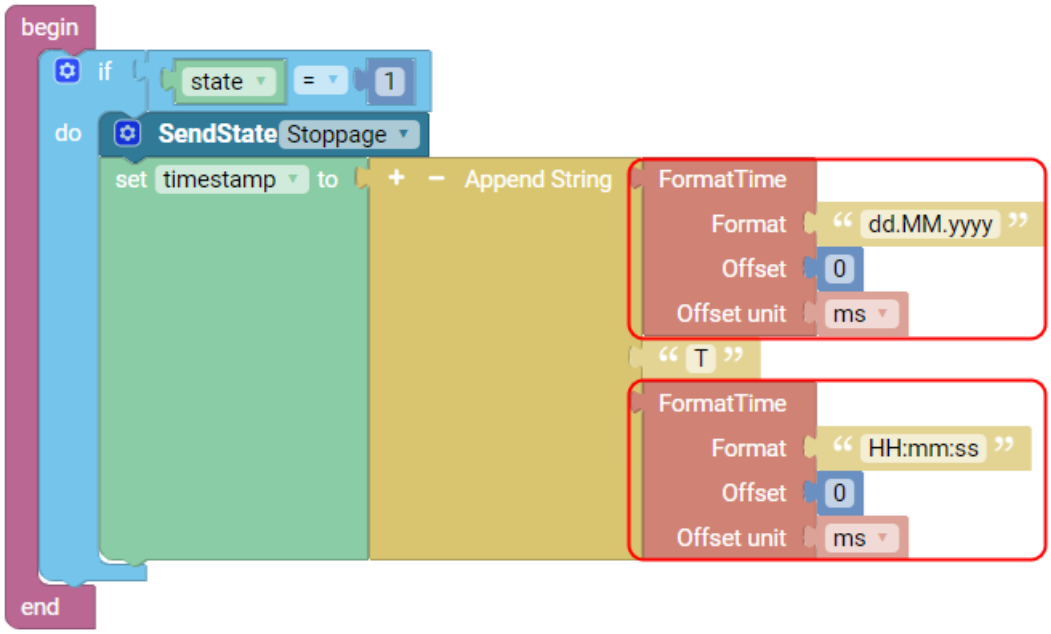


Bild 55: Grafisches Beispiel für FormatTime

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 der Text-String **`T`** für Time, anschließend die Uhrzeit in der Reihenfolge Stunde:Minute:Sekunde.

13.2AtTime Do

Verwendung

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

Input/Output

Input sind ausschließlich Numbers.

Beispiel

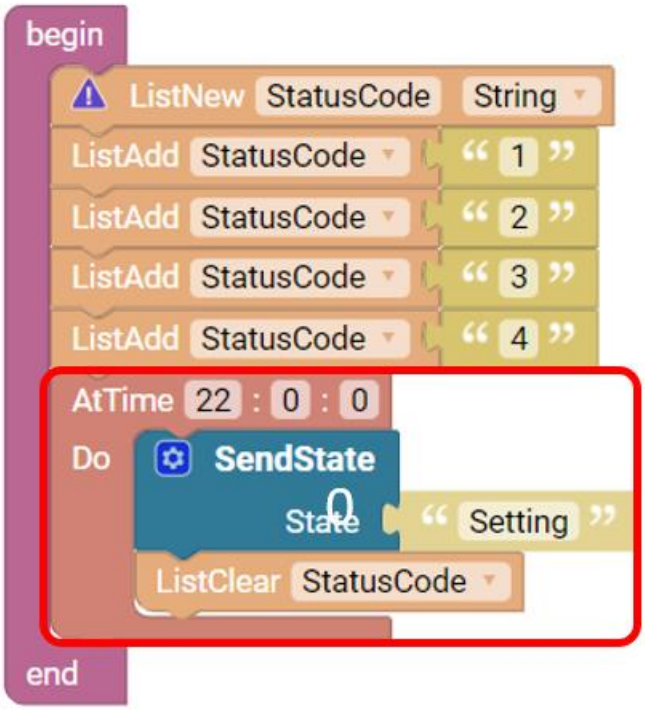


Bild 56: Beispiel für AtTime Do

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.
Anschließend wird die Liste wieder geleert.

13.3Sleep

Verwendung

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/Output

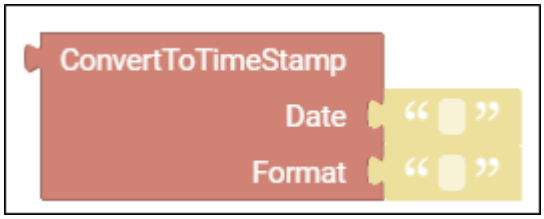
Input sind ausschließlich Numbers. Einschränkungen für den Output gibt es nicht.

Beispiel

Bild 57: **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



Verwendung

Der Block **ConvertToTimeStamp** gibt einen Zeitstempel aus. **`Date`** gibt das zu konvertierende Datum an, das übergebene Datumsformat wird darunter in dem String **`Format`** definiert. Der Output ist ein Unix-Wert, also die Zeit in Millisekunden nach dem 01.01.1970 um 0.00 Uhr.

Input/Output

Input und Output sind ausschließlich Strings.

Beispiel

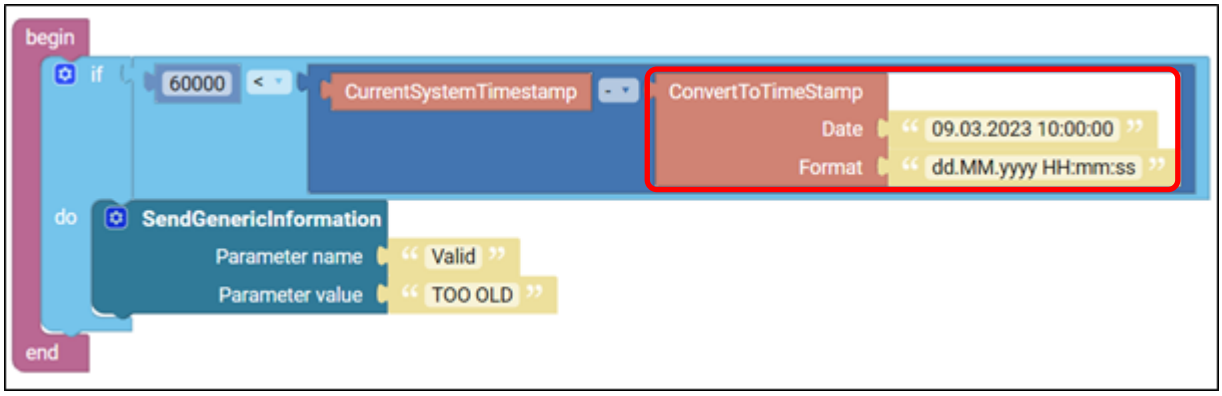


Bild 58: Beispiel für ConvertToTimeStamp

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 **`SendGenericInformation`** versendet. Diese sagt, dass der empfangene Zeitpunkt zu alt ist.

13.5 CurrentSystemTimestamp

Verwendung

`CurrentSystemTimestamp` trägt immer den Unix-Wert der aktuellen Uhrzeit ein. Sie gibt an, wie viele Sekunden seit dem 01.01.1970 vergangen sind.

Input/Output

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

Beispiel

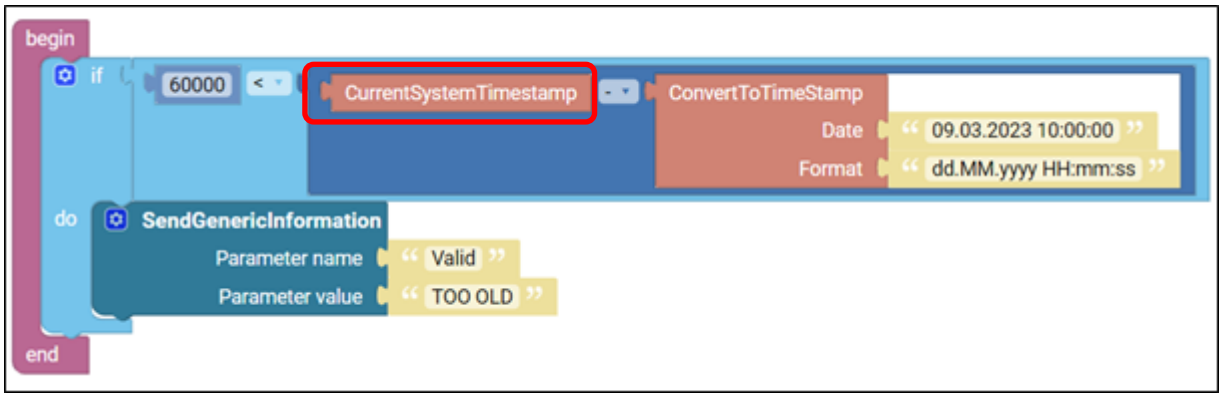


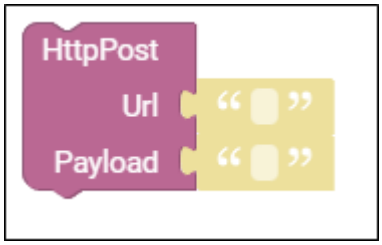
Bild 59: Beispiel für CurrentSystemTimestamp

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 **`SendGenericInformation`** versendet. Diese sagt, dass der empfangene Zeitpunkt zu alt ist.

14 Weitere Aktionen (Misc)

Diese Kategorie ist eine Sammlung von weiteren Befehlen und Blöcken, die eine Verbindung zu anderen Systemen herstellen. Die Blöcke können zum Beispiel Daten aus dem Internet integrieren, den Asset-Status abrufen oder die IP-Adresse/den Host-Namen eines Assets ausgeben.

14.1 HttpPost



Verwendung

Der Block **`HttpPost`** sendet eine Nachricht an ein Drittsystem. Unter **`Url`** wird die Internetadresse (das Ziel) eingetragen. **`Payload`** bezeichnet den Teil der übermittelten Daten, also die eigentliche Nachricht.
Wir empfehlen die Schreibweise mit den zwei Primes (hochgestellten Anführungsstrichen, z. B.: "k").

Input/Output

Inputs sind Strings. Einschränkungen für den Output gibt es nicht.

Beispiel

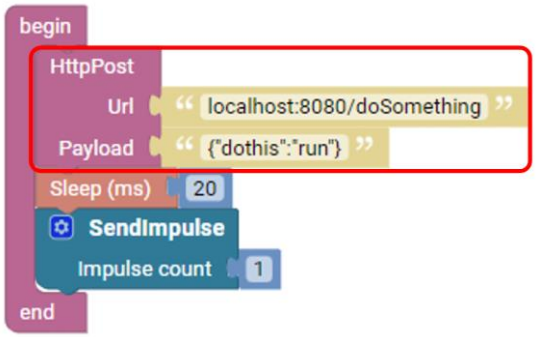
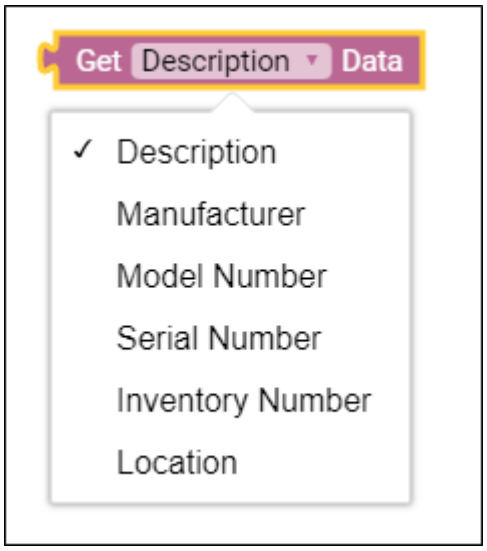


Bild 60: Beispiel für HttpPost

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



Verwendung

`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.
Diese Parameter werden automatisch in das Drop-down-Menü hinzugefügt.

Input/Output

Der Input wird im Drop-down-Menü gewählt. Output sind Strings.

Beispiel

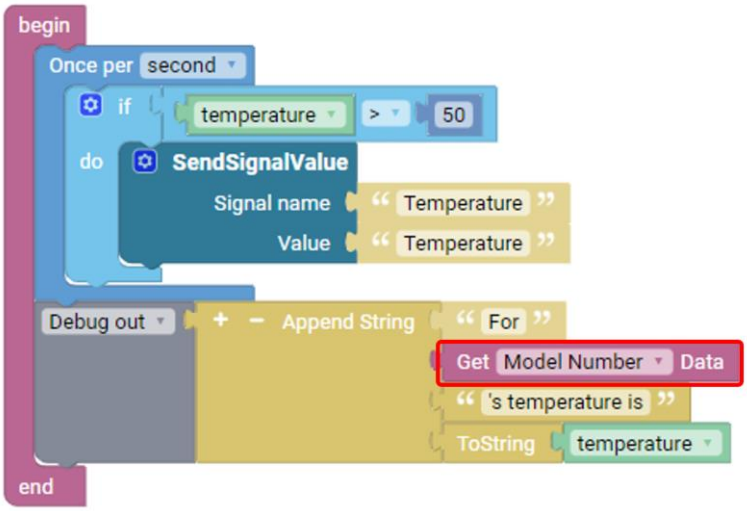


Bild 61: Beispiel für Get [specific] Data

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

Verwendung

`GetMachineStatus` gibt den aktuellen Maschinenstatus aus.

Input/Output

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

Beispiel

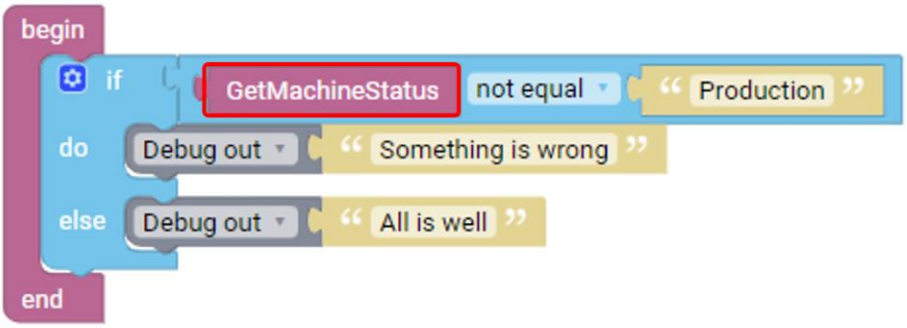


Bild 62: **Beispiel von GetMachineStatus**

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

Verwendung

Wenn ein System oder eine Maschine nicht in Betrieb ist, kann die Statusabfrage **`Offline`** verwendet werden.

Input/Output

Einschränkungen für den Input gibt es nicht. Output sind Booleans.

Beispiel

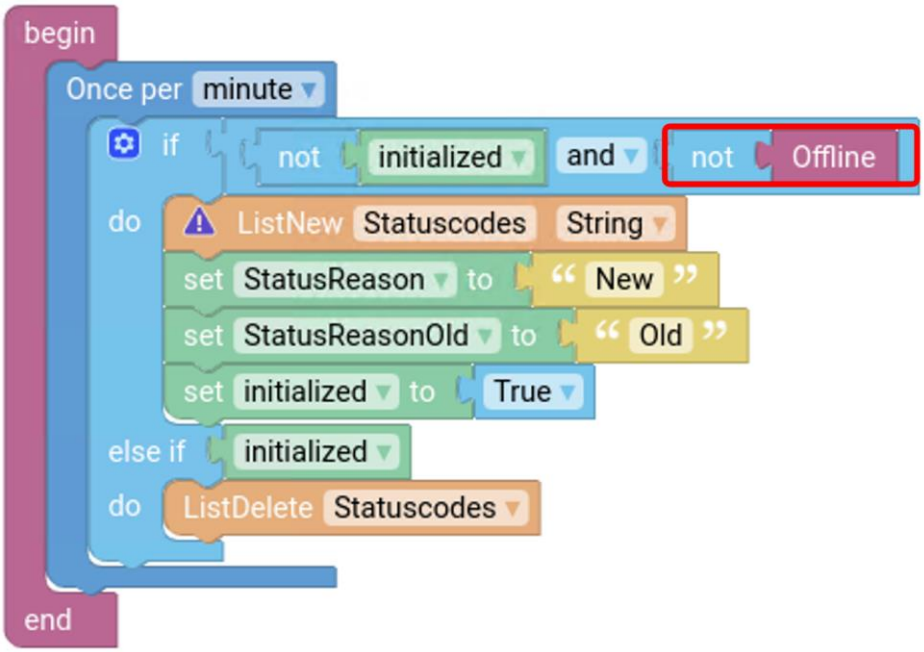


Bild 63: **Beispiel für Offline**

In dem Beispiel wird ein Mal pro Minute überprüft, ob folgender Status vorliegt:

- das Programm wurde nicht gerade gestartet (**`not initialized`**)
  und
- das Asset ist nicht offline (**`not Offline`**)

Wenn beides zutrifft, dann läuft das Asset. In diesem Fall werden Listen mit den aktuellen und alten Statusgründen erstellt. Anschließend wird mit **`True`** bestätigt, dass das Programm gerade gestartet (**`initialized`**) wurde. Dadurch wird der obere Teil der Liste nicht nochmal durchlaufen. Der Block **`ListDelete`** löscht anschließend die Liste der Statuscodes.

14.5IpAddress

Verwendung

`IPAdress` gibt die IP-Adresse eines Assets aus. Bei der IP-Adresse handelt es sich um eine individuelle Adresse, die ein Gerät im Internet oder auf einem lokalen Netzwerk identifiziert.

Input/Output

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

Beispiel

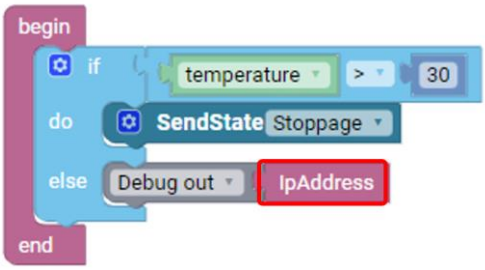


Bild 64: Beispiel für IpAddress

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

14.6HostName

Verwendung

`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.

Input/Output

Einschränkungen beim Input gibt es nicht. Output sind Stings.

Beispiel

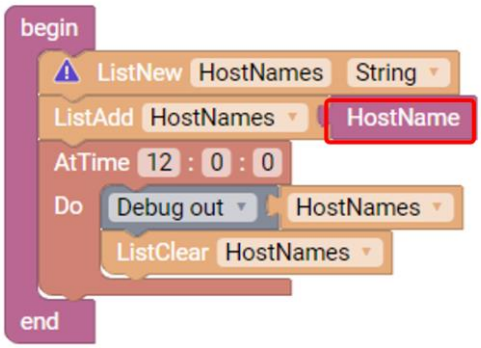


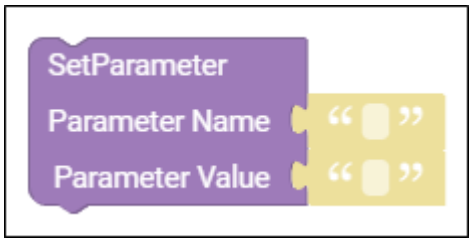
Bild 65: Beispiel für HostName

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 Asset-Eigenschaften verarbeiten (Business Parameters)

Business-Parameter sind Eigenschaften der Maschine, wie zum Beispiel eine Beschreibung, Herstellername, Modell- und Seriennummer oder Aufstellungsort. Die zugehörigen Daten werden im Configuration Wizard in den vorherigen Konfigurationsschritten hinterlegt (siehe Handbuch zu EDGE CONNECT).

15.1 SetParameter



Verwendung

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 (siehe Handbuch zu EDGE CONNECT).
Soll ein bereits definierter Parameter verwendet werden, wird **`GetParameter`** (Kapitel 15.2) genutzt.

Input/Output

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

Beispiel

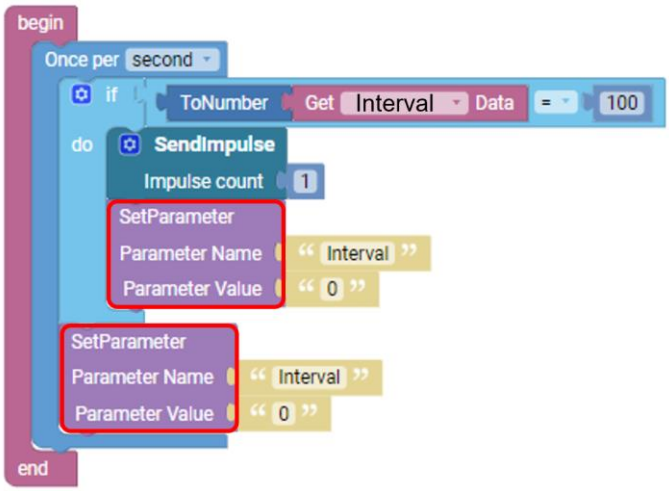
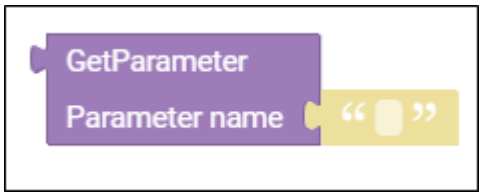


Bild 66: **Beispiel für SetParameter**

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 (**`Parameter Value`**) 0 gesetzt (**`SetParameter`**). Andernfalls wird das Intervall weiter mit 1 hochgezählt.

15.2 GetParameter



Verwendung

Der Block **`GetParameter`** zieht den Wert eines Parameters.

Input/Output

Input und Output sind ausschließlich Strings.

Beispiel

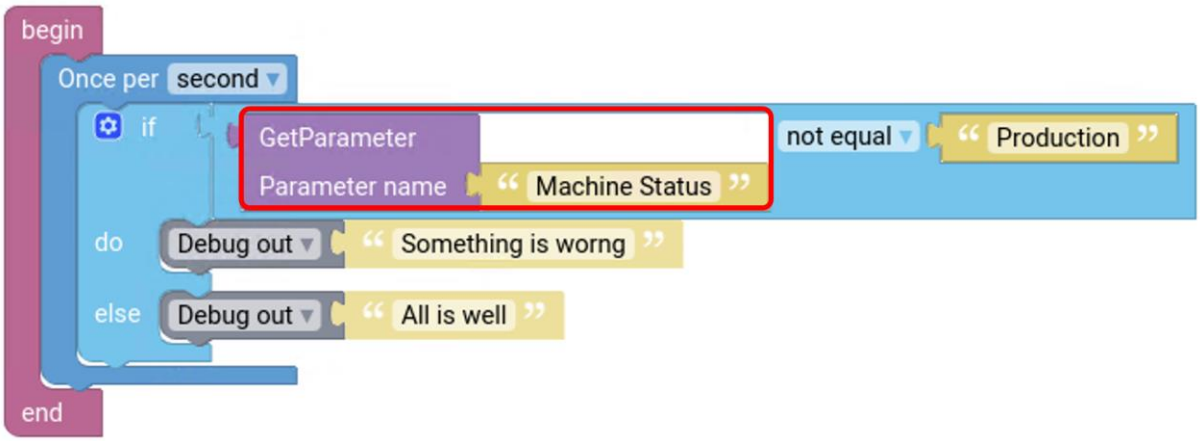
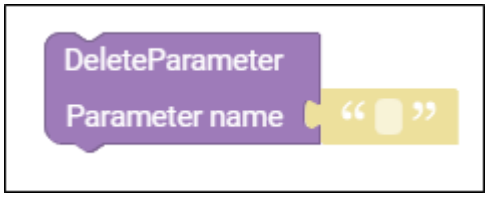


Bild 67: Beispiel GetParameter

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



Verwendung

Der Block **`DeleteParameter`** setzt den Parameterwert in der Datenbank auf 0 zurück.

Input/Output

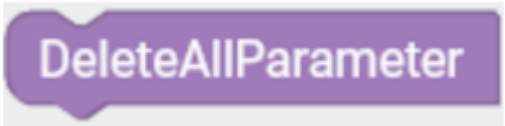
Input sind ausschließlich Strings. Einschränkungen beim Output gibt es nicht.

Beispiel

Bild 68: Beispiel für DeleteParameter

Wenn das Signal I1 gleich (=) 1 ist, wird die Aussage mathematischen Vergleichs = **`True`** (wahr/1). Dann setzt der Block **`DeleteParameter`** den **`Parameter name`** COUNTER auf 0 zurück.

15.4 DeleteAllParameter



Verwendung

Der Block **`DeleteAllParameter`** löscht alle Parameter. Verwendet wird der Block wie **`DeleteParameter`**.

⚠ Alle bereits verwendeten Parameter werden auch auf null zurückgesetzt.

Input/Output

Einschränkungen für Input und Output gibt es nicht.

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 Reference Customer specific settings	 Optional Optional	 Number String String	 N/A N/A N/A
SendQuantity Quantity Unit Quality details Reference Customer specific settings	 Optional Optional Optional Optional	 Number String String String String	 N/A N/A N/A N/A N/A
SendState State Status codes Reference Customer specific settings	 Optional Optional Optional	 String String String String	 N/A N/A N/A N/A
SendSignalValue Signal name Value Unit Reference Customer specific settings Timestamp	 Optional Optional Optional Optional	 String String String String String String	 N/A N/A N/A N/A N/A N/A
SendSignalPackage Signal name		String String	N/A N/A

Blöcke	Sonstiges	Input	Output
Value Unit Reference Customer specific settings	 Optional Optional Optional	String String String String	N/A N/A N/A N/A
SendGenericInformation Parameter name Parameter value Reference Customer specific settings	 Optional Optional	 String String String String	 N/A N/A N/A 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

Blöcke	Sonstiges	Input	Output
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 Parameter 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	Umschaltzeichen

Dec	Char	Beschreibung
	Data link escape	
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 - 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 synchroner Datenübertragung, dass auch bei Abwesenheit von zu übertragenden Signalen synchronisiert wird
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	Trennung von Gruppen

Dec	Char	Beschreibung
	Group separator	
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	/	
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	

Dec	Char	Beschreibung
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	
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	

Dec	Char	Beschreibung
121	y	
122	z	
123	{	
124	|	
125	}	
126	~	
127	DEL Delete	Löschen des letzten Zeichens