DACQ Skriptsprache

Version 5.12

Handbuch

	Dokument: Handbuch - DACQ Skriptsprache.docx
	Freigabedatum: 24.11.22
	Dokumentversion: 1
	Autor: PMüller/ABöer

FB_8-41 ERSTELLT AM: 12.01.22 GEÄNDERT AM: 24.11.22 VERSION: V2
FREIGEGEBEN DURCH: HEAD OF TTE AUTOR: TECHNICAL WRITER VERTRAULICHKEITSSTUFE: ÖFFENTLICH

Inhaltsverzeichnis

1

1 Allgemein*

ⓘ Dieses Handbuch setzt Kenntnisse im Umgang mit FORCAM FORCE IIOT.
Sollten Sie keine Kenntnisse im Umgang mit FORCAM FORCE IIOT haben, nehmen Sie sich die Zeit, sich mit den Grundlagen vertraut zu machen.

Wir empfehlen Ihnen die Nutzung unserer Academy.

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

Die Maschinenkommunikation in FORCAM FORCE IIOT erfolgt über die DCU. An eine Maschine wird dafür ein Controller (Steuereinheit) angeschlossen, die Daten der Maschine ausliest.

Die DCU beinhaltet alle relevanten Informationen (Controller-Typ, IP-Adresse, Port, Signale usw.) einer Maschine. Eine DCU kann Daten von bis zu 100 Maschinen sammeln. Um die Stabilität aller Prozesse nicht zu gefährden, ist es empfehlenswert, an eine DCU nicht mehr als 50 Maschinen anzubinden.

Die DCU kommuniziert mit der Maschine und fragt Daten in kurzen Abständen ab (z. B. alle 100 ms oder einmal in der Sekunde) oder empfängt sie von einem zwischenliegenden OPC-Server oder einer WAGO-Box. Die DCU sammelt unverarbeitete Signale und überträgt sie (via RMI) an die DACQ.
Die DACQ normalisiert die empfangenen Daten und weist sie Betriebszuständen zu. Die DACQ sendet dann relevante Informationen wie Maschinenstatus oder Mengen an den Server. Ein Skript innerhalb der DACQ regelt die Interpretation der empfangenen Daten.

* 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.

Skripte werden immer dann ausgeführt, wenn sich der Wert von einem referenzierten Signal oder Variable geändert hat. Mögliche Ausnahme sind Datums- und Zeitwerte (siehe Abschnitt 2.7.3).

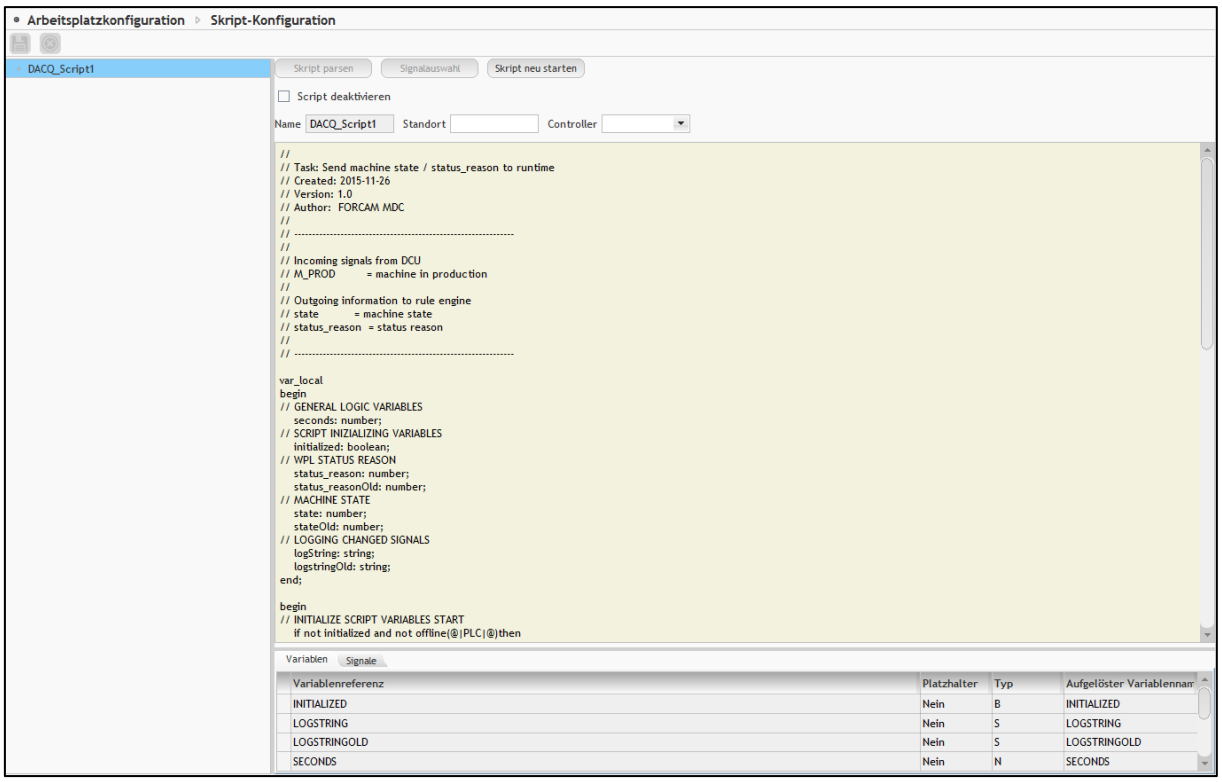

Bild 1: Skripterstellung in der Workbench (Beispiel)

Ein aus mehreren Statements (einzelnen Anweisungen) bestehendes Skript muss stets mit **begin** … **end** zu einem Block zusammengefasst werden.
Ein Statement wird immer mit einem Strichpunkt abgeschlossen. Ausnahmen hierzu sind:

- Begin/end
  Hinter **begin** darf nie ein Strichpunkt stehen. Hinter **end** kann ein Strichpunkt stehen, ist jedoch nicht notwendig.
- if … then … else
  Statements vor **then** und **else** dürfen nicht mit einem Strichpunkt abgeschlossen werden.

2 Formelelemente

Dieser Abschnitt beschreibt alle verwendbaren Formelelemente. Bei allen Formeln wird zwischen Groß- und Kleinschreibung nicht unterschieden.

2.1 Numerische Konstanten

Als Dezimaltrennzeichen sind sowohl Punkt als auch Komma erlaubt.
Beispiel: 3,14 oder 3.14.

2.2 String-Konstanten

Strings werden in doppelte obere Anführungszeichen eingeschlossen (z. B. "hallo"). Anführungszeichen innerhalb von Strings wird ein Backslash vorangestellt (z. B. "Dies ist ein \"echtes\" Anführungszeichen"). Durch einen Backslash können folgende Sonderzeichen in einen String eingefügt werden:

Tabelle 1: Sonderzeichen in Strings

Sonderzeichen	Funktion
\"	(doppeltes) Anführungszeichen
****	Backslash
\n	Zeilenwechsel
\t	Tabulator
\b	Backspace
\r	Carriage Return
\xnn	Zeichen mit ASCII-Wert nn (hexadezimal), z. B. \x0 für Null-Zeichen

2.3 Logische (boolesche) Konstanten

Die logischen Konstanten sind **true** für wahr und **false** für falsch. Anstatt true für wahr, kann auch eine 1 für wahr und für false kann auch eine 0 für falsch verwendet werden, allerdings dürfen diese nicht in einer Abfrage zusammen kombiniert werden, z. B. If X = 1 and Y = true. Solch eine Verknüpfung verursacht einen Fehler. Man darf nur entweder true/false oder 1/0 verwenden.

2.4 Signalwerte

Ein Signalwert hat folgende Form:
controller:device:name oder **controller:name**

Der Wert **device** kann weggelassen werden, wenn das entsprechende Signal ohne Device definiert ist oder **controller** und **name** bereits zur eindeutigen Bezeichnung ausreichen.

2.5 Variable

Variable sind programminterne Merker, die von mehreren Objekten bzw. Skripten ausgelesen oder gesetzt werden können. Variable können von folgenden Typen sein:

Tabelle 2: Typen von Variablen

Variable	Abkürzung	Bedeutung
boolean	%B%	Boolesche Variable: Ausdruck, der nur zwei Werte haben kann (wahr vs. falsch)
number	%N%	Numerische Variable: Ausdruck, der sich nur aus Ziffern zusammensetzt
string	%S%	String-Variable: Ausdruck bestehend aus einer Zeichenkette

Der Typ der Variablen wird zu Beginn eines Skripts definiert. Im folgenden Beispiel werden etwa Sekunden numerisch und die Skript-Initialisierung als boolescher Ausdruck dargestellt bzw. ausgeführt:

```
var_local
begin
// GENERAL LOGIC VARIABLES
   seconds: number;
// SCRIPT INIZIALIZING VARIABLES
   initialized: boolean;
// WPL STATUS REASON
   status_reason: number;
   status_reasonOld: number;
// MACHINE STATE
   state: number;
   stateOld: number;
// LOGGING CHANGED SIGNALS
   logString: string;
   logstringOld: string;
end;
```

Bild 2: Definition von Variablen-Typen

2.6 Textersatz in Text-Objekten

Im festen Text eines String-Objekts kann das Ergebnis der Text-Formel mit **%f** oder **%[Formatangabe]f** eingefügt werden. Dieses Ergebnis wird gemäß der Formatangabe formatiert (siehe Abschnitt 2.7).

2.7 Formatangaben

2.7.1 Zahlen

Formatangaben für Zahlen haben folgende Form:
[-][0][Gesamtlänge[.Nachkomma]][x|X]

Der Ergebnisstring wird auf **Gesamtlänge** aufgefüllt, stellt aber immer die gesamte Zahl dar, auch wenn er dadurch länger wird. Folgendes Verhalten beachten:

- Ist **Gesamtlänge** angegeben und **Nachkomma** nicht, werden keine Nachkommastellen angezeigt. Durch Angabe von **0** wird mit Nullen aufgefüllt, sonst mit Leerzeichen [Blanks].
- Bei Angabe von **-** ist die Formatierung linksbündig, sonst rechtsbündig.
- Durch Angabe von **x** oder **X** erfolgt hexadezimale Darstellung mit Klein- oder Großbuchstaben bei kleinem oder großem **X**. In diesem Fall werden Nachkommastellen immer abgeschnitten.

Beispiele (Leerzeichen sind durch Punkte dargestellt):

Tabelle 3: Beispiele für Formatangaben für Zahlen

Formatangabe	Zahl	Ergebnis
3	9	..9
03.3	3.1	003.100
-5	9	9....
3X	255	0FF (Hex)
x	10	a

2.7.2 Strings

Formatangaben für Strings haben folgende Form:
[-][Minlänge[.Maxlänge]]

Der Ergebnisstring wird auf **Maxlänge** aufgefüllt bzw. auf **Minlänge** gekürzt.

Bei Angabe von **-** ist die Formatierung linksbündig, sonst rechtsbündig.

Beispiele (Leerzeichen sind durch Punkte dargestellt):

Tabelle 4: Beispiele für Formatangaben für Strings

Formatangabe	String	Ergebnis
8	hallo	…hallo
8.9	hallo	…hallo
8.9	hallo Welt	hallo Wel
-8	hallo	hallo…

2.7.3 Datum und Uhrzeit

Datum und Uhrzeit werden mit **%d** bzw. **%t** formatiert. Bei Datum und Uhrzeit werden folgende Abkürzungen verwendet:

Tabelle 5: Abkürzungen für Datum und Uhrzeit

Bereich	Abkürzung	Bedeutung
Datum	y	Jahr
	m	Monat
	d	Tag
Uhrzeit	h	Stunde
	m	Minute
	s	Sekunde
	t	Millisekunde

ⓘ Eine 0 hinter **d** bzw. **t** verhindert, dass bei Änderung der Uhrzeit das Objekt aktualisiert bzw. das Skript aufgerufen wird.

Beispiele:

Tabelle 6: Beispiele für Formatangaben für Datum/Uhrzeit

Formatangabe	Datum/Uhrzeit	Ergebnis	Aktualisierung bei Zeitänderung
%d0(yyyy-mm-dd)	13. Dez. 2004	2004-12-13	Nein
%t0(hh.mm.ss.ttt)	10:30 Uhr und 30,123 Sekunden	10.30.30.123	Nein
%t(hh:mm:ss)	10:30 Uhr und 30 Sekunden	10:30:30	Ja

3 Operationen

3.1 Numerisch

Tabelle 7: Numerische Operationen

Operation	Formel
Addition	<numerischer Ausdruck1> **+** < numerischer Ausdruck2>
Subtraktion	<numerischer Ausdruck1> **-** < numerischer Ausdruck2>
Multiplikation	<numerischer Ausdruck1> ***** < numerischer Ausdruck2>
Division	<numerischer Ausdruck1> **/** < numerischer Ausdruck2>
Exponent	<numerischer Ausdruck1> **^** < numerischer Ausdruck2>
Sinus	**sin**(<numerischer Ausdruck>)
Cosinus	**cos**(<numerischer Ausdruck>)
Tangens	**tan**(<numerischer Ausdruck>)
Unäres Minus	**-** <numerischer Ausdruck>
Bitweises UND	<numerischer Ausdruck1> **AND** < numerischer Ausdruck2>
Bitweises ODER	<numerischer Ausdruck1> **OR** < numerischer Ausdruck2>
Bitweise Invertierung	**NOT** <numerischer Ausdruck>
Quadratwurzel	**SQRT** <numerischer Ausdruck>

3.2 Logisch

Tabelle 8: Logische Operationen

Operation	Formel
Logisches UND	<Boolescher Ausdruck1> **AND** <Boolescher Ausdruck12>
Logisches ODER	<Boolescher Ausdruck1> **OR** <Boolescher Ausdruck2>
Negation	**NOT** <Boolescher Ausdruck>

3.3 Zeichenketten (Strings)

Tabelle 9: Operationen für Zeichenketten

Operation	Formel
Verkettung	<String1> **+** <String2>
Teilstring	**SUBSTRING**(<String>, <numerischer Ausdruck1>, <numerischer Ausdruck2>) **SUBSTRING**(<String>, <numerischer Ausdruck1>) <numerischer Ausdruck1> ist der Anfangs-Index des Teilstrings, beginnend mit 0. <numerischer Ausdruck2> ist der Index des ersten Zeichens, das nicht mehr in dem Teilstring enthalten ist. Fehlt <numerischer Ausdruck2>, geht der Teilstring bis zum Ende des Originalstrings.
Wandlung String in Zahl	**TONUMBER**(<String>) <String> wird in eine Zahl umgewandelt. Wenn <String> keine Zahl darstellt, ist das Ergebnis 0.
Wandlung Zahl in String	**TOSTRING**(<numerischer Ausdruck>) **TOSTRING**(<numerischer Ausdruck>, <String>)
String-Länge	**LENGTH**(<String>)
Beispiele	
Formel	**Ergebnis**
SUBSTRING("hamburger", 4, 8)	urge
TONUMBER ("10”) + 2	12
LENGTH("hamburger")	9

3.4 Vergleich

Tabelle 10: Vergleichs-Operationen

Operation	Formel
gleich	<Ausdruck1> **=** <Ausdruck2> <Ausdruck1> **==** <Ausdruck2>
ungleich	<Ausdruck1> **!=** <Ausdruck2> <Ausdruck1> **<>** <Ausdruck2>
kleiner	<numerischer Ausdruck1> **<** <numerischer Ausdruck2>
kleiner-gleich	<numerischer Ausdruck1> **<=** <numerischer Ausdruck2>
größer	<numerischer Ausdruck1> **>** <numerischer Ausdruck2>
größer-gleich	<numerischer Ausdruck1> **>=** <numerischer Ausdruck2>

ⓘ <Ausdruck1> und <Ausdruck2> müssen jeweils vom gleichen Typ (logisch, numerisch oder String) sein.

3.5 Sonstiges

Tabelle 11: Sonstige Operationen

Operation	Formel
Verzweigung	**if** <boolescher Ausdruck> **then** <Ausdruck1> **else** <Ausdruck2> <Ausdruck1> und <Ausdruck2> müssen den gleichen Typ liefern (logisch oder numerisch). Wenn <boolescher Ausdruck> wahr ist, ist das Ergebnis der Wert von <Ausdruck1>, sonst von <Ausdruck2>.
Verbindungszustand	**OFFLINE** liefert **TRUE**, wenn Verbindungsprobleme mit einer DCU oder mit einem Controller bestehen. **OFFLINESTRING** liefert dann den Namen einer Einheit, zu der keine Verbindung besteht. Beispiel: OFFLINE(SPS4711): Wahr, wenn der Controller SPS4711 nicht erreichbar ist. OFFLINE(DCU1): Wahr, wenn DCU1 nicht erreichbar ist.
Zuweisung	variable **:=** <Ausdruck> Variable erhält den Wert, den <Ausdruck>zurückliefert. Variable kann ein Signal oder eine VU- bzw. DACQ- lokale Variable sein. Der Datentyp der rechten Seite muss mit der linken übereinstimmen.
Block	**begin** <Ausdruck1>; <Ausdruck2>;

Operation	Formel
	end <Ausdruck1>, <Ausdruck2>usw. werden nacheinander ausgewertet.
Zyklische Wiederholung	**oncePerSecond** <Ausdruck> **oncePerMinute** <Ausdruck> **oncePerHour** <Ausdruck> **oncePerDay** <Ausdruck> <Ausdruck>wird einmal pro Sekunde/Minute/Stunde/Tag aufgerufen
Flanke	**risingEdge** <boolescher Ausdruck> **fallingEdge** <boolescher Ausdruck> Liefert TRUE, wenn der Wert von <boolescher Ausdruck> von FALSE auf TRUE (risingEdge) bzw. von TRUE auf FALSE (fallingEdge) wechselt
Log	**stdlog**(appname, msgclass, errornr, text) text wird ins Standardlog ausgegeben, mit Anwendungsname appname (String), Meldungsklasse msgclass (String der Länge 1) und Fehlernummer errornr (numerisch). **fileLog**(path\filename, text) text wird an die durch path\filename vollständig beschriebene Datei (z.B. C:\MDE-Log\log.txt) angehängt. Der Pfad muss existieren, die Datei wird erzeugt.
Testausgabe	**debugOut**(text) text wird auf die Java-Konsole ausgegeben, wenn eine vorhanden ist.
Daten senden	**sendToForcam**(text) **sendToClient**(text) text wird an einen definierten Empfänger geschickt, der üblicherweise ein anderes Programm von Forcam ist. Adresse und Port des Empfängers werden in javis.ini im Abschnitt [forcamsend] bzw. [clientSend] mit den Parametern address und port angegeben (default: localhost mit Port 10.000). Hier kann mit dataport auch festgelegt werden, auf welchem Port die Antwort dieses Programms empfangen wird.
Daten empfangen	Definierte Variable einer Javis DACQ können von Clienten durch das Senden einer XML-Message beeinflusst werden. **1. Direktes Setzen von Signalen in Devices** (z.B. speicherprogrammierbaren Steuerungen). Die angesprochenen Signale müssen in der Javis-DB definiert sein. Beispiel: Setzen des Signals SPS1:D4711: SOLLTEMP1: <?xml version="1.1"?> <SETSIGNAL CONTROLLER="SPS1" DEVICE="D4711" VARNAME="SOLLTEMP1" VALUE="9"> </SETSIGNAL> **2. Setzen von DACQ-internen Javis-Variablen.**

Operation	Formel
	Beispiel: Numerische Variable %N%MYNUMBERVAR setzen: <?xml version="1.0"?> <TCO NUMBER="9701"> <DOUBLE NAME="MYNUMBERVAR" VALUE="0.5"/> </TCO> Beispiel: String-Variable %S%MYSTRINGVAR setzen: <?xml version="1.0"?> <TCO NUMBER="9701"> <STRING NAME="MYSTRINGVAR" VALUE="Hallo"/> </TCO> Der Vorsatz **%N%** bzw. **%S%** wird in Javis automatisch erzeugt, je nachdem ob der Tag-Name DOUBLE bzw. STRING lautet. Die TCO-Nummer muss 9701 sein.
Datenbankzugriff	**execSql**([database,) statement) **execSqlAsync**([database,] statement) Es wird der in statement enthaltene Update- bzw. Insert-Befehl unmittelbar bzw. asynchron via resistenter Queue in database ausgeführt. **selectFromDatabase**([database,] query) Es wird die in query angegebene Abfrage ausgeführt. Als Ergebnis wird stets der erste gefundene Wert als String zurückgegeben (unabhängig vom Typ des Tabellenwertes). Ein Leerstring wird geliefert, wenn die Abfrage kein Ergebnis liefert oder der Tabellenwert null ist. ⚠ ACHTUNG: Sowohl **execSql** als auch **selectFromDatabase** werden unmittelbar ausgeführt. Es muss programmtechnisch sichergestellt werden, dass diese blockierenden Funktionen nur dann aufgerufen werden, wenn sie unbedingt erforderlich sind. Weniger kritisch sind Datenbankeinträge mit Hilfe der Funktion **execSqlAsync**, da hier nur ein Eintrag in eine Queue erfolgt. Hier ist das Datenbanksystem selbst das begrenzende Element. Es muss nur sichergestellt sein, dass die Datenbank die Einträge im zeitlichen Mittel verarbeiten kann. Ist der optionale Parameter **database** nicht angegeben, wird **scriptdb** angenommen. Ansonsten werden die Parameter der <database>-entsprechenden Paragrafen in javis.ini verwendet. Die Namen der Parameter entsprechen denen der normalen Datenbank. **Beispiele**: [javisdb] connectionurl=jdbc:oracle:thin:@kfcoracle:1521:fact45 username=kh password=kh driver=oracle.jdbc.driver.OracleDriver [scriptdb] connectionurl=jdbc:oracle:thin:@kfcorcl10:1521:fact username=t0409 password=t0409 driver=oracle.jdbc.driver.OracleDriver

Operation	Formel
	Ist der Abschnitt [**scriptdb**] nicht vorhanden, wird die normale Datenbankverbindung benutzt.

4 Beispiel-Skripte

4.1 Auslesen und Versenden des Betriebszustands

```
//
// Task: Send machine state / status_reason to runtime
// Created: 2015-11-26
// Version: 1.0
// Author:  FORCAM MDC
//
// ---------------------------------------------------------------
//
// Incoming signals from DCU
// M_PROD       = machine in production
//
// Outgoing information to rule engine
// state        = machine state
// status_reason  = status reason
//
// ---------------------------------------------------------------

var_local
begin
// GENERAL LOGIC VARIABLES
   seconds: number;
// SCRIPT INIZIALIZING VARIABLES
   initialized: boolean;
// WPL STATUS REASON
   status_reason: number;
   status_reasonOld: number;
// MACHINE STATE
   state: number;
   stateOld: number;
// LOGGING CHANGED SIGNALS
   logString: string;
   logstringOld: string;
end;

begin
// INITIALIZE SCRIPT VARIABLES START
   if not initialized and not offline(@|PLC|@)then
   begin
//    set initialized to perform initializing once
     initialized := true;
   end;
// INITIALIZE SCRIPT VARIABLES END

// ACTIONS ONCE PER SECOND START
```

```
   oncePerSecond
   begin
      seconds:= seconds + 1;
   end;
// ACTIONS ONCE PER SECOND END

// LOGGING SIGNALS WHEN CHANGED START
   logstring := "@|PLC|@ Signals : " + " @|PLC|@ offline : " + toString(offline(@|PLC|@))
                              + " M_PROD : "        + toString(@|PLC|@:M_PROD);
   if logString <> logstringOld then
   begin
    stdlog("Javis-DACQ", "I", 0, logString);
    logstringOld := logString;
   end;
// LOGGING SIGNALS WHEN CHANGED END

// DEFINITION state status_reason START
   if offline(@|PLC|@) then
   begin
    state := 1;                          // 1 = No production 2 = production
    status_reason := 12;                 // no connection
   end
   else if @|PLC|@:M_PROD then
   begin
    state := 2;                          // 1 = No production 2 = production
    status_reason := 0;                  // 0 = no malfunction
   end
   else
   begin                                 // all other cases
    state := 1;                          // 1 = No production 2 = production
    status_reason := 1;                  // 1 = 999 = undefined stoppage
   end;
// DEFINITION state status_reason END

// SEND state status_reason START
   if (status_reason <> status_reasonOld) or (state <> stateOld) then
   begin
      stdlog("Javis-DACQ", "I", 0, "@|WPL|@ send state " + toString(state) + " reason " +
toString(status_reason));
      debugOut("@|WPL|@ send state " + toString(state) + " reason " + toString(status_reason)+
"\n");
      sendStateWorkplace("@|WPL|@", state, status_reason);
      status_reasonOld := status_reason;
      stateOld := state;
   end;
// SEND state status_reason END
end;
```

4.2 Auslesen und Versenden des Betriebszustands und Mengen

```
//
// Task: Send machine state / status_reason / quantities to runtime
// Created: 2015-11-26
// Version: 1.0
// Author:  FORCAM MDC
//
// ------------------------------------------------------------
//
// Incoming signals from DCU
// M_PROD        = machine in production
// ABS_CNT1      = absolute counter 1 on PLC
//
// Outgoing information to rule engine
// state         = machine state
// status_reason  = status reason
// counterSEND    = machine strokes / quantity
//
// ------------------------------------------------------------

var_local
begin
// GENERAL LOGIC VARIABLES
   seconds: number;
// SCRIPT INIZIALIZING VARIABLES
   initialized: boolean;
// PIECE COUNT VARIABLES
   counter: number;
   counterOLD: number;
   counterSend: number;
// WPL STATUS REASON
   status_reason: number;
   status_reasonOld: number;
// MACHINE STATE
   state: number;
   stateOld: number;
// LOGGING CHANGED SIGNALS
   logString: string;
   logstringOld: string;
end;

begin
// INITIALIZE SCRIPT VARIABLES START
   if not initialized and not offline(@|PLC|@)then
   begin
//    initialize counter
     counter := @|PLC|@:ABS_CNT1;
     counterOld := @|PLC|@:ABS_CNT1;
```

```
//    set initialized to perform initializing once
      initialized := true;
    end
    else if initialized then
    begin
      counter := @|PLC|@:ABS_CNT1;
    end;
// INITIALIZE SCRIPT VARIABLES END

// ACTIONS ONCE PER SECOND START
    oncePerSecond
    begin
       seconds:= seconds + 1;
    end;
// ACTIONS ONCE PER SECOND END

// LOGGING SIGNALS WHEN CHANGED START
    logstring := "@|PLC|@ Signals : " + " @|PLC|@ offline : " + toString(offline(@|PLC|@))
                              + " M_PROD : "        + toString(@|PLC|@:M_PROD)
                              + " ABS_CNT1 : "      + toString(@|PLC|@:ABS_CNT1);
    if logString <> logstringOld then
    begin
      stdlog("Javis-DACQ", "I", 0, logString);
      logstringOld := logString;
    end;
// LOGGING SIGNALS WHEN CHANGED END

// DEFINITION state status_reason START
    if offline(@|PLC|@) then
    begin
      state := 1;                       // 1 = No production 2 = production
      status_reason := 12;              // no connection
    end
    else if @|PLC|@:M_PROD then
    begin
      state := 2;                       // 1 = No production 2 = production
      status_reason := 0;               // 0 = no malfunction
    end
    else
    begin                               // all other cases
      state := 1;                       // 1 = No production 2 = production
      status_reason := 1;               // 1 = 999 = undefined stoppage
    end;
// DEFINITION state status_reason END

// DEFINITION COUNTER START
   if counter > counterOLD then // counter on PLC is incremented
   begin
     counterSend := counter - counterOLD;
     counterOLD := counter;
```

```
  end
  else if counter < counterOLD then // counter on PLC is reset
  begin
    counterSend := counter;
    counterOLD := counter;
  end
  else
  begin
    counterSend := 0;
  end;
// DEFINITION COUNTER END

// SEND state status_reason START
   if (status_reason <> status_reasonOld) or (state <> stateOld) then
   begin
      stdlog("Javis-DACQ", "I", 0, "@|WPL|@ send state " + toString(state) + " reason " +
toString(status_reason));
      debugOut("@|WPL|@ send state " + toString(state) + " reason " + toString(status_reason)+
"\n");
      sendStateWorkplace("@|WPL|@", state, status_reason);
      status_reasonOld := status_reason;
      stateOld := state;
   end;
// SEND state status_reason END

// SEND STROKES / QUANTITY START
   if counterSend > 0 then
   begin
      stdlog("Javis-DACQ", "I", 0, "@|WPL|@ send quantity : " + toString(counterSend));
      debugOut("@|WPL|@ send strokes or quantity : " + toString(counterSend) + " \n");
      sendCountWorkplace("@|WPL|@", 1, counterSend);     //send quantity to workplace
      counterSend := 0;
   end;
// SEND STROKES / QUANTITY END
end;
```

4.3 Auslesen und Versenden des Betriebszustands und Hüben

```
//
// Task: Send machine state / status_reason / strokes to runtime
// Created: 2015-11-26
// Version: 1.0
// Author:  FORCAM MDC
//
// ------------------------------------------------------------
//
// Incoming signals from DCU
// M_PROD        = machine in production
// ABS_CNT1      = absolute counter 1 on PLC
//
// Outgoing information to rule engine
// state         = machine state
// status_reason  = status reason
// counterSEND    = machine strokes / quantity
//
// ------------------------------------------------------------

var_local
begin
// GENERAL LOGIC VARIABLES
   seconds: number;
// SCRIPT INIZIALIZING VARIABLES
   initialized: boolean;
// PIECE COUNT VARIABLES
   counter: number;
   counterOLD: number;
   counterSend: number;
// WPL STATUS REASON
   status_reason: number;
   status_reasonOld: number;
// MACHINE STATE
   state: number;
   stateOld: number;
// LOGGING CHANGED SIGNALS
   logString: string;
   logstringOld: string;
end;

begin
// INITIALIZE SCRIPT VARIABLES START
   if not initialized and not offline(@|PLC|@)then
   begin
//    initialize counter
     counter := @|PLC|@:ABS_CNT1;
     counterOld := @|PLC|@:ABS_CNT1;
```

```
//    set initialized to perform initializing once
      initialized := true;
    end
    else if initialized then
    begin
      counter := @|PLC|@:ABS_CNT1;
    end;
// INITIALIZE SCRIPT VARIABLES END

// ACTIONS ONCE PER SECOND START
    oncePerSecond
    begin
       seconds:= seconds + 1;
    end;
// ACTIONS ONCE PER SECOND END

// LOGGING SIGNALS WHEN CHANGED START
    logstring := "@|PLC|@ Signals : " + " @|PLC|@ offline : " + toString(offline(@|PLC|@))
                              + " M_PROD : "        + toString(@|PLC|@:M_PROD)
                              + " ABS_CNT1 : "      + toString(@|PLC|@:ABS_CNT1);
    if logString <> logstringOld then
    begin
      stdlog("Javis-DACQ", "I", 0, logString);
      logstringOld := logString;
    end;
// LOGGING SIGNALS WHEN CHANGED END

// DEFINITION state status_reason START
    if offline(@|PLC|@) then
    begin
      state := 1;                       // 1 = No production 2 = production
      status_reason := 12;              // no connection
    end
    else if @|PLC|@:M_PROD then
    begin
      state := 2;                       // 1 = No production 2 = production
      status_reason := 0;               // 0 = no malfunction
    end
    else
    begin                               // all other cases
      state := 1;                       // 1 = No production 2 = production
      status_reason := 1;               // 1 = 999 = undefined stoppage
    end;
// DEFINITION state status_reason END

// DEFINITION COUNTER START
   if counter > counterOLD then // counter on PLC is incremented
   begin
     counterSend := counter - counterOLD;
     counterOLD := counter;
```

```
  end
  else if counter < counterOLD then // counter on PLC is reset
  begin
    counterSend := counter;
    counterOLD := counter;
  end
  else
  begin
    counterSend := 0;
  end;
// DEFINITION COUNTER END

// SEND state status_reason START
   if (status_reason <> status_reasonOld) or (state <> stateOld) then
   begin
      stdlog("Javis-DACQ", "I", 0, "@|WPL|@ send state " + toString(state) + " reason " +
toString(status_reason));
      debugOut("@|WPL|@ send state " + toString(state) + " reason " + toString(status_reason)+
"\n");
      sendStateWorkplace("@|WPL|@", state, status_reason);
      status_reasonOld := status_reason;
      stateOld := state;
   end;
// SEND state status_reason END

// SEND STROKES / QUANTITY START
   if counterSend > 0 then
   begin
      stdlog("Javis-DACQ", "I", 0, "@|WPL|@ send strokes : " + toString(counterSend));
      debugOut("@|WPL|@ send strokes or quantity : " + toString(counterSend) + " \n");
      sendStrokesWorkplace("@|WPL|@", counterSend, 1);   //send strokes to workplace
      counterSend := 0;
   end;
// SEND STROKES / QUANTITY END
end;
```

4.4 Senden einer Status-Information als XML

Bild 3: Skript SendStatus (Beispiel)

Es wird alle 10 Sekunden eine Status-Information als XML-Message an den Client gesendet, dessen TCP/IP-Adresse der Funktion **sendToForcam** zugeordnet ist, wenn sich der Status des Device **1001746** (Funktionseinheit, Maschine) in der Variablen **%S%@d** geändert hat.
@d ist ein Platzhalter für den Namen des Device, dem dieses Script gehört.

5 Weitere Funktionen

ⓘ Die mit * markierten Funktionen benötigen eine spezielle Buchungslogik. Mit ** markierte Funktionen benötigen zusätzlich zu einer speziellen Buchungslogik eine Konfiguration im Shop Floor Terminal.

Tabelle 12: Liste weiterer Funktionen

	Funktionsname	Parameter		Ergebnis	Runtime Command	Detail

set	SENDSTATEWORKPLACE	string
		int
		int
		int
		[int]
		
		or
		
		string
		int
		int
		int
		int
		Int
		int
		int
		int
		“MDE”
			WPL
			State
			Reason
			Reason detail
			Reason detail 2
			
			or
			
			WPL
			State
			Reason level 1
			Reason level 2
			Reason level 3
			Reason level 4
			Reason level 5
			Reason level 6
			Reason level 7
			“MDE”
					MachineStatusCommand	Ändern Sie den WPL-Maschinenstatus mit optionalen Störungsgründen in nachfolgenden Detailebenen.
						
						für 1 Statusgrund-Detailebene:
						`sendStateWorkplace("@|WPL|@", state, reason, reason_detail);`
						
						für 2 Statusgrund-Detailebenen:
						`sendStateWorkplace("@|WPL|@", state, reason, reason_detail, reason_detail2);`
						
						oder für mehr als 2 Statusgrund-Detailebenen:
						`sendStateWorkplace("@|WPL|@", state, reason_level1, reason_level2, reason_level3, reason_level4, reason_level5, reason_level6,, reason_level7, "MDE");`
						
						⚠ Störgrundlevel 2 (oder 2-7 nach Code, getrennt durch Komma (müssen alle gesetzt und 0 sein, falls leer), zusätzlicher Parameter „MDE“ muss als letzter Parameter gesetzt werden.

	Funktionsname	Parameter		Ergebnis	Runtime Command	Detail
set	SENDSTATEMACHINE	string int int int [int] **or** string int int int int Int int int int “MDE”	MA State Reason Reason detail Reason detail 2 **or** WPL State Reason level 1 Reason level 2 Reason level 3 Reason level 4 Reason level 5 Reason level 6 Reason level 7 “MDE”		MachineStatusCommand	see SENDSTATEWORKPLACE
set	SENDSTROKESWORKPLACE	string int int	WPL Strokes Counter number		MachineStrokeCommand	sendStrokesWorkplace("@|WPL|@", strokes, 1);
set	SENDCOUNTWORKPLACE	string int int	WPL Counter number Count		MachineCountCommand	sendCountWorkplace("@|WPL|@", counter_number, count);

set	SENDCOUNTMACHINE	string int int	MA Counter number Count		MachineCountCommand	SENDCOUNTMACHINE bewirkt dasselbe wie SENDCOUNTWORKPLACE, mit dem Unterschied, dass der Maschinenname anstelle des Arbeitsplatznamens angegeben werden muss. Im Prinzip ist diese Funktion überflüssig, da es eine 1:1-Beziehung zwischen Arbeitsplatz und Maschine gibt.
set	SENDSTROKESMACHINE	string Int int	MA Strokes Counter number		MachineStrokeCommand	SENDSTROKESMACHINE bewirkt dasselbe wie SENDSTROKESWORKPLACE, mit dem Unterschied, dass der Maschinenname anstelle des Arbeitsplatznamens angegeben werden muss. Im Prinzip ist diese Funktion überflüssig, da es eine 1:1-Beziehung zwischen Arbeitsplatz und Maschine gibt.
set	SENDQUANTITYWORKPLACE **	string int string int	WPL Quantity amount Quantity reason Counter number		MachineQuantityCommand	Mengenmeldung mit einem Mengenbetrag und einem Mengengrund (Mnemonik) für eine bestimmte Zählernummer (0 bis 7) senden. sendQuantityWorkplace("@|WPL|@", quantity_amount, quantity_reason, 0)
set	SENDQUANTITYMACHINE	string int string int	MA Quantity amount Quantity reason Counter number		MachineQuantityCommand	Mengenmeldung mit einem Mengenbetrag und einem Mengengrund (Mnemonik) für eine bestimmte Zählernummer (0 bis 7) senden.
set	SENDCOUNTERARRAYWORKPLACE	string int… boolean…	APL Counter Flag for counter		MachineCounterArrayCommand	⚠ Die Felder müssen gleich groß sein.
set	SENDCOUNTERARRAYMACHINE	string int… boolean…	MA Counter Flag for counter		MachineCounterArrayCommand	⚠ Die Felder müssen gleich groß sein.
set	SENDEVENTS	int… string… long… long…	Event Code WPL or MA Value			⚠ Die Felder müssen gleich groß sein.
get	INSPECTIONNOTIFICATION	string	WPL	string		Gibt die Prüfbenachrichtigung für den WPL zurück. Gibt on, off oder null zurück, wenn keine Prüfung existiert.

get	WORKPLACEFIELD	string string	WPL Field name	string		Gibt ein Feld für einen Arbeitsplatz zurück. Gibt "" zurück, wenn die Verbindung zur Runtime nicht verfügbar ist. Gibt "-1" zurück, wenn ein weiterer Fehler auftritt. Wenn die Autostatus-Berechnung im Skript erfolgen soll, muss die LCUPDATE CONF INTERVAL LEADING OP oderUPDATE CONF INTERVAL SEQUENTIAL SEQUENTIAL im Logik-Realtime-Prozess sein, um das Feld maxConfidenceDuration, (nötig für die Statusableitung) abfragen zu können. Für weitere Informationen über die notwendigen LCs, die die Informationen an DACQ (Realtime Verarbeitung) senden, siehe **Design of logic component library - Concepts - Confluence (forcam.com)** Für weitere Informationen zu den abfragbaren Attributen über den Parameter **Field name** siehe **WorkplaceField - Forcam FORCE Documentation - Confluence**

get	OPERATIONFIELD	string
		string
		[int]
			WPL
			Field name
			Index
				string		Gibt ein Feld für eine Operation zurück.
						Gibt "" zurück, wenn kein aktiver Führungsvorgang gefunden wurde oder die Verbindung zur Runtime nicht verfügbar ist.
						Gibt "-1" zurück, wenn ein weiterer Fehler auftritt.
						
						Index 0: Leading operation (Standard)
						Index 1-7: Side operation
						
						Die Vorgänge werden von der Runtime über LC an DACQ übermittelt
						`OPERATION TRIGGER ON OPERATION PHASE CHANGE`
						
						Example:
						`timePerUnit := toNumber(operationField("@|WPL|@","timePerUnit"));`

set	SENDCLAMPINGCHANGEWORKPLACE *	string int int	WPL Counter Pallet number		ClampingChangeCommand	SENDCLAMPINGCHANGEWORKPLACE sendet einen Wechsel der Aufspannung. Argumente sind Arbeitsplatzname, Palettennummer und Palettenseite.
set	SENDCLAMPINGCHANGEMACHINE *	string int int	MA Pallet number Pallet side number		ClampingChangeCommand	SENDCLAMPINGCHANGEMACHINE sendet einen Wechsel der Aufspannung. Argumente sind Maschinenname, Palettennummer und Palettenseite.
set	SENDOPERATIONAUTOSTARTID **	string string	WPL Autostart ID			Sendet die AutostartID an ffworker (Shop Floor Terminal).
set	MAPNEW	string	MAP			Erzeugt eine neue MAP im System mit dem angegebenen Namen. Vorhandene MAPs werden überschrieben.
set	MAPPUT	string	MAP			Fügt einen Parametereintrag in die MAP ein. Ein bestehender Eintrag wird überschrieben. Wenn es keine MAP mit dem angegebenen Namen gibt, wird sie erstellt.
set	MAPCLEAR	string string string	MAP Parameter Value			Löscht alle Einträge in der MAP.
set	MAPDELETE	string	MAP			Löscht die MAP. ⚠ Muss nach dem Aufruf SENDGENERICCOMMAND erfolgen.
get	SENDGENERICCOMMAND	string string string	WPL Terminal ID MAP	string		Sendet ein Server-Ereignis an die SFT mit der angegebenen MAP. Die TERMINAL-ID muss in der MAP angegeben werden und mit der entsprechenden Terminal-ID der SFT übereinstimmen. Gibt die GUID des gesendeten Serverereignisses zurück. MAPDELETE muss anschließend aufgerufen werden, um Speicher freizugeben.
get	RESPONSEMAPRECEIVED	string	GUID	boolean		Gibt **true** zurück, wenn eine Antwort-MAP für die angegebene GUID empfangen wurde. Gibt **false** zurück, wenn nicht.
get	GETRESPONSEVALUE	string string	GUID Parameter	string		Gibt den Wert des angegebenen Parameters in der MAP zurück. Gibt einen leeren String zurück, wenn der Parameter in der MAP nicht vorhanden ist. DELETERESPONSEMAP muss aufgerufen werden, nachdem alle Parameterwerte in Variablen gespeichert wurden.

set	DELETERESPONSEMAP	string	GUID			Löscht die MAP mit ihrem Inhalt und gibt den belegten Speicher frei. ⚠ Muss nach dem Speichern aller Parameterwerte in Variablen erfolgen.
set	SENDSAPBARCODE	string string	WPL or MA SAP Barcode			SENDSAPBARCODE sendet einen Barcode an SAP. Das erste Argument ist der Arbeitsplatz. Das zweite Argument ist der Barcode.
set	SENDSAPLABELPRINT	string string	WPL or MA SAP Label Print			SENDSAPLABELPRINT sendet einen Etikettendruck an SAP. Das erste Argument ist der Arbeitsplatz. Das zweite ist der Etikettentext.
get	FOLLOWOPERATIONFIELD	string string [int]	WPL Field name Index	string		Gibt ein Feld für eine Nachfolge-AVO zurück. Index 0: Leading operation (Standard) Index 1-7: Side operation
set	SENDQUANTITYOPERATION	string int string [int]	WPL Quantity amount Quantity reason Index		OperationQuantityCommand	Sendet eine Mengenmeldung mit Mengenbetrag und Mengengrund (Mnemonik) von der Runtime über LC an das DACQ `OPERATION TRIGGER ON OPERATION PHASE CHANGE` Index 0: Leading operation (Standard) Index 1-7: Side operation
set	SENDQUANTITYFOROPERATION	string int string int	WPL Quantity amount Quantity reason Operation number		OperationQuantityCommand	Sendet eine Mengenmeldung mit dem Mengenbetrag und dem Mengengrund (Mnemonik) für einen Vorgang.
get/ set	MALFUNCTIONREASON	string string	WPL Controller	string		Legt den Namen des Pseudosignals basierend auf dem WPL und dem Controller fest. Registriert die Signale. Gibt den letzten Wert Grund für die WPL.
get/ set	MALFUNCTIONDETAIL	string string	WPL Signal name	string		Legt den Namen des Pseudosignals basierend auf dem WPL und dem Controller fest. Registriert die Signale. Gibt den letzten Wert Grund für die WPL.

set	CLOSEWORKERSU	string	WPL			Hinzufügen einer neuen Methode zum Schließen einer SU-Box im Worker: CLOSEWORKERSU("@|APL|@", "@|SU_NUMBER|@", "workerID", "COMP_NUMBER_CHECK_ASSIGNMENT");
set	SENDWORKERSU	string string string string	WPL			Sendet die SU-Nummer an den Arbeiter, der den Wert auswertet.
get	GETWORKERSUFEEDBACK	[string] [string]	WPL SU number	string		Gibt die Rückmeldung eines SU über: variable := getworkersignalfeedback("@|WPL|@", 0);
set	DELETEWORKERSU	string [string]	WPL SU number			Löscht den SU-Eintrag aus dem MAP des WPL. 0: Löscht die gesamte MAP 1: Löscht den Eintrag der WPL 2: Löscht den Eintrag für den WPL
get	GETWORKERGENERALFEEDBACK	string string	WPL Feedback	string		Gibt das allgemeine Prüfergebnis für den WPL und die zugeordnete SU zurück. Gibt 99 zurück, wenn ein Fehler aufgetreten ist (siehe Protokolldatei javis_DACQ-overall.log)
get	GETWORKERSIGNALFEEDBACK	string int	WPL Type	string		Gibt das Ergebnis der Signalprüfung für den WPL und die zugeordnete SU Type 0: Key Type 1: Value
set	DELETEWORKERGENERAL	[string] [string]	WPL SU number			Löscht den allgemeinen Eintrag aus der HashMap des WPL. 0: Löscht die gesamte MAP 1: Löscht den Eintrag der WPL 2: Löscht den Eintrag für den WPL
set	DELETEWORKERSIGNAL	[string] [string]	WPL SU number			Löscht den Signaleintrag aus dem MAP des WPL. 0: Löscht die gesamte MAP 1: Löscht den Eintrag der WPL 2: Löscht den Eintrag für den WPL

set	SETSIGNAL	string string string string	WPL SPS Signal Value			Setzt ein definiertes Signal am WPL. ⚠ Der WPL muss den gleichen Namen haben wie der Controller. Ein vorangestelltes "SPS" im Namen wird akzeptiert.
get	SELECTFROMDATABASE	string string	iniParagraph (random name for further use) SQL statement	string		Startet eine SQL-Datenbankabfrage und gibt das Ergebnis zurück. ⓘ Die Anweisung muss mit "SELECT" beginnen.
get	GETFILENAME	string [int]	Filepath Type	string		Gibt einen Dateinamen aus dem Dateipfad vom Typ zurück: 0: den Dateinamen mit der Erweiterung 1: nur die Erweiterung 2: den vollständigen Dateipfad ⓘ Ohne einen Typ-Parameter wird nur der Dateiname zurückgegeben.
get	LENGTH	string	Value	int		Gibt die Länge einer Zeichenkette zurück.
get	SUBSTRING	string int [int]	Value Index begin Index end			Der dritte Parameter ist frei. Ist kein Wert gesetzt, wird der maximale Index des Input-Strings verwendet.

set	SENDDOMAINCHANGE	string
		string
		string
		string
			WPL
			Domain
			Field
			Value
						Sendet einen DomainAttributeChange-Befehl an die Runtime.
						
						Als Domäne verwenden Sie "OPERATION" oder "WORKPLACE".
						
						Wenn der Domäne OPERATION ist, wird der Befehl mit der aktiven führenden Operation gesendet.
						
						sendDomainChange("@|WPL|@", "OPERATION", "strokeFactor", 5); // Hubfaktor für aktiven Vorlaufbetrieb einstellen
						
						sendDomainChange("@|WPL|@", "WORKPLACE", "DB_WORKPLACE_USER_FIELD:1", "Hello World"); // Benutzerfeld 1 von WPL einstellen
						
						Als Referenz siehe
						Booking Commands#DomainAttributeChangeCommand (forcam.com)
						
						Die Operationen werden von der Runtime über LC an DACQ übermittelt
						`OPERATION TRIGGER ON OPERATION PHASE CHANGE.`
						
						Für weitere Informationen siehe
						Design of logic component library - Concepts - Confluence (forcam.com)

get	SPLITSTRING	string string int	Value Regular expression Index		string	Zerlegt eine Zeichenkette anhand eines regulären Ausdrucks in ein Feld von Teilen und gibt den Teil des Indexes zurück. Beispiel: Eine Zeichenkette "test1-test2-test3" mit dem regulären Ausdruck "-" und dem Index 1 ergibt "test2". Für weitere Informationen siehe **https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#split-java.lang.String-**

set	SENDBGACTIVITYCOMMAND	string string string	WPL Activity MAP			Sendet ein Server-Ereignis mit dem Geltungsbereich SERVER_SIDE_ACT und dem Aktivitätsbezeichner (Hashkey) zusammen mit der angegebenen MAP-ID an die HEADLESS-SFT. Die Aktivitätskennung muss mit der Aktivitätskennung der entsprechenden HEADLESS-SFT übereinstimmen. Gibt die GUID des gesendeten Server-Ereignisses zurück. ⚠ MAPDELETE muss anschließend aufgerufen werden, um Speicher freizugeben.
set	SENDGLOBALEVENTCOMMAND	string string string … string	Event MAP Value 2 … Value 5	string		Sendet eine MAP mit dem IGlobalEventMessageSenderService Die MAP sollte vorher mit Hilfe der Funktionen MAPNEW und MAPPUT gefüllt werden.
set	SENDGLOBALEVENTCOMMAND	string string string … string	Event Value 1 Value 2 … Value 5	string		Erzeugt eine MAP mit den angegebenen Werten und sendet sie mit dem IGlobalEventMessageSenderService.
set	CURRENTSYSTEMTIMESTAMP	number	Time			current_time := CURRENTSYSTEMTIMESTAMP(); // UTC-TimeZone in ms
set	SENDSTATEOPERATIONCOMMAND	string string string string string number	WPL Order number Order split Operation number Operation split Operation phase		OperationPhaseCommand	Senden Sie einen Zustandsoperationsbefehl für einen AVO- und AVO-Phasencode. Es kann ein Auftrags- und Vorgangssplitwert definiert werden. Wenn kein Auftrag oder keine Betriebsaufteilung erforderlich ist, verwenden Sie stattdessen eine leere Zeichenfolge.

set	SENDOPERATIONQTYCOMMAND	string string string string string number string	WPL Order number Order split Operation number Operation split Quantity amount Quantity reason		OperationPhaseCommand	Sendet einen AVO-Mengenbefehl für einen Vorgang mit einem Mengenbetrag und einem Mengengrund (Mnemonik). Es kann ein Auftrags- und AVO-Splitwert definiert werden. Wenn kein Auftrags- oder AVO-Split erforderlich ist, verwenden Sie stattdessen eine leere Zeichenfolge.
set	SENDORDERCOMPLETIONCOMMAND	string string string	WPL Order number Order split		DomainAttributeChangeCommand	Sendet einen Befehl zur Änderung eines Domänenattributs ORDER_COMPLETION mit der WPL-Domäne für einen bestimmten Auftrag. Es kann ein AVO-Splitwert definiert werden. Wenn kein Auftrags- oder AVO-Split benötigt wird, ist stattdessen eine leere Zeichenkette zu verwenden. Die AVOs werden über LC abgeschlossen `COMPLETE_ORDER_ON_ATTRIBUTE_CHANGE`

[]	optional parameter	MA	Machine	SU	Storage unit
…	Array of values	MAP	Parameter map (generic hash map)	SFT	Shopfloor Terminal
WPL	Workplace	LC	Logic Component (see Design of logic component library - Concepts - Confluence (forcam.com))		

5.1 Von WorkplaceField abfragbare Felder

Durch die Funktion WorkplaceField können verschiedene Felder abgefragt werden. Die nötige Struktur ist folgend:
WorkplaceField("@|WPL|@", Feldname)

Die folgenden Felder können abgefragt werden (Feldname aus oberer Struktur mit einem Feld ersetzen):

- maxConfidenceDuration
  (wichtig für die Autostatus-Berechnung im DACQ-Skript)
- ASSIGNEDOPERATIONS
- BOOKINGTYPEID
- CLIENT
- COMPANYCODE
- CUREPERIOD
- CUREPERIODFACTOR
- DERIVEDCOLOR
- DERIVEDDESCRIPTION
- DERIVEDMNEMONIC
- DESCRIPTION
- ERPCYCLETIME
- EXTERNALKEY
- GENERICUSERFIELD.1
- GENERICUSERFIELD.2
- GENERICUSERFIELD.3
- GENERICUSERFIELD.4
- GENERICUSERFIELD.5
- GENERICUSERFIELD.6
- GENERICUSERFIELD.7
- GENERICUSERFIELD.8
- GENERICUSERFIELD.9
- GENERICUSERFIELD.10
- GENERICUSERFIELD.11
- GENERICUSERFIELD.12
- GENERICUSERFIELD.13
- GENERICUSERFIELD.15
- GENERICUSERFIELD.16
- GENERICUSERFIELD.17
- GENERICUSERFIELD.18
- GENERICUSERFIELD.19
- GENERICUSERFIELD.20

- GENERICUSERFIELD.21
- GENERICUSERFIELD.22
- GENERICUSERFIELD.23
- GENERICUSERFIELD.24
- GENERICUSERFIELD.25
- GENERICUSERFIELD.26
- GENERICUSERFIELD.27
- GENERICUSERFIELD.28
- GENERICUSERFIELD.29
- GENERICUSERFIELD.30
- GENERICUSERFIELD.31
- GENERICUSERFIELD.32
- GENERICUSERFIELD.33
- GENERICUSERFIELD.34
- GENERICUSERFIELD.35
- GENERICUSERFIELD.36
- GENERICUSERFIELD.37
- GENERICUSERFIELD.38
- GENERICUSERFIELD.39
- GENERICUSERFIELD.40
- GENERICUSERFIELD.41
- GENERICUSERFIELD.42
- GENERICUSERFIELD.43
- GENERICUSERFIELD.44
- GENERICUSERFIELD.45
- GENERICUSERFIELD.46
- GENERICUSERFIELD.47
- GENERICUSERFIELD.48
- GENERICUSERFIELD.49
- GENERICUSERFIELD.50
- ISAUTOMATICRECODINGSET
- LASTWORKPLACESTATECHANGETIMESTAMP
- OBJECTREFERENCE
- PLANT
- SHIFTDAYCHANGEOFFSET
- STROKECOUNTER
- STROKEFREQUENCY
- UNDEFINEDSTOPPAGES

5.2 Von OperationField abfragbare Felder

Durch die Funktion OperationField können verschiedene Felder abgefragt werden. Die nötige Struktur ist folgend:
OperationField ("@|WPL|@", Feldname, index)

Die folgenden Felder können abgefragt werden (Feldname aus oberer Struktur mit einem Feld ersetzen):

- ALTERNATEOPERATIONNUMBER
- BLOCKNUMBER
- BOOKEDQUANTITY.1 or YIELDCOUNT
- BOOKEDQUANTITY.2 or SCRAPCOUNT
- BOOKEDQUANTITY.3 or REWORKCOUNT
- BOOKEDYIELDQUANTITY
- CLIENT
- COMPANYCODE
- CONFIRMATIONNUMBER
- CONTROLKEY
- COUNTERNUMBER
- DEFAULTSTROKEFACTOR
- DEFAULTTRANSPORTQUANTITY
- DERIVEDCOLOR
- DERIVEDDESCRIPTION
- DERIVEDMNEMONIC
- DESCRIPTION
- DISPLAYQUANTITYUNIT
- ERPPLANNEDSCRAPQUANTITY
- ERPREWORKQUANTITY
- ERPSCRAPQUANTITY
- ERPSTATUS
- ERPSTATUSIDS
- ERPSTATUSLA
- ERPYIELDQUANTITY
- EXTERNALKEY
- FUNCTIONTYPE
- GENERICDOUBLEVALUE.SET_TRANSPORT_QUANTITY
- GENERICUSERFIELD.1
- GENERICUSERFIELD.2
- GENERICUSERFIELD.3
- GENERICUSERFIELD.4
- GENERICUSERFIELD.5
- GENERICUSERFIELD.6
- GENERICUSERFIELD.7
- GENERICUSERFIELD.8

- GENERICUSERFIELD.9
- GENERICUSERFIELD.10
- GENERICUSERFIELD.11
- GENERICUSERFIELD.12
- GENERICUSERFIELD.13
- GENERICUSERFIELD.14
- GENERICUSERFIELD.15
- GENERICUSERFIELD.16
- GENERICUSERFIELD.17
- GENERICUSERFIELD.18
- GENERICUSERFIELD.19
- GENERICUSERFIELD.20
- GENERICUSERFIELD.21
- GENERICUSERFIELD.22
- GENERICUSERFIELD.23
- GENERICUSERFIELD.24
- GENERICUSERFIELD.25
- GENERICUSERFIELD.26
- GENERICUSERFIELD.27
- GENERICUSERFIELD.28
- GENERICUSERFIELD.29
- GENERICUSERFIELD.30
- GENERICUSERFIELD.31
- GENERICUSERFIELD.32
- GENERICUSERFIELD.33
- GENERICUSERFIELD.34
- GENERICUSERFIELD.35
- GENERICUSERFIELD.36
- GENERICUSERFIELD.37
- GENERICUSERFIELD.38
- GENERICUSERFIELD.39
- GENERICUSERFIELD.40
- GENERICUSERFIELD.41
- GENERICUSERFIELD.42
- GENERICUSERFIELD.43
- GENERICUSERFIELD.44
- GENERICUSERFIELD.45
- GENERICUSERFIELD.46
- GENERICUSERFIELD.47
- GENERICUSERFIELD.48
- GENERICUSERFIELD.49
- GENERICUSERFIELD.50

- OBJECTREFERENCE
- OPERATIONNUMBER
- OPERATIONSPLIT
- OPERATIONTEXT
- ORDERNUMBER
- ORDERSPLIT
- ORDERTYPE
- OVERDELIVERYCHECK
- OVERDELIVERYQUANTITY
- PHASE$CODE
- PLANT
- PRODUCTIONVERSION
- SEQUENCE
- STANDARDUNIT1
- STANDARDUNIT2
- STANDARDUNIT3
- STANDARDUNIT4
- STANDARDUNIT5
- STANDARDUNIT6
- STANDARDVALUE1
- STANDARDVALUE1MS
- STANDARDVALUE2
- STANDARDVALUE2MS
- STANDARDVALUE3
- STANDARDVALUE3MS
- STANDARDVALUE4
- STANDARDVALUE4MS
- STANDARDVALUE5
- STANDARDVALUE5MS
- STANDARDVALUE6
- STANDARDVALUE6MS
- STROKECOUNTER
- STROKEFACTOR
- TARGETEND
- TARGETQUANTITY
- TARGETSETUPTIME
- TARGETSTART
- TEMPQUANTITY
- TIMEPERSTROKE
- TIMEPERUNIT
- UNDERDELIVERYCHECK
- UNDERDELIVERYQUANTITY

- USERSTATUS
- WORKPLACEGROUP

6 Anhang

6.1 Änderungstabelle

Tabelle 13: Änderungen in Version 5.12

Version	Datum	Name	Änderung
1	2019-02-01	Ali Egilmez	Initiale Dokument-Erstellung
2	2022-08-19	Peter Müller	Kapitel **5 Weitere Funktionen** angepasst Kapitel **5.2 Von OperationField abfragbare Felder** ergänzt Kapitel **6.1 Änderungstabelle** e