



# DACQ Script Language

Version 5.12

---

*Manual*

- 
- DOC Document: Manual - DACQ Script Language.docx
  - P Release date: 2022-11-08
  - E Document version: 1
  - U Author: ABöer
-

## Content

<b>1</b>	<b>General.....</b>	<b>3</b>
<b>2</b>	<b>Formula Elements.....</b>	<b>5</b>
2.1	Numeric Constants .....	5
2.2	String Constants .....	5
2.3	Logical (Boolean) Constants.....	5
2.4	Signal Values.....	6
2.5	Variable .....	6
2.6	Text Replacement in Text Objects.....	7
2.7	Format Specifications .....	7
2.7.1	Numbers .....	7
2.7.2	Strings .....	8
2.7.3	Date and Time.....	9
<b>3</b>	<b>Operations.....</b>	<b>10</b>
3.1	Numerical .....	10
3.2	Logical.....	10
3.3	Strings.....	11
3.4	Comparison.....	12
3.5	Miscellaneous .....	12
<b>4</b>	<b>Sample Script.....</b>	<b>15</b>
4.1	Reading and Sending the Operating State .....	15
4.2	Reading and Sending the Operating State and Quantities .....	17
4.3	Reading and Sending the Operating State and Strokes .....	20
4.4	Sending Status Information as XML .....	23
<b>5</b>	<b>Additional Functions.....</b>	<b>24</b>
5.1	Field to poll from WorkplaceField .....	35
5.2	Field to poll from OperationField .....	37
<b>6</b>	<b>Document History.....</b>	<b>41</b>

## General

---

### 1 General

- i** This manual assumes knowledge in the use of FORCAM FORCE IIOT.  
If you do not have any knowledge of using FORCAM FORCE IIOT, take the time to familiarize yourself with the basics.

We recommend that you use our Academy.

The FORCAM Academy (<https://forcam.com/academie/>) provides the knowledge to effectively use the methods for digital transformation and the technologies for the Smart Factory.  
Based on lean manufacturing and TPM methods, our institute team will guide you to initiate changes in the company and to use the technologies correctly.

The machine communication in FORCAM FORCE™ is carried out via the DCU. In order to do so, a controller (control unit) is connected to the machine that reads out the machine data.

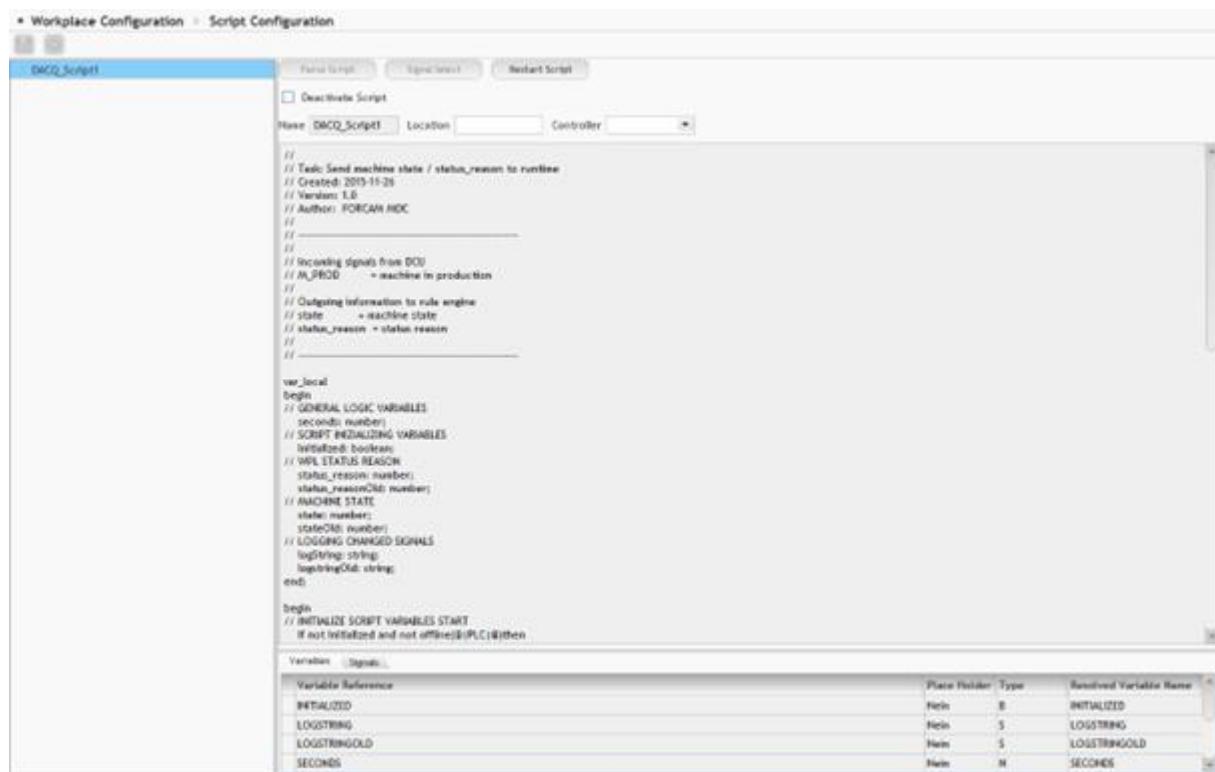
The DCU contains all relevant information (controller type, IP-address, Port, signals a.s.o.) of a machine. One DCU is able to collect data of up to 100 machines. In order not to endanger the stability of all processes it is advised to connect not more than 50 machines to one DCU.

The DCU communicates with the machine and polls data in short intervals (e.g. every 100msec or once per second) or receives them from an intermediate OPC server or a WAGO box. The DCU collects unprocessed signals and transfers them to the DACQ (via RMI).

The DACQ normalizes the received data and assigns them to operating states. Then the DACQ sends relevant information like the machine status or quantities to the server. A script within the DACQ controls the interpretation of the received data.

## General

Scripts are executed whenever the value of a referenced signal or variable has changed. Possible exceptions are date and time values (see section 0).



The screenshot shows the 'Script Configuration' dialog box. The script is named 'DACQ\_Script1'. The code is as follows:

```

// Task: Send machine state / status_reason to runtime
// Created: 2015-11-28
// Version: 1.0
// Author(s): FORCAM HOC
//
// Recording signals from DCO
// M_PCODE = machine in production
//
// Outgoing information to rule engine
// state: > machine state
// status_reason = status reason
//
var_local
Begin
// GENERAL LOGIC VARIABLES
    seconds(number)
// SCRIPT INITIALIZING 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

```

The 'Variables' tab is selected, showing the following table:

Variable Reference	Place Holder	Type	Received Variable Name
INITIALIZED	Hein	B	INITIALIZED
LOGSTRING	Hein	S	LOGSTRING
LOGSTRINGOLD	Hein	S	LOGSTRINGOLD
SECONDS	Hein	N	SECONDS

**Fig. 1: Scripting in the workbench (example)**

A script that consists of multiple single statements always has to be combined to one block with **begin ... end**.

A statement is always completed with a semicolon. Exceptions to this are:

- Begin/end  
**begin** can never be followed by a semicolon. It is possible to put a semicolon after **end**, but it is not necessary.
- if ... then ... else  
Statements before **then** and **else** can never be completed by a semicolon.

## 2 Formula Elements

This section describes all useable formula elements. Formulas are not case sensitive.

### 2.1 Numeric Constants

Period as well as comma are permitted to be used as decimal separators.

Example: 3,14 or 3.14.

### 2.2 String Constants

Strings are enclosed in quotation marks (e.g. "hello").

Quotation marks within strings are prefixed by a backslash (e.g. "This is a \"real\" quotation mark").

The following special characters can be inserted in a string with a backslash:

**Table 1: Special characters in strings**

Special character	Function
\"	(Double) quotation mark
\\	Backslash
\n	Word-wrap
\t	Tabulator
\b	Backspace
\r	Carriage return
\xnn	Sign with ASCII value nn (hexadecimal), e.g. \x0 for zero sign

### 2.3 Logical (Boolean) Constants

The logical constants are **true** for true and **false** for false. Instead of true for true, a "**1**" can also be used for true and for false a "**0**" can also be used for false, but these must not be combined together in a query, e.g. if X = 1 and Y = true. Such a combination will cause an error. You may only use either true/false or 1/0.

## 2.4 Signal Values

A signal value has the following form:

**controller:device:name** or **controller:name**

The value **device** can be omitted if the corresponding signal is defined without device or if **controller** and **name** are enough to achieve an unambiguous designation.

## 2.5 Variable

Variables are program-intern flags that can be read out or be set by multiple objects or scripts, respectively. Variables can be of the following types:

**Table 2: Types of variables**

Variable	Prefix	Meaning
<b>boolean</b>	%B%	Boolean variable: expression which can have only 2 values (true vs. false)
<b>number</b>	%N%	Numeric variable: expression which consists only of numbers
<b>string</b>	%S%	String variable: expression which consists of a character string

The variable type is defined at the beginning of a script. In the following example, seconds are displayed numeric and the script initialization are Boolean.

```

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;

```

**Fig. 2: Definition of variable types**

## 2.6 Text Replacement in Text Objects

The result of a text formula can be inserted into the fixed text of a string object with **%f** or **%[format specification]f**. This result is formatted according to the format specification (see section 2.7).

## 2.7 Format Specifications

### 2.7.1 Numbers

Format specification for numbers have the following form:

**[-][0][total length [.decimal]][x|X]**

The result string is filled to the **total length**, but also always depicts the complete number, even if it becomes longer because of it. Take the following behavior into account:

- If **total length** is indicated and not **decimal**, decimal places are not displayed. By indicating **0** it is filled with zeros, otherwise with blanks.
- When indicating **-** the formatting is left aligned, otherwise right aligned.
- By indicating **x** or **X** hexadeciml display with lower case or upper-case letters when lower- or upper-case **X**. Decimal places are always cut off in this case.

**Examples** (blanks are shown as dots):

**Table 3: Examples for the format specification for numbers**

Format specification	Number	Result
<b>3</b>	9	..9
<b>03.3</b>	3.1	003.100
<b>-5</b>	9	9....
<b>3X</b>	255	0FF
<b>x</b>	10	a

## 2.7.2 Strings

Format specifications for strings have the following form:

**[-][min length[.max length]]**

The result string is filled to the **max length** or is cut to **min length**, respectively.

When indicating - the formatting is left aligned, otherwise right aligned.

**Examples** (blanks are shown as dots):

**Table 4: Examples for the format specification for strings**

Format specification	String	Result
<b>8</b>	hello	...hello
<b>8.9</b>	hello	...hello
	hello Mama	hello Mam
<b>-8</b>	Hello	hello...

### 2.7.3 Date and Time

Date and time are formatted with **%d** or **%t** respectively. The following abbreviations are used with date and time:

**Table 5: Abbreviations for date and time**

Area	Abbreviation	Meaning
<b>Date</b>	y	year
	m	month
	d	day
<b>Time</b>	h	hour
	m	minute
	s	second
	t	millisecond

- i** A 0 behind **d** or **t**, respectively, prevents that - in case of a time change - the object is updated, or the script is called-up.

**Examples:**

**Table 6: Examples for format specifications for date/time**

Format specification	Date/time	Result	Update in case of a time change
<b>%d0(yyyy-mm-dd)</b>	13. Dec. 2004	2004-12-13	no
<b>%t0(hh.mm.ss.ttt)</b>	10:30 o'clock and 30,123 seconds	10.30.30.123	no
<b>%t(hh:mm:ss)</b>	10:30 o'clock and 30 seconds	10:30:30	yes

## 3 Operations

### 3.1 Numerical

**Table 7: Numerical operations**

Operation	Formula
<b>Addition</b>	<numeric expression1> + < numeric expression2>
<b>Subtraction</b>	<numeric expression1> - < numeric expression2>
<b>Multiplication</b>	<numeric expression1> * < numeric expression2>
<b>Division</b>	<numeric expression1> / < numeric expression2>
<b>Exponent</b>	<numeric expression1> ^ < numeric expression2>
<b>Sinus</b>	<b>sin</b> (<numeric expression>)
<b>Cosine</b>	<b>cos</b> (<numeric expression>)
<b>Tangent</b>	<b>tan</b> (<numeric expression>)
<b>Unary minus</b>	- <numeric expression>
<b>Bitwise AND</b>	<numeric expression1> <b>AND</b> < numeric expression2>
<b>Bitwise OR</b>	<numeric expression1> <b>OR</b> < numeric expression2>
<b>Bitwise inversion</b>	<b>NOT</b> <numeric expression>
<b>Square root</b>	<b>SQRT</b> <numeric expression>

### 3.2 Logical

**Table 8: Logical Operations**

Operation	Formula
<b>Logic AND</b>	<Boolean expression1> <b>AND</b> <Boolean expression12>
<b>Logic OR</b>	<Boolean expression1> <b>OR</b> <Boolean expression2>
<b>Negation</b>	<b>NOT</b> <Boolean expression>

### 3.3 Strings

**Table 9: Operations for strings**

Operation	Formula
<b>Linking</b>	<code>&lt;string1&gt; + &lt;string2&gt;</code>
<b>Substring</b>	<p><b>SUBSTRING(&lt;string&gt;, &lt;numeric expression1&gt;, &lt;numeric expression2&gt;)</b>  <b>SUBSTRING(&lt;string&gt;, &lt;numeric expression1&gt;)</b></p> <p>&lt;numeric expression1&gt; is the start index of the substring, beginning with 0.</p> <p>&lt;numeric expression2&gt; is the index of the first sign, that is not contained in the string anymore.</p> <p>If &lt;numeric expression2&gt; is missing, the substring extends up to the end of the original string.</p>
<b>Conversion string into number</b>	<p><b>TONUMBER(&lt;string&gt;)</b>  &lt;string&gt; is converted into a number. If &lt;string&gt; does not represent a number, the result is 0.</p>
<b>Conversion number into string</b>	<p><b>TOSTRING(&lt;numeric expression&gt;)</b>  <b>TOSTRING(&lt;numeric expression&gt;, &lt;string&gt;)</b></p>
<b>String length</b>	<b>LENGTH(&lt;string&gt;)</b>
<b>Example</b>	
Formula	Result
<b>SUBSTRING("hamburger", 4, 8)</b>	urge
<b>TONUMBER ("10") + 2</b>	12
<b>LENGTH("hamburger")</b>	9

## 3.4 Comparison

**Table 10: Comparison operations**

Operation	Formel
<b>equal</b>	<expression1> = <expression2> <expression1> == <expression2>
<b>unequal</b>	<expression1> != <expression2> <expression1> <> <expression2>
<b>less than</b>	<numeric expression1> < <numeric expression2>
<b>less than - equal</b>	<numeric expression1> <= <numeric expression2>
<b>greater than</b>	<numeric expression1> > <numeric expression2>
<b>greater than - equal</b>	<numeric expression1> >= <numeric expression2>

**i** <expression1> and <expression2> always must be of the same type (logical, numerical or string).

## 3.5 Miscellaneous

**Table 11: Other operations**

Operation	Formula
<b>Branching</b>	<b>if</b> <Boolean expression> <b>then</b> <expression1> <b>else</b> <expression2>  <expression1> and <expression2> have to deliver the same type (logical or numerical). If <Boolean expression> is true, the result is the value of <expression1>, otherwise of <expression2>.
<b>Connection status</b>	<b>OFFLINE</b> renders <b>TRUE</b> , if connection problems with a DCU or with a controller occur. <b>OFFLINESTRING</b> then delivers the name of a unit to which no connection exists.  Example: <b>OFFLINE(SPS4711)</b> : True, if the controller SPS4711 cannot be reached. <b>OFFLINE(DCU1)</b> : True, if DCU1 cannot be reached.
<b>Assignment</b>	<b>variable := &lt;expression&gt;</b>  Variable receives the value, which <expression> returns. Variable can be a signal or a VU or - respectively - DACQ local variable. The data type of the right side has to match the left one.
<b>Block</b>	<b>begin</b> <expression1>; <expression2>; <b>end</b> <expression1>, <expression2> a.s.o. are analyzed successively.

## Operations

Operation	Formula
Cyclic repetition	<b>oncePerSecond</b> <expression> <b>oncePerMinute</b> <expression> <b>oncePerHour</b> <expression> <b>oncePerDay</b> <expression>  <expression> is called-up once per second/minute/hour/day.
Edge	<b>risingEdge</b> <Boolean expression>  <b>fallingEdge</b> <Boolean expression>  Renders TRUE, if the value of <Boolean expression> changes from FALSE to TRUE (risingEdge) or from TRUE to FALSE (fallingEdge), respectively.
Log	<b>stdlog</b> (appname, msgclass, errornr, text)  text is output into the standard log, with application name appname (string), message class msgclass (string with length 1) and error number errornr (numeric).  <b>fileLog</b> (path\filename, text)  text is suffixed to the file that is completely described with path\filename (e.g. C:\MDE-Log\log.txt). The path has to exist, the file is created.
Test output	<b>debugOut</b> (text)  text is output on the Java console if one is present.
Send data	<b>sendToForcam</b> (text) <b>sendToClient</b> (text)  text is sent to a predefined recipient, which is usually another program by Forcam. Address and port of the recipient are specified in javis.ini in section [forcamsend] or [clientSend] respectively, with the parameters address and port (default: localhost with port 10.000). It is possible to define here with dataport, with which port the answer of this program should be received.
Receive data	Defined variables of a Javis DACQ can be influenced by clients by sending a XML message.  <b>1. Direct setting of signals in devices</b> (e.g. programmable logic controllers). The addressed signals have to be defined in the Javis DB.  Example: Setting of signal SPS1:D4711: TARTEMP1:  <?xml version="1.1"?> <SET SIGNAL CONTROLLER="SPS1" DEVICE="D4711" VARNAME="TARTEMP1" VALUE="9"> </SET SIGNAL>  <b>2. Setting of DACQ internal Javis variables.</b>  Example: Set numeric variable %N%MYNUMBERVAR:  <?xml version="1.0"?> <TCO NUMBER="9701"> <DOUBLE NAME="MYNUMBERVAR" VALUE="0.5"/>

## Operations

Operation	Formula
	<p>&lt;/TCO&gt;</p> <p>Example: Set string variable %S%MYSTRINGVAR:</p> <pre>&lt;?xml version="1.0"?&gt; &lt;TCO NUMBER="9701"&gt; &lt;STRING NAME="MYSTRINGVAR" VALUE="Hello"/&gt; &lt;/TCO&gt;</pre> <p>The prefix %N% bzw. %S% is created automatically in Javis, depending on whether the day name is DOUBLE or STRING, respectively. The TCO number has to be 9701.</p>
Database access	<p><b>execSql([database,] statement)</b>  <b>execSqlAsync([database,] statement)</b></p> <p>The update or insert command, respectively which is contained in statement, is executed immediately or asynchronous, respectively via resistant queue in database.</p> <p><b>selectFromDatabase([database,] query)</b></p> <p>The poll specified in query is executed. The first found value is returned as string as the result (independent of the table value type). A blank string is delivered if the queue does not produce a result or if the table value is zero.</p> <p><b>⚠ ATTENTION:</b>  <b>execSql</b> as well as <b>selectFromDatabase</b> are executed immediately. It is necessary to ensure by programming that these blocking functions are only called-up if they are essential. Less critical are database entries with help of the function <b>execSqlAsync</b>, because only an entry into a queue occurs here. The database system itself is here the limiting element. It is necessary to ensure that the database is able to process the entries within the averaged time.  If the optional parameter <b>database</b> is not specified, <b>scriptdb</b> is assumed. Otherwise the parameters of the paragraphs that correspond with the &lt;database&gt; in javis.ini are used. The parameter names are equal to those of the normal database.</p> <p><b>Examples:</b></p> <pre>[javisdbs] 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</pre> <p>If section <b>[scriptdb]</b> does not exist, the normal database connection is used.</p>

## 4 Sample Script

### 4.1 Reading and Sending the Operating State

```

// 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 INITIALIZING 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

```

## Sample Script

---

```

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

// LOGGING SIGNALS WHEN CHANGED START
logstring := "@|PLC|@ Signals : " + "@|PLC|@ offline : " + toString(@|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
    state := 1;           // all other cases
    status_reason := 1;    // 1 = No production 2 = production
                           // 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;

```

## Sample Script

---

### 4.2 Reading and Sending the Operating State and Quantities

```

// 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 INITIALIZING 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;

```

## Sample Script

---

```

// 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(@|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
    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

```

## Sample Script

---

```

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;

```

## Sample Script

---

### 4.3 Reading and Sending the Operating State and Strokes

```

// 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 INITIALIZING 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;

```

## Sample Script

---

```

// 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(@|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
    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

```

## Sample Script

---

```

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;

```

## Sample Script

### 4.4 Sending Status Information as XML

**Script sendStatus (Device 1001746)**

Name	sendStatus	Device	1001746
------	------------	--------	---------

```

oncePerSecond
begin
if %N%count@d <= 10 then
begin
  %N%count@d := %N%count@d + 1
end
else
begin
  %N%count@d := 1;
  if %S%stat@d <> %S%oldstat@d and %S%stat@d <> ""
  then begin
    sendToForcam
    (
      "<?xml version='1.0'?>\n"
      +"<TCO SENDER='DCU' RECEIVER='KSMDE"
      +"SUBSTRING(@d,6,1)+""
      +"format(' MSGTIME=%d0(yyyy-mm-dd)-%t0(hh.mm.ss.tt)000')"
      +" NUMBER='9502'\n"
      +"<STRING NAME='APL' VALUE=@d>\n"
      +"</STRING>\n"
      +"<STRING NAME='STATUS' VALUE=%S%stat@d"
      +">\n"
      +"</STRING>\n"
      +"</TCO>\n"
    );
    %S%oldstat@d := %S%stat@d;
  end
end
end

```

<a href="#">Speichern</a>	<a href="#">Abbrechen</a>	<a href="#">Hilfe</a>
---------------------------	---------------------------	-----------------------

**Fig. 3: Script SendStatus (example)**

A status information is sent as XML message every 10 seconds to the client, whose TCP/IP address is assigned to the function **sendToForcam**, if the status of the device **1001746** (functional unit, machine) has changed in the variable **%S%@d**.

**@d** is a place holder for the device name this script belongs to.

## 5 Additional Functions

**i** Functions marked with \* require a special booking logic. In addition to that, functions marked with \*\* require a configuration in the Shopfloor Terminal.

Table 12: List of additional functions

	Function name	Parameter		Return	Runtime Command	Detail/Example
set	SENDSTATEWORKPLACE	string int int int [int]  <b>or</b> string int int int int Int int int int "MDE"	WPL State Reason Reason detail Reason detail 2  <b>or</b> 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	<p>Change the WPL machine state with optional malfunction reasons in subsequent detail levels.</p> <p><b>for 1 status reason detail level:</b>  <code>sendStateWorkplace("@ WPL @", state, reason, reason_detail);</code></p> <p><b>for 2 status reason detail levels:</b>  <code>sendStateWorkplace("@ WPL @", state, reason, reason_detail, reason_detail2);</code></p> <p><b>or for more than 2 status reason detail levels:</b>  <code>sendStateWorkplace("@ WPL @", state, reason_level1, reason_level2, reason_level3, reason_level4, reason_level5, reason_level6, reason_level7, "MDE");</code></p> <p><b>⚠️</b> For more than 2 status reason levels, you have to set all 7 levels (set 0 for no reason) and "MDE" as last parameter.</p>

## Additional Functions

	Function name	Parameter		Return	Runtime Command	Detail/Example
set	SENDSTATEMACHINE	string int int int [int] <b>or</b> string int int int int Int int int int “MDE”	MA State Reason Reason detail Reason detail 2 <b>or</b> 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 does the same as SENDCOUNTWORKPLACE, with the difference that the machine name must be specified instead of the workstation name. In principle, this function is superfluous, since there is a 1:1 relationship between workstation and machine.

## Additional Functions

	Function name	Parameter		Return	Runtime Command	Detail/Example
set	SENDSTROKESMACHINE	string Int int		MA Strokes Counter number	MachineStrokeCommand	SENDSTROKESMACHINE does the same as SENDSTROKESWORKPLACE, with the difference that the machine name must be specified instead of the workstation name. In principle, this function is superfluous, since there is a 1:1 relationship between workstation and machine.
set	SENDQUANTITYWORKPLACE **	string int string int		WPL Quantity amount Quantity reason Counter number	MachineQuantityCommand	Send a quantity booking with a quantity amount and quantity reason (mnemonic) for a specific counter number (0 to 7).  sendQuantityWorkplace("@ WPL @", quantity_amount, quantity_reason, 0)
set	SENDQUANTITYMACHINE	string int string int		MA Quantity amount Quantity reason Counter number	MachineQuantityCommand	Send a quantity booking with a quantity amount and quantity reason (mnemonic) for a specific counter number (0 to 7).
set	SENDCOUNTERARRAYWORKPLACE	string int... boolean...		APL Counter Flag for counter	MachineCounterArrayCommand	 The arrays need the same size.
set	SENDCOUNTERARRAYMACHINE	string int... boolean...		MA Counter Flag for counter	MachineCounterArrayCommand	 The arrays need the same size.
set	SENDEVENTS	int... string... long... long...		Event Code WPL or MA Value		 The arrays need the same size.
get	INSPECTIONNOTIFICATION	string		WPL	string	Returns the inspection notification for the WPL. Returns on, off or null, if no inspection exists.

## Additional Functions

---

get	WORKPLACEFIELD	string string	WPL Field name	string		Returns a field for a WPL. Returns "", if the connection to the Runtime is not available. Returns "-1" if another error occurs.  If Autostatus calculation shall be done in the script, the LC UPDATE CONF INTERVAL LEADING OP or UPDATE CONF INTERVAL SEQUENTIAL must be in the logic realtime process to be able to query the field maxConfidenceDuration, needed for the status derivation.  For further information about necessary LCs which send the information to DACQ (realtime process) see <a href="#">Design of logic component library - Concepts - Confluence (forcam.com)</a>  For further information about the queryable attributes by the parameter <b>Field name</b> see <a href="#">WorkplaceField - Forcam FORCE Documentation - Confluence</a>
-----	----------------	------------------	-------------------	--------	--	--

## Additional Functions

get	OPERATIONFIELD	string string [int]	WPL Field name Index	string		Returns a field for an operation. Returns "", if no active leading operation found or the connection to the Runtime is not available. Returns "-1", if another error occurred.  Index 0: Leading operation (Standard) Index 1-7: Side operation  The operations are transmitted to DACQ by the Runtime via LC OPERATION TRIGGER ON OPERATION PHASE CHANGE  Example: timePerUnit := toNumber(operationField("@ WPL @", "timePerUnit"));
set	SENDCLAMPINGCHANGEWORKPL ACE *	string int int	WPL Counter Pallet number		ClampingChangeCommand	
set	SENDCLAMPINGCHANGEMACHINE *	string int int	MA Pallet number Pallet side number		ClampingChangeCommand	
set	SENDOPERATIONAUTOSTARTID **	string string	WPL Autostart ID			Sends the Autostart ID to the Shopfloor Terminal
set	MAPNEW	string	MAP			Creates a new MAP inside the system with the given name. Existing ones will be overridden.
set	MAPPUT	string	MAP			Adds a parameter entry to the MAP. An existing entry will be overridden. If there is no MAP with the given name, it will be created.
set	MAPCLEAR	string string string	MAP Parameter Value			Clears all entries of the MAP.

## Additional Functions

---

set	MAPDELETE	string	MAP			Deletes the MAP.  ⚠ Must be done after calling SENDGENERICCOMMAND.
get	SENDGENERICCOMMAND	string string string	WPL Terminal ID MAP	string		Sends a server event to the SFT with the given MAP. The TERMINAL ID must be supplied in the MAP and match the corresponding terminal id of the SFT. Returns the GUID of the server event sent. MAPDELETE must be called afterwards to free memory.
get	RESPONSEMAPRECEIVED	string	GUID	boolean		Returns <b>true</b> , if a response MAP for the given GUID has been received. Returns <b>false</b> , if not.
get	GETRESPONSEVALUE	string string	GUID Parameter	string		Returns the value of the given parameter in the MAP. Returns an empty string if the parameter does not exist in the MAP. DELETERESPONSEMAP must be called after storing all parameter values in variables.
set	DELETERESPONSEMAP	string	GUID			Deletes the MAP with its content and frees the used memory.  ⚠ Must be done after storing all parameter values in variables.
set	SENDSAPBARCODE	string string	WPL or MA SAP Barcode			SENDSAPBARCODE sends a barcode to SAP. The first argument is the workstation. The second argument is the barcode.
set	SENDSAPLABELPRINT	string string	WPL or MA SAP Label Print			SENDSAPLABELPRINT sends a label print to SAP. The first argument is the workstation. The second is the label text.
get	FOLLOWOPERATIONFIELD	string string [int]	WPL Field name Index	string		Returns a field for a successor operation.  Index 0: Leading operation (Standard) Index 1-7: Side operation

## Additional Functions

set	SENDQUANTITYOPERATION	string int string [int]	WPL Quantity amount Quantity reason Index		OperationQuantityCommand	Sends a quantity booking with the quantity amount and quantity reason (mnemonic) to the DACQ by the Runtime via LC <b>OPERATION TRIGGER ON OPERATION PHASE CHANGE</b>  Index 0: Leading operation (Standard) Index 1-7: Side operation
set	SENDQUANTITYFOROPERATION	string int string int	WPL Quantity amount Quantity reason Operation number		OperationQuantityCommand	Sends a quantity booking with the quantity amount and quantity reason (mnemonic) for an operation.
get/ set	MALFUNCTIONREASON	string string	WPL Controller	string		Sets pseudo signal name based on the WPL and the controller. Register the signals. Returns the last value reason for the WPL.
get/ set	MALFUNCTIONDETAIL	string string	WPL Signal name	string		Sets pseudo signal name based on the WPL and the controller. Register the signals. Returns the last value detail for the WPL.
set	CLOSEWORKERSU	string	WPL			Add a new method to close a SU box in the worker: CLOSEWORKERSU("@ APL @", "@ SU_NUMBER @", "workerID", "COMP_NUMBER_CHECK_ASSIGNMENT");
set	SENDWORKERSU	string string string string	WPL			Sends the SU number to the worker for evaluating the value.
get	GETWORKERSUFEEDBACK	[string] [string]	WPL SU number	string		Returns feedback of a SU about: variable := getworkersignalfeedback("@ WPL @", 0);

## Additional Functions

Set	DELETEWORKERSU	string [string]	WPL SU number			Deletes the SU entry out of the MAP of the WPL. 0: Deletes the whole MAP 1: Deletes the entry of the WPL 2: Deletes the entry for the WPL
get	GETWORKERGENERALFEEDBACK	string string	WPL Feedback	string		Returns the general check result for the WPL and the assigned SU. Returns 99, if an error occurred (see log file javis_DACQ-overall.log)
get	GETWORKERSIGNALFEEDBACK	string int	WPL Type	string		Returns the signal check result for the WPL and the assigned SU Type 0: Key Type 1: Value
set	DELETEWORKERGENERAL	[string] [string]	WPL SU number			Deletes the general entry out of the HashMap of the WPL. 0: Deletes the whole MAP 1: Deletes the entry of the WPL 2: Deletes the entry for the WPL
set	DELETEWORKERSIGNAL	[string] [string]	WPL SU number			Deletes signal entry out of the MAP of the WPL. 0: Deletes the whole MAP 1: Deletes the entry of the WPL 2: Deletes the entry for the WPL
set	SETSIGNAL	string string string string	WPL SPS Signal Value			Sets a defined signal at the WPL. <span style="color: blue;">⚠</span> The WPL must have the same name as the controller. A prefixed "SPS" in the name is accepted.
get	SELECTFROMDATABASE	string string	iniParagraph (random name for further use) SQL statement	string		Starts a SQL database query and return the result. <span style="color: blue;">ℹ</span> The statement must start with "SELECT".
get	GETFILENAME	string [int]	Filepath Type	string		Returns a file name from the filepath of type: 0: the file name with extension 1: only the extension 2: the complete filepath <span style="color: blue;">ℹ</span> Without a type parameter, just the file name will be returned.

## Additional Functions

---

get	LENGTH	string	Value	int		Returns the length of a string.
get	SUBSTRING	string int [int]	Value Index begin Index end			The third parameter is free. If no value has been set, the max index will be used of the input string.
set	SENDDOMAINCHANGE	string string string string	WPL Domain Field Value			<p>Sends a DomainAttributeChange command to the Runtime.</p> <p>As domain use "OPERATION" or "WORKPLACE".</p> <p>If the domain is OPERATION, the command will be sent with the active leading operation.</p> <pre>sendDomainChange("@ WPL @", "OPERATION", "strokeFactor", 5); // set stroke factor for active leading operation</pre> <pre>sendDomainChange("@ WPL @", "WORKPLACE", "DB_WORKPLACE_USER_FIELD:1", "Hello World"); // set user field 1 of WPL</pre> <p>For reference see  <a href="#">Booking</a>  <a href="#">Commands#DomainAttributeChangeCommand</a>  <a href="#">(forcam.com)</a></p> <p>The operations are transmitted to DACQ by the Runtime via LC      OPERATION TRIGGER ON OPERATION PHASE CHANGE.</p> <p>For further information see  <a href="#">Design of logic component library - Concepts - Confluence</a> <a href="#">(forcam.com)</a></p>

## Additional Functions

get	SPLITSTRING	string string int	Value Regular expression Index		string	Splits a string by a regular expression in a field of parts and returns the part of index.  Example: A string "test1-test2-test3" with the regular expression "-" and index 1 will return "test2".  For further information see <a href="https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#split-java.lang.String-">https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#split-java.lang.String-</a>
set	SENDDBGACTIVITYCOMMAND	string string string	WPL Activity MAP			Sends a server event with scope SERVER_SIDE_ACT and the activity identifier (hashkey) along with the given MAP-ID to the HEADLESS-SFT.  The activity identifier must match with the activity id of the corresponding headless SFT.  Returns the GUID of the server event sent.   MAPDELETE must be called afterwards to free memory
set	SENDGLOBALEVENTCOMMAND	string string string ... string	Event MAP Value 2 ... Value 5	string		Sends a MAP with the IGlobalEventMessageSenderService  The MAP should be filled before by using the functions MAPNEW and MAPPUT.
set	SENDGLOBALEVENTCOMMAND	string string string ... string	Event Value 1 Value 2 ... Value 5	string		Creates a MAP with the given values and sends it with the IGlobalEventMessageSenderService.
set	CURRENTSYSTEMTIMESTAMP	number	Time			current_time := CURRENTSYSTEMTIMESTAMP(); // UTC-TimeZone in ms

## Additional Functions

set	SENDSTATEOPERATIONCOMMAND	string string string string string number	WPL Order number Order split Operation number Operation split Operation phase	OperationPhaseCommand	Send a state operation command for an operation and operation phase code.  An order and operation split value can be defined. If there is no order or operation split needed, use an empty string instead.
set	SENDOPERATIONQTYCOMMAND	string string string string string number string	WPL Order number Order split Operation number Operation split Quantity amount Quantity reason	OperationPhaseCommand	Sends an operation quantity command for an operation with a quantity amount and quantity reason (mnemonic).  An order and operation split value can be defined. If there is no order or operation split needed, use an empty string instead.
set	SENDORDERCOMPLETIONCOMMAND	string string string	WPL Order number Order split	DomainAttributeChangeCommand	Sends a domain attribute change command ORDER_COMPLETION with the WPL domain for a specified order.  An order split value can be defined. If there is no order or operation split needed, use an empty string instead.  The operations are completed via LC COMPLETE_ORDER_ON_ATTRIBUTE_CHANGE

[] optional parameter

... Array of values

WPL Workplace

MA Machine

MAP Parameter map (generic hash map)

LC Logic Component (see Design of logic component library - Concepts - Confluence (forcam.com))

SU Storage unit

SFT Shopfloor Terminal

## 5.1 Field to poll from WorkplaceField

You can poll different fields of a workplace using the function WorkplaceField. The structure required is as follows:

WorkplaceField("@|WPL|@", fieldname)

You can poll the following fields (replace fieldname in the above structure with a field):

- maxConfidenceDuration  
(important for the Autostatus-calculation in the DACQ-script)
- 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

## Additional Functions

---

- 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

---

## Additional Functions

### 5.2 Field to poll from OperationField

You can poll different fields of a current active operation at a workplace using the function OperationField. The structure required is as follows:

OperationField("@|WPL|@", fieldname, index)

You can poll the following fields (replace fieldname in the above structure with a field):

- 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
- ERPSTATUSSLA
- ERPYIELDQUANTITY
- EXTERNALKEY
- FUNCTIONTYPE
- GENERICDOUBLEVALUE.SET\_TRANSPORT\_QUANTITY
- GENERICUSERFIELD.1
- GENERICUSERFIELD.2
- GENERICUSERFIELD.3
- GENERICUSERFIELD.4
- GENERICUSERFIELD.5
- GENERICUSERFIELD.6

## Additional Functions

---

- 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

## Additional Functions

---

- GENERICUSERFIELD.49
- GENERICUSERFIELD.50
- OBJECTREFERENCE
- OPERATIONNUMBER
- OPERATIONSPLIT
- OPERATIONTEXT
- ORDERNUMBER
- ORDERSPLIT
- ORDERTYPE
- OVERDELIVERYCHECK
- OVERDELIVERYQUANTITY
- PHASE\$CODE
- PLANT
- PRODUCTIONVERSION
- SEQUENCE
- STANDARDUNIT1
- STANDARDUNIT2
- STANDARDUNIT3
- STANDARDUNIT4
- STANDARDUNITS5
- STANDARDUNIT6
- STANDARDVALUE1
- STANDARDVALUE1MS
- STANDARDVALUE2
- STANDARDVALUE2MS
- STANDARDVALUE3
- STANDARDVALUE3MS
- STANDARDVALUE4
- STANDARDVALUE4MS
- STANDARDVALUE5
- STANDARDVALUE5MS
- STANDARDVALUE6
- STANDARDVALUE6MS
- STROKECOUNTER
- STROKEFATOR
- TARGETEND
- TARGETQUANTITY
- TARGETSETUPTIME
- TARGETSTART
- TEMPQUANTITY
- TIMEPERSTROKE
- TIMEPERUNIT

---

## Additional Functions

- UNDERDELIVERYCHECK
- UNDERDELIVERYQUANTITY
- USERSTATUS
- WORKPLACEGROUP

## Document History

### 6 Document History

Version	Date	Name	Change
<b>1</b>	2019-02-01	Ali Egilmez	Initial document creation
<b>2</b>	2022-08-19	Peter Müller	Chapter <b>Fehler! Verweisquelle konnte nicht gefunden werden.</b> Additional Functions adjusted for 5.12 Chapter Fehler! Verweisquelle konnte nicht gefunden werden. <b>Field to poll from OperationField</b> added Chapter Fehler! Verweisquelle konnte nicht gefunden werden. <b>Document History</b> adjusted