



# Version 5.9

## DACQ Script Language

Manual

Document: **Manual - DACQ Script Language**

Created: **2017-04-18**

Last change: **2019-07-02**

Author: **AEgilmez**



COPYRIGHT 2019 BY **FORCAM GMBH**, D-88214 Ravensburg  
ALL RIGHTS RESERVED. COPY OR TRANSLATION, ALSO IN EXTRACTS  
ONLY WITH WRITTEN PERMISSION BY FORCAM GMBH

## Table of Contents

|          |  |           |
|----------|--|-----------|
| <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 .....  | 7         |
| 2.7.3    | Date and Time.....   | 8         |
| <b>3</b> | <b>Operations.....</b>                                       | <b>9</b>  |
| 3.1      | Numerical .....  | 9         |
| 3.2      | Logical.....   | 9         |
| 3.3      | Strings.....   | 10        |
| 3.4      | Comparison.....  | 11        |
| 3.5      | Miscellaneous .....  | 11        |
| <b>4</b> | <b>Sample Script.....</b>                                    | <b>14</b> |
| 4.1      | Reading and Sending the Operating State .....                | 14        |
| 4.2      | Reading and Sending the Operating State and Quantities ..... | 16        |
| 4.3      | Reading and Sending the Operating State and Strokes .....    | 19        |
| 4.4      | Sending Status Information as XML .....                      | 22        |
| <b>5</b> | <b>Additional Functions.....</b>                             | <b>23</b> |
| 5.1      | Field to poll from WorkplaceField .....                      | 26        |

## 1 General

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.

Scripts are executed whenever the value of a referenced signal or variable has changed. Possible exceptions are date and time values (see section 2.7.3).

```

• Workplace Configuration > Script Configuration
DACQ_Script1
Parse Script Signal Select Restart Script
Deactivate Script
Name DACQ_Script1 Location Controller
// 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

```

| Variable Reference | Place Holder | Type | Resolved Variable Name |
|--------------------|--------------|------|------------------------|
| INITIALIZED        | Nein         | B    | INITIALIZED            |
| LOGSTRING          | Nein         | S    | LOGSTRING              |
| LOGSTRINGOLD       | Nein         | S    | LOGSTRINGOLD           |
| SECONDS            | Nein         | N    | SECONDS                |

**Figure 1: Scripting in the workbench (example)**

## General

---

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** and **false**.

## 2.4 Signal Values

A signal value has the following form:

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

**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** hexadecimal 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.

## Formula Elements

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

### 3.2 Logical

**Table 8: Logical Operations**

| Operation | Formula   |
|-----------|---|
| Logic AND | <Boolean expression1> <b>AND</b> <Boolean expression12> |
| Logic OR  | <Boolean expression1> <b>OR</b> <Boolean expression2>   |
| Negation  | <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>                     | <b>SUBSTRING</b> ( <code>&lt;string&gt;, &lt;numeric expression1&gt;, &lt;numeric expression2&gt;</code> )<br><b>SUBSTRING</b> ( <code>&lt;string&gt;, &lt;numeric expression1&gt;</code> )<br><br><code>&lt;numeric expression1&gt;</code> is the start index of the substring, beginning with 0.<br><code>&lt;numeric expression2&gt;</code> is the index of the first sign, that is not contained in the string anymore.<br>If <code>&lt;numeric expression2&gt;</code> is missing, the substring extends up to the end of the original string. |
| <b>Conversion string into number</b> | <b>TONUMBER</b> ( <code>&lt;string&gt;</code> )<br><code>&lt;string&gt;</code> is converted into a number. If <code>&lt;string&gt;</code> does not represent a number, the result is 0.  |
| <b>Conversion number into string</b> | <b>TOSTRING</b> ( <code>&lt;numeric expression&gt;</code> )<br><b>TOSTRING</b> ( <code>&lt;numeric expression&gt;, &lt;string&gt;</code> )   |
| <b>String length</b>                 | <b>LENGTH</b> ( <code>&lt;string&gt;</code> )  |
| <b>Example</b>                       |  |
| <b>Formula</b>                       | <b>Result</b>  |
| <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><br><expression1> == <expression2>  |
| <b>unequal</b>              | <expression1> != <expression2><br><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>                   |

- ⓘ <expression1> and <expression2> always have to 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><br><br><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.<br><b>OFFLINESTRING</b> then delivers the name of a unit to which no connection exists.<br><br>Example:<br><b>OFFLINE(SPS4711)</b> : True, if the controller SPS4711 cannot be reached.<br><b>OFFLINE(DCU1)</b> : True, if DCU1 cannot be reached. |
| <b>Assignment</b>        | <b>variable := &lt;expression&gt;</b><br><br>Variable receives the value, which <expression> returns.<br>Variable can be a signal or a VU or - respectively - DACQ local variable.<br>The data type of the right side has to match the left one.  |
| <b>Block</b>             | <b>begin</b><br><expression1>;<br><expression2>;<br><b>end</b><br><br><expression1>, <expression2> a.s.o. are analyzed successively.  |

|                          |  |
|--------------------------|--|
| <b>Cyclic repetition</b> | <b>oncePerSecond</b> <expression><br><b>oncePerMinute</b> <expression><br><b>oncePerHour</b> <expression><br><b>oncePerDay</b> <expression><br><br><expression> is called-up once per second/minute/hour/day.  |
| <b>Edge</b>              | <b>risingEdge</b> <Boolean expression><br><br><b>fallingEdge</b> <Boolean expression><br><br>Renders TRUE, if the value of <Boolean expression> changes from FALSE to TRUE (risingEdge) or from TRUE to FALSE (fallingEdge), respectively.   |
| <b>Log</b>               | <b>stdlog</b> (appname, msgclass, errornr, text)<br><br>text is output into the standard log, with application name appname (string), message class msgclass (string with length 1) and error number errornr (numeric).<br><br><b>fileLog</b> (path\filename, text)<br><br>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.   |
| <b>Test output</b>       | <b>debugOut</b> (text)<br><br>text is output on the Java console if one is present.  |
| <b>Send data</b>         | <b>sendToForcam</b> (text)<br><b>sendToClient</b> (text)<br><br>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.   |
| <b>Receive data</b>      | Defined variables of a Javis DACQ can be influenced by clients by sending a XML message.<br><br><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.<br><br>Example: Setting of signal SPS1:D4711: TARTEMP1:<br><br><pre>&lt;?xml version="1.1"?&gt; &lt;SET SIGNAL CONTROLLER="SPS1"       DEVICE="D4711"       VARNAME="TARTEMP1"       VALUE="9"&gt; &lt;/SET SIGNAL&gt;</pre><br><b>2. Setting of DACQ internal Javis variables.</b><br><br>Example: Set numeric variable %N%MYNUMBERVAR:<br><br><pre>&lt;?xml version="1.0"?&gt; &lt;TCO NUMBER="9701"&gt; &lt;DOUBLE NAME="MYNUMBERVAR" VALUE="0.5"/&gt; &lt;/TCO&gt;</pre><br>Example: Set string variable %S%MYSTRINGVAR: |

|                        |  |
|------------------------|--|
|                        | <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>   |
| <b>Database access</b> | <p><b>execSql([database,] statement)</b><br/> <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><br/> <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.<br/> 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><br/> [javisdb]<br/> connectionurl=jdbc:oracle:thin:@kfcoracle:1521:fact45<br/> username=kh<br/> password=kh<br/> driver=oracle.jdbc.driver.OracleDriver</p> <p>[scriptdb] connectionurl=jdbc:oracle:thin:@kfcorcl10:1521:fact<br/> username=t0409<br/> password=t0409<br/> driver=oracle.jdbc.driver.OracleDriver</p> <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(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
    state := 1;           // all other cases
    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 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;

```

## Sample Script

---

```

        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(@|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;           // all other cases
    status_reason := 1; // 1 = No production 2 = production
    status_reason := 999; // 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;

```

## Sample Script

---

```

        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 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;

```

## Sample Script

---

```

        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(@|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;           // all other cases
    status_reason := 1; // 1 = No production 2 = production
    status_reason := 999; // 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;

```

## Sample Script

---

```

        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 Sending Status Information as XML



```

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
end

```

**Figure 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 Shop Floor Terminal.

**Table 12: List of additional functions**

|     | Function name         | Parameter 1 |     | Parameter 2 |                        | Parameter 3 |                                    | Parameter 4 |  | Return | Runtime Command      | Detail  |
|-----|-----------------------|-------------|-----|-------------|------------------------|-------------|------------------------------------|-------------|--|--------|----------------------|---|
| set | SEND-STATE-WORK-PLACE | string      | WPL | int         | Machine status by code | int         | Malfunction reason level 1 by code | int         | Malfunction reason level 2 (or level 2 to 7, divided by comma (must all be set and be 0 if empty), additional parameter "MDE" must be set as last parameter) by code |        | MachineStatusCommand | level 1 (default): sendStateWorkplace("@ WPL @", machine-state, status_reason, reason_detail);<br>level 2 (malfunction reason mapping): sendStateWorkplace("@ WPL @", machine-state, status_reason, reason_detail, reason_detail2);<br>complete vector: sendStateWorkplace("@ WPL @", machine-state, reason_level1, reason_level2, reason_level3, reason_level4, reason_level5, reason_level6, reason_level7, "MDE"); |
| set | SEND-STATEMA-CHINE    | string      | MA  | int         | Machine status by code | int         | Malfunction reason level 1 by code | int         | Malfunction reason level 2 (or level 2 to 7, divided by comma (must all be set and be 0 if empty), additional  |        | MachineStatusCommand |   |

## Additional Functions

---

|     |                            |        |     |        |                 |        |                          |  |        |                         |   |
|-----|----------------------------|--------|-----|--------|-----------------|--------|--------------------------|--|--------|-------------------------|---|
|     |                            |        |     |        |                 |        |                          | parameter "MDE" must be set as last parameter) by code |        |                         |   |
| set | SENDSTROKES-WORKPLACE      | string | MA  | int    | Strokes         | int    | Counter Number           |  |        | MachineStroke Command   | sendStrokes-Workplace("@ WPL @", strokes, 1);   |
| set | SEND-COUNT-WORK-PLACE      | string | WPL | int    | Counter Number  | int    | Counter                  |  |        | MachineCount-Command    | sendCountWork-place("@ WPL @", counter, count);   |
| set | SEND-COUNTMACHINE          | string | MA  | int    | Counter Number  | int    | Counter                  |  |        | MachineCount-Command    |   |
| set | SENDSTROKESMACHINE         | string | MA  | int    | Strokes         | int    | Counter Number           |  |        | MachineStroke Command   |   |
| set | SEND-QUANTITYWORK-PLACE ** | string | WPL | int    | Quantity Amount | string | Quantity reason Mnemonic | Counter Number   | 0-7    | MachineQuanti-tyCommand | sendQuantityWork-place("@ WPL @", quantity, qty_reason_mnemonic, 0)   |
| set | SEND-QUANTITYMACHINE       | string | MA  | int    | Quantity Amount | string | Quantity reason Mnemonic | Counter Number   | 0-7    | MachineQuanti-tyCommand |   |
| get | WORK-PLACE-FIELD           | string | WPL | string | Counter         |        |                          |  | string |                         | Returns a field for a workplace. If Autostatus calculation shall be done in the script, the LC UPDATE CONF INTERVAL LEADING OP or UPDATE CONF INTERVAL SEQUENTIAL has to be in the logic realtime process to be able to query field maxConfidenceDuration needed for the status derivation. |

## Additional Functions

|     |  |        |       |        |                  |      |                  |  |  |                                 |   |   |
|-----|--|--------|-------|--------|------------------|------|------------------|--|--|---------------------------------|---|---|
|     |  |        |       |        |                  |      |                  |  |  |                                 |   | Please see Design of logic component library#MessagingtoDACQ for necessary LCs which send the information to DACQ (relatime process).<br>Following attributes can be queried by parameter "fieldname": WorkplaceField |
| set | SENDCLA<br>MPINGCH<br>ANGE-<br>WORK-<br>PLACE *    | string | WPL   | int    | Counter          | int  | Pallet<br>Number |  |  | Clamp-<br>ingChan-<br>geCommand |   |   |
| set | SENDCLA<br>MPINGCH<br>ANGEMA-<br>CHINE *           | string | MA    | int    | Pallet<br>Number | int  | Pallet<br>Number |  |  | Clamp-<br>ingChan-<br>geCommand |   |   |
| set | SEN-<br>DOPERA-<br>TIONAU-<br>TO-<br>STARTID<br>** | string | WPL   | string | Autostart<br>ID  |      |                  |  |  |                                 | Sends the AutostartID to ffworker<br>(Shop Floor Terminal)  |   |
| get | SUB-<br>STRING                                     | string | Value | int    | Index<br>Begin   | _int | Index<br>End     |  |  |                                 | The third parameter is free. If no value has been set, the max index will be used of the input sting. |   |

Int = integer

WPL = Workplace

MA = Machine

## 5.1 Field to poll from WorkplaceField

You can poll different fields 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
- GENERICUSERFIELD.21

## Additional Functions

---

- 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