Created by Werner Krings, Tani GmbH
Creation date: 2019.7.1, last modified 2022.2.15

Technical Data PLC Engine

The **PLC Engine** allows data processing in production lines. It connects controllers, devices, SCADA systems and databases from various manufacturers. It processes the data and dispatches them between the connected devices and other systems. It collects data, manages them and creates overviews for OEE applications.
Additional all the data can be handled over OPC. This are processed data and all data from the connected controllers, devices, OPC systems and databases.

PLC Engine is the perfect component for using it for communications in Industrie 4.0.

You will implement important use cases with **PLC Engine**

Boosts the PLC communication with logical optimizing of the requests. Often this will happen to error texts from the controllers. The messages need to be fetched normally if the corresponding error number is changing and a new error condition becomes true. Without any change to Your SCADA system the communication speeds up. The main reason is less load on the communication line.

Exchanges data between controllers and devices without PLC programming or SCADA functionality. That's how it's done: PLC Engine is reading the data from a controller or device and writes them into another controller or device. The writing normally takes place if the data are changing. With configurable triggers data which are depending on other data conditions for the writing can be defined. The data will converted if necessary. This is important if both controllers are different: One Siemens S7, the other a Rockwell Control Logix.
More complex handling is possible, too: Collecting of data from one or multiple controllers, do some calculation with the data, and write the result to a controller or device. Of course, any OPC compatible system can use the same data also.

Collects and processes data
PLC Engine is reading data from one or more controllers.
The data are collected and possibly calculated or normalized.
If all data are in place they are ready to process for other systems or OPC access.
During the logical processing the data are blocked for access.
Syncronous data access are waiting until the collection is finished.
Configurable error conditions are configurable: If one of the controllers is not available constant data are used, or an error number is processed.

Exchanges data with common databases. You can read data from controllers and write them directly into common databases - with optional calculations. Data readed from databases can be written into controllers, too. All the data are available for OPC systems also.

Collects data over long time for OEE into a local or external database. So easily OEE applications can be build. The integrated websites are offering the collected and calculated data directly as curves or other. Additionally the data can be handled over OPC.

Functionality

With logic tables you define all the things your plant will need:

- Read and write data from industrial controllers and devices.
- Exchange data with databases. Read, write, update, delete.
- Read and write files, check for changes, create files, delete files.
- Send emails.
- Read and write data over OPC UA and Classic OP.
- Sequence chains with conditions for changing the actual step.
- Calculate data. Round, basic calculations, constants, conversions, reinterpretations.
- Check data. AND OR XOR NOT. Comparisons for equal, less, greater. Float number plausability.
- Manage data structures. Create them, decode structures.
- Collect data. Combine and separate texts. Combine and separate binary data.
- Lots of triggers: Time trigger, data change trigger, bit trigger, file create trigger, file modify trigger, directory change trigger.
- Subroutines.
- Handle UPC UA function calls (RPC).
- Generate OPC UA events.

Online diagnostics for shortening the configuration time

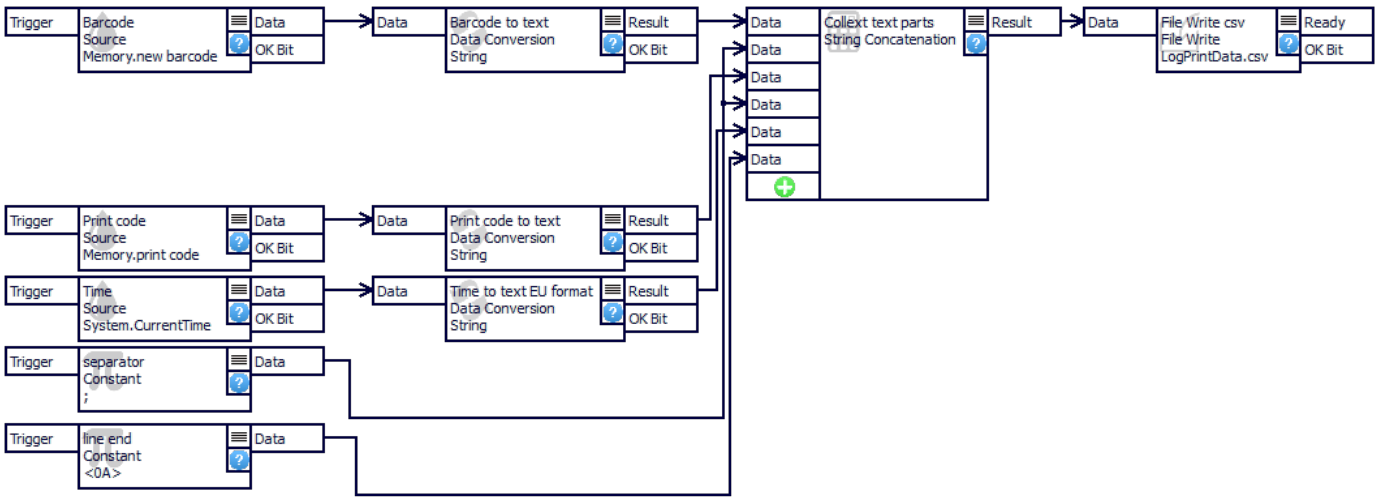
- Watch connections and their state.
- Watch variables. Status, value, who needs them.
- Watch logic tables and their behaviour.
- Watch and modify single variables (Status variable).
- Diagnostics logger for long term watching.

Configuration

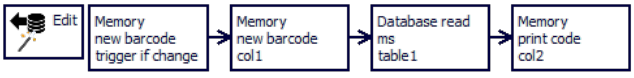
The configuration works in graphical style or with simple lists.

Lot of Wizards simplify this.

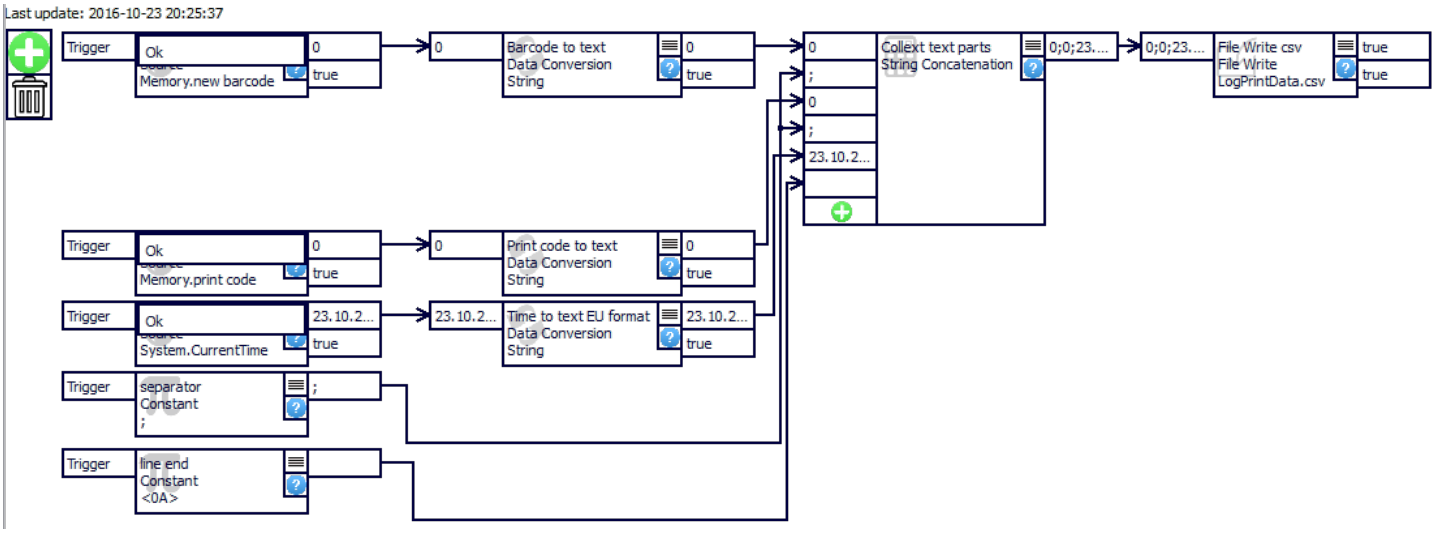
The example logs from barcode reader, adds a timestamp and writes both into a .csv file.



This example reads a barcode, gets the print data from a database and prints the result.



Diagnostics boost the startup in plants. Use Your smartphone or tablet and all of the status displays.



The diagnostics logger handles longer running information collecting.

OPC Interfaces

- **OPC Pipe** Open interface
- **OPC UA** (Unified Architecture)
- **OPC DA** (Classic OPC over DCOM, available under Windows only)

The maximum number of OPC clients is depending on used resources only. A PC from 2014 can handle multiple hundred connections.
All OPC interfaces are working locally in one PC or over network.
in case of Classic OPC Classic please do not use DCOM over networks, but it will be supported.
OPC UA supports the fast binary protocol. Security is supported in all variants. Multicast discovery is supported.
Data access data items are supported up to 200K each.

OPC UA functionality and limitations

The OPC UA implementation conforms to the specification 1.05.
The OPC UA Standard Model is supported, some extensions exist.
The maximum single request and answerlen is 16m
The OPC UA Alarms & Conditions module is supported. This includes filters, history.
An internal discovery server is active on standard, it supports multicast discovery also. It can be used as a global discovery server. Alternatively an external discovery server can be configured.
The certificate management GDS Push is supported.

The session timeout will be limited to one hour.
The server and client certificate will be renewed if the Tani self signed certificate is used. All other certificates remain unaffected on expiring. The certificate validity is checked all 12h. It will be renewed seven days before it expires. Running connections will not be affected, new connections will use the new certificate.
AddNodes is supported with the following restrictions:

- Reference type must be OpcUaId_Organizes
- NodeId can't be specified
- BrowseName can't contain a dot
- NodeClass must be Variable or Object
- NodeAttributes for Variable:
  - DisplayName: unspecified or equal to BrowseName
  - Description: unspecified or any text
  - Value: is ignored; new variables will always be initialized to 0 (if numeric) or "" (if string type)
  - DataType:
    - OpcUaType_Boolean
    - OpcUaType_SByte, OpcUaType_Byte
    - OpcUaType_Int16/32/64, OpcUaType_UInt16/32/64
    - OpcUaType_Float, OpcUaType_Double
    - OpcUaType_String
    - OpcUaType_LocalizedString. This will be handled outside OPC UA as a normal string. The LocaldId always is a null string
    - OpcUaType_DateTime
    - OpcUaType_ExtendedObject, OpcUaType_ExtendedObjectEx. Mostly this are structures. One of the structure types under Types -> DataTypes -> BaseDataType -> Structure -> UserStructures; these are the structures known to the PLC Engine core.
      - if the structure is given both here and via TypeDefinition, both settings must match
    - if unspecified, OpcUaType_Byte or the structure type of the TypeDefinition is used
  - ValueRank, ArrayDimensions: unspecified (= scalar), scalar or a one-dimensional array of any size
  - AccessLevel, UserAccessLevel: unspecified or (OpcUa_AccessLevels_CurrentRead | OpcUa_AccessLevels_CurrentWrite)
  - MinimumSamplingInterval: unspecified or 0
  - WriteMask, UserWriteMask: unspecified or OpcUa_NodeAttributesMask_Value
- NodeAttributes for Object:
  - DisplayName: unspecified or equal to BrowseName
  - Description: unspecified or any text
  - EventNotifier, WriteMask, UserWriteMask: unspecified or 0
- TypeDefinition for Variable:
  - OpcUaId_BaseDataVariableType
  - one of the structure types under Types -> VariableTypes -> BaseVariableType -> BaseDataVariableType -> UserStructures; these are the structures known to the PLC Engine core.
- TypeDefinition for Object:
  - OpcUaId_FolderType
- Each RPC as a calling queue of 10. If the requests are coming faster before handled they will return a memory error.

Machine models from the OPC Foundation or the VDMA directly can be loaded with its corresponding XML file.

The security certificate key minimum length are

- Basic128Rsa15: RSA Key Length 1024 .. 4096
- Basic256: RSA Key Length 1024 .. 4096
- Basic256Sha256: RSA Key Length 2048 .. 4096

Traffic between different OPC interfaces (tunneling) is supported. It will be used for the OPC DA tunnels.

MQTT Interfaces

- **MQTT Version** 3 and 5
- **MQTT Client** if a station need to be a device
- **MQTT Broker**, the server

MQTT comes from the Internet of Things world. It is simple and fast.
A device can simultanously send data to multiple devices.
You can use the client and broker on the same device at the same time.

Controller Interfaces

All controllers will be connected over network. Often this is Ethernet, WLAN or other networks. All serial Ethernet and MPI Ethernet gateways for industrial controllers usage are supported.

Configuration Interfaces

The configuration can be done with the shipped configuration software or over OPC with the System topic.
The connection for the configuration is encrypted with TLS 1.2. The encryption can be switched off for usage in countries where encryption is forbidden.

Network Redundancy for connections to controllers and devices

Connections to devices and controllers are supporting network redundancy.
Double and triple redundancy can be selected.
Two redundancy operation modi are possible.
In **dynamic redundancy** any of the connections is working as master. If it breaks another connection becomes the master connection.
In **static redundancy** the first connection is the master. If it breaks another connection becomes the master. If the first connection works again it will become the master connection again.

The connections of the redundancy should work on different network adapters.The adapters need different IP subnets for properly work.

Controller Types and Controller Protocols

- Siemens **S7 1200 and 1500** family. The **optimized data blocks** are supported, also structures, alarms, events. All symbols and comments are browsed online. The 2021 PLC firmware 2.9 works fine, the version 3.x from 2022 also.
- Siemens S7 over RFC1006 and Sinec H1. Supported are**S7 200, 300 and 400, Logo 8, ET200**. Siemens CP or the Ethernet interface onboard the CPU can be used. Mainstream MPI Gateways as Hilscher Netlink, Helmholz Netlink, IBH Softec Netlink, INAT Echolink, Process Informatik S7Lan or Softing Netlink are supported, too. S7 compatible systems as VIPA Speed7 can be used, too.
- Siemens S5 over RFC1006, PLC Header, RAW or Sinec H1. Supported are Siemens CPs, INAT CPs, Helmholz CPs, IBH Softec S5Net, Process Informatik S5Lan.
- Rockwell **Compact Logix**, **Control Logix** and **GuardLogix**, all firmware versions.
  Rockwell **Micro 8** series as the 800, 810, 820 and more.
  Rockwell **PLC5** and **SLC** series, all firmware versions.
  Routing paths are fully supported. This enables connecting to any PLC and accessing other PLC connected to an internal bus as DH+. So you will reach a PLC5 as example over another Rockwell as a CompactLogiX as router.
- GE PACSystems from General Electric. This controller family works with CIP from Rockwell.
- Mitsubishi **Melsec Q, QL** and **FX5** family using the SLMP protocol (3E protocol).
- **BACnet devices**. BACnet is common used in building automation. BBMD and COV, alarms, events, trend, calendar, shedule, rpc, lists and more are supported.
- **KNX**. KNX also is used in building automation. All known symbol imports are supported.
- Devices using the **Modbus TCP** protocol.
  - Modicon
  - Schneider
  - Wago
  - Beckhoff
  - Phoenix Contact
  - Omron
  - B&R
  - Fanuc
  - ABB
- **MQTT** version 3 and 5. For the OPC usage a simple item syntax odders the IoT world.
- IEC 60870-5-104. This often is used in long distance management of energy, oil and water.
- All systems and devices which can be accessed with**OPC UA** or **Classic OPC**. OPC server and client are available.
- Raw data. This are data which do not have any format by a standard.

Communicates via Ethernet.

BACnet

BACnet will be used over IP / UDP.
Maximum length of strings: 256 Byte
Status text elements are supported (state_text)
Supported charsets: UTF-8, UTF-16, Latin-1
Unions ("Choice") and structures ("Sequence") are existing for important values as trend, shedule, calendar, prioriry.
The trend data are offered as history data. All unimplemented instances will not be shown.
Enum values are represented as UINT32. Some special enum are handled as bool.
Values in "Octet-String" and "Bit-String" can be written in whole only.

BBMD (BACnet Broadcast Management Device) details

BBMD will be used during the connection establishing and the device search if the devices do not be all in the same collision domain. BACnet uses broadcast during ist connection establishing.
There are several procedures in BBMD:

- Search ussing broadcast.
- Search using the IP device address, receive the BACnet ID.
- Search using the BACnet id, receice the IP address.

Additionally BBMD can be used connecting older serial only installations to the IP network.

COV (Change Of Values) details

COV represents the event subsystem of BACnet. Events will be offered in browsing the variables, they will be subscribed. If the device will send the data the event will be generated.
Because BACnet is working with UDP the COV receive can not be guaranteed. Tani is offering an option: If no event will be received during the reconnection time from the configured connection it will be polled. If the value did not change no event is send for this polling.

BACnet - Writing values with priority-array

These object types have a priority-array in addition to their present-value property:

- analog-output
- analog-value
- binary-output
- binary-value
- multi-state-output
- multi-state-value
- access-door

The BACnet spec says:

- priority-array is read-only and contains 16 entries (that can be a valid value or NULL).
- present-value is read-write and contains 1 value (the non-NULL value with the lowest priority from priority-array, or the value from relinquish-default if no non-NULL value in priority-array exists).
- Writing to present-value uses an optional priority parameter to write to the correct entry in priority-array.

The Tani implementation works as follows:

- priority-array is read-write and contains 16 structure entries with 2 fields:
  * Value: the data value in this entry (or 0 if no valid value is present)

* ValueValid: a boolean value; 1 if Value is valid, 0 if not (NULL value).

- Writing to an element of priority-array implicitly uses a "write present-value with priority" operation to change the desired value.
- Writing to priority-array[i].Value always creates a non-NULL entry.
- Writing 0 to priority-array[i].ValueValid creates a NULL entry.
- Writing 1 to priority-array[i].ValueValid creates a non-NULL entry with value 0 (this is usually not very useful).
- Writing to priority-array[i] (as a structured data type) creates a NULL entry when ValueValid is 0. Else a non-NULL entry with the specified Value is created.
- present-value is read-write and contains the value obtained by BACnet protocol.
- Writing to present-value doesn't transfer the priority parameter. The BACnet device will implicitly write to priority entry 16 in this case.

This mechanism was chosen to allow choosing the write priority via OPC without changing the read syntax for present-value property. This also allows writing NULL values via OPC.

Implemented Properties

The following object properties are implemented:

Object Type	Property	BACnet Type	OPC Type	Remarks
all	all	BACnetObjectIdentifier	UInt32	
all	all	Bit String	Array of Boolean	
all	all	Boolean	Boolean	
all	all	Character String	String	
all	all	Double	Double	
all	all	Enumerated	UInt32	
all	all	Octet String	Array of UInt8	
all	all	Real	Float	
all	all	Signed	Int32	
all	all	Unsigned	UInt32	
all	Change of State Time (16)	BACnetDateTime	DateTime	
all	Event Time Stamps (130)	Sequence of BACnetTimeStamp	Array of Structure "Timestamp"	
all	Object Type (79)	BACnetObjectType	UInt32	
all	Time of Active Time Reset (114)	BACnetDateTime	DateTime	
all	Time of State Count Reset (115)	BACnetDateTime	DateTime	
Access Door (30)	Door Alarm State (226)	BACnetDoorAlarmState	UInt32	
Access Door (30)	Present Value (85)	BACnetDoorValue	UInt32	
Access Door (30)	Priority Array (87)	BACnetPriorityArray	Array(1..16) of Structure "UnsignedPriorityValue"	see section "Priority Array"
Access Door (30)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Analog Input (0)	Present Value (85)	Real	Float	
Analog Input (0)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Analog Output (1)	Present Value (85)	Real	Float	
Analog Output (1)	Priority Array (87)	BACnetPriorityArray	Array(1..16) of Structure "AnalogPriorityValue"	see section "Priority Array"
Analog Output (1)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Analog Value (2)	Present Value (85)	Real	Float	
Analog Value (2)	Priority Array (87)	BACnetPriorityArray	Array(1..16) of Structure "AnalogPriorityValue"	see section "Priority Array"
Analog Value (2)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Averaging (18)	Maximum Value Timestamp (149)	BACnetDateTime	DateTime	
Averaging (18)	Minimum Value Timestamp (150)	BACnetDateTime	DateTime	
Binary Input (3)	Present Value (85)	BACnetBinaryPV	UInt32	
Binary Input (3)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Binary Output (4)	Present Value (85)	BACnetBinaryPV	UInt32	
Binary Output (4)	Priority Array (87)	BACnetPriorityArray	Array(1..16) of Structure "UnsignedPriorityValue"	see section "Priority Array"
Binary Output (4)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Binary Value (5)	Present Value (85)	BACnetBinaryPV	UInt32	
Binary Value (5)	Priority Array (87)	BACnetPriorityArray	Array(1..16) of Structure "UnsignedPriorityValue"	see section "Priority Array"
Binary Value (5)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Calendar (6)	Datelist (23)	List of BACnetCalendarEntry	Array() of Structure "BACnet.CalendarEntry"	
Device (8)	Last Restore Time (87)	BACnetTimeStamp	Structure "Timestamp"	
Device (8)	Local Date (56)	Date	Structure "Date"	
Device (8)	Local Time (57)	Time	Structure "Time"	
Device (8)	Object List (76)	Sequence of BACnetObjectIdentifier	Array of UInt32	
Device (8)	Protocol Object Types Supported (96)	BACnetObjectTypesSupported	Array of Boolean	
Device (8)	Protocol Services Supported (97)	BACnetServicesSupported	Array of Boolean	
Device (8)	Segmentation Supported (107)	BACnetSegmentation	UInt32	
Device (8)	System Status (112)	BACnetDeviceStatus	UInt32	
Device (8)	Time of Device Restart (203)	BACnetTimeStamp	Structure "Timestamp"	
Event Enrollment (9)	Object Property Reference (78)	BACnetDeviceObjectPropertyReference	Structure "DeviceObjectPropertyReference"	
File (10)	Modification Date (149)	BACnetDateTime	DateTime	
Life Safety Point (21)	Present Value (85)	BACnetLifeSafetyState	UInt32	
Life Safety Point (21)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Life Safety Zone (22)	Present Value (85)	BACnetLifeSafetyState	UInt32	
Life Safety Zone (22)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Load Control (28)	Actual Shed Level (212)	BACnetShedLevel	Structure "ShedLevel"	
Load Control (28)	Duty Window (213)	Unsigned	UInt32	
Load Control (28)	Expected Shed Level (214)	BACnetShedLevel	Structure "ShedLevel"	
Load Control (28)	Present Value (85)	BACnetShedState	UInt32	
Load Control (28)	Requested Shed Level (218)	BACnetShedLevel	Structure "ShedLevel"	
Load Control (28)	Shed Duration (219)	Unsigned	UInt32	
Load Control (28)	Start Time (142)	BACnetDateTime	DateTime	
Loop (12)	Controlled Variable Reference (19)	BACnetDeviceObjectPropertyReference	Structure "DeviceObjectPropertyReference"	
Loop (12)	Manipulated Variable Reference (60)	BACnetDeviceObjectPropertyReference	Structure "DeviceObjectPropertyReference"	
Loop (12)	Setpoint Reference (109)	BACnetSetpointReference	Structure "SetpointReference"	
Loop (12)	Present Value (85)	Real	Float	
Loop (12)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	

Object Type	Property	BACnet Type	OPC Type	Remarks
Multi State Input (13)	Alarm Values (7)	Sequence of Unsigned	Array of UInt32	
Multi State Input (13)	Fault Values (39)	Sequence of Unsigned	Array of UInt32	
Multi State Input (13)	Present Value (85)	Unsigned	UInt32	
Multi State Input (13)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Multi State Output (14)	Present Value (85)	Unsigned	UInt32	
Multi State Output (14)	Priority Array (87)	BACnetPriorityArray	Array(1..16) of Structure "UnsignedPriorityValue"	see section "Priority Array"
Multi State Output (14)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Multi State Value (19)	Alarm Values (7)	Sequence of Unsigned	Array of UInt32	
Multi State Value (19)	Fault Values (39)	Sequence of Unsigned	Array of UInt32	
Multi State Value (19)	Present Value (85)	Unsigned	UInt32	
Multi State Value (19)	Priority Array (87)	BACnetPriorityArray	Array(1..16) of Structure "UnsignedPriorityValue"	see section "Priority Array"
Multi State Value (19)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Notification Class (15)	Recipient List (102)	List of BACnetDestination	Array() of Structure "BACnet.Destination"	
Schedule (17)	Effective Period (32)	BACnetDateRange	Structure "DateRange"	
Schedule (17)	Exception Schedule (38)	Sequence of BACnetSpecialEvent	Array of Structure "SpecialEvent"	
Schedule (17)	List of Object Property References (54)	Sequence of BACnetDeviceObjectPropertyReference	Array of Structure "DeviceObjectPropertyReference"	
Schedule (17)	Present Value (85)	ABSTRACT-SYNTAX.&Type	Structure "Any"	
Schedule (17)	Schedule Default (174)	ABSTRACT-SYNTAX.&Type	Structure "Any"	
Schedule (17)	Weekly Schedule (123)	Sequence Size(7) Of BACnetDailySchedule	7 sub-objects ("Monday", "Tuesday", ...) of Structure "TimeValue"	
Pulse Converter (24)	Present Value (85)	Real	Float	
Pulse Converter (24)	Status Flags (111)	BACnetStatusFlags	Array(0..3) of Boolean	
Structured View (29)	Subordinate List (211)	Sequence of BACnetDeviceObjectReference	Array of Structure "DeviceObjectReference"	
Trend Log (20)	Client COV Increment (127)	BACnetClientCov	Structure "ClientCov"	
Trend Log (20)	Log Buffer (131)	BACnetLogRecord	Structure "LogRecord"	Accessed via "HistoryRead" function, "Read" shows only one record.
Trend Log (20)	Log Device Object Property (132)	BACnetDeviceObjectPropertyReference	Structure "DeviceObjectPropertyReference"	
Trend Log (20)	Start Time (142)	BACnetDateTime	DateTime	
Trend Log (20)	Stop Time (143)	BACnetDateTime	DateTime	

KNX and EIB

KNX will be used over IP / TCP and IP / UDP.
The symbol import is using the standarized ESF files.

Databases

PLC Engine is a database client. It connects with user and password to the database.
The standard SQL statements INSERT INTO, UPDATE, SELECT, DELETE, FUNCTION and PROCEDURE will be used over the wizards. Other statements will be configured directly.
Supported are:

- My SQL (from version 1.9 not under Windows XP)
- PostGre Sql (not for Windows XP)
- Microsoft SQL
- Sybase SQL Server, Sybase ASE, SAP ASE (Adaptive Server Enterprise)
- ODBC
- Oracle can be used with ODBC

Multiple databases can be handled simultanously.
In one database multiple sub databases can be used.
During the configuration the databases will be browsed. This will require depending on the type of the database or the interface a username and a password, possibly more.
The database itself need to be configured that it can be accessed.
PLC-Engine will need an account on each database.

For using PLC Engine on a PC the locally installed databases can be used also.
On PLC Engine Device a My SQL database is installed. This database will be managed completely by PLC Engine.

TANI PostgreSQL Integration

The TANI PostgreSQL driver explicitly supports all data types listed in the following table. All types not listed here are implicitly converted to the "text" type and handed to the PLC Engine as a single String.

PostgreSQL type	Array type	OPC type	Description	Notes
boolean	boolean[]	Bit	Truth state (true or false)	
bit(n) bit varying(n)	bit(n)[] bit varying(n)[]	Bit	Bit vector of (maximum) length n	
smallint smallserial	smallint[] smallserial[] int2vector int2vector[]	Int16	16-bit signed integer	
integer serial	integer[] serial[]	Int32	32-bit signed integer	
bigint bigserial	bigint[] bigserial[]	Int64	64-bit signed integer	
oid	oid[] oidvector oidvector[]	UInt32	32-bit unsigned integer	
real	real[]	Float	32-bit floating-point value	
double precision	double precision[]	Double	64-bit floating-point value	
numeric decimal	numeric[] decimal[]	Double	exact fixed-point value	the values are converted to floating-point values, this may lead to loss of precision

PostgreSQL type	Array type	OPC type	Description	Notes
abstime timestamp timestamp with time zone	abstime[] timestamp[] timestamp with time zone[]	Timestamp	Timestamp values (date + time)	a time zone (if present) is ignored
date	date[]	Timestamp	Date value	The time part of the OPC Timestamp is set to 00:00:00
reltime time time with time zone	reltime[] time[] time with time zone[]	Timestamp	Time values	a time zone (if present) is ignored The date part of the OPC timestamp is set to 1970-01-01
char(n) char varying(n) text	char(n)[] char varying(n)[] text[]	String	Character strings (of (maximum) length n)	

General note: all multi-dimensional arrays (including arrays of vectors) are flattened to a single-dimensional array. The identity of the multiple dimensions is lost. This is a limitation of the internal data handling of the TANI software. Example: {{1,2},{3,4}} is returned in the logic table as {1,2,3,4}. Writing of multi-dimensional arrays (using INSERT or UPDATE) is not supported.

Logger for diagnostics

The OPC Server contains a logger for diagnostics purposes during plant startup. The logger can be configured. The system load can be big if all controller data in big plants are logged.

Limits

Maximum number of configurable client connections: 4000.
Maximum length of a single item: 4GB.
Maximum number of elements each connection: 1 million.
Maximum number of elements (Items): 16 million.
Maximum OPC groups each connection: 100.
Maximum number of passive connection for each port is 999.
The OPC synchronous functions returning a bad quality immediately if the PLC connection is not established.
Changes in controller configuration will be checked all 10 seconds if the PLC does not offer a mechanism for this check during write.
Fields can be up to 64K in length each.
Multi dimensional arrays can have up to six dimensions.

Fields can be up to 64K in length each.The maximum amount of configurable logic tables is 60000.
The maximum length of one logic table is 4GB.
Maximum number of entries on a logic element 60000.
File operations can handle up to 64K in each request. Adding data to a file is limited only by the space on the disk.
Multi dimensional arrays can have up to seven dimensions.

Virtual Connections

Virtual connections provide a start point mostly used as destination for redirects.
A virtual connection has connection rights. The redirect destination in this connection will offer the rignts of the virtual connection. As in the logic tables and the status variable lists the source rights are ignored.
Symbols for the virtual connection can be defined in the symbol editor.
One PLC element can be used once in one virtual connection.

Depending on the license the limits can be less.

Speed

The throughput will be mainly limited by the controller speed or the reaction time of OPC applications.
Read requests to the controller will be optimized as much the controller is supporting this. For that elements will be collected to blocks reading more than requested, but not for inputs and outputs. These optimizing can be affected by configuration separately for each connection. Optimizing can be switched off, too.
Write requests to the controller are collected or handled in that order the application did called the system.
On OPC all optimizing the individual OPC uses is supported.
The normal time in cyclic controller requests is 50ms.It can be faster if the controller polling interval is set to zero.
Only data are sent to OPC which did change in the controller between two read requests.

The throughput of one logic table normally will be handled below 10ms. Lot amounts of logic tables containing much of mathematics functions may rise the 10ms. This is true especially on embedded devices without a mathematical co processor.
No handling can be faster than the controller and device data acquisition. The same is true for database access.

Functions

Logic tables

Logic tables are constructed for linear logic. Loops are not possible.

Sequence chains

A sequence chain supports the maximum of 65535 steps each.

Error Handling in Logic Tables

All logic elements which functions can fail are supporting an OK bit. The user need handle this preventing unexpected run-time behavior.

Variables and Structures

Structures can not contain loops.
A structure or variable can be up to 4GB in its size.

Status Diagnostics Lists

The status diagnostics lists are supporting arrays up to 100 values each. If an array is longer than 100 values the first 100 elements are shown only. Writing this array is not possible.

Field and text optimizings

The from version 1.8 existing field optimizings will prevent reading the long fields too often, the index is requested on standard only.This optimizing bases of the fact that the index does net changed too frequently.

Usage of memory

- Program code: A minimum of 6MB is used. The exactly memory usage is depending of the internal behavior of the operating systems. So dynamic libraries are loaded once for all running instances using them. Example: If the standard library is not loaded already it will use additional 4MB of memory.
- User data: The minimum data usage is 2MB internally. Additional the controller data are held in memory for comparing new data. Each item uses the length of data and additional 64 bytes. Each configured connection occupies 4KB.

- Fields can be up to 64K in length each.The maximum amount of configurable logic tables is 60000.
- The maximum length of one logic table is 4GB.
- Maximum number of entries on a logic element 60000.
- Depending on the license the limits can be less.

Usage of computation time

The consumed computation time is depending on the load with communication. Most the time it will be waited for controller data or OPC application reaction.
All software is working with events. This maximizes the throughput and minimizes the usage of computation time.
Multiple CPU are supported. Up to ten CPU will be used, the main work will be handled by three CPU.

Installation

The installation does depending on the product install multiple parts separately. On uninstall not all products are deleted automatically. But all installed products can be deleted over the menu or the software part in the system control manager.
The user settings will be preserved and not deleted during uninstall.

Automatic structure import

Type Auto-Import works for all client protocols that are able to use structures/enumerations and have online browsing functions. This includes:

- OPC UA
- OpcPipe
- Siemens S7-1500
- Rockwell CompactLogix/ControlLogix/MicroLogix
- IEC104
- KNX

These protocols have a fixed list of structures and don't need Auto-Import:

- BACnet

These protocols have online browsing, but don't use structures/enumerations:

- OPC DA
- MQTT

All other protocols don't have online browsing.

Type Auto-Import is implemented in two steps:

1. A structure or enumeration type which has not been imported is assigned a Node ID when:
    - the Item is being monitored (by calling CreateMonitoredItems):
    - the Item is being read/written (by calling Read/Write):
    - the Item is being registered (by calling RegisterNodes):
    - the DataType attribute of an Item with this type is accessed:
2. A structure or enumeration type which has not been imported is actually imported when:
    - the Item is being monitored (by calling CreateMonitoredItems):
    - the Item is being read/written (by calling Read/Write):
    - the Item is being registered (by calling RegisterNodes):
    - the DataTypeDefinition attribute of the DataType node is read (after it has been created by step 1):
    - the EnumValues property node is read (for Enumerations, after it has been created by step 1):

Limitations:

Before Auto-Import Step 1, any types that have not been imported yet:

- are not available anywhere

Before Auto-Import Step 2, any types that have not been imported yet:

- have a DataType Node ID assigned
- are not browseable in Types/DataTypes/BaseDataType/Structure/UserStructures or Types/DataTypes/BaseDataType/Enumeration
- are not present in the XML data in Types/DataTypes/OPC Binary/UserStructures
- are not browseable in Types/VariableTypes/BaseVariableType/BaseDataVariableType/UserStructures
- don't have the type comment available
- don't have Encoding Node IDs available

After Auto-Import Step 2:

- the newly imported types behaves exactly as any manually imported type
- if the type later changes in the source systen, the import cache will NOT be updated

A client wishing to use a variable with a structure/enumeration type that has not been imported should

- either read the DataType attribute of the variable, then read the DataTypeDefinition attribute/EnumValues property of the type node,
- or monitor/read the Value attribute of the variable before checking the data types

to trigger the type import. Only after completing one of these the structure type is available in the server.

Operating systems

- Windows 10, 11 (all versions). Older versions as Vista and Windows 7/8 also. 64 and on older Windows 32 bit.
- Windows Server 2012, 2016, 2019, 2022.
- Linux on the Raspberry and Odroid computers (64 and 32 bit).
- Linux on many Phytec devices as the Regor and Tauri S (32 bit) and Tauri L (64 bit).
- Linux on a PC with Debian, Ubuntu, Suse, Arch, Centos, Redhat and other Distributions.
- Linux 64 Bit as Docker or Kubernetes or Containerd Container.

- OPC DA will require Microsoft Windows. All from Microsoft supported operating systems for Intel and all user languages will be supported. The latest service pack must be present.
- OPC DA will require minimum of 2 CPU cores for a good performance.
- Under Windows the OPC server are working as service, Linux runs them as daemon.
- The Raspberry version supports all Linux distributions offered for this platform.
- All other will run under lot of operation systems also, mostly Linux based.
- Under Linux the OPC Server needs a POSIX compatible System. The Standard Library needs V2.2 as minimum. The configuration software is bases on KDE 5 and is needing the kdelibs. Please use actual distributions like Debian, Ubuntu, Suse, Redhat or similar. Tested is: Windows Intel 32 and 64 bit, Linux Intel 32 and 64 Bit, Linux MIPS 32 CPU, Linux ARM 32 and 64 Bit CPU.
- Running in virtual machines is supported. Docker or Kubernetes containers are supported, too.
- Windows 7 needs as minimum service pack 1 for using the drivers. All Windows service packs for SHA512 must be installed.
- The web based configuration will run on actual browsers as Firefox, Chrome, Safari on any platform. It should be newer than 5 years. The browser need support SVG graphics, CSS and JavaScript for the responsive design. Mobile browsers can be used also.

- All configurations are compatible to all OPC servers, also over mixed operating systems.