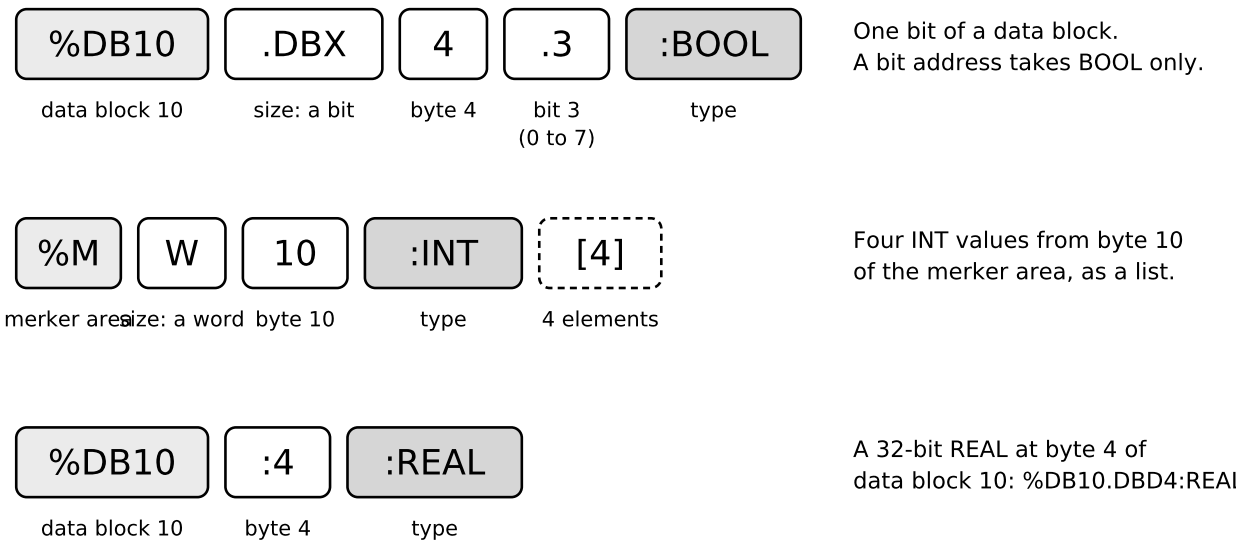
Amtiri

REFERENCE MANUAL

Siemens S7

Generated on November 25, 2026



The `S7-Client` module reads and writes the memory of a Siemens PLC directly, over the S7comm protocol on TCP port 102. It is for the controllers that have no OPC UA server of their own (S7-300, S7-400, LOGO! 8), and for an S7-1200 or S7-1500 whose OPC UA server is not switched on.

Module	What it does	Typical use
`S7-Client` (Siemens S7 Client)	Reads PLC memory into variables on a schedule, and writes variables to PLC memory when they change	A Siemens PLC with no OPC UA server

Enable, **Name** and the **Status and messages** tab work the same way in every module. They are described in Settings every module shares.

> **Warning**
>
> **On an S7-1200 or S7-1500, two settings in the CPU have to change, and both weaken it.** PUT/GET communication must be enabled in the CPU's protection settings, and optimized block access must be turned off on every data block this module reads. Neither change is specific to Data Orchester: both open the PLC to every client on the network. The editor says so beside the connection settings. Where the controller offers OPC UA, use the OPC UA client instead: it needs no such change, and it authenticates and encrypts.

Siemens S7 Client

The module has the two lists every client module has: **collectors** read an address into a variable on a schedule, and **forwarders** write a variable to an address whenever the variable changes. An S7 address states its own type, so there is no separate type column to keep in step with it.

Settings

Key	Setting	Type	Default	Meaning
`host`	Host	string	(empty)	The PLC's address. Required: without it the module does not open
`controllerType`	Controller	string	(empty)	`S7_300`, `S7_400`, `S7_1200`, `S7_1500` or `LOGO`, shown in the editor as S7-300 to LOGO! 8. Passed to the driver as the controller type; empty leaves it to the driver. The driver also accepts `S7_200` and `ANY`, which the editor does not offer; any other value makes every connection fail, reported as `PLC {plc} unreachable`. Choosing one in the editor fills in **Rack** and **Slot**
`rack`	Rack	int	0	The rack the CPU sits in
`slot`	Slot	int	1	The slot the CPU sits in
`requestTimeoutMs`	Timeout (ms)	int, ms	5000	How long one request, a read or a write, may take before it fails. Opening the connection may take twice as long. Before engine 6.12.0 this setting had no effect, and the driver waited 10 s
`collectors`	Collector	list of rows	empty	The addresses read into variables
`forewarders`	Forwarder	list of rows	empty	The variables written to addresses. The key is spelled `forewarders`

What choosing a controller fills in:

Controller	Stored as	Rack	Slot
S7-300	`S7_300`	0	2
S7-400	`S7_400`	0	2
S7-1200	`S7_1200`	0	1
S7-1500	`S7_1500`	0	1
LOGO! 8	`LOGO`	0	0

These are where each family's CPU usually sits. A CPU in another rack or slot needs its own numbers, typed after the controller is chosen.

S7-200 Smart is not offered: the driver has no controller type for it. Editors before Data Orchester 1.25.0 offered it as `S7_200_SMART`, and a module set to it could never connect.

Collectors and forwarders

A collector row uses every key below. A forwarder row uses `field` and `address`.

Key	Column	Type	Default	Meaning
`field`	Variable	variable code	(none)	Collector: the variable the value is written to. Forwarder: the variable whose changes are written to the PLC
`address`	Address	string	(empty)	The memory address and its type, written as the next section says
`cron`	CRON	Quartz cron	`0 * * * * ?`	When the collector is read. Required for a collector

The **Check** column beside each address is not stored. It shows what is wrong with the address, or, when nothing is, the variable type the address yields.

Addresses

An address is `%`, a memory area, a byte offset, an optional bit, a colon and a type, and optionally an element count in brackets:

```
%DB10:4:REAL        data block 10, byte 4, a 32-bit REAL
%DB10.DBD4:REAL     the same address, in the notation TIA Portal shows
%DB10.DBX4.3:BOOL   data block 10, byte 4, bit 3
%M0.1:BOOL          merker byte 0, bit 1
%I0.0:BOOL          input byte 0, bit 0
%Q0.1:BOOL          output byte 0, bit 1
%MW10:INT           the merker word at byte 10
%IW64:INT           the input word at byte 64
%DB1:0:INT[10]      ten INT values from byte 0 of data block 1
```

Part	Written as
Data block	`DB` and its number, then `:` or `.`, optionally followed by the size `DBX` (bit), `DBB` (byte), `DBW` (word) or `DBD` (double word)
Other areas	`I` inputs, `Q` outputs or `M` merkers, optionally followed by the size `X`, `B`, `W` or `D`
Byte offset	A whole number
Bit	`.0` to `.7`, after the byte offset
Type	`:` and a type from the table below
Count	`[n]`, with `n` at least 1: `n` consecutive values, read as a list

- **A bit address takes `BOOL`, and `BOOL` takes a bit.** An address is a bit address when it has `.0` to `.7`, the size `DBX` or the size `X`. The two mistakes are refused, because a PLC would otherwise answer them without complaint: `%DB10.DBX4:REAL` asks for one bit and calls it a REAL, and `%M0:BOOL` names no bit.
- **There is no symbol browsing.** S7comm has no symbol table, so addresses are typed, or pasted from TIA Portal.

Type	Variable type
`BOOL`	boolean
`BYTE`, `WORD`, `DWORD`, `LWORD`, `SINT`, `USINT`, `INT`, `UINT`, `DINT`, `UDINT`, `LINT`, `ULINT`	int
`REAL`, `LREAL`	float
`CHAR`, `WCHAR`, `STRING`, `WSTRING`	string
`TIME`, `LTIME`, `S5TIME`	int
`DATE`, `TIME_OF_DAY`, `TOD`, `DATE_AND_TIME`, `DT`, `LDT`	string

An address with a count yields a list of its type. The variable type is what the **Check** column shows, and what the pencil beside **Variable** creates a variable as.

What the **Check** column, and the error of a module that will not open, say about an address:

Text	Meaning
`No address`	The address is empty
`'{address}' is not an address this module understands`	It does not follow the grammar above
`'{type}' is not a type this module understands`	The type is not in the table
`'{address}' addresses a single bit, which can only be BOOL, not {type}`	A bit address with another type
`'{address}' is BOOL but names no bit; write it as {start}.0:BOOL`	`BOOL` without a bit. `{start}` is the address up to its first colon: `%M0:BOOL` gets `write it as %M0.0:BOOL`
`'{address}' asks for an array of no elements`	A count of 0

Test read

Type an address in the box beside **Test read** and press the button. The engine reads that address once, with the **Host**, **Controller**, **Rack** and **Slot** on the screen and a timeout of 5000 ms, and shows the value, or why it could not read it: the address's problem, `No host`, `The PLC answered {code}`, or the driver's reason. Nothing is saved, and the address does not have to be in a table. It is the quickest way to check the rack, the slot and the PUT/GET setting before the module is enabled.

Behaviour

- **Opening.** The host, every collector's schedule and every address, collectors and forwarders alike, are checked before anything starts. A problem stops the module opening, naming the variable of the row.
- **Collectors sharing a schedule are one request.** Collectors whose cron expressions are written exactly the same are read together, and the driver splits the request into as many messages as the CPU needs. Forty addresses on one schedule cost one round trip, not forty.
- **Every collector is read once when the module opens**, then on its schedule. The first read opens the connection: a PLC that is switched off does not stop the module opening, it shows as a lost connection.
- **There are no subscriptions.** Every collector is polled.
- **A refused address is not a lost link.** When the PLC refuses one address, its variable is left as it was, the rest of the request is written, and the module warns.
- **Reconnection is the module's own.** A request that fails, or takes longer than **Timeout (ms)**, drops the connection, and the next read or write opens a new one. The schedules pace the retries.

- **Forwarders** write their variable's current value once when the module opens, then every change, one request per write. A value is converted to the width of the address's type before it is sent. Keep it within the type's range: the module does not check it.

Messages

A problem is written into the module's message variables, and the next successful read clears it. A refused address is the exception. A read in which the PLC refuses one address still counts as a successful read, and every other address of it is written, but the warning about the refused address stays while the PLC keeps refusing it, and ends when that address is read again. With several refused, it names the latest one. Before engine 6.18.8 the end of the same read cleared it at once, so it showed only for a moment. "then `ERROR`" follows the thresholds in Settings every module shares: a lost connection becomes an error after `erroAfterSeconds`. All texts are in English; the full list is in Module messages. `{server}` and `{plc}` are the **Host**, `{mapping}` is the variable's code, and `{address}` and `{node}` are the address as its row gives it. `{code}` is the driver's answer for one address, such as `NOT_FOUND`, `ACCESS_DENIED` or `INVALID_ADDRESS`.

When	Level	Text
Opening: no host	`ERROR`	`No host`
Opening: a collector has no schedule	`ERROR`	`No schedule given`
Opening: a schedule does not parse	`ERROR`	`Invalid schedule '{cron}': {reason}`
Opening: an address the module cannot use	`ERROR`	`{mapping}: {reason}`, the problem as the **Check** column says it
A connection opened	`INFO`	`Connected to {server}`
A read failed, and the connection was dropped	`WARNING`, then `ERROR`	`Disconnected from {server}`, then `PLC {plc} unreachable`
The PLC refused one address of a read	`WARNING`	`{address} returned {code}`
The PLC refused a write	`WARNING`	`{address} refused the write: {code}`
A write failed, and the connection was dropped	`WARNING`	`Writing {node} failed: {reason}`

Example

An oxygen reading and a blower's run state every five seconds, and an oxygen setpoint written back when an operator moves it, on an S7-1500:

```
Host         10.20.3.40
Controller   S7-1500          Rack 0   Slot 1
Timeout      5000

Collector    AIT_201_O2     %DB10:4:REAL      */5 * * * * ?
Collector    B_201_RUN      %Q0.1:BOOL        */5 * * * * ?
Forwarder    AIT_201_SP     %DB10:8:REAL
```

The two collectors share a schedule, so they are one request every five seconds. Writing `AIT_201_SP` from a dashboard or a formula sends it to the PLC as a REAL.

Next steps

- Module catalogue

- Settings every module shares
- Module messages
- OPC UA, for an S7-1200 or S7-1500 with its OPC UA server switched on
- Connectivity

This page describes Data Orchester engine 6.18.8.