

SIMATIC NET

PC software Industrial Communication with PG/PC Volume 2 - Interfaces

Programming Manual

Preface

1

OPC process variables for
SIMATIC NET

2

OPC Alarms & Events server
for SIMATIC NET

3

Using the OPC server

4

SIMATIC Computing

5

Sample programs

6

Reference Automation
Interface

7

Related literature

8

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

DANGER

indicates that death or severe personal injury **will** result if proper precautions are not taken.

WARNING

indicates that death or severe personal injury **may** result if proper precautions are not taken.

CAUTION

indicates that minor personal injury can result if proper precautions are not taken.

NOTICE

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions.

Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

WARNING

Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Table of contents

1	Preface	17
1.1	How do I work with the documentation?	18
1.2	Regulations	19
1.2.1	Which legal regulations do you need observe?	19
1.2.2	Which safety regulations do you need to know?	20
2	OPC process variables for SIMATIC NET	21
2.1	Which communication functions exist?	21
2.2	What are process variables?	21
2.3	How are the ItemIDs of the process variables formed?	22
2.4	PROFIBUS DP	24
2.4.1	Class 1 DP master	25
2.4.1.1	Powerful SIMATIC NET in-process server for the PROFIBUS DP protocol	25
2.4.1.2	Powerful SIMATIC NET OPC server for the PROFIBUS DP protocol	26
2.4.1.3	DPC1 and DPC2 services	28
2.4.1.4	Process variables for services of the class 1 master	29
2.4.1.5	Syntax of process variables for a class 1 master	30
2.4.1.6	Protocol ID	31
2.4.1.7	Configured CP name	31
2.4.1.8	Examples of process variables for a class 1 master	32
2.4.1.9	DPC1 services	33
2.4.1.10	Syntax of the process variables for DPC1 services	33
2.4.1.11	Examples of process variables for DPC1 services	35
2.4.1.12	Fast Logic for CP 5613/CP 5614 and CP 5623/CP 5624	35
2.4.1.13	Syntax of the control variables for fast logic	36
2.4.1.14	DPSpecific information variables	38
2.4.1.15	Syntax of the DP-specific information variables	38
2.4.1.16	Examples of DP-specific information variables	42
2.4.1.17	Syntax of the systemspecific information variables	43
2.4.2	Class 2 DP master	43
2.4.2.1	Protocol ID	44
2.4.2.2	Configured CP name	44
2.4.2.3	Syntax of the process variables for master diagnostics	44
2.4.2.4	Syntax of the process variables for slave diagnostics	51
2.4.2.5	Syntax of the process variables for I/O data	57
2.4.2.6	Syntax of the process variables for data records	59
2.4.2.7	Examples of process variables for data records	62
2.4.2.8	Syntax of the DP2-specific information variables	63
2.4.2.9	Syntax of the systemspecific information variables	64
2.4.3	DP slave	64
2.4.3.1	Variable services for access to local slave data	65
2.4.3.2	Syntax of the process variables for the DP slave	65
2.4.3.3	Examples of process variables for the DP slave	66
2.4.3.4	DP slavespecific information variables	67

2.4.3.5	Syntax of the DP slave-specific information variables	67
2.5	PROFIBUS DP with OPC UA	68
2.5.1	SIMATIC NET OPC UA server for the DP protocol	69
2.5.2	Support of DP services with OPC UA	71
2.5.3	How is the DP OPC UA server addressed?	73
2.5.4	Which namespaces does the DP OPC UA server provide?	75
2.5.5	The Nodeld	76
2.5.6	Board objects for DP modules	79
2.5.6.1	Overview of board objects for DP modules	79
2.5.6.2	Board names	80
2.5.6.3	Type definition of the DP master board object	81
2.5.6.4	Type definition of the DP slave board object	82
2.5.7	DP master class 1	83
2.5.7.1	Process variables for services of the DP master class 1	83
2.5.7.2	Syntax of the process variables for the master class 1	83
2.5.7.3	Examples of process variables for the DP master class 1	85
2.5.7.4	DPC1 services	86
2.5.7.5	Syntax of the process variables for DPC1 services	86
2.5.7.6	Examples of process variables for DPC1 services	89
2.5.7.7	Fast Logic for CP 5613/CP 5614/CP 5623/CP 5624 (master only)	90
2.5.7.8	Syntax of the control variables for fast logic	90
2.5.7.9	Control frames Sync and Freeze	92
2.5.7.10	Syntax of the control variables for Sync and Freeze	93
2.5.7.11	DP-specific information variables	95
2.5.8	DP slave	106
2.5.8.1	Variable services for accessing local DP slave data	106
2.5.8.2	Syntax of the process variables for the DP slave	106
2.5.8.3	Examples of the process variables for the DP slave	108
2.5.8.4	Status of the DP slave	109
2.5.8.5	Mode of the DP slave	109
2.5.8.6	DP slave-specific information variables	110
2.5.8.7	Syntax of the DP slave-specific information variables	110
2.5.9	DP OPC UA template data variables	113
2.5.9.1	DP OPC UA template data variables	113
2.6	S7 communication	116
2.6.1	Powerful SIMATIC NET OPC server for the S7 protocol	117
2.6.2	Protocol ID	118
2.6.3	Connection names	119
2.6.4	Variable services	120
2.6.4.1	Syntax of the process variables for S7 variable services	120
2.6.4.2	Examples of process variables for S7 variable services	125
2.6.4.3	Examples of items with an optimum structure	125
2.6.5	Bufferorientated services	126
2.6.5.1	Syntax of the process variables for buffer-oriented services	127
2.6.5.2	Examples of process variables for bufferorientated services	130
2.6.6	S7specific information variables	131
2.6.6.1	Syntax of the S7-specific information variables	131
2.6.6.2	S7-specific diagnostics variables	134
2.6.6.3	Syntax of the systemspecific information variables	138
2.6.7	Block management services	139
2.6.7.1	Syntax of the control variables for block management services	140

2.6.7.2	Examples of using the block management services.....	143
2.6.8	Passwords	144
2.6.8.1	Syntax of the control variables for passwords	145
2.6.8.2	Examples of using passwords	146
2.6.9	Server services	146
2.6.9.1	Example of using server services	147
2.6.10	S7 template data variables	149
2.6.10.1	Template data variables in the namespace.....	151
2.6.10.2	Syntax of the template data variables.....	151
2.6.11	Unconfigured S7 connection.....	153
2.6.12	COML S7 connection.....	158
2.7	S7 communication with OPC UA	159
2.7.1	Properties of S7 communication with OPC UA	159
2.7.2	SIMATIC NET OPC UA server for the S7 protocol.....	160
2.7.3	How is the S7 OPC UA server addressed?	162
2.7.4	Which namespaces does the S7 OPC UA server provide?	164
2.7.5	The Nodeld	165
2.7.6	Connection objects	167
2.7.6.1	Connection names	168
2.7.7	Structure and functions of the productive S7 connection object	169
2.7.7.1	Type definition of the S7 connection object.....	170
2.7.7.2	S7 connection information objects.....	171
2.7.7.3	Examples of S7-specific information variables and return values	172
2.7.7.4	Methods for the S7 block services	173
2.7.8	Variable services.....	176
2.7.8.1	Variable services.....	176
2.7.8.2	Syntax of the variable services	176
2.7.8.3	Examples of process variables for S7 OPC UA variable services	181
2.7.9	Buffer-oriented services	183
2.7.9.1	Syntax of the buffer-oriented services	183
2.7.9.2	Examples of process variables for buffer-oriented services	185
2.7.10	Block information objects of an S7 connection	187
2.7.10.1	Length information	187
2.7.10.2	Template objects.....	188
2.7.10.3	Diagnostic and configuration information.....	188
2.7.11	S7 OPC UA template data variables	194
2.7.12	Events, conditions and alarms	196
2.7.12.1	What alarms are there?	196
2.7.12.2	What are events?.....	196
2.7.12.3	Which events of the S7 UA Alarming server are there?	196
2.7.12.4	Event type hierarchy of S7 UA Alarming servers	197
2.7.13	Standard event types	199
2.7.13.1	Standard event type with the display name "BaseEventType".....	199
2.7.13.2	Standard event type with the display name "ConditionType"	205
2.7.13.3	Standard event type with the display name "AcknowledgableConditionType"	208
2.7.13.4	Standard event type with the display name "AlarmConditionType"	208
2.7.13.5	Standard event type with the display name "ExclusiveLimitAlarmType"	210
2.7.13.6	Standard event type with the display name "ExclusiveLevelAlarmType"	210
2.7.13.7	Standard event type with the display name "ExclusiveDeviationAlarmType"	211
2.7.13.8	Standard event type with the display name "OffNormalAlarmType"	211
2.7.14	S7 event types	212
2.7.14.1	S7 event type with the display name "S7StatepathAlarmType"	212

2.7.14.2	S7 event type with the display name "S7ExclusiveLevelAlarmType"	213
2.7.14.3	S7 event type with the display name "S7ExclusiveDeviationAlarmType"	215
2.7.14.4	S7 event type with the display name "S7OffNormalAlarmType"	216
2.7.14.5	S7 event type with the display name "S7DiagnosisEventType"	218
2.7.15	Area tree and source space	219
2.7.16	Receiving events	220
2.7.17	Methods of UA alarms	221
2.8	Optimized S7 communication with OPC UA	222
2.8.1	Properties of optimized S7 communication with OPC UA	222
2.8.2	SIMATIC NET OPC UA server for the optimized S7 protocol	222
2.8.3	How is the S7OPT OPC UA server addressed?	224
2.8.4	Which namespaces does the S7OPT OPC UA server provide?	226
2.8.5	Die Nodeld	228
2.8.6	Connection objects	229
2.8.7	Connection names	233
2.8.8	Structure and functions of the productive S7OPT connection object	233
2.8.9	Type definition of the S7OPT connection object	234
2.8.10	S7OPT connection information objects	234
2.8.11	Examples of S7OPT-specific information variables and return values	235
2.8.12	Variable services	236
2.8.12.1	Standard access	236
2.8.12.2	Optimized access	244
2.8.13	Block information objects of an S7 connection	249
2.8.13.1	Length information	249
2.8.13.2	Template objects	250
2.8.13.3	Diagnostics and configuration information	250
2.8.14	S7OPT OPC UA template data variables	252
2.8.15	OPC UA events, conditions and alarms	253
2.8.15.1	What OPC UA alarms are there?	253
2.8.15.2	What are OPC UA events?	254
2.8.15.3	Which S7OPT event types are supported by the S7OPT OPC UA server?	254
2.8.15.4	Event type hierarchy of S7OPT OPC UA servers	255
2.8.16	Standard event types and the use of their properties	257
2.8.16.1	Standard event type with the display name "BaseEventType"	257
2.8.16.2	Forming the SourceName, Message and Severity	259
2.8.16.3	Standard event type with the display name "ConditionType"	262
2.8.16.4	Formation of ConditionName	263
2.8.16.5	Standard event type with the display name "AcknowledgeableConditionType"	264
2.8.16.6	Standard event type with the display name "AlarmConditionType"	264
2.8.16.7	Standard event type with the display name "OffNormalAlarmType"	265
2.8.17	S7OPT event types	266
2.8.17.1	S7OPT event type with the display name "S7OPTStatepathAlarmType"	266
2.8.17.2	S7OPT event type with the display name "S7OPTConsistencyAlarmType"	266
2.8.17.3	S7OPT event type with the display name "S7OPTOffNormalAlarmType"	267
2.8.17.4	S7OPT event type with the display name "S7OPTInfoReportEventType"	270
2.8.18	Area tree and source space	272
2.8.19	Receiving events	274
2.8.20	Methods of UA alarms	276
2.9	Open communication services (SEND/ RECEIVE)	276
2.9.1	Open communication services (SEND/ RECEIVE) via Industrial Ethernet	276
2.9.1.1	Powerful SIMATIC NET OPC server for the SR protocol	277

2.9.1.2	Protocol ID	279
2.9.1.3	Connection names	279
2.9.1.4	Variable services.....	279
2.9.1.5	Bufferoriented services	282
2.9.1.6	SEND/RECEIVEspecific information variables.....	286
2.9.1.7	Syntax of the systemspecific information variables	287
2.9.2	Open communication services (SEND/ RECEIVE) over PROFIBUS.....	288
2.9.2.1	Protocol ID	288
2.9.2.2	Connection names	288
2.9.2.3	Bufferoriented services	289
2.9.2.4	FDLspecific information variables.....	294
2.9.2.5	Syntax of the systemspecific information variables	296
2.10	Open communication services (SEND/ RECEIVE) with OPC UA via Industrial Ethernet....	296
2.10.1	Properties of SR communication with OPC UA.....	296
2.10.2	SIMATIC NET OPC UA server for the SR protocol	297
2.10.3	How is the SR OPC UA server addressed?	299
2.10.4	Protocol ID	301
2.10.5	Which namespaces does the SR OPC UA server provide?.....	301
2.10.6	Connection names	302
2.10.7	The Nodeld	303
2.10.8	Data variables for S5 data blocks and areas (S5-compatible communication)	304
2.10.9	Buffer-oriented services	308
2.10.10	SR-specific information variables	311
2.10.11	SR OPC UA template data variables.....	312
2.11	SNMP Communication over Industrial Ethernet	316
2.11.1	Protocol ID	316
2.11.2	Data types of the SNMP protocol	316
2.11.3	Process variables for SNMP variable services	317
2.11.4	SNMPspecific information variables	318
2.11.5	SNMPspecific traps.....	321
2.12	PROFINET IO communication over Industrial Ethernet	321
2.12.1	Powerful SIMATIC NET OPC server for the PROFINET IO protocol.....	322
2.12.2	How can IO data be addressed?	323
2.12.3	How can the configured PROFINET IO namespace be browsed?	327
2.12.4	Which OPC variables are available for PROFINET IO?	328
2.12.5	Protocol ID	328
2.12.6	Controller name	328
2.12.7	PROFINET IO process variables	328
2.12.8	PROFINET IO data status	334
2.12.9	PROFINET IO data records	337
2.12.10	Syntax of the systemspecific information variables	340
2.12.11	PROFINET IOspecific information variables	341
2.13	PROFINET IO communication with OPC UA over Industrial Ethernet	343
2.13.1	SIMATIC NET OPC UA server for the PROFINET IO protocol	344
2.13.2	How is the PROFINET IO OPC UA server addressed?	346
2.13.3	Which namespaces does PROFINET IO communication with OPC UA provide?	348
2.13.4	The Nodeld	351
2.13.5	Board object and PROFINET IO controller.....	352
2.13.6	Data variables of the PROFINET IO controller	355
2.13.7	Device objects.....	355

2.13.8	Data variables and methods of the device object	357
2.13.9	PROFINET IO module objects.....	358
2.13.9.1	Data variables of the PROFINET IO modules	360
2.13.9.2	IO data variables of the PROFINET IO modules	361
2.13.9.3	Data variables for IOPS and IOCS	362
2.13.9.4	Data record data variables of the PROFINET IO modules	366
2.13.10	PROFINET IO OPC UA templates.....	368
2.13.10.1	Template data variables.....	368
2.13.10.2	Directory of the template data variables	369
2.14	Server diagnostics	371
2.14.1	Protocol ID	371
2.14.2	OPC DA server diagnostics items.....	372
2.15	Buffer-oriented services with the OPC interface.....	376
2.15.1	Using buffer send/receive services	376
2.15.2	Properties of buffer-oriented communication	376
2.15.3	Mapping data buffers on OPC variables	377
2.15.4	Using buffer-oriented services	378
2.15.5	Points to note when using buffer-oriented services with TCP/IP native	378
2.16	Restricting access rights of the OPC variables.....	379
3	OPC Alarms & Events server for SIMATIC NET	383
3.1	Event server for S7 communication	383
3.1.1	Functions and alarm categories	383
3.1.2	Parameters for events.....	387
3.1.3	Event attributes	388
3.1.4	Attributes for entries in the diagnostic buffer of the module	399
3.1.5	Attributes for alarms indicating an interrupted connection.....	403
3.1.6	Configuring alarm texts, source and area.....	405
3.1.7	How can source information, sources and conditions be browsed and filtered in the namespace?.....	410
3.1.8	Mapping STEP 7 configuration values to OPC S7 Alarms & Events parameters	412
3.1.9	S7 demo alarms	413
3.2	Simple event server for SNMP communication	414
3.3	Alarms & Events server for S7 and SNMP communication	417
3.3.1	The A&E servers from SIMATIC NET.....	417
3.3.2	What are the names (ProgID) of the servers?	418
3.3.3	How can the protocol of the alarm source be detected?	419
4	Using the OPC server	421
4.1	Programming the automation interface.....	421
4.1.1	Programming the automation interface for data access	421
4.1.1.1	What does the object model of OPC data access provide?	421
4.1.1.2	Points to remember when programming.....	422
4.1.1.3	Objects of the automation interface for data access.....	423
4.1.2	Programming the automation interface for Alarms & Events	431
4.1.2.1	What does the object model of OPC Alarms & Events provide?	432
4.1.2.2	Points to remember when programming.....	433
4.1.2.3	Objects of the automation interface for Alarms & Events	433
4.2	Programming the custom interface	442

4.2.1	Creating COM objects and querying the status of the OPC server	443
4.2.2	Objects of the custom interface for data access.....	443
4.2.2.1	OPCServer object	443
4.2.2.2	OPCGroup object.....	448
4.2.3	Objects of the custom interface for Alarms & Events	456
4.2.3.1	OPCEventServer object.....	456
4.2.3.2	OPCEventSubscription object	459
4.2.3.3	OPCEventAreaBrowser object	461
4.2.4	Interfaces of the client for alarms and events.....	461
4.2.4.1	IOPCEventSink interface	461
4.2.4.2	IOPCShutdown interface	462
4.2.5	Error messages for OPC DA process variables	463
4.3	Programming the XML interface	466
4.3.1	Description of the elements	468
4.3.2	Basic schemas.....	469
4.3.2.1	ItemProperty	469
4.3.2.2	ItemValue	470
4.3.2.3	OPCError	471
4.3.2.4	ReplyBase.....	471
4.3.2.5	RequestOptions	472
4.3.3	Read and write jobs	473
4.3.3.1	Read.....	473
4.3.3.2	ReadResponse	475
4.3.3.3	Write.....	477
4.3.3.4	WriteResponse	479
4.3.4	Using subscriptions	480
4.3.4.1	Subscribe	482
4.3.4.2	SubscribeResponse.....	484
4.3.4.3	SubscriptionPolledRefresh	485
4.3.4.4	SubscriptionPolledRefreshResponse	486
4.3.4.5	SubscriptionCancel	487
4.3.4.6	SubscriptionCancelResponse.....	488
4.3.5	Further queries.....	488
4.3.5.1	Browse	488
4.3.5.2	BrowseResponse.....	490
4.3.5.3	GetProperties	491
4.3.5.4	GetPropertiesResponse.....	493
4.3.5.5	GetStatus	494
4.3.5.6	GetStatusResponse.....	495
4.4	Programming the OPC UA interface.....	496
4.4.1	Configuring the OPC UA server.....	497
4.4.1.1	Authentication	497
4.4.1.2	Endpoint security	498
4.4.2	Certificate management for the OPC UA server	499
4.4.3	Certificate management in the OPC UA client "OPC Scout V10"	501
4.4.4	Certificate management with a graphic user interface	506
4.4.5	OPC UA services	508
4.4.6	Creating OPC UA clients	511
4.4.6.1	Interfaces under OPC UA	511
4.4.6.2	The C interface with OPC UA	512
4.4.6.3	The .NET interface with OPC UA	512

4.4.7	Messages of the OPC UA server.....	513
4.4.8	Migration from OPC Data Access / Alarms & Events to OPC UA	515
4.5	Programming the OPC UA redundancy interface.....	516
4.5.1	Installing and configuring the "Network Load Balancing" feature	519
4.5.2	Configuration of the cluster	524
4.5.3	Locating the server endpoints with the help of the discovery server	532
4.5.4	The OPC UA reconnect procedure	532
4.5.5	Redundancy support by OPC UA clients	534
4.5.6	When does a server failover occur?	535
4.5.7	Details of the server failover	536
4.5.8	Reaction of the SIMATIC NET OPC UA S7 A&C to a redundancy failover	538
4.5.9	General recommendations and notes on redundant OPC UA S7 communication.....	540
4.5.10	Reading out the status of the cluster by the OPC UA client	541
5	SIMATIC Computing	545
5.1	.NET OPC client control.....	546
5.1.1	Overview	546
5.1.2	Step 1 - Installation	547
5.1.2.1	Referencing the controls	547
5.1.2.2	Setting a sample program.....	550
5.1.3	Step 2 - Controls in a WINDOWS FORM	552
5.1.3.1	Adding Windows controls.....	555
5.1.3.2	Adding the .NET OPC client control.....	557
5.1.4	Step 3 - Configuring data links.....	558
5.1.4.1	Browsing OPC items	559
5.1.4.2	Browsing Windows control properties	562
5.1.4.3	Creating a list of links	563
5.1.4.4	Trigger for writing	565
5.1.4.5	OPC quality and error	569
5.1.5	Step 4 - Starting the sample program	571
5.1.5.1	Debug and release	572
5.1.5.2	Distributing the application	572
5.1.6	Troubleshooting	573
5.1.7	Creating an OPC UA certificate for the NET OPC data control	574
5.1.8	OPC UA certificates when connecting to an OPC UA server	574
5.2	.NET OPC client API	576
5.2.1	SIMATICNET.OPCDAClient namespace	577
5.2.1.1	Classes of the data model	577
5.2.1.2	The interface of the DaServerMgt object	586
5.2.1.3	Troubleshooting	603
5.2.2	Example of namespace SIMATICNET.OPCDAClient	604
5.2.3	SIMATICNET.OPCCMN namespace.....	608
5.2.3.1	The interface of the OPCServerEnum object	608
5.2.3.2	ServerIdentifier class	610
5.2.3.3	ServerCategory enumerator	611
5.2.3.4	EndpointIdentifier class	611
5.2.3.5	PkiCertificate class	612
5.2.3.6	Creating a new certificate	612
5.2.3.7	toDER method.....	613
5.2.3.8	fromDER method	613
5.2.3.9	toWindowsStore method	613

5.2.3.10	toWindowsStoreWithPrivateKey method	614
5.2.3.11	fromWindowsStore method	614
5.2.3.12	fromWindowsStoreWithPrivateKey method.....	615
5.2.3.13	fromWindowsStoreWithPrivateKey method.....	615
5.2.3.14	Properties.....	616
5.2.3.15	Enumerator WinStoreLocation.....	616
6	Sample programs.....	619
6.1	OPC Automation interface (synchronous communication) in VB.NET	619
6.1.1	Activating the simulation connection.....	619
6.1.2	Working with the sample program	620
6.1.3	How the program runs	621
6.1.4	Using the OPC automation interface with .NET-Framework	622
6.2	OPC Custom interface (synchronous communication) in C++	625
6.2.1	Activating the simulation connection.....	625
6.2.2	Working with the sample program	626
6.2.3	How the program runs	627
6.2.4	Description of the OPCDA_SyncDlg.cpp program	630
6.2.4.1	OnInitDialog	631
6.2.4.2	OnStart.....	631
6.2.4.3	OnRead.....	639
6.2.4.4	OnWrite.....	641
6.2.4.5	OnStop.....	643
6.2.4.6	DestroyWindow.....	645
6.2.4.7	GetQualityText.....	645
6.2.5	Notes on writing your own programs	646
6.3	OPC Custom interface (asynchronous communication) in C++	646
6.3.1	Activating the simulation connection.....	647
6.3.2	Working with the sample program	647
6.3.3	Start program	647
6.3.4	Read and write values	648
6.3.5	Activate group	648
6.3.6	Stop the program	649
6.3.7	How the program runs	649
6.3.8	Description of the Program	656
6.3.9	The "OPCDA_AsyncDlg.cpp" file	656
6.3.10	Callback.cpp and Callback.h	677
6.3.11	Notes on writing your own programs	682
6.4	OPC Custom interface (asynchronous communication) in VB.NET	682
6.4.1	Working with the sample program	683
6.4.2	Description of the program	684
6.5	OPC Custom interface (synchronous communication) in C#	689
6.5.1	Working with the sample program	689
6.5.2	Description of the program	690
6.6	OPC Custom interface (asynchronous communication) in C#	693
6.6.1	Working with the sample program	694
6.6.2	Description of the program	695
6.7	OPC XML interface in C#	700
6.7.1	Working with the sample program	700

6.7.2	Add Web service to project.....	702
6.7.3	The MainForm class	703
6.7.4	The constructor of MainForm and the Dispose method.....	704
6.7.5	Creating the dialog box elements	705
6.7.6	The Main method	710
6.7.7	The Button_Start_Sample_Click method	711
6.7.8	The Button_Stop_Sample_Click method	713
6.7.9	The Button_Read_Value_Click method	713
6.7.10	The Button_Write_Value_Click method	715
6.8	OPC Alarms & Events Custom interface in C++.....	717
6.9	OPC UA interface in C	720
6.9.1	Activating the simulation connection.....	720
6.9.2	Importing the client certificates	721
6.9.3	Working with the sample program	721
6.9.4	Start program	722
6.9.5	Reading and writing values.....	722
6.9.6	Monitoring variables	722
6.9.7	Stop the program	723
6.9.8	Description of the program sequence	723
6.9.8.1	Connection establishment.....	723
6.9.8.2	Reading and writing variables.....	724
6.9.8.3	Monitoring variables and alarms	725
6.9.8.4	Connection termination	725
6.9.9	Notes on converting to real variables	726
6.10	OPC UA interface (asynchronous communication) in C#.....	726
7	Reference Automation Interface.....	727
7.1	General information	727
7.1.1	What is an interface?	727
7.1.2	The two Interface types of OPC.....	728
7.1.3	COM/OLE objects	728
7.1.4	Collection objects	729
7.1.5	Object model	729
7.1.6	Data synchronization	731
7.1.7	Exceptions.....	731
7.1.8	Events	731
7.1.9	Arrays.....	731
7.1.10	Parameters.....	732
7.1.11	Type library	732
7.2	The OPCServer object	732
7.2.1	Properties of the OPCServer object	733
7.2.1.1	StartTime.....	733
7.2.1.2	CurrentTime	733
7.2.1.3	LastUpdateTime.....	734
7.2.1.4	MajorVersion	734
7.2.1.5	MinorVersion	735
7.2.1.6	BuildNumber	735
7.2.1.7	VendorInfo.....	736
7.2.1.8	ServerState	736
7.2.1.9	LocaleID	737

7.2.1.10	Bandwidth	738
7.2.1.11	OPCGroups	738
7.2.1.12	PublicGroupNames	739
7.2.1.13	ServerName	739
7.2.1.14	ServerNode	740
7.2.1.15	ClientName	740
7.2.2	Methods of the OPCServer object	741
7.2.2.1	GetOPCServers	741
7.2.2.2	Connect	742
7.2.2.3	Disconnect	743
7.2.2.4	CreateBrowser	743
7.2.2.5	GetErrorString	744
7.2.2.6	QueryAvailableLocaleIDs	745
7.2.2.7	QueryAvailableProperties	745
7.2.2.8	GetItemProperties	746
7.2.2.9	LookupItemIDs	747
7.2.3	Events of the OPCServer object	748
7.2.3.1	ServerShutDown	749
7.3	The collection object OPCGroups	749
7.3.1	Properties of the OPCGroups object	750
7.3.1.1	Parent	750
7.3.1.2	DefaultGroupsActive	751
7.3.1.3	DefaultGroupUpdateRate	751
7.3.1.4	DefaultGroupDeadband	752
7.3.1.5	DefaultGroupLocaleID	752
7.3.1.6	DefaultGroupTimeBias	753
7.3.1.7	Count	753
7.3.2	Methods of the OPCGroups object	754
7.3.2.1	Item	754
7.3.2.2	Add	755
7.3.2.3	GetOPCGroup	755
7.3.2.4	Remove	756
7.3.2.5	RemoveAll	756
7.3.2.6	ConnectPublicGroup	757
7.3.2.7	RemovePublicGroup	757
7.3.3	Events of the OPCGroups object	758
7.3.3.1	GlobalDataChange	758
7.4	The OPCGroup object	760
7.4.1	Properties of the OPCGroup object	761
7.4.1.1	Parent	761
7.4.1.2	Name	761
7.4.1.3	IsPublic	762
7.4.1.4	IsActive	762
7.4.1.5	IsSubscribed	763
7.4.1.6	ClientHandle	764
7.4.1.7	ServerHandle	764
7.4.1.8	LocaleID	765
7.4.1.9	TimeBias	765
7.4.1.10	DeadBand	766
7.4.1.11	UpdateRate	767
7.4.1.12	OPCItems	767

7.4.2	Methods of the OPCGroup object.....	768
7.4.2.1	SyncRead.....	768
7.4.2.2	SyncWrite.....	770
7.4.2.3	AsyncRead.....	771
7.4.2.4	AsyncWrite	773
7.4.2.5	AsyncRefresh.....	774
7.4.2.6	AsyncCancel	775
7.4.3	Events of the OPCGroup object.....	776
7.4.3.1	DataChange	776
7.4.3.2	AsyncReadComplete	777
7.4.3.3	AsyncWriteComplete	778
7.4.3.4	AsyncCancelComplete.....	779
7.5	The collection object OPCItems.....	780
7.5.1	Properties of the OPCItems collection object	780
7.5.1.1	Parent.....	781
7.5.1.2	DefaultRequestedDataType.....	781
7.5.1.3	DefaultAccessPath.....	781
7.5.1.4	DefaultIsActive	782
7.5.1.5	Count.....	782
7.5.2	Methods of the collection object OPCItems	783
7.5.2.1	Item	783
7.5.2.2	GetOPCItem.....	784
7.5.2.3	AddItem.....	784
7.5.2.4	AddItems	785
7.5.2.5	Remove	786
7.5.2.6	Validate	787
7.5.2.7	SetActive	788
7.5.2.8	SetClientHandles	789
7.5.2.9	SetDataTypes	790
7.6	The OPCItem object	790
7.6.1	Properties of the OPCItem object	791
7.6.1.1	Parent.....	791
7.6.1.2	ClientHandle.....	791
7.6.1.3	ServerHandle	792
7.6.1.4	AccessPath	792
7.6.1.5	AccessRights	793
7.6.1.6	ItemID.....	793
7.6.1.7	IsActive.....	794
7.6.1.8	RequestedDataType	794
7.6.1.9	Value	795
7.6.1.10	Quality	796
7.6.1.11	TimeStamp	796
7.6.1.12	CanonicalDataType	796
7.6.1.13	EUType	797
7.6.1.14	EUInfo	798
7.6.2	Methods of the OPCItem object.....	798
7.6.2.1	Read.....	799
7.6.2.2	Write	800
7.7	Definitions	800
7.7.1	State of the server	800
7.7.2	Error messages.....	801

7.8	Appendix on the Automation Interface reference	803
8	Related literature.....	805
8.1	OPC Specifications	806

Preface

What's new in the SIMATIC NET OPC server V13?

In OPC Unified Architecture Version 1.01, the previous functionality has been enhanced and expanded.

The SIMATIC NET OPC UA servers also support:

- S7OPT UA alarming

What's new in this manual?

- The description of S7OPT UA Alarming was added in the section "Optimized S7 communication".

SIMATIC NET - Pioneering successful solutions in black and white

This documentation will be your companion on your way to successful application of SIMATIC NET. It will provide you with a clear introduction to the topics and will show you how to install and configure individual components and how to create programs based on OPC. You will see the opportunities that industrial communication with SIMATIC NET can open up for you, for your automation solutions, and, above all, for the success of your company.

SIMATIC NET - The right decision

You know the advantages of distributed automation systems and want to make optimum use of industrial communication. You expect a strong partner and innovative, reliable products. With SIMATIC NET, you've made the right choice.

This documentation makes use of your basic knowledge and allows you to profit from the know-how of specialists.

Are you a beginner?

Then you can familiarize yourself systematically. Start in Volume 1 with the introduction to industrial communication. In Volume 1, you will find all the necessary information on the communications principles and range of functions of the SIMATIC NET OPC server. Read up on the basics of the OPC interface and familiarize yourself with the SIMATIC NET communications protocols and their advantages and functions.

Are you a professional?

Then you can get going straight away. This volume 2 provides you with all the information you require to work with SIMATIC NET.

1.1 How do I work with the documentation?

Do you find examples useful?

The supplied sample programs will provide you with a flexible basis with which you can put your own ideas into practice.

SIMATIC NET glossary

Explanations of the specialist terms used in this documentation can be found in the SIMATIC NET glossary.

You will find the SIMATIC NET glossary here:

- SIMATIC NET Manual Collection

The DVD ships with certain SIMATIC NET products.

- On the Internet under the following entry ID:

50305045 (<http://support.automation.siemens.com/WW/view/en/50305045>)

Security messages

Note

Siemens offers IT security mechanisms for its automation and drive product portfolio in order to support the safe operation of the plant/machine. Our products are also continuously developed further with regard to IT security. We therefore recommend that you regularly check for updates of our products and that you only use the latest versions. You will find information in:

(<http://support.automation.siemens.com/WW/lisapi.dll?func=cslib.csinfo2&aktprim=99&lang=en>)

Here, you can register for a product-specific newsletter.

For the safe operation of a plant/machine, however, it is also necessary to integrate the automation components into an overall IT security concept for the entire plant/machine, which corresponds to the state-of-the-art IT technology. You will find information on this in: (<http://www.siemens.com/industrialsecurity>)

Products from other manufacturers that are being used must also be taken into account.

1.1 How do I work with the documentation?

To help you to work with the documentation, various conventions and symbols have been used. These are intended to highlight different types of information and to speed up access to the information you require.

Conventions

The documentation uses a variety of conventions to highlight different types of information and to allow you to locate specific types of information at a glance. The table below shows the conventions used.

Convention	indicates:
<i>Italics</i>	These are <i>highlighted</i> expressions or <i>names</i> of items, variables, programs, dialogs etc.
<Text in pointed brackets>	These are <variable terms> that must be replaced by the currently relevant term.
Terms printed in bold at the start of the text column	Syntax elements , that are described below.
Text in the Courier New font	This is program code.
{Syntax text in braces}	Optional information in syntax descriptions
"Syntax text in quotes"	Variable terms in syntax descriptions

1.2 Regulations

You must observe the following regulations relating to SIMATIC NET products:

1.2.1 Which legal regulations do you need observe?

We would point out that the contents of this document shall not become a part of or modify any prior or existing agreement, commitment or legal relationship. The Purchase Agreement contains the complete and exclusive obligations of Siemens. Any statements contained in this document do not create new warranties or restrict the existing warranty.

We would further point out that, for reasons of clarity, this document cannot deal with every possible problem arising from the use of the OPC Server. Should you require further information or if any special problems arise which are not sufficiently dealt with here, please contact your local Siemens representative.

Disclaimer

We have checked the contents of this document for agreement with the hardware and software described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this document are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcomed.

Technical data subject to change.

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility or design, are reserved.

Siemens AG
IIA SC CI

Postfach 4848
D-90026 Nürnberg

1.2.2 Which safety regulations do you need to know?

Qualified personnel

Only qualified personnel should be allowed to install and work with this product. Qualified persons are defined as persons who are authorized to commission, to ground, and to tag circuits, equipment, and systems in accordance with established safety practices and standards.

Correct usage

WARNING

This device and its components may only be used for the applications described in the relevant catalog or the technical description, and only in connection with devices or components from other manufacturers which have been approved or recommended by Siemens.

This product can only function correctly and safely if it is transported, stored, set up, and installed correctly, and operated and maintained as recommended.

With the products described in this document, it is easy to access process data and to modify process data. Modifying process data can lead to unforeseeable reactions in the process that can lead to death, serious injury to persons and/or damage to equipment.

Proceed cautiously and make sure that you do not access data that could cause unexpected reactions from the controlled devices and equipment.

OPC process variables for SIMATIC NET

This chapter explains the syntax of the names of process variables. These names specify which process values are addressed on the OPC interface. You specify the variable names when you program or configure an OPC client.

2.1 Which communication functions exist?

The OPC server provides standardized access to the SIMATIC NET industrial communications networks.

The SIMATIC NET OPC Server supports the interfacing of applications with any automation components networked over PROFIBUS or Industrial Ethernet. It provides the following communication functions:

- S7 communication
- Open communication services (SEND/ RECEIVE)
- PROFIBUS DP and FDL
- SNMP
- PROFINET IO

The communication functions are specially optimized for the various requirements. Several communication functions can be supported at the same time by the OPC server for Data Access and Unified Architecture. S7 communication, PROFINET IO communication, SEND/RECEIVE communication and DP communication are also available for OPC UA.

2.2 What are process variables?

A process variable is a writable and/or readable data item of the process I/O such as the temperature of a tank used as the input value of a programmable controller.

OPC COM (Data Access)

Process variables are represented in the OPC Data Access class model by the class OPC item. Only elements of this class represent a real value from the process in OPC.

ItemID

The ItemID is a string that identifies the process variable uniquely. It tells the server which process variables are assigned to the OPC item. The process value can then be accessed via the OPC item.

The OPC Server of SIMATIC NET maps the various communication services of the SIMATIC NET communications protocols using OPC items by using parts of the ItemID as parameters for calling the communication function.

OPC UA

All elements of a process are represented in the OPC UA object model by suitable objects, for example process variables by data variables or properties, alarms by alarm instances or communications services by methods. The objects may be interrelated.

Nodeld

The Nodeld is made up of a namespace index and an identifier (character string, numeric value, byte string or GUID). The Nodeld identifies an object from the process on an OPC UA server.

The OPC UA servers of SIMATIC NET map the various communication services of the protocols via access to objects.

2.3 How are the ItemIDs of the process variables formed?

Process variables are identified on the OPC interface by a unique name, the ItemID. The ItemID is made up as follows:

Syntax

```
<protocolID>:[<connectionname>]<variablename>
```

Explanations

<protocolID>

Specifies the protocol for access to the process variable.

The following protocol IDs exist:

DP	DP protocol including DP master, DP slave, and DPC1
S7	S7 functions over PROFIBUS and Industrial Ethernet
SR	Open communications services via Industrial Ethernet
FDL	Open communications services via PROFIBUS
SNMP	SNMP Communication over Industrial Ethernet
PN IO	PROFINET IO communication over Industrial Ethernet
DP2	Class 2 DP Master and DPC2

<connectionname>

The connection name identifies the connection or communications module via which the partner (for example PLC, other PC station or DP slave) can be accessed. This is specified in STEP 7 / SIMATIC NCM PC during network configuration.

The connection name specified depends on the protocol being used. The connection name must be unique within a protocol.

Note**Permitted characters for connection names**

The following characters are permitted in the character set for connection names:

A-Z, a-z, 0-9, _, -, ^, !, #, \$, %, &, ',(,), <, >, =, ?, ~, +, *, ' ', :, @, {, }, " and the space (space not at the start or end of a connection name)

The following characters are not permitted in the character set for connection names:

- Period ".."
- Pipe symbol "|"
- Backslash "\"
- Slash "/"
- Square brackets "[" and "]"

Spaces at the start and end of the connection name also not permitted.

<variablename>

Variable to be addressed.

The variable name must be unique for the connection specified in the connection name. The structure of the variable name depends on the selected protocol.

Note

The ItemIDs are not case-sensitive.

Note

The syntax of OPC UA NodeIDs is described in the section "Programming the OPC UA interface (Page 496)".

Note

Curly brackets indicate optional syntax elements.

2.4 PROFIBUS DP

Process variables for the DP master

The OPC Server of SIMATIC NET for the DP master mode provides process variables for the following services:

- Services for a class 1 master
 - Access and monitoring of DP inputs and outputs
- DPC1 services
 - Acyclic transmission of blocks of data
- Fast Logic for
 - CP 5613/ CP 5613 A3 and CP 5614/ CP 5614 A3 (DP master only)
Automatic monitoring of slave data
 - the CP 5623 and CP 5624 (DP master only)
Automatic monitoring of slave data
- Diagnostics variables
 - Evaluation of static diagnostics information

Process variables for class 2 DP master

- Configuration services
 - Reading the input and output data of a slave
 - Reading the slave configuration
 - Writing data records for a slave
- Online diagnostics
 - Reading the diagnostics data of a slave
 - Reading the class 1 master diagnostics data
- Writing and reading data records for a slave (DPC2)

Process variables for the DP slave

The OPC Server of SIMATIC NET for the DP slave mode provides process variables for the following services:

- Variable services for access to the local slave data
 - Access to the inputs and outputs of the slave
- Diagnostics variables
 - Evaluation of static diagnostics information of the slave

2.4.1 Class 1 DP master

2.4.1.1 Powerful SIMATIC NET in-process server for the PROFIBUS DP protocol

The PROFIBUS DP protocol contains an image of the input and output data on the communications processor of the computer. Access to this process data is within the local computer and can therefore be extremely fast particularly when using the SIMATIC NET modules CP 5613/CP 5614 and CP 5623/ CP 5624 via the "DP Base" interface.

In some situations, for example, when using PCbased controllers, extremely fast access to process data is necessary. For the extremely fast DP protocol, SIMATIC NET offers an in-process server that provides practically the full performance of the DP protocol for OPC clients as well.

Using OPC as COMbased clientserver architecture involves certain internal execution times depending on the implementation of the OPC server. These result in the main when using a local server (also known as an "outprocess server"; an "*.exe" file with its own process space) due to process changeovers and transferring the function parameters from the client to the server (marshaling).

If the OPC server is implemented as an inprocess server, the times for changing processes and marshaling are avoided since the OPC server takes the form of a dynamic link library (DLL) and runs in the process of the client.

When using an in-process server, however, the following must be taken into account:

- Only 1 client can use the server at any one time.

The simultaneous use of the inprocess OPC server by several clients would mean generating the server more than once in different process spaces and would result in simultaneous but uncoordinated access to the same hardware. As a result, only the client that starts first has access to the process data while access by other clients would be denied.

- The stability of the OPC server depends on the client.

If the OPC client behaves in an uncontrolled manner and, for example causes access violations, the OPC server will also be affected. The result is that it will no longer be possible for the OPC server to reset the communications module when necessary. Explicit closing of the OPC server using the configuration programs would also not be possible.

Calling the inprocess server for DP

The inprocess server for DP is addressed using a separate ProgID. The ProgID is called "OPC.SimaticNET.DP".

The ProgID is specified in function calls or in the OPC clients to select a server.

Examples of function calls:

Visual Basic:

```
ServerObj.Connect ("OPC.SimaticNET.DP")
```

Visual C++:

```
r1 = CLSIDFromProgID(L"OPC.SimaticNET.DP", &clsid);  
r1 = CoCreateInstance (clsid, NULL, CLSCTX_INPROC_SERVER,  
IID_IOPCServer, (void**)&m_pIOPCServer);
```

2.4.1.2 Powerful SIMATIC NET OPC server for the PROFIBUS DP protocol

Improved performance even with several clients

As described in the section above, the high-performance inproc server for PROFIBUS DP can be used by only one client. To allow the same utilization by two or more clients with the same performance requirements, a second configuration variant is available.

To use this variant, all underlying DP protocol libraries and the COM server as inproc server are loaded on the outproc OPC server. The protocol is handled in the process of the OPC server, additional execution times for changing between processes and multiprotocol mode are avoided. The process change between the OPC client and OPC server is, however, still necessary.

Configuration

This high-speed variant is enabled implicitly by selecting the "DP" protocol as the only protocol in the "Communication Settings" configuration program (if other protocols or the OPC UA interface are selected, the performance advantage described is lost):

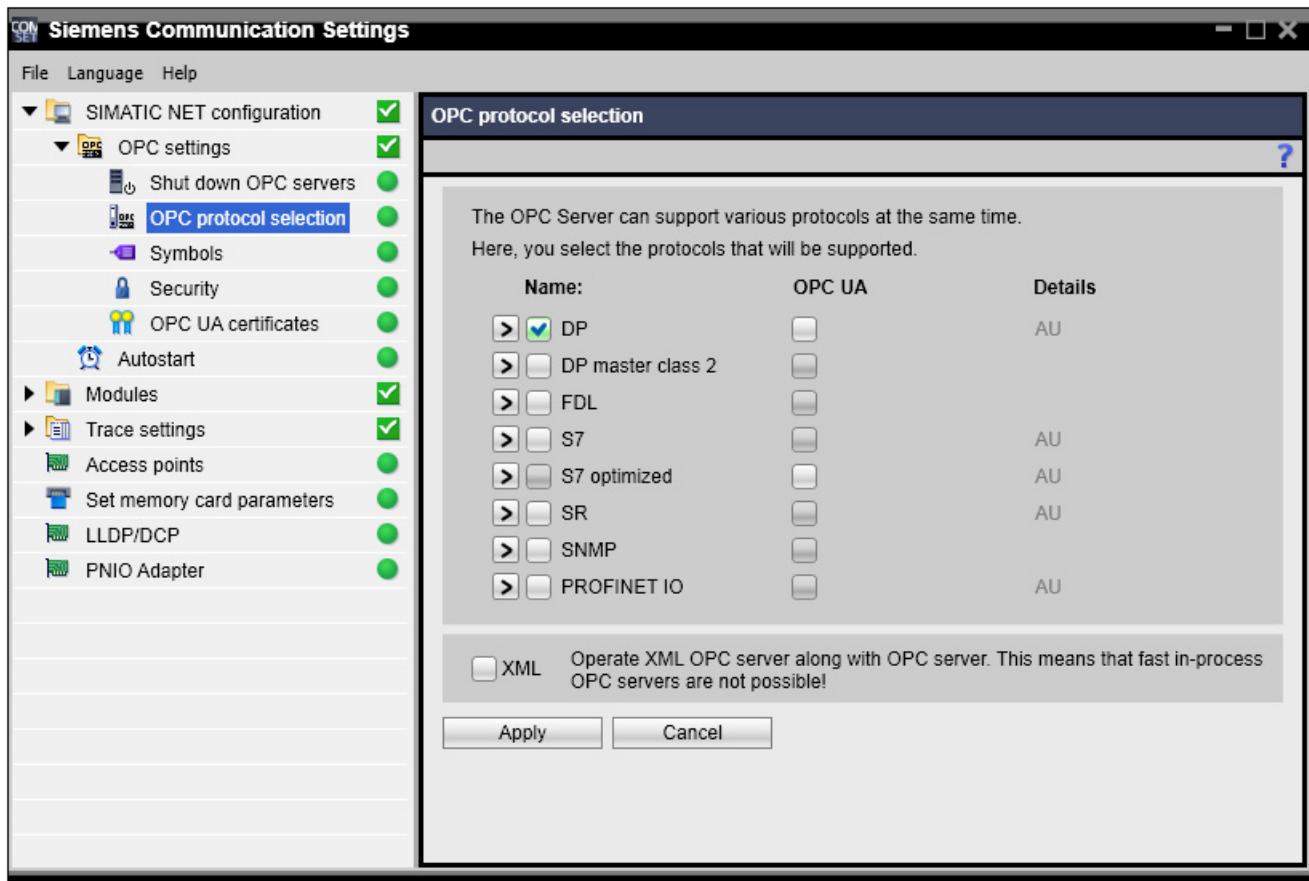


Figure 2-1 Window in the "Communication Settings" configuration program for selecting the DP protocol

"Symbols" can also be selected.

Note

When creating symbols with the symbol editor, the use of the following characters is permitted: A-Z, a-z, 0-9, _, -, ^, !, #, \$, %, &, ', /, (,), <, >, =, ?, ~, +, *, '!', :, |, @, [], {, }, ". When creating symbols with STEP 7, you should also remember that problems can arise when resolving the array if your symbol file includes symbols in the form <symbolname> and <symbolname>[<index>] at the same time.

Advantages / disadvantages

Using the powerful DP OPC server has the disadvantage that only the single protocol (DP) mode is possible. On the other hand, it does have the following advantages:

- Higher performance than with multiprotocol mode.
- Simple configuration.
- Several clients can use the server at the same time.
- The stability of the OPC server does not depend on the clients.

2.4.1.3 DPC1 and DPC2 services

Class 1 DP master

The DP class 1 master communicates cyclically with the DP slaves. With DPC1, a master operating cyclically can handle additional acyclic data traffic (read data record, write data record) and react to alarms from slaves. To be able to use DPC1, the DP slaves must support these DPV1 additional functions.

Communication includes central functions such as:

- Configuring and assigning parameter values to the slaves
- Cyclic data transfer with the DP slaves
- Monitoring the DP slaves
- Providing diagnostics information

Class 2 DP master

With DPC2, a DP class 2 master can establish additional communication relations with the DP slaves. The DP slaves must support the DPV1 additional functions. The most important DPC2 additional functions are as follows:

- Connection establishment
- Connection termination
- Reading data records
- Writing data records

Slaves with DP-V1 additional functions

Slaves with DPV1 additional functions can communicate with class 1 and class 2 DP masters.

Overview of the DP protocol

The following figure illustrates the parts of the DP or DPV1 protocol:

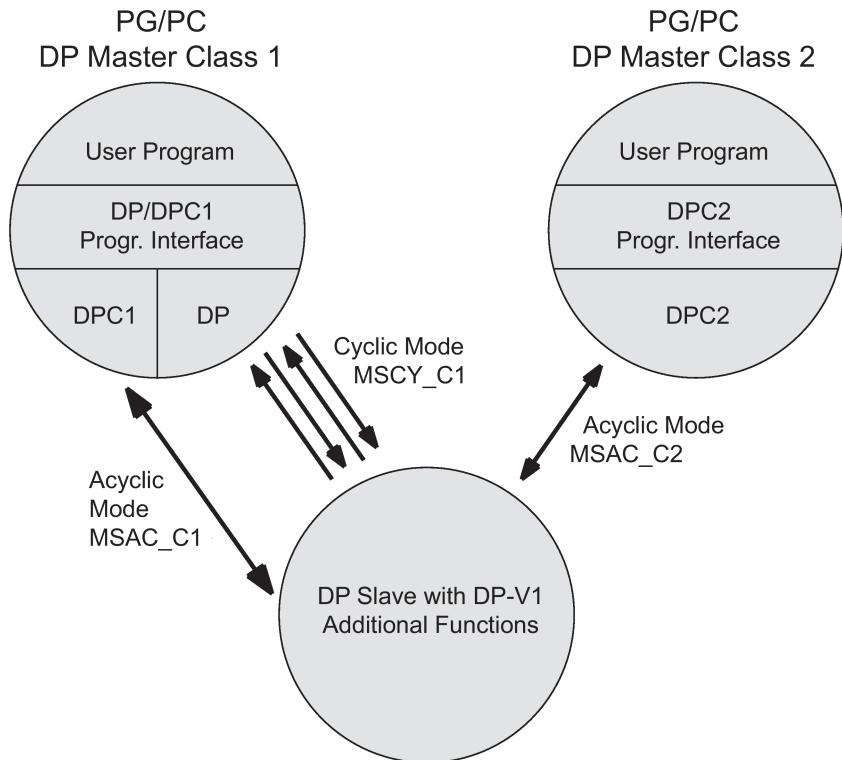


Figure 2-2 The parts of the DP or DP-V1 protocol

MSCY_C1	Master-slave, cyclic C1 mode
MSAC_C1	Master-slave, acyclic C1 mode
MSAC_C2	Master-slave, acyclic C2 mode

2.4.1.4 Process variables for services of the class 1 master

With the services for accessing cyclic data, you can access, monitor and control the inputs and outputs of the slaves.

Access is via:

- Slave number.
This number corresponds to the PROFIBUS address.
- Submodule number.
A DP slave can include several submodules with different input/output areas.
- Input/output area.

2.4.1.5 Syntax of process variables for a class 1 master

Syntax

Inputs:

```
DP: [<connectionname>] slave<address>{M<number>}_I{<format>  
<offset>{.<bit>}{},<quantity>}}
```

Outputs:

```
DP: [<connectionname>] slave<address>{M<number>}_Q{<format>  
<offset>{.<bit>}{},<number>}}
```

Explanations

DP

DP protocol for access to the process variable.

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

slave

Indicates that a DP slave is addressed.

<address>

PROFIBUS address of the slave.

Range: 0 ... 126.

M

Identifier for the number of the submodule that contains the input or output area.

<number>

Number of the submodule containing the input or output area.

_I

Identifier for an input. Inputs are read only.

_Q

Identifier for an output. Outputs can be read and written.

<format>

Format of the delivered data.

Specifying the format specifies the data type.

In principle, all listed OLE data types can be read via the Automation and Custom interface of OPC. However, certain development tools such as Visual Basic only offer a restricted number of data types.

The following table lists the corresponding Visual Basic type in which the variable value can be represented.

Format identifier	Description	OLE data type	Visual Basic type
X	Bit	VT_BOOL	Boolean
BYTE or B	Byte (unsigned 8)	VT_UI1	Byte
CHAR	Character (signed 8)	VT_I1	Integer
WORD or W	Word (unsigned 16)	VT_UI2	Long
INT	Integer (signed 16)	VT_I2	Integer
DWORD or D	Double word (unsigned 32)	VT_UI4	Double
DINT	Double integer (unsigned 32)	VT_I4	Long
REAL	Floating-point number (IEEE 4 bytes)	VT_R4	Single

<offset>

Byte offset in the address space of the slave at which the element to be addressed is located. If a submodule is specified, the offset applies within the submodule. If no submodule is specified, the offset relates to the entire input/output area of the slave.

<bit>

Bit number in the addressed byte.

Specifying a bit is permitted only with the Xformat identifier, the range is between 0 and 7.

<quantity>

Number of elements.

The data type (VT_ARRAY) of the variable is an array with elements of the specified format. If quantity is omitted, the quantity 1 is assumed and the data type of the variable is not an array.

Note

If no submodule is specified, the variable returns the entire input or output area over all submodules. Structured access by specifying the format, offset and quantity is possible.

2.4.1.6 Protocol ID

The protocol ID for the *DP protocol* is "DP".

2.4.1.7 Configured CP name

The connection name specifies the communication access by specifying the communications processor attached to the PROFIBUS network.

In the DP protocol, the CP name is the configured name of the communications module.

Syntax

DP: [<configuredCPname>]

Explanation

<configuredCPname>

The configured CP name is specified with SIMATIC STEP 7 / NCM.

Examples of "configured CP name"

Typical configured CP names are:

CP 5611 A2

CP 5621

CP 5613

CP 5614

CP 5623

CP 5624

2.4.1.8 Examples of process variables for a class 1 master

Here you will find examples illustrating the syntax of variable names for DP variables.

Inputs

DP:[CP 5623]Slave005M003_IB0

Slave005M003_IB0

Input byte 0 of submodule 3 of slave 5

DP:[CP 5623]Slave005M003_IB1,3

Slave005M003_IB1,3

Array with 3 bytes starting at input byte 1 of submodule 3 of slave 5

DP:[CP 5623]Slave005M003_ID2

Slave005M003_ID2

Double word starting at input byte 2 of submodule 3 of slave 5

DP:[CP 5623]Slave004M003_IReal0

Slave004M003_IReal0

Floating-point number in the input area of slave 4, submodule 3

DP:[CP 5623]Slave004_IB0,8

Slave004_IB0,8

The first 8 bytes of the entire input area of slave 4 over all submodules

Outputs

DP:[CP 5623]Slave005M007_QB1

Slave005M007_QB1

Output byte 1 of submodule 7

DP:[CP 5623]Slave005M007_QX2.5

Slave005M007_QX2.5

Bit 5 in output byte 2 of slave 5, submodule 7

*DP:[ICP 5623]Slave004_QW0,8
Slave004_QW0,8
Array with 8 words from the output area of slave 4 over all submodules*

2.4.1.9 DPC1 services

Using DPC1 services, you can access the data records of the DPC1 slaves. The data records are transferred acyclically.

The meaning of the data records is specified by the vendor of the slave. They could, for example, be used for the configuration data of a drive.

When a DPC1 variable is registered, the OPC server can only check that the syntax is correct and not whether the variable is valid on the partner device and the size of the data record is adequate based on the configuration of the DPC1 slave.

2.4.1.10 Syntax of the process variables for DPC1 services

Syntax

```
DP: [<connectionname>] slave<address>S<slot>data<index>
{,<length>} {,<subarea>}
```

Explanations

DP

DP protocol for access to the process variable.

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

slave

Identifier for access to a slave over the DP protocol.

<address>

PROFIBUS address of the slave.

Range: 0 ... 126.

S

Identifier for the slot of the slave, typically a submodule.

<slot>

Slot in the extended memory area of a slave for acyclic services. The slot and index identify a data record.

data

Identifier for access to a data record.

<index>

Index within a slot in the extended memory area of a slave for acyclic services. The slot and index identify a data record.

<length>

Length of the record. Range between 1 and 240.

<subarea>

The subarea is made up as follows:

```
<format><offset>{.<bit>}{,<quantity>}
```

<format>

Format in which the data is delivered.

If no format is specified, the *byte* format is used.

Format identifier	Description	OLE data type	Visual Basic type
X	Bit	VT_BOOL	Boolean
BYTE or B	Byte (unsigned 8)	VT_UI1	Byte
CHAR	Character (signed 8)	VT_I1	Integer
WORD or W	Word (unsigned 16)	VT_UI2	Long
INT	Integer (signed 16)	VT_I2	Integer
DWORD or D	Double word (unsigned 32)	VT_UI4	Double
DINT	Double integer (unsigned 32)	VT_I4	Long
REAL	Floating point number	VT_R4	Single

<offset>

Byte address in the data record for the element to be addressed.

<bit>

Bit number in the addressed byte.

A bit can only be specified with the format identifier X.

<quantity>

Number of elements

The data type of the variable is an array (VT_ARRAY) with elements of the specified format.

If <quantity> is omitted, the quantity 1 is assumed. With quantity 1, the data type is variable and not array.

Value of the data record

Return value of the data record item

The data type is VT_ARRAY | VT_UI1 if no subarea is specified.

The value is read-only; if the data record length is specified, it can also be written.

Note

The data length resulting from the parameters *quantity* and *format* must not exceed the size of the data record on the slave. The size of a data record depends on the particular slave and cannot be checked by the OPC server.

If you define subareas, remember the following point: If you read a data record, the partner device initially always reads the entire data record. The subarea is only evaluated afterwards.

When writing, the entire data record is also sent to the partner device. If several subareas are written using a group call, all the subareas are first entered in the data record before the data record is sent.

You should therefore put all OPC items with part access to a data record into one group and write the entire group.

Note**Writing data records to the partner device**

When writing data records, make sure that you avoid overlaps and gaps since it is not possible to predict which value will be written.

2.4.1.11 Examples of process variables for DPC1 services

Here you will find examples illustrating the syntax of variable names for DPC1 services.

Variable names for DPC1

DP:[CP 5623]Slave005S003Data2,120,DWORD7

Slave005S003Data2,120,DWORD7

Access to the double word starting at offset 7 in a data record with a length of 120 bytes in slot 3, index 2 of slave 5.

DP:[CP 5623]Slave005S003Data2,120,B8,4

Slave005S003Data2,120,B8,4

Access to an array with 4 bytes from offset 8 in a data record with a length of 120 bytes in slot 3, index 2 of slave 5.

2.4.1.12 Fast Logic for CP 5613/CP 5614 and CP 5623/CP 5624

The CP 5613/CP 5623 and DP master section of the CP 5614/CP 5624 support the Fast Logic property. This means that you can set the CP parameters so that it writes values to the same or other slaves as a reaction to the data change of a slave. The user application is also informed of the data change.

The CP 5613/CP 5614 and CP 5623/CP 5624 provide four Fast Logic triggers that can be configured and evaluated using the OPC control variables.

Advantages of fast logic

Using Fast Logic has the following advantages:

- There is less load on the OPC server and OPC client.
- The data transmission is faster because it is not dependent on the software running on the CP and takes place in the hardware of the CP.

Note

After a fast logic trigger has been triggered, it is automatically deactivated again afterwards. You must then activate the trigger again using the FLActivate variable.

Fast logic functions correctly only when the DP master is in the OPERATE mode and the slaves involved are in the READY state. A fast logic trigger should therefore only be activated by the DP application when the user program has brought the DP master to the OPERATE mode.

As long as fast logic triggers are active, no DP user program may write to the output bytes linked to input bytes with fast logic .

2.4.1.13 Syntax of the control variables for fast logic

Syntax

DP: [<connectionname>] FL<parameter><N>

Explanations

DP

DP protocol for access to the process variable.

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

FL

Identifier for fast logic.

<parameter>

Possible values are as follows:

State

Returns the fast logic status.

Return values:

CLEARED

Trigger N is not activated.

ACTIVATED

Trigger N is activated.

TRIGGERED

Trigger *N* has executed monitoring.

OLE data type	Visual Basic type
VT_BSTR	String

Activate

Activate can only be written.

Writing a parameter field specifies the Fast Logic property for the trigger specified in the ItemID.

The parameter field that is written is an array with 8 bytes and has the following structure:

slave_addr_in_byte

Address of the slave whose inputs are selected for the trigger

index_in_byte

Offset of the input byte of the trigger

cmp_value_in_byte

Comparison value for the input byte

mask_in_byte

You can mask individual bits in the input byte so that they are ignored in the comparison. A bit is masked by setting the value "1" in the appropriate bit, this means that if *mask_in_byte*==0x00 is set, all bits of *cmp_value_in_byte* are used for the comparison. The trigger is triggered when all unmasked bits in the selected input byte match the bits in *cmp_value_in_byte*.

slave_addr_out_byte

Selects the slave whose output byte will be changed when the trigger condition occurs

index_out_byte

Offset of the output byte

value_out_byte

Value of the output byte that will be written

mask_out_byte

Bits in the output byte can be masked individually so that they are not modified when the trigger condition arrives. A bit is masked by setting the value "1" in the appropriate bit; this means that if *mask_out_byte*==0x00 is set, all bits of *value_out_byte* are written to the selected output byte.

OLE data type	Visual Basic type
VT_ARRAY of VT_UI1	Byte()

Clear

Clear can only be written.

If the Boolean value TRUE is written, the Fast Logic trigger specified in the ItemID is deactivated.

OLE data type	Visual Basic type
VT_BOOL	Boolean

<N>

Number of the fast logic trigger being used. Value between 1 and 4.

2.4.1.14 DPspecific information variables

The DPspecific information variables for the class 1 master are used to query information on the DP master and the attached DP slaves.

You can query the following information:

- Mode of the DP master
- Event messages of the DP master
- Sign of life monitoring of the CP
- Mode of a DP slave
- The type of a DP slave
- Other information for more detailed diagnostics

2.4.1.15 Syntax of the DP-specific information variables

Syntax

```
DP: [<connectionname>]<informationparameter>
```

Explanations

DP

DP protocol for access to the process variable.

<connectionname>

Protocolspecific connection name. The connection name is specified in the configuration.

<informationparameter>

Possible values are as follows:

- Masterstate
- EvAutoclear
- EvWatchdog
- EvClass2Master
- WatchdogTimeout
- Slave/nSlvState
- Slave/nSlvDiag
- Slave/nSlvRestart
- Slave/nMiscSlvType
- Other slave information

Masterstate

Current mode of the DP master.

The current mode can be both written and read. Setting the mode by writing one of the values shown below is only possible within the context of the DP application environment.

Input and return values:

OFFLINE

No communication between master and slave.

STOP

No communication between master and slave except for diagnostics data

CLEAR

Parameter assignment and configuration phase

AUTOCLEAR

Autoclear phase, the DP master can no longer access all slaves

OPERATE

Productive phase

OLE data type	Visual Basic type
VT_BSTR	String

EvAutoclear

Signals an error during communication with DP slaves. The DP system automatically closes down and changes to the CLEAR mode.

Return values:

True

An error has occurred in communication with DP slaves and the system has closed down and changed to the CLEAR mode

False

No error occurred

OLE data type	Visual Basic type
VT_BOOL	Boolean

Note

You can only use EvAutoclear if AUTOCLEAR was set during configuration.

If this information parameter is output, no reaction is necessary.

EvWatchdog

Signals that job monitoring timed out on the module. The OPC server has not made a DP function call during the monitoring time. The OPC client or OPC server may no longer be available.

You set this monitoring time in the configuration.

Return values:

True

Monitoring timed out on the module

False

Monitoring did not time out on the module

OLE data type	Visual Basic type
VT_BOOL	Boolean

EvClass2Master

Signals access by a class 2 DP master.

Return values:

True

A class 2 DP master is participating in data traffic and accessing internal diagnostics lists of the class 1 DP master

False

No class 2 DP master is involved

OLE data type	Visual Basic type
VT_BOOL	Boolean

Note

No reaction to the event by the user program is required.

WatchdogTimeout

Watchdog on the CP. If a value is written, the watchdog can be set. The default value is set in the configuration.

Return values:

0

Monitoring off

400 - 102000

Any value in ms. The value is rounded to a multiple of 400.

6000

Default

OLE data type	Visual Basic type
VT_UI4	Double

Slave/nSlvState

Current status of the DP slave with address *n*.

Return values:

OFFLINE

No communication between master and slave

NOT_ACTIVE

DP slave is not activated

READY

DP slave is in the data transfer phase

READY_DIAG

DP slave is in the data transfer phase; diagnostics data also exists

NOT_READY

DP slave is not in the data transfer phase

NOT_READY_DIAG

DP slave is not in the data transfer phase; diagnostics data also exists

OLE data type	Visual Basic type
VT_BSTR	String

Slave/*n*SlvDiag

Last diagnostics data of the DP slave with address *n*. The structure of the diagnostics data is slave-specific and can be found in the documentation of the slave.

OLE data type	Visual Basic type
VT_ARRAY VT_UI1	Byte()

Slave/*n*SlvRestart

Item for restarting DP slaves; in other words, deactivating and immediately reactivating DP slaves with an item. These two internal function calls are executed with an interval in between corresponding to the configured cycle time.

OLE data type	Visual Basic type
VT_BOOL	Boolean

Return values:

True

Restart slave *n*

False

No function

Slave/*n*MiscSlvType

Type of the DP slave with address *n*.

Return values:

NO_SLV

No DP slave

NORM

Standard DP slave

ET200_U

Nonstandard slave: ET 200 U

ET200K_B

Nonstandard slave: ET 200 K/B

ET200_SPM

Non-standard slave: General SPM station

UNDEFINED

Unknown DP slave

OLE data type	Visual Basic type
VT_BSTR	String

Other slave information

The following slave-specific information parameters are not intended for productive communication. If problems occur, it may, nevertheless, be useful to evaluate the variables listed below.

All variables are generally assigned directly to low level DP services. These variables are read in the same way as other information variables. The result of the call is the byte dump of the particular low level DP service.

SlavenMiscReadSlvParCfgData

Configuration data of the slave at address *n* for describing the input and output data areas and the data consistency

SlavenMiscReadSlvParUserData

Application data of slave *n* (part of the parameter file)

SlavenMiscReadSlvParPrmData

Parameter assignment data of slave *n*

SlavenMiscReadSlvParType

SI flag and type of slave *n* (slave-related flags, slave type and other reserved data)

SlavenMiscSlvDiag

Diagnostics data of slave *n*

2.4.1.16 Examples of DP-specific information variables

Here, you will find several examples of return values of DP-specific information variables.

DP master mode

DP:[CP 5623]MasterState

MasterState

can, for example, return the following value:

OPERATE

The DP master is in the productive phase.

Current status of a slave*DP:[CP 5623]Slave3SlvState*

Slave3SlvState

can, for example, return the following value:

READY

The DP slave with address 3 is in the data transfer phase.

Slave type*DP:[CP 5623]Slave3MiscSlvType*

Slave3MiscSlvType

can, for example, return the following value:

ET200_U

Slave 3 is an ET 200 U.

2.4.1.17 Syntax of the systemspecific information variables

`DP: [SYSTEM] &version()`

&version()

Returns a version ID for the DP OPC Server, here, for example, the string
SIMATIC NET Core Server DP V 7.xxxx.yyyy.zzzz Copyright 2012

Data type: VT_BSTR

Access right: readonly

2.4.2 Class 2 DP master**Class 2 DP master**

A class 2 DP master can configure and perform online diagnostic jobs. The essential functions of a class 2 DP master are:

- Master diagnostics: Reading the class -1 master diagnostic data
- Slave diagnostics: Reading the diagnostic data of a slave and the slave configuration
- I/O data: Reading the input and output data of a slave
- Data records: Writing and reading data records for a slave

With the acyclic services for accessing I/O data, you can read the inputs and outputs of the slaves and access and monitor their diagnostic data. The slaves must support the DP-V1 additional functions. You also have access to the data of a class 1 master.

Access is:

- via the PROFIBUS address of the slave
Slave devices cannot initiate communication. They do not receive the token and respond only to polls from the master. Slaves are therefore known as passive nodes. This number is the same as the PROFIBUS address.
- via the PROFIBUS address of the master
Access to the diagnostic data of a class 1 master from a class 2 master.

2.4.2.1 Protocol ID

All items begin with the prefix "DP2".

2.4.2.2 Configured CP name

The configured CP name specifies the communications processor used (CP).

Syntax

`DP2 : [<configuredCPname>]`

Explanation

<configuredCPname>

The configured CP name is specified with SIMATIC STEP 7 / NCM.

Example

`DP2:[CP 5623]`

2.4.2.3 Syntax of the process variables for master diagnostics

Introduction

The following section describes the various forms of syntax with which you can access the diagnostics data of a DP class 1 master.

Syntax

`DP2 : [<configuredCPname>]master<masteraddress>MstDiag`

DP2 items of the bus node

master

Identifier for access to a master using the DP protocol.

<masteraddress>

PROFIBUS station address of the bus node.
Range: 0 to 126

Explanation**DP2 items of the bus node - item for master diagnostics****MstDiag**

Last system diagnostics data of the DP master.

Data type: VT_ARRAY

Access right: readonly

OLE data type: VT_BOOL

Description: 126 elements

Each element in the array signals whether the assigned slave has sent diagnostics data; the array index corresponds to the PROFIBUS address.

Value	Meaning
FALSE	The slave has not sent diagnostics data
TRUE	The slave has sent diagnostics data

Syntax

DP2 : [<configuredCPname>]master<masteraddress>MstState

Explanation**DP2 items of the bus node - master diagnostics items****MstState**

Status of the DP master. Contains the current status and some version information.

Data type: VT_ARRAY

Access right: readonly

OLE data type: VT_UI1

Description: 16 elements

Byte	Description
1	Operating status 0x40 – STOP 0x80 – CLEAR 0xC0 - OPERATE
2 and 3	Ident number
4	Version of the hardware (DDLM/user interface)
5	Version of the software (DDLM/user interface)
6	Version of the hardware

Byte	Description
7	Version of the software
8 to 16	Reserved

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>
DataTransferList
```

Explanation

DP2 items of the bus node - master diagnostics items

DataTransferList

Data transfer list of the DP master.

Data type: VT_ARRAY

Access right: readonly

OLE data type: VT_BOOL

Description: 126 elements

Each element in the array signals whether the assigned slave is in the productive phase and has sent data; the array index corresponds to the PROFIBUS address.

Value	Description
FALSE	Slave is not in the data transfer phase
TRUE	Slave is in the data transfer phase

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiag
```

Explanation

DP2 items of the bus node - master diagnostics items

<address>

PROFIBUS station address of the bus node. The bus node does not necessarily need to be attached to PROFIBUS for the item to be created and used (possibly with an access error).

Range: 0 to 126

Note

The SIMATIC NET OPC Server can only recognize whether the addressed bus node is actually DP-compliant after a slave service has executed successfully. With critical bus nodes that may react incorrectly to such services, the creation of the corresponding items is the responsibility of the user.

SlvDiag

Last diagnostics data of the DP slave stored on the DP master.

Data type: VT_ARRAY

Access right: readonly

OLE data type: VT_UI1

The following extra individual items exist for the specific contents and description of the elements of the *SlvDiag* field:

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagMasterLock
```

This bit is set when the DP slave has already been assigned parameters by another master, in other words, its own master does not currently have access to this slave.

Bit 7 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagPrmFault
```

This bit is set by the DP slave when the last parameter assignment frame for the slave was bad (for example wrong length, wrong Ident number, invalid parameter).

Bit 6 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagInvalidSlaveResponse
```

This bit is set when an implausible response is received from an addressed DP slave.

Bit 5 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagNotSupported
```

This bit is set when a function is requested that is not supported by this slave (for example the SYNC mode is requested but not supported by the slave).

Bit 4 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
      <address>SlvDiagExtDiag
```

This bit is set by the DP slave. When the bit is set, there must be a diagnostics entry in the diagnostics area (Ext_Diag_Data) specific to the slave.

If the bit is not set, there may be a status message in the diagnostics area (Ext_Diag_Data) specific to the slave. The meaning of this status message must be defined for the specific application.

Bit 3 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
      <address>SlvDiagCfgFault
```

This bit is set when the configuration data last sent by the DP master does not match the data detected by the DP slave. This means that there is a configuration error.

Bit 2 of the 1st station status byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
      <address>SlvDiagStationNotReady
```

This bit is set when the DP slave is not yet ready for productive data exchange.

Bit 1 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
      <address>SlvDiagStationNonExistent
```

The DP master sets this bit when the DP slave cannot be reached over the bus. If the bit is set, the diagnostics bits contain the status of the last diagnostics message or the initial value.

The DP slave sets this bit permanently to zero.
Bit 0 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagDeactivated
```

This bit is set when the DP slave is indicated as being inactive in the local parameter set and has been taken out of cyclic processing.

Bit 7 of the 2nd (station status) byte of the *SlvDiag* field.

Bit 6 of the 2nd (station status) byte of the *SlvDiag* field is reserved.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagSyncMode
```

The DP slave sets this bit when this DP slave has received the Sync control command.

Bit 5 of the 2nd (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagFreezeMode
```

The DP slave sets this bit when this DP slave has received the Freeze control command.
Bit 4 of the 2nd (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagWDOn
```

This bit is set by the DP slave. If this bit is set to 1, the watchdog is activated on the DP slave.

Bit 3 of the 2nd (station status) byte of the *SlvDiag* field.

Bit 2 of the 2nd (station status) byte of the *SlvDiag* field is set permanently to 1.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagStatDiag
```

If the DP slave sets this bit, the DP master fetches diagnostics information until the bit is cleared again. The DP slave sets this bit, for example, when it is not able to provide any valid user data.

Bit 1 of the 2nd (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagPrmReq
```

If the DP slave sets this bit, it must be reassigned parameters and reconfigured. The bit remain set until parameters have been reassigned.

Bit 0 of the 2nd (station status) byte of the *SlvDiag* field.

Note

When bit 1 (*SlvDiagStatDiag*) and bit 0 (*SlvDiagPrmReq*) are set, bit 0 has the higher priority.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagExtDiagOverflow
```

When this bit is set, there is more diagnostics information available than indicated in *SlvDiagExtDiagData*. The DP slave sets this bit if there is more channel diagnostics information available than the DP slave can enter in its send buffer. Or the DP master sets this bit when the DP slave sends more diagnostics data than the DP master can handle in its diagnostics buffer.

Bit 7 of the 3rd (station status) byte of the *SlvDiag* field.

Bits 6 to 0 of the 3rd (station status) byte of the *SlvDiag* field are reserved.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>]master<masteraddress>slave
<address>SlvDiagMasterAddr
```

The address of the DP master that assigned the parameters for this DP slave is entered in byte 4 of the *SlvDiag* field (the *Diag.MasterAdd* byte). If the DP slave has not been assigned parameters by a DP master, the DP slave sets the address 255 in this byte.
 Byte 4 (*Diag.MasterAdd* byte) of the *SlvDiag* field.

Data type: VT_UI1

Access right: readonly

Syntax

```
DP2:[<configuredCPname>]master<masteraddress>slave
<address>SlvDiagIdentNumber
```

The manufacturer's ID for a DP slave type is indicated in this word. This ID can be used for testing as well as for precise identification.

5th and 6th (IdentNumber) bytes of the *SlvDiag* field.

Data type: VT_UI2

Access right: readonly

Syntax

```
DP2:[<configuredCPname>]master<masteraddress>slave
<address>SlvDiagExtDiagData
```

The DP slave can enter its specific diagnostics data in this field of bytes, (the *ExtDiagData* bytes) with a length of up to 26 bytes. A block structure with a separate header byte for device- and ID-related diagnostics data is specified.

7th to 32nd (*ExtDiagData*) bytes of the *SlvDiag* field.

Data type: VT_ARRAY

Access right: readonly

OLE data type: VT_UI1

2.4.2.4 Syntax of the process variables for slave diagnostics

Introduction

The following section describes the various forms of syntax with which you can access the diagnostics data of a slave.

Syntax

```
DP2:[<configuredCPname>]slave<address>SlvDiag
```

Explanation**DPMCL2 items of the bus node - slave diagnostics items****SlvDiag**

Last diagnostics data of the DP slave.

Data type: VT_ARRAY

Access right: readonly

OLE data type: VT_UI1

The following extra individual items exist for the specific contents and description of the elements of the *SlvDiag* field:

Syntax

```
DP2 : [<configuredCPname>] slave<address>SlvDiagMasterLock
```

This bit is set when the DP slave has already been assigned parameters by another master, in other words, its own master does not currently have access to this slave.

Bit 7 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>] slave<address>SlvDiagPrmFault
```

This bit is set by the DP slave when the last parameter assignment frame for the slave was bad (for example wrong length, wrong Ident number, invalid parameter).

Bit 6 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>] slave<address>SlvDiagInvalidSlaveResponse
```

This bit is set when an implausible response is received from an addressed DP slave.

Bit 5 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>] slave<address>SlvDiagNotSupported
```

This bit is set when a function is requested that is not supported by this slave (for example the SYNC mode is requested but not supported by the slave).
Bit 4 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>] slave<address>S1vDiagExtDiag
```

This bit is set by the DP slave. When the bit is set, there must be a diagnostics entry in the diagnostics area (*Ext_Diag_Data*) specific to the slave.
If the bit is not set, there may be a status message in the diagnostics area (*Ext_Diag_Data*) specific to the slave. The meaning of this status message must be defined for the specific application.
Bit 3 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: Read-only

Syntax

```
DP2 : [<configuredCPname>] slave<address>S1vDiagCfgFault
```

This bit is set when the configuration data last sent by the DP master does not match the data detected by the DP slave. This means that there is a configuration error.
Bit 2 of the 1st station status byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>] slave<address>S1vDiagStationNotReady
```

This bit is set when the DP slave is not yet ready for productive data exchange.
Bit 1 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>] slave<address>S1vDiagStationNonExistent
```

The DP master sets this bit when the DP slave cannot be reached over the bus. If the bit is set, the diagnostics bits contain the status of the last diagnostics message or the initial value.
The DP slave sets this bit permanently to zero.
Bit 0 of the 1st (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

DP2 : [<configuredCPname>] slave<address>SlvDiagDeactivated

This bit is set when the DP slave is indicated as being inactive in the local parameter set and has been taken out of cyclic processing.

Bit 7 of the 2nd (station status) byte of the *SlvDiag* field.

Bit 6 of the 2nd (station status) byte of the *SlvDiag* field is reserved.

Data type: VT_BOOL

Access right: readonly

Syntax

DP2 : [<configuredCPname>] slave<address>SlvDiagSyncMode

The DP slave sets this bit when this DP slave has received the Sync control command.

Bit 5 of the 2nd (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

DP2 : [<configuredCPname>] slave<address>SlvDiagFreezeMode

The DP slave sets this bit when this DP slave has received the Freeze control command.

Bit 4 of the 2nd (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

DP2 : [<configuredCPname>] slave<address>SlvDiagWDOn

This bit is set by the DP slave. If this bit is set to 1, the watchdog is activated on the DP slave.

Bit 3 of the 2nd (station status) byte of the *SlvDiag* field.

Bit 2 of the 2nd (station status) byte of the *SlvDiag* field is set permanently to 1.

Data type: VT_BOOL

Access right: readonly

Syntax

DP2 : [<configuredCPname>] slave<address>SlvDiagStatDiag

If the DP slave sets this bit, the DP master fetches diagnostics information until the bit is cleared again. The DP slave sets this bit, for example, when it is not able to provide any valid

user data.

Bit 1 of the 2nd (station status) byte of the *SlvDiag* field.

Data type: VT_BOOL

Access right: readonly

Syntax

DP2 : [<configuredCPname>] slave<address>S1vDiagPrmReq

If the DP slave sets this bit, it must be reassigned parameters and reconfigured. The bit remain set until parameters have been reassigned.

Bit 0 of the 2nd (station status) byte of the *SlvDiag* field.

Note

When bit 1 (*SlvDiagStatDiag*) and bit 0 (*SlvDiagPrmReq*) are set, bit 0 has the higher priority.

Data type: VT_BOOL

Access right: readonly

Syntax

DP2 : [<configuredCPname>] slave<address>S1vDiagExtDiagOverflow

When this bit is set, there is more diagnostics information available than indicated in *S1vDiagExtDiagData*. The DP slave sets this bit if there is more channel diagnostics information available than the DP slave can enter in its send buffer. Or the DP master sets this bit when the DP slave sends more diagnostics data than the DP master can handle in its diagnostics buffer.

Bit 7 of the 3rd (station status) byte of the *S1vDiag* field.

Bits 6 to 0 of the 3rd (station status) byte of the *S1vDiag* field are reserved.

Data type: VT_BOOL

Access right: readonly

Syntax

DP2 : [<configuredCPname>] slave<address>S1vDiagMasterAddr

The address of the DP master that assigned the parameters for this DP slave is entered in byte 4 of the *S1vDiag* field (the *Diag.MasterAdd* byte). If the DP slave has not been assigned parameters by a DP master, the DP slave sets the address 255 in this byte.

Byte 4 (*Diag.MasterAdd* byte) of the *S1vDiag* field.

Data type: VT_UI1

Access right: readonly

Syntax

DP2 : [<configuredCPname>] slave<address>S1vDiagIdentNumber

The manufacturer's ID for a DP slave type is indicated in this word. This ID can be used for testing as well as for precise identification.

5th and 6th (IdentNumber) bytes of the *SlvDiag* field.

Data type: VT_UI2

Access right: readonly

Syntax

```
DP2 : [<configuredCPname>] slave<address>SlvDiagExtDiagData
```

The DP slave can enter its specific diagnostics data in this field of bytes, (the *ExtDiagData* bytes) with a length of up to 26 bytes. A block structure with a separate header byte for device- and ID-related diagnostics data is specified.

7th to 32nd (*ExtDiagData*) bytes of the *SlvDiag* field.

Data type: VT_ARRAY

Access right: readonly

OLE data type: VT_UI1

Syntax

```
DP2 : [<configuredCPname>] slave<address>SlvCFGData
```

Explanation

DPMCL2 items of the bus node - slave diagnostics items

SlvCFGData

Configuration data of the DP slave stored on the DP master.

Data type: VT_ARRAY

Access right: readonly

OLE data type: VT_U_I1

Syntax

```
DP2 : [<configuredCPname>] slave<address>SetSlaveAddress
```

Explanation

DPMCL2 items of the bus node - slave diagnostics items

SetSlaveAddress

Sets a new PROFIBUS address for the slave

Data type: VT_ARRAY

Access right: write-only

OLE data type: VT_VARIANT

When writing the item, the corresponding DPMCL2 service is executed. The value to be written also contains an array with the parameters necessary for executing the service.

Array element	Data type	Meaning
1	VT_UI1	New slave address to be assigned
2	VT_BOOL	Flag indicating whether or not the DP slave address can be changed again at a later point in time.
3	VT_I4	Device type (PROFIBUS Ident number) of the node
4	VT_ARRAY VT_UI1	Userspecific data. An empty array can be transferred.

After writing the new slave address successfully, it may be necessary to create new items for the slave. The items with the old slave address (for example DPMCL2_I/O Items) continue to access the slave at the old address and can therefore no longer be read successfully.

2.4.2.5 Syntax of the process variables for I/O data

The English abbreviations I/O (input/output) correspond to the German abbreviations E/A (Eingang/Ausgang).

Syntax

When accessing the inputs in the I/O area of a slave, the slave variable must have the following syntax:

```
DP2:[<configuredCPname>]slave<address>_E{<format>
<offset{.bit}>{,quantity}}
```

or (English):

```
DP2:[<configuredCPname>]slave<address>_I{<format>
<offset{.bit}>{,quantity}}
```

Explanation

DPMCL2 items of the bus node - slave I items

<address>_E or English <address>_I
 <address>_A or English <address>_Q

The inputs and the outputs can only be read by the OPC client of the SIMATIC NET OPC server.

The submodules and the data size (byte or word) as assigned in the defined configuration and the consistency are ignored by the DP class 2 master. This distinguishes it from a DP class 1 master. Although the DP class 2 master can query the configuration of the slave during runtime, the slave and corresponding DP class 1 master can nevertheless change its configuration during runtime without informing the DP class 2 master. The result of this is inconsistent configuration data even if timeconsuming polling of the configuration data is implemented (not forgetting the considerable load on the network).

An I/O area is accessed by specifying the slave number <address> and the I/O area, where _/ (or _E, German) identifies the input area. If the optional format information is omitted, the data will be returned as an array of bytes with the entire length of the inputs.

Data type: VT_ARRAY

Access right: readonly

OLE data type: VT_UI1

<format>

The *format* element specifies the format in which the data will be delivered. If no format is specified, the *byte* format is used. Specifying the format also specifies the data type.

Format identifier	Description	OLE data type	Visual Basic data type
X	Bit	VT_BOOL	Boolean
BYTE or B	Byte (unsigned 8)	VT_UI1	Byte
CHAR	Character (signed 8)	VT_I1	Integer
WORD or W	Word (unsigned 16)	VT_UI2	Long
INT	Integer (signed 16)	VT_I2	Integer
DWORD or D	Double word (unsigned 32)	VT_UI4	Double
DINT	Double integer (unsigned 32)	VT_I4	Long
REAL	Floating point number	VT_R4	Single

<offset{.bit}>

Offset in bytes of the element to be accessed (offset addressing). A bit can only be specified with the type X . The offset is specified in bytes.

Example: X2.3 returns the 3rd bit of the 2nd byte.

<quantity>

Number of elements. The data type of the variable is an array with elements (data type VT_ARRAY) of the specified format. It is not possible to specify the number for the format X . If this part of the name is omitted, or the number 1 is specified, the number 1 is assumed and the data type of the variable is not an array.

Syntax

When accessing the outputs in the I/O area of a slave, the slave variable must have the following syntax:

```
DP2:[<configuredCPname>]slave<address>_A{<format>
<offset{.bit}>{,quantity}}
```

or (English):

```
DP2:[<configuredCPname>]slave<address>_Q{<format>
<offset{.bit}>{,quantity}}
```

Explanation**DPMCL2 items of the bus node - slave Q items**

An I/O area is accessed by specifying the slave number <address> and the I/O area, where _I/Q (or _A, German) identifies the output area. If the optional format information is omitted, the data will be returned as an array of bytes with the entire length of the outputs.

Data type: VT_ARRAY

Access right: Read-only

OLE data type: VT_UI1

Formatting is as for the I items of the slave.

2.4.2.6 Syntax of the process variables for data records

Syntax

```
DP2:[<configuredCPname>]slave<address>S<slot>data<index>
{,<length>}{},<subarea>
where subarea = <subarea> = <format><offset>{.<bit>}{},<quantity>
```

DP2 items of the bus node**slave**

Identifier for access to a slave over the DP protocol.

address

PROFIBUS station address of the bus node. The bus node does not necessarily need to be attached to PROFIBUS for the item to be created and used (possibly with an access error).

Range: 0 to 126

Note

The SIMATIC NET OPC Server can only recognize whether the addressed bus node is actually DP-compliant after a slave service has executed successfully. With critical bus nodes that may react incorrectly to such services, the creation of the corresponding items is the responsibility of the user.

Explanation**DPC2 items of the bus node - DPC2 Read/Write****S**

Identifier for the slot of the slave; typically a submodule.

<slot>

Slot in the extended memory area of a slave. The slot and index identify a data record.

Range: 0 to 255 (on the C interface: 0 to 255)

data

Identifier for a data record.

<index>

Index of the data in the slave slot.

Range: 0 to 255

<length>

Length of the record.

Range: 1 to 240

<subarea>

Identifier for a subarea.

<format>

The *format* element specifies the format in which the data will be delivered. Specifying the format also specifies the data type.

Format identifier	Description	OLE data type	Visual Basic data type
X	Bit	VT_BOOL	Boolean
BYTE or B	Byte (unsigned 8)	VT_UI1	Byte
CHAR	Character (signed 8)	VT_I1	Integer
WORD or W	Word (unsigned 16)	VT_UI2	Long
INT	Integer (signed 16)	VT_I2	Integer
DWORD or D	Double word (unsigned 32)	VT_UI4	Double
DINT	Double integer (unsigned 32)	VT_I4	Long
REAL	Floating point number	VT_R4	Single

<offset{.bit}>

Offset in bytes of the element to be accessed. A bit can only be specified with the type X .

Example: X2.3 returns the 3rd bit of the 2nd byte.

<quantity>

Number of elements (not permitted with X format). The data type of the variable is an array with elements (data type VT_ARRAY) of the specified format. If this part of the name is omitted, or the number 1 is specified, the number 1 is assumed and the data type of the variable is not an array.

Value of the data record

Return value of the data record item.

The data type is VT_ARRAY | VT_UI1 if no subarea is specified.

Read-only; if the data record length is specified, it can also be written.

Note

The data length resulting from the parameters quantity and format must not exceed the size of the data record on the slave. The size of a data record depends on the particular slave and cannot be checked by the OPC server.

When a subarea of the data record is read, the partner device first reads the entire data record and then evaluates the relevant subarea.

When a subarea is written, the entire data record is also transferred to the partner device. If several subareas of the data record are written in a write job, the data record is only written to the partner device after all the subareas of the data record have been updated.

When using the OPC server, it makes sense to put all the items with partial access to a particular data record together in one group and to write the entire group.

Note**Writing data records to the partner device**

When writing data records, make sure that you avoid overlaps and gaps since it is not possible to predict which value will be written.

Note

A written value of an OPC DPC2 data record item cannot be read out identically using the same item or an item with the same address, slot and index. The written and read values of DPC2 data records may differ depending on the specific device.

Syntax

```
DP2:[<configuredCPname>]slave<address>DTS<slot>data
<index>{,<length>}{},<subarea>
where <subarea> = <format><offset>{.<bit>}{},<quantity>}
```

Explanation**DPC2 items of the bus node - DPC2 data transport**

The DPC2 data transport items are read/write if the length is specified (otherwise read-only). If no subarea is specified, the data type is as follows:

Data type: VT_ARRAY

OLE data type: VT_UI1

This item implements a data exchange in the protocol. Read access of these items is local and returns the data record obtained with last write access.

DTS

Identifier for data transport.

<slot>

Slot in the extended memory area of a slave. The slot and index identify a data record.
Range: 0 to 255

<offset>{.<bit>}

Offset in bytes of the element to be accessed. A bit can only be specified with the type X .
Example: X2.3 returns the 3rd bit of the 2nd byte.

<quantity>

Number of elements (not permitted with X format). The data type of the variable is an array with elements (data type: VT_ARRAY) of the specified format. If this part of the name is omitted, or the number 1 is specified, the number 1 is assumed and the data type of the variable is not an array.

Note

When a subarea of the data record is read, the subarea of the last data record received from the partner device is evaluated.

When a subarea is written, the entire data record is also transferred to the partner device. If several subareas of the data record are written in a write job, the data record is only written to the partner device after all the subareas of the data record have been updated.

When using the OPC server, it makes sense to put all the items with partial access to a particular data record together in one group and to write the entire group.

Note**Writing data records to the partner device**

When writing data records, make sure that you avoid overlaps and gaps since it is not possible to predict which value will be written.

2.4.2.7 Examples of process variables for data records

Here you will find examples illustrating the syntax of variable names for DPC2 data records.

Variable names for DPC2

DP2:[CP 5623]Slave005S003Data2,120,DWORD7

Access to the double word from offset 7 in a data record with a length of 120 bytes in slot 3, index 2 of slave 5.

DP2:[CP 5623]Slave005S003Data2,120,B8,4

Access to an array with 4 bytes from offset 8 in a data record with a length of 120 bytes in slot 3, index 2 of slave 5.

DP2:[CP 5623]Slave005DTS006Data2,10,DWORD2

Data transport access to the double word from offset 2 in a data record with a length of 10 bytes in slot 6, index 2 of slave 5.

2.4.2.8 Syntax of the DP2-specific information variables

Syntax

```
DP2 : [<configuredCPname>] &identify()
```

Explanation

&identify()

Returns the station identifier of the specified device as an array with 4 strings:

Data type: VT_ARRAY

Description: 4 array elements (0-3)

Access right: readonly

OLE data type: VT_BSTR

Elements of the return value:

- Vendor
- Controller
- Hardware version
- Software version

Example: {Siemens AG|FW-CP 5613A2 EL (E2)|1.0|V 6.2.1.3175 20.08.2004}

OLE data type Visual Basic type

VT_ARRAY | VT_BSTR String()

Syntax

```
DP2 : [<configuredCPname>] lifelist()
```

Explanation

lifelist()

General items

Returns a list with all the connected bus nodes.

Data type: VT_ARRAY

Description: 127 array elements (0-126)

Access right: readonly

OLE data type: VT_UI1

Each array element stands for a PROFIBUS station address. The values of the array elements have the following meaning:

Element	Meaning
0x00	STATION_PASSIVE
0x10	STATION_NON_EXISTENT
0x20	STATION_ACTIVE_READY (ready to enter the logical ring)
0x30	STATION_ACTIVE (already in the logical ring)

Note

When using the lifelist as an OPC item, for example DP2:[CP5614A2]lifelist(), remember the following:

The "Lifelist" FDL service is a service that can be called by only one application. If it is set as an OPC item, it is no longer available for other applications. In particular, the lifelist can no longer be displayed in the "Communication Settings" configuration program.

Since the lifelist call also causes a not insignificant load on the bus that occurs cyclically with OPC, it should, where possible, be avoided.

2.4.2.9 Syntax of the systemspecific information variables

DP2:[SYSTEM]&version()

&version()

Returns a version ID for the DP2 OPC Server, here, for example, the string
SIMATIC NET Core Server DP2 V 7.xxxx.yyyy.zzzz Copyright 2012

Data type: VT_BSTR

Access right: readonly

2.4.3 DP slave

Note

A PROFIBUS communications processor as DP slave can only be operated via OPC as a DP-V0 slave. With the HARDNET modules named, no data records can therefore be read or written via OPC.

2.4.3.1 Variable services for access to local slave data

The OPC Server for SIMATIC NET can operate in a PROFIBUS DP network not only as a DP master but also in the role of DP slave. The OPC server manages memory areas for the inputs and outputs of this DP slave and maps them to OPC variables. An OPC client can read the outputs set by the master and set values in the inputs that will be fetched by the master in the next cycle.

DP slaves have a modular structure. A DP slave can include several submodules with different input/output areas. The submodules are assigned during configuration.

The variable name identifies an input or output area in the submodule of a slave. The inputs and outputs of the slave are accessed by specifying the submodule number and the input or output area.

Status information of the slaves and the DP master can also be queried using variable names.

Note

Parallel operation of the DP master and DP slave with the CP 5614, CP 5614 A2 or CP 5614 FO module is possible only when the "DP Base" mode was set during configuration. As an alternative, you can also use the CP 5624.

2.4.3.2 Syntax of the process variables for the DP slave

Syntax

Inputs:

```
DP: [<connectionname>] slave{M<number>}_I{<format><offset>
{.<bit>} {,<quantity>}}
```

Outputs:

```
DP: [<connectionname>] slave{M<number>}_Q{<format><offset>
{.<bit>} {,<quantity>}}
```

Explanations

DP

Protocol for access to the process variable.

<connectionname>

Name of the communications module specified in the configuration.

slave

Identifier for access to a slave over the DP protocol.

M

Identifier for the number of the submodule.

<number>

Number of the submodule containing the input or output area.

I

Identifier for an input.

Q

Identifier for an output.

<format>

Format in which the data is delivered.

Specifying the format also specifies the data type.

All specified OLE data types can be read via the automation interface of OPC. However, certain development tools (for example Visual Basic) only offer a restricted number of data types. The following table therefore lists the corresponding Visual Basic type in which the variable value can be represented.

Format identifier	Description	OLE data type	Visual Basic type
X	Bit	VT_BOOL	Boolean
BYTE or B	Byte (unsigned 8)	VT_UI1	Byte
CHAR	Character (signed 8)	VT_I1	Integer
WORD or W	Word (unsigned 16)	VT_UI2	Long
INT	Integer (signed 16)	VT_I2	Integer
DWORD or D	Double word (unsigned 32)	VT_UI4	Double
DINT	Double integer (unsigned 32)	VT_I4	Long
REAL	Floating point number	VT_R4	Single

<offset>

Byte address in the address space of the slave at which the element to be addressed is located.

<bit>

Bit number in the addressed byte. The range is between 0 and 7.

A bit can only be specified with the format identifier X.

<quantity>

Number of elements.

The data type (VT_ARRAY) of the variable is an array with elements of the specified format.

If quantity is omitted, the quantity 1 is assumed and the data type of the variable is not an array.

Do not use the quantity parameter with the Xformat identifier.

2.4.3.3 Examples of process variables for the DP slave

Here you will find examples illustrating the syntax of variable names for DP slave variables.

Inputs

DP:[CP 5611]SlaveM003_IB0
SlaveM003_IB0

Input byte 0 (offset 0) in submodule 3 of the DP slave.

DP:[CP 5611]SlaveM003_IB1,3

SlaveM003_IB1,3

An array with 3 bytes starting at input byte 1 (offset 1) in submodule 3 of the DP slave.

DP:[CP 5624]SlaveM003_IX0.0

SlaveM003_IX0.0

Input bit 0 in byte 0 in submodule 3 of the DP slave.

DP:[CP 5614]SlaveM003_IB3,8

SlaveM003_IB3,8

An array with 8 input bytes starting at offset 3 in submodule 3 of the DP slave.

Outputs

DP:[CP 5611]SlaveM003_QW3

SlaveM004_QW3

An output word at address 3 in submodule 4 of the DP slave.

DP:[CP 5611]SlaveM003_QDWORD2

SlaveM003_QDWORD2

An output double word at address 2 in submodule 3 of the DP slave.

DP:[CP 5624]SlaveM003_QX3.7

Slave_QX3.7

An output bit 7 in byte 3 of the DP slave.

DP:[CP 5624]SlaveM001_QW0,4

SlaveM001_QW0,4

An array with 4 output words in submodule 1 of the DP slave.

2.4.3.4 DP slavespecific information variables

For DP slave diagnostics, there are several predefined information variables.

2.4.3.5 Syntax of the DP slave-specific information variables

Syntax

You have two options:

DP: [<connectionname>]<diagnosticitem>

DP: [<connectionname>]<parameteritem>

Explanations

DP

Protocol for access to the process variable.

<connectionname>

Name specified in the configuration of the communications module.

<diagnosticitem>

Predefined item.

You have the following option:

devicestate

State of the module on which the DP slave is located.

The following states are possible:

- ONLINE
- OFFLINE

<parameteritem>

Predefined parameter items.

The available options are as follows:

- *SlaveMiscReadSlvParCfgData*
Configuration data of the slave
- *SlaveSlvState*
State of the slave.

The following states are possible:

- DATA_EXCHANGE
- NO_DATA_EXCHANGE

2.5 PROFIBUS DP with OPC UA

Process variables for the DP master with OPC UA

The OPC server of SIMATIC NET for the DP master mode via OPC UA provides process variables for the following services:

- Services for a class 1 master
Access and monitoring of DP inputs and outputs
- Sync / Freeze
Acyclic sending of control frames to slave groups
- Fast Logic for
 - the CP 5613 A2 and CP 5614 A2 (DP master only)
Automatic monitoring of slave data
 - the CP 5623 and CP 5624 (DP master only)
Automatic monitoring of slave data
- Diagnostics variables
Evaluation of static diagnostics information

Process variables for the DP slave

The OPC server of SIMATIC NET for the DP slave mode via OPC UA provides process variables for the following services:

- Variable services for access to the local slave data
 - Access to the inputs and outputs of the slave
- Diagnostics variables
 - Evaluation of static diagnostics information of the slave

2.5.1 SIMATIC NET OPC UA server for the DP protocol

Introduction

The following section describes a configuration variant for the DP protocol that also supports OPC UA alongside the DP COM OPC Data Access server. To do this, the DP COM OPC Data Access server needs to be set up as an outproc OPC server.

Since the PROFIBUS DP protocol maintains an image of the input and output data in the DP RAM of the communications processor in the PC, access to process data takes place only locally within the PC. Particularly when using the SIMATIC NET modules CP 5613 A2, CP 5613 A3, CP 5614 A2, CP 5614 A3, CP 5623 and CP 5624 over the DP Base interface, this makes access extremely fast.

In some situations, for example, when using PCbased controllers, very short times for access to process data are necessary.

Configuration

The DP OPC UA server is activated by selecting "DP" and "OPC UA" in the "Communication Settings" configuration program in "OPC protocol selection":

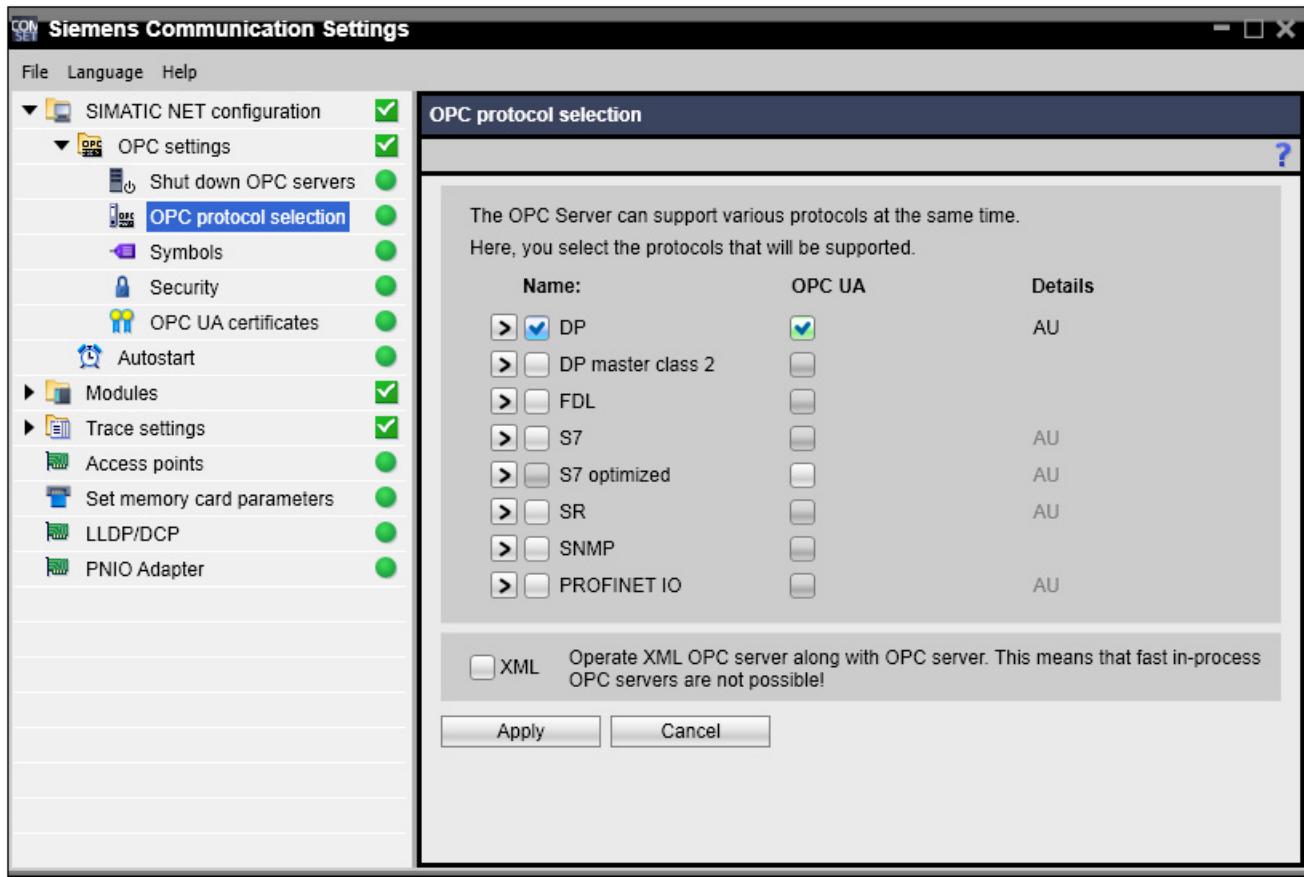


Figure 2-3 Window in the "Communication Settings" configuration program for selecting OPC UA for the DP protocol

Advantages / disadvantages

When using the DP OPC UA server, only the outproc mode of the DP OPC server is possible. The DP OPC UA server process must be started to maintain UA readiness to receive. Even after all OPC UA clients have logged off, the DP OPC UA server is not exited. If the DP OPC UA server process is stopped, the UA functionality is no longer available.

Compared with the DP COM server, the advantages are as follows:

- COM/DCOM configuration is no longer necessary.
- High-speed, secure communication

2.5.2 Support of DP services with OPC UA

Class 1 DP master

The DP OPC UA server supports DP master class 1. The DP master class 1 handles the cyclic communication with the DP slaves. Communication includes central functions such as:

- Configuring and assigning parameter values to the slaves
- Cyclic data transfer with the DP slaves
- Monitoring the DP slaves
- Providing diagnostics information

With DPC1, a master operating cyclically can also handle acyclic data traffic.

Overview of the DP master class 1 in OPC UA

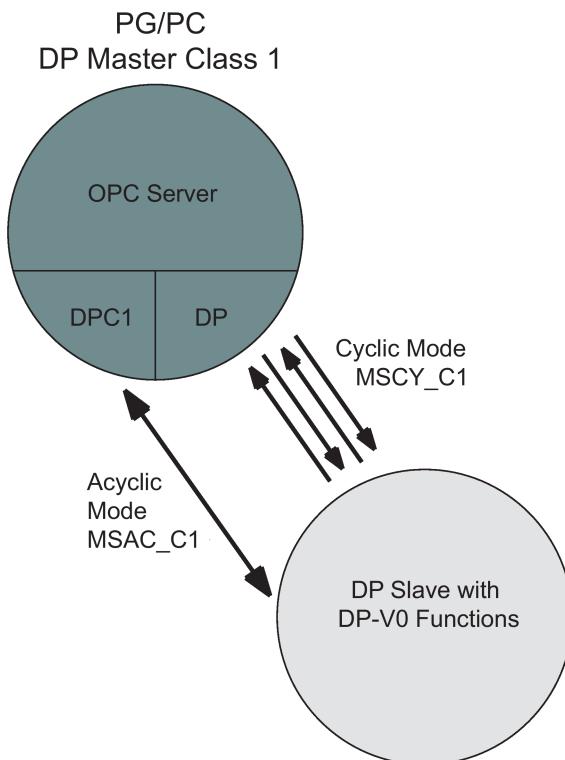


Figure 2-4 The protocol sections of DP master class 1

MSCY_C1	Master-slave, cyclic C1 mode
MSAC_C1	Master-slave, acyclic C1 mode

DP slave (DP-V0)

The DP OPC UA server supports the DP slave function DP-V0. Communication includes central functions such as:

- Parameter assignment and configuration by the DP master class 1
- Cyclic data transfer to the DP master class 1
- Monitoring by the DP master class 1
- Provision of diagnostics information by the DP master class 1

With DPC1, a master operating cyclically can also handle acyclic data traffic.

Overview of the DP OPC UA slave (V0)

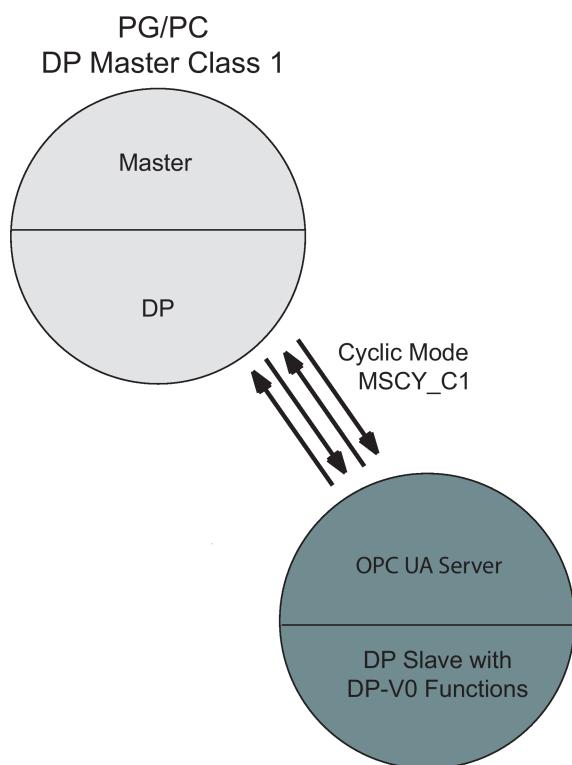


Figure 2-5 The protocol sections of DP slave (V0)

MSCY_C1 Master-slave, cyclic C1 mode

MSAC_C1 Master-slave, acyclic C1 mode

2.5.3 How is the DP OPC UA server addressed?

Server URL

For the native binary TCP protocol, there are two ways for the OPC client to address the server:

- Direct addressing:
 - `opc.tcp://<hostname>:55103`
or
 - `opc.tcp://<IP-Adresse>:55103`
or
 - `opc.tcp://localhost:55103`

The DP OPC UA server uses port 55103.

- The Discovery service is a tool for listing all available server URLs. For this, use the following server URL:
 - `opc.tcp://<hostname>:4840`
or
 - `opc.tcp://<IP-Adresse>:4840`
or
 - `http://<hostname>:52601/UADiscovery/`
or
 - `http://<IP-Adresse>:52601/UADiscovery/`

The Discovery server has port 4840 (for TCP connections) and port 52601 (for HTTP connections).

IPv6 address

The data exchange between the OPC client application and the OPC server can also be implemented using IPv6 addressing. The address must be in parentheses, for example `[fe80:e499:b710:5975:73d8:14]`

Endpoints and security modes

The SIMATIC NET DP OPC UA server supports communication using the TCP protocol secured by encryption and signature. These secure modes should be preferred in productive operation.

The Discovery service on the addressed host signals the endpoints of the servers, in other words, their security requirements and protocol support.

The server URL "opc.tcp://<hostname>:55103" of the DP OPC UA server provides the following endpoints:

- Endpoint in "SignAndEncrypt" security mode:

A signature and encryption are required to communicate with the server. Communication is protected by exchanging certificates and entering a password.

In addition to the security mode, the security policy Basic128Rsa15 is also displayed.

- Endpoint in "None" security mode:

In this mode, no security functions are required by the server (security policy "None") and this is suitable for testing and commissioning.

For more detailed information on the security functions, refer to the section "Programming the OPC UA interface (Page 496)".

The security policies "Basic128Rsa18" and "None" are in the UA specification of the OPC Foundation at the following Internet address:

[> "Specifications" > "Part 7"](http://opcfoundation.org/UA)

You will find more detailed information on the following Internet page:

OPC Foundation ([> "Security Category" > "Facets" > "Security Policy"](http://www.opcfoundation.org/profilereporting/index.htm))

The OPC UA Discovery of the OPC Scout V10

The OPC Scout V10 allows you to use the Discovery service to enter UA endpoints (server URL) in the navigation area of the OPC Scout V10.

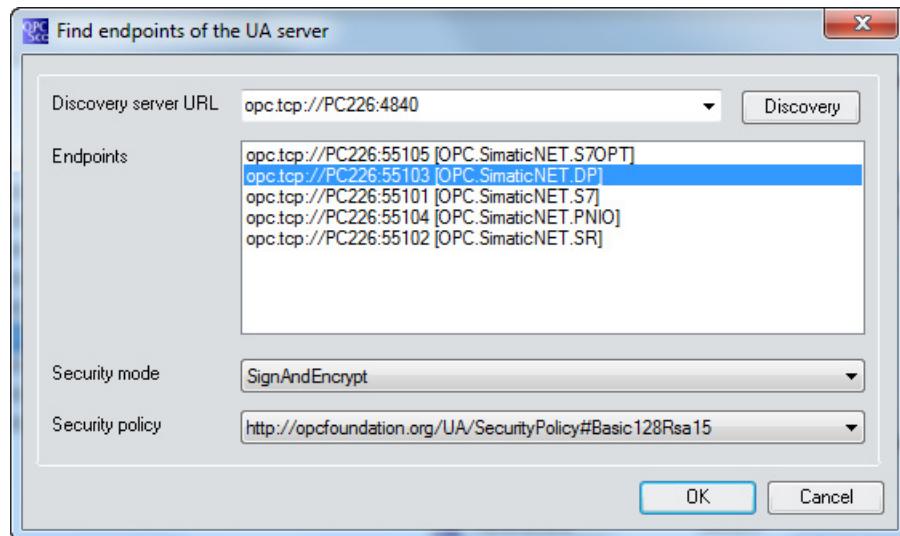


Figure 2-6 The "Find endpoints of the UA server" dialog of OPC Scout V10

All available OPC UA servers (including the DP OPC UA server) can be listed using the OPC UA Discovery service.

The OPC Scout V10 recognizes all the OPC UA endpoints supported by SIMATIC NET. The Discovery service on the addressed host then signals the registered DP OPC UA servers for these endpoints and their ports and security modes.

For more detailed information, refer to the online help of OPC Scout V10.

2.5.4 Which namespaces does the DP OPC UA server provide?

The DP OPC UA server provides the following namespaces:

Table 2- 1 The namespaces of DP OPC UA:

Namespace index	"Identifier" (namespace URI) / comment
0	"http://opcfoundation.org/UA/" specified by the OPC Foundation
1	"urn:Siemens.Automation.SimaticNET.DP:(GUID)" Unique identifier of the DP OPC UA server.
2	"DPTYPES:" Definitions for DP-specific object types.
3	"DP:" Identifier of the DP OPC UA server with new simplified syntax (browsable and can be used with OPC UA)
4	"DPCOM:" Identifier of the server with the old syntax, DP OPC DA-compatible (can be used with OPC UA but cannot be browsed)

The namespace indexes 0 and 1 are reserved and their significance is specified by the OPC Foundation.

The assignment of the remaining namespace indexes to the identifiers (namespace URI) must be obtained via the data variable "NamespaceArray" at the beginning of an OPC UA session by the client specifying the identifier. The identifiers "DPTYPES:", "DP:" and "DPCOM" always exist with the DP OPC UA server.

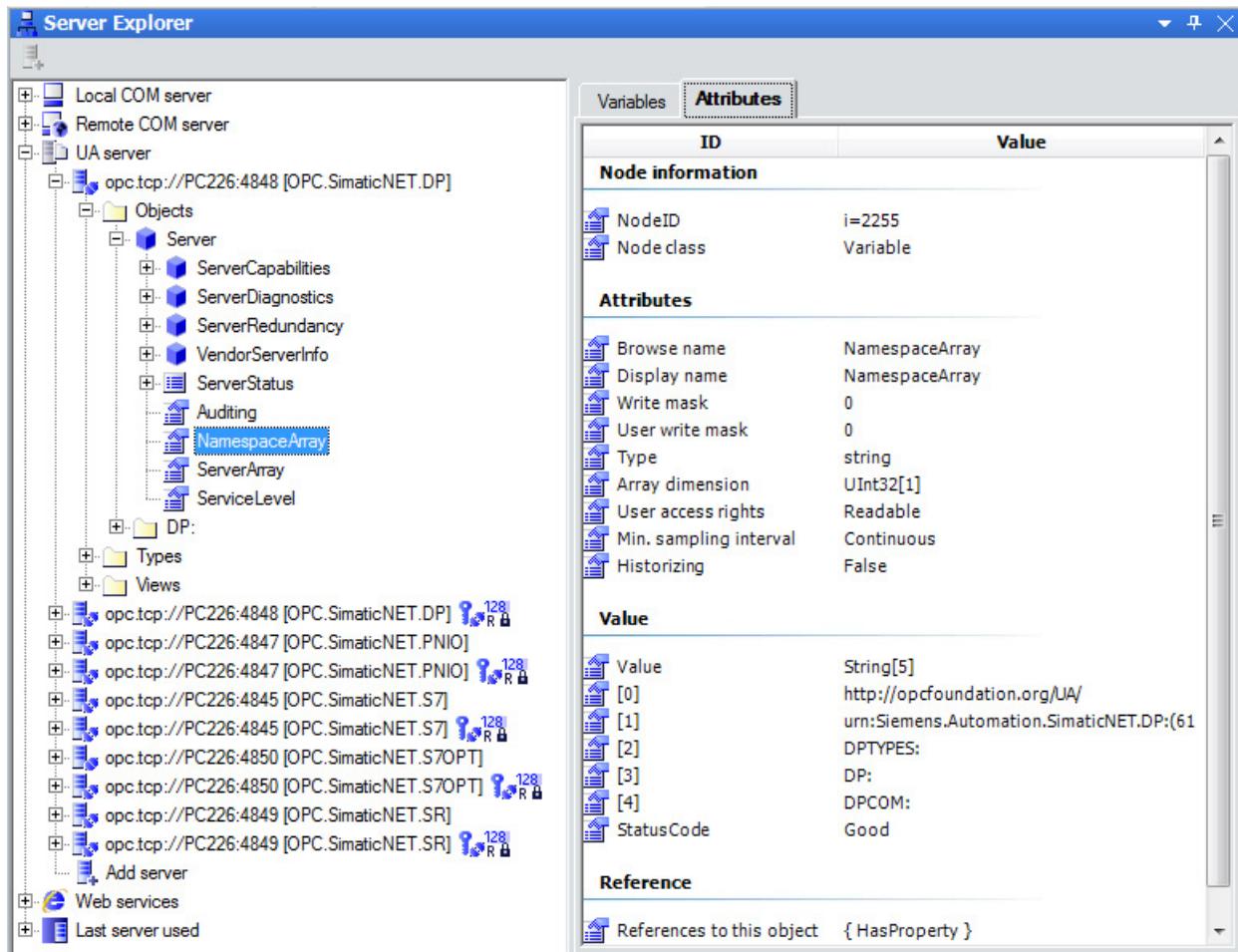


Figure 2-7 Display of the DP OPC UA namespaces using the browse function of the OPC Scout V10

2.5.5 The Nodeld

Identification of a DP process variable

The Nodeld uniquely identifies a DP process variable during runtime with the aid of the following tuple:

- Namespace index
- Identifier (character string, numeric value)

Examples

- Nodeld:
 - Namespace URI:
DP:
(= namespace index 3) for Siemens.Automation.SimaticNET.DP
 - Identifier:
DPMasterName.slv17.q0.244

- Nodeld:
 - Namespace URI:
DPCOM:
(= namespace index 4) for OPC.SimaticNET; the syntax is DP OPC DA-compatible
 - Identifier:
DPMasterName.slv17.q0.244

How does the new namespace adapted to OPC UA behave?

The world of the OPC Data Access items of a COM server for reading and writing process variables is self-contained. Alongside it but independent, there is the alarm world.

On the other hand, the OPC UA view of automation objects is also related to various properties of the objects. OPC UA no longer accesses items alone, but also objects and their subobjects.

- Slave variables and methods are, for example, subobjects of a DP master object. Attributes and properties define the objects in greater detail.
- An OPC Data Access item for slave access corresponds to an OPC UA data variable.
- An OPC Data Access item for Fast Logic corresponds to an OPC UA data method.

The qualified identifiers of the Nodelds have a greater significance in OPC UA than in OPC Data Access. Each individual access to an object, subobject, property and attribute uses its Nodeld.

OPC UA provides the display name among other things to support local languages. This means that the same objects, for example in different language environments specified by the OPC UA client, can be browsed differently although the same Nodeld is presented every time. Selection of the display name is analogous to the relevant Nodeld. The texts of the entire namespace are in English.

Syntax of the DP OPC UA data objects

OPC UA defines an optimized syntax for access to the individual objects. The Nodelds of all OPC UA objects have the following structure:

<boardobject>. <object>". "<subobject>". "<property>

A subobject can contain further subobjects.

Access to a Nodeld that cannot be interpreted is rejected with an error. The letters "A-Z" are not case-sensitive for any items.

Symbolic object representation

The OPC UA specification recommends a uniform symbolic representation for the hierarchical description of the address space. The following symbols are used in this document:

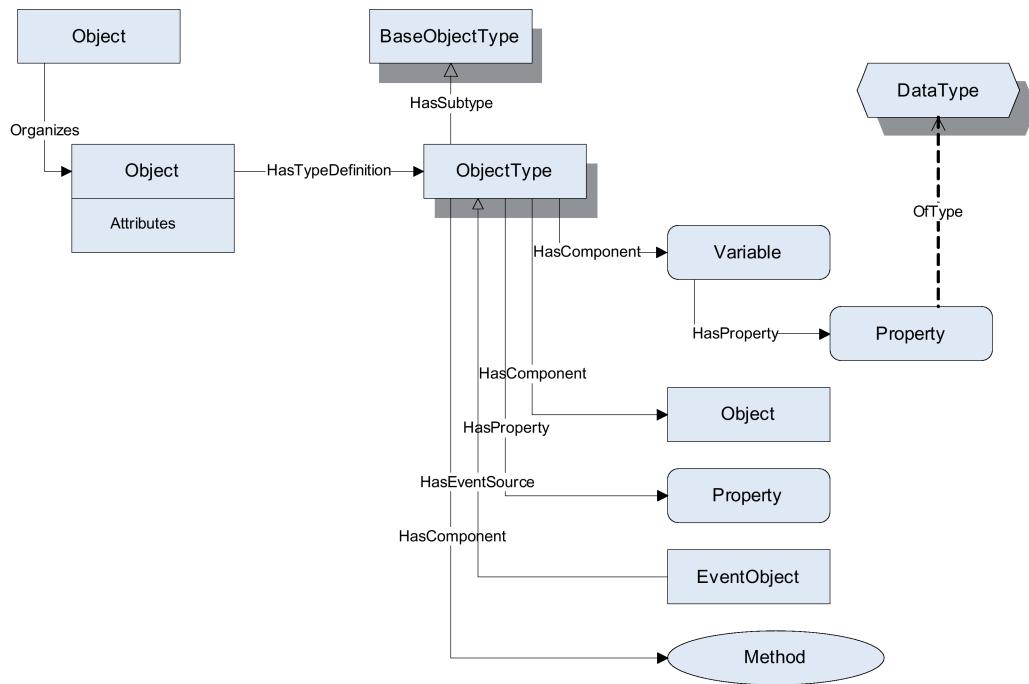


Figure 2-8 Symbols of the OPC UA address space

2.5.6 Board objects for DP modules

2.5.6.1 Overview of board objects for DP modules

DP board types

All protocol-specific objects are always assigned to a board, with DP this is normally the DP master module or the DP slave module. Exceptions here are the so-called system board and the DP demo module.

- DP master module
DP master modules are used for the data exchange with DP slave modules and are generally configured using STEP 7.
- DP slave module
DP slave modules are used for the data exchange with DP master modules and are generally configured using STEP 7.
- The demo module
The demo module with the name "DEMO" is used to simulate a DP master with several simple slaves.

Demo module

<boardobject>:= "DEMO"

With the demo module with the name "DEMO", there are objects that have a namespace similar to that of the DP master modules. The demo module is intended to familiarize you with the SIMATIC NET OPC systems and can be activated via the configuration.

<boardobject>:= "DEMO_S"

With the demo module with the name "DEMO_S", there are objects that have a namespace similar to that of the DP slave modules. The demo module is intended to familiarize you with the SIMATIC NET OPC systems and can be activated via the configuration.

Note

The demo modules with the names "DEMO" and "DEMO_S" must not be used at the same time with a DP master module or DP slave module with the same name. A DP master module or DP slave module configured with this name is ignored if a demo module is added to the configuration.

What are DP board objects?

All productive protocol-specific objects are always assigned to a board. In DP, these are the DP master modules or DP slave modules (board). Exceptions to this are the demo board and the DEMO_S board.

2.5.6.2 Board names

The board name of a DP module

The board name is the DP name configured in STEP 7 or the TIA Portal to identify the module. In STEP 7, this name is known as the "local ID". The local ID is unique within the OPC server.

Board types

The OPC server supports the following board types:

- DP master module
- DP slave module

What characters are permitted for DP board names?

For the <boardname>, you can use numbers "0-9", upper and lower case alphabetic characters "A-z" and the special characters "_-+()". The board name can be a maximum of 24 characters long. The names are not case-sensitive.

Note

In SIMATIC NET the module names are typically used as in the example below.

The board names "SYSTEM", "DEMO" and "DEMO_S" are reserved and must not be used.

Examples of board names

Typical examples are:

- CP5613A2
- CP5624Slave
- DPMaster

2.5.6.3 Type definition of the DP master board object

Type definition of the DP master board object

For the objects and functionalities that can be used via a productive DP master board, a specific OPC UA object type is defined:

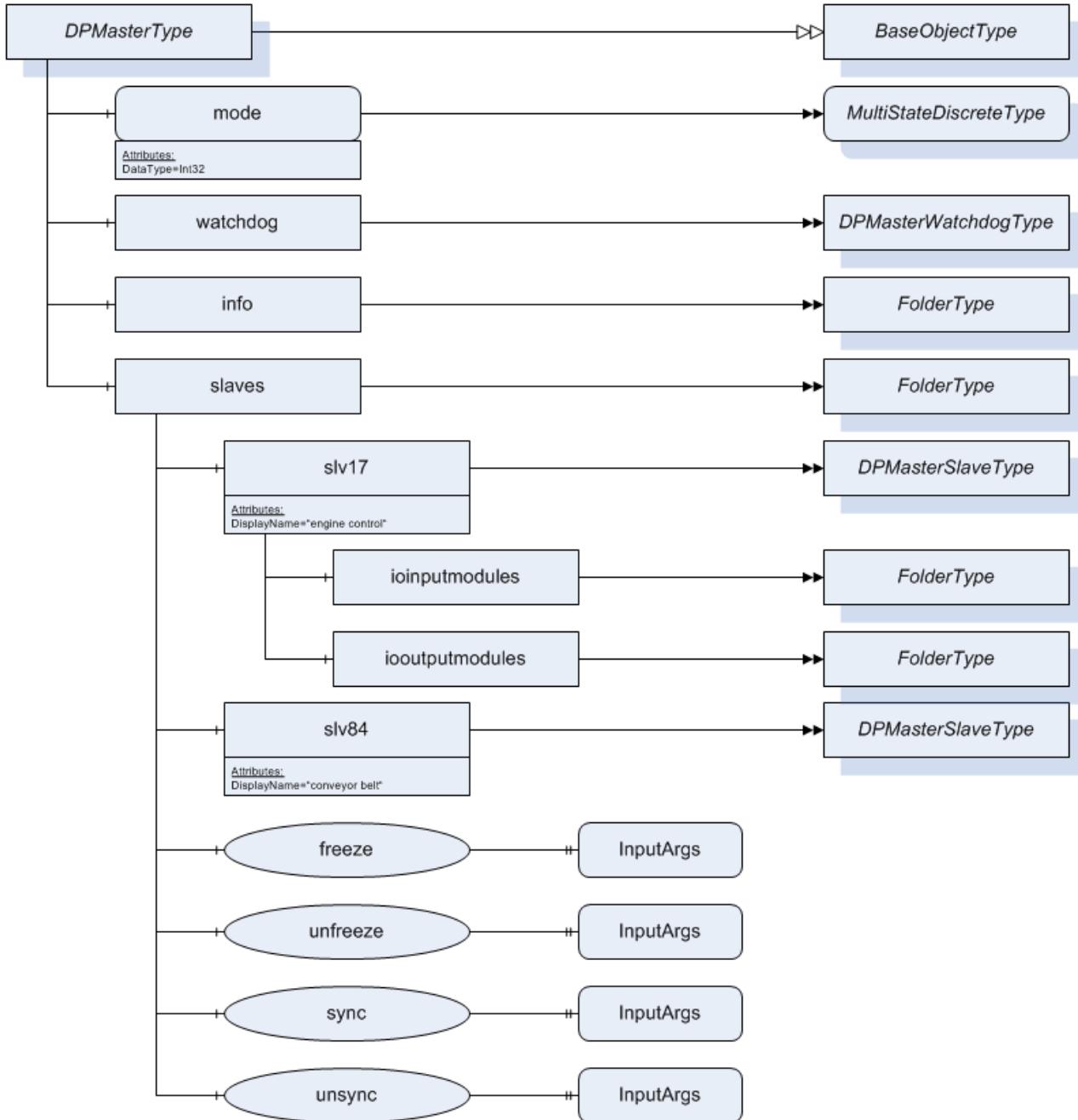


Figure 2-9 The type of the DP master board object in the namespace of OPC UA

Instances of this type are displayed in the OPC UA namespace for objects. The type itself can be read out structured under "Types".

2.5.6.4 Type definition of the DP slave board object

Type definition of the DP slave board object

For the objects and functionalities that can be used via a productive DP slave board, a specific OPC UA object type is defined:

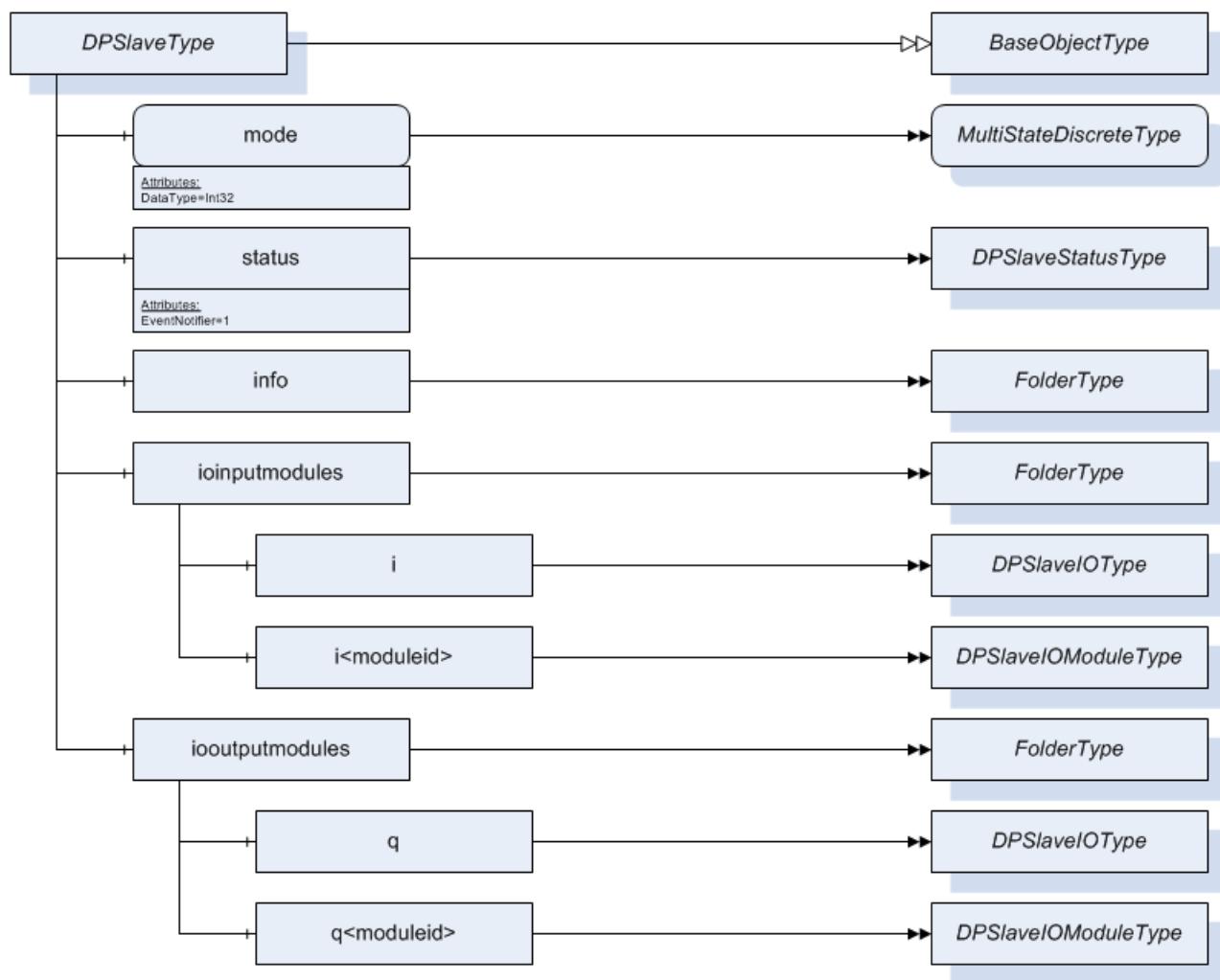


Figure 2-10 The type of the DP slave board object in the namespace of OPC UA

Instances of this type are displayed in the OPC UA namespace for objects. The type itself can be read out structured under "Types".

2.5.7 DP master class 1

2.5.7.1 Process variables for services of the DP master class 1

With the services for accessing cyclic data, you can access the inputs and outputs of the slaves and monitor and control them.

Access is via:

- Slave number
This number corresponds to the PROFIBUS address.
- Module number
DP slaves can include several submodules with different input/output areas.
- Input/output area

2.5.7.2 Syntax of the process variables for the master class 1

Syntax of the process variables

Optimized syntax of the process variables of the DP OPC UA NodeID for the cyclic reading and writing of IO data variables:

Namespace URI: DP: (Namespace index: 3)

Syntax

The available options are as follows:

- <DPmaster>.<slave>.i{<number>} { .<offset>, <DPtype>{ , <quantity> } }
- <DPmaster>.<slave>.q{<number>} { .<offset>, <DPtype>{ , <quantity> } }

Explanations

<DPmaster>

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

<slave>

Symbolic slave name as identifier for access to a DP slave. The symbolic slave name can be assigned in STEP 7 / NCM PC (must be configured with OPC server V8.2 or higher).

Note

If an OPC server < V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier "slv" followed by the configured slave address is used as the symbolic slave name.

If an OPC server >= V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier of the slave is used as the slave name. The consistency check in STEP 7 / NCM PC ensures that unique slave names are used and that the slave name is not empty.

Note**Permitted characters for slave names**

The following characters are permitted in the character set for slave names:

A-Z, a-z, 0-9, _, -, ^, !, #, \$, %, ', (,), =, ~, +, ',', @, {, }

The following characters are not permitted in the character set for slave names:

- Period ".."
- Colon ":"
- Pipe symbol "|"
- Backslash "\\"
- Square brackets "[" and "]"
- Quotation marks """"
- Ampersand "&"
- Question mark "?"
- Angle brackets "<" and ">"
- Asterisk "**"

The space cannot be used at the start or end of a slave name.

q

Identifier for an output. Outputs can be read and written.

i

Identifier for an input. Inputs are read only.

<number>

Number of the submodule containing the input or output area.

<offset> (offset zero-based - first element is zero)

Byte offset in the address space of the slave on which the element to be addressed is located. If a submodule is specified, the offset applies within the submodule. If no submodule

is specified, the offset relates to the entire input/output area of the slave. The byte offset is zero-based.

<DPtype>

A DP data type is converted to the corresponding OPC UA data type on the OPC UA server. The following table lists the type identifier and the corresponding OPC UA data type in which the variable value can be represented.

Data type	OPC UA data type	Note
x<bitaddress>	Boolean	Bit (bool) In addition to the byte offset in the area, the <bitaddress> in the relevant byte must be specified. Range of values 0 to 7
b	Byte	Byte (unsigned)
	Byte string	Used as the default if no <DPtype> is specified. OPC UA does not know "Byte[]", but uses the scalar data type "ByteString" for this.
w	UInt16	Word (unsigned)
dw	UInt32	Double word (unsigned)
lw	UInt64	Long word (unsigned)
c	SByte	Byte (signed)
i	Int16	Word (signed)
di	Int32	Double word (signed)
li	Int64	Long word (signed)
r	Float	Floating point (4 bytes)
lr	Double	Floating point (8 bytes)
s<stringlength>	String	The <stringlength> reserved for the string must be specified. Range of values 1 to 254 When writing, shorter strings can also be written whereby the transferred data length is always the reserved string length in bytes. The unnecessary bytes are filled with the value 0.

<quantity>

Number of elements. The data type of the variable is an array with elements of the specified format. Specifying a number of array elements causes an array of the corresponding type to be formed even when only a single array element is addressed.

2.5.7.3 Examples of process variables for the DP master class 1

Here you will find examples illustrating the syntax of variable names for DP variables.

Inputs

Namespace URI: DP: (Namespace index: 3)

- CP 5623.conveyorbelt.i3.0
identifies input byte 0 of submodule 3 of the "conveyorbelt" slave
- CP 5623.motorcontrol.i3.1,b,3
identifies an array with 3 bytes starting at input byte 1 of submodule 3 of the "motorcontrol" slave
- CP 5623.weldingrobot.i3.2,dw
identifies a double word starting at input byte 2 of submodule 3 of the "weldingrobot" slave
- CP 5623.slv4.i3.0,r
identifies a floating-point number starting at input byte 0 of slave 4, submodule 3
- CP 5623.slv4.i.0,b,8
identifies the first 8 bytes of the entire input area of slave 4 over all submodules

Outputs

Namespace URI: DP: (Namespace index: 3)

- CP 5623.conveyorbelt.q7.1
identifies output byte 1 of submodule 7 of the "conveyorbelt" slave
- CP 5623.motorcontrol.q.7,2,x5
identifies bit 5 in output byte 2 of the "motorcontrol" slave, submodule 7
- CP 5623.slv4.q.0,w,8
identifies an array with 8 words from the output area of slave 4 over all submodules

2.5.7.4 DPC1 services

Using DPC1 services, you can access the data records of the DPC1 slaves. The data records are transferred acyclically.

The meaning of the data records is specified by the vendor of the slave. They could, for example, be used for the configuration data of a drive.

When a DPC1 variable is registered, the OPC server can only check that the syntax is correct and not whether the variable is valid on the partner device and the size of the data record is adequate based on the configuration of the DPC1 slave.

2.5.7.5 Syntax of the process variables for DPC1 services

Syntax of the process variables

Optimized syntax of the process variables of the DP OPC UA Node ID for data record data variables:

Namespace URI: DP: (Namespace index: 3)

Classic syntax

Readable and writable data records (without length information no writable data records):

- <DPmaster>.<slave>.s<slot>.dr<index>{,<length>} { .<offset>{,
 <DPtype>{,<quantity>}} }

Readable data records:

- <DPmaster>.<slave>.s<slot>.dr<index>{.<offset>{,
 <DPtype>{,<quantity>}} }

Explanations

<DPmaster>

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

<slave>

Symbolic slave name as identifier for access to a DP slave. The symbolic slave name can be assigned in STEP 7 / NCM PC (must be configured with OPC server V8.2 or higher).

Note

If an OPC server < V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier "slv" followed by the configured slave address is used as the symbolic slave name.

If an OPC server >= V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier of the slave is used as the slave name. The consistency check in STEP 7 / NCM PC ensures that unique slave names are used and that the slave name is not empty.

Note

Permitted characters for slave names

The following characters are permitted in the character set for slave names:

A-Z, a-z, 0-9, _, -, ^, !, #, \$, %, ', (,), =, ~, +, ', @, {, }

The following characters are not permitted in the character set for slave names:

- Period ".."
- Colon ":"
- Pipe symbol "|"
- Backslash "\\"
- Square brackets "[" and "]"
- Quotation marks """"
- Ampersand "&"
- Question mark "?"
- Angle brackets "<" and ">"
- Asterisk "**"

The space cannot be used at the start or end of a slave name.

s

Identifier for the slot of the slave

<slot>

Slot in the extended memory area of a slave for acyclic services. The slot and index identify a data record.

dr

Identifier for access to a data record.

<index>

Index within a slot in the extended memory area of a slave for acyclic services. The slot and index identify a data record.

<length>

Length of the record. Range between 1 and 240.

<offset>

Byte address in the data record for the element to be addressed.

<DPtype>

Data type.

The data type is converted to the corresponding OLE data type on the OPC UA server.

Data type	OPC UA data type	Note
x<bitaddress>	Boolean	Bit (bool) In addition to the byte offset in the area, the <bitaddress> in the relevant byte must be specified. Range of values 0 to 7
b	Byte	Byte (unsigned)
	Byte string	Used as the default if no <DPType> is specified. OPC UA does not know "Byte[]", but uses the scalar data type "ByteString" for this.
w	UInt16	Word (unsigned)
dw	UInt32	Double word (unsigned)
lw	UInt64	Long word (unsigned)
c	SByte	Byte (signed)
i	Int16	Word (signed)
di	Int32	Double word (signed)
li	Int64	Long word (signed)
r	Float	Floating point (4 bytes)
lr	Double	Floating point (8 bytes)
s<stringlength>	String	The <stringlength> reserved for the string must be specified. Range of values 1 to 254 When writing, shorter strings can also be written whereby the transferred data length is always the reserved string length in bytes. The unnecessary bytes are filled with the value 0.

<quantity>

Number of elements. The data type of the variable is an array with elements of the specified format. Specifying a number of array elements causes an array of the corresponding type to be formed even when only a single array element is addressed.

2.5.7.6 Examples of process variables for DPC1 services

Here you will find examples illustrating the syntax of variable names for DPC1 services.

Variable names for DPC1

CP 5613.motorcontrol.s3.dr2,120.7,dw

identifies access to the double word from offset 7 in a data record with a length of 120 bytes in slot 3, index 2 of the "motorcontrol" slave.

CP 5613.slv5.s3.dr2,120.8,b,4

identifies access to an array with 4 bytes from offset 8 in a data record with a length of 120 bytes in slot 3, index 2 of slave 5.

2.5.7.7 Fast Logic for CP 5613/CP 5614/CP 5623/CP 5624 (master only)

The CP 5613/CP 5623 and DP master section of the CP 5614/CP 5624 support the Fast Logic property. This means that you can set the CP parameters so that it writes values to the same or other slaves as a reaction to the data change of a slave. The user application is also informed of the data change.

The CP 5613/CP 5614 and CP 5623/CP 5624 provide 6 Fast Logic triggers that can be configured and evaluated using the OPC UA data variables and methods.

Advantages of fast logic

Using Fast Logic has the following advantages:

- There is less load on the OPC server and OPC client.
- The data transmission is faster because it is not dependent on the software running on the CP and takes place in the hardware of the CP.

Note

After a fast logic trigger has been triggered, it is automatically deactivated again afterwards. You then need to activate the trigger again using the "fl<fastlogicid>.on" method.

Fast Logic functions correctly only when the DP master is in the OPERATE mode and the slaves involved are in the "READY" state. A Fast Logic trigger should therefore only be activated by the DP application program when the user program has brought the DP master to the "OPERATE" state and the slaves involved are in the "READY" state.

As long as fast logic triggers are active, no DP user program may write to the output bytes linked to input bytes with fast logic .

2.5.7.8 Syntax of the control variables for fast logic

Syntax of the OPC UA data variables

Namespace URI: DP: (Namespace index: 3)

```
<DPmaster>.fl<fastlogicid>.state  
<DPmaster>.fl<fastlogicid>.fastlogicid
```

Syntax of the OPC UA methods

Namespace URI: DP: (Namespace index: 3)

```
<DPmaster>.fl<fastlogicid>.activate
```

```
<DPmaster>.fl<fastlogicid>.clear
```

Explanation

<DPmaster>

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

fl

Identifier for fast logic.

<fastlogicid>

Number of the fast logic trigger being used. Value between 1 and 4.

state

Returns the fast logic status.

Return values:

- CLEARED (0)
Trigger <fastlogicid> is not activated.
- ACTIVATED (1)
Trigger <fastlogicid> is activated.
- TRIGGERED (2)
Trigger <fastlogicid> has performed the monitoring.

fastlogicid

Returns the ID of the fast logic trigger being used.

activate

Activate and call the fast logic trigger.

The "activate" OPC UA method is sent as a parameter with the call.

If a parameter block is sent, the fast logic property is set for the trigger <fastlogicid> specified in the OPC UA method. The parameter block that is sent is an array with 8 bytes and has the following structure:

- slave_addr_in_byte
Address of the slave whose inputs are selected for the trigger
- index_in_byte
Offset of the input byte of the trigger
- cmp_value_in_byte
Comparison value for the input byte
- mask_in_byte
You can mask individual bits in the input byte so that they are ignored in the comparison. A bit is masked by setting the value "1" in the appropriate bit, this means that if mask_in_byte==0x00 is set, all bits of cmp_value_in_byte are used for the comparison. The trigger is fired when all unmasked bits in the selected input byte match the bits in cmp_value_in_byte.
- slave_addr_out_byte
Selects the slave whose output byte will be changed when the trigger condition occurs

- **index_out_byte**
Offset of the output byte
- **value_out_byte**
Value of the output byte that will be written
- **mask_out_byte**
Bits in the output byte can be masked individually so that they are not modified when the trigger condition occurs. A bit is masked by setting the value "1" in the appropriate bit; this means that if `mask_out_byte==0x00` is set, all bits of `value_out_byte` are written to the selected output byte.

clear

Deactivate fast logic.

The "off" OPC UA method can only be written. It does not have any arguments.

Examples

OPC UA variable: CP 5614.fl4.state

OPC UA method: CP 5623.fl3.clear

2.5.7.9 Control frames Sync and Freeze

For particular applications, there are four control commands available with which a control frame can be sent. In standard applications, these are not required.

A control frame is a frame that the master sends to one slave, a group, several groups, or to all slaves. These frames are not acknowledged by the slaves.

Control frames are used to transfer control commands (known as global controls) to the selected slaves to allow synchronization. A control command contains three components:

- Identifier indicating whether one or more DP slaves are being addressed
- Identification of the slave group
- Control command(s)

Creating groups

During configuration, you can assign a group identifier to a slave; in other words, it is possible to include several slaves in one group.

Which slaves belong to a group is specified when you create the database. During this phase, each DP slave can be assigned a group number. The DP slave is informed of this group number during the parameter assignment phase. You can specify a maximum of eight groups.

2.5.7.10 Syntax of the control variables for Sync and Freeze

Syntax of the OPC UA methods

Namespace URI: DP: (Namespace index: 3)

```
<DPmaster>.q.sync()
<DPmaster>.q.unsync()
<DPmaster>.i.freeze()
<DPmaster>.i.unfreeze()
<DPmaster>.<slave>.q.sync()
<DPmaster>.<slave>.q.unsync()
<DPmaster>.<slave>.i.freeze()
<DPmaster>.<slave>.i.unfreeze()
<DPmaster>.<slave>.state.<status>
```

Explanation

<DPmaster>

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

<slave>

Symbolic slave name as identifier for access to a DP slave. The symbolic slave name can be assigned in STEP 7 / NCM PC (must be configured with OPC server V8.2 or higher).

Note

If an OPC server < V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier "slv" followed by the configured slave address is used as the symbolic slave name.

If an OPC server >= V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier of the slave is used as the slave name. The consistency check in STEP 7 / NCM PC ensures that unique slave names are used and that the slave name is not empty.

Note**Permitted characters for slave names**

The following characters are permitted in the character set for slave names:

A-Z, a-z, 0-9, _, -, ^, !, #, \$, %, ', (,), =, ~, +, ',', @, {, }

The following characters are not permitted in the character set for slave names:

- Period ".."
- Colon ":"
- Pipe symbol "|"
- Backslash "\\"
- Square brackets "[" and "]"
- Quotation marks """"
- Ampersand "&"
- Question mark "?"
- Angle brackets "<" and ">"
- Asterisk "***"

The space cannot be used at the start or end of a slave name.

q

Identifier for an output.

sync()

With the "sync" OPC UA method call, the current states of all DP slave outputs of the selected groups are frozen. All output data sent by the DP master following this is initially not accepted by the DP slaves.

The parameter block that is written is a byte with the following structure:

- **slavegroups**

Each bit of the byte represents one of 8 configurable slave groups. Bit 0 represents the first slave group.

If you repeat the sync call, the current output data of the master is transferred to the outputs of the slaves and the outputs are frozen again.

unsync()

The "unsync" OPC UA method call cancels the freezing of the DP slave outputs. After receiving the unsync call, the DP slaves once again accept all the output data sent by the DP master cyclically.

The parameter block that is written is a byte with the following structure:

- **slavegroups**

Each bit of the byte represents one of 8 configurable slave groups. Bit 0 represents the first slave group.

i

Identifier for an input.

freeze()

With the "freeze" OPC UA method call, the current states of all DP slave inputs of the selected groups are frozen. In the read cycles that follow, the DP master receives this frozen input data.

The parameter block that is sent is a byte with the following structure:

- slavegroups

Each bit of the byte represents one of 8 configurable slave groups. Bit 0 represents the first slave group.

If you repeat the sending of the freeze call, the current input data of the master is transferred to the inputs of the slaves and the inputs are frozen again.

unfreeze()

The "unfreeze" OPC UA method call cancels the freezing of the DP slave inputs. After receiving the unfreeze call, the DP slaves once again provide the DP master with the current input data.

The parameter block that is sent is a byte with the following structure:

- slavegroups

Each bit of the byte represents one of 8 configurable slave groups. Bit 0 represents the first slave group.

state

Identifier for a slave status method or variable.

<status>

Status	Description
activate()	Activates the slave
deactivate()	Deactivates the slave
restart()	Restarts the slave
state	Variable that indicates the status of the slave (EnumString).

Examples

CP 5614.q.sync()

CP 5614.motorcontrol.state.state

2.5.7.11 DP-specific information variables**DP-specific data variables for information**

There are DP master-specific data variables with which you can obtain information about the status, properties or type of DP master.

The following information can be obtained:

- Mode of the DP master (mode)
- Module information of the DP master (info)

- Watchdog of the DP master (watchdog)
- Slave parameters that the DP master knows from its slaves.

Syntax of the DP-specific information variables

Nodeld:

Namespace index: 3

<masterobject>. <informationparameter>

Mode of the DP master

Syntax for the mode of the DP master (mode)

Namespace URI: DP: (Namespace index: 3)

<DPmaster>. <informationparameter>

Explanations

<DPmaster

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

<informationparameter>

Module information	Description	
mode	Current mode of the DP master. The current mode can both be read and written. Setting the mode by writing one of the values shown below is only possible within the context of the DP application environment. Data variable of the OPC UA type "MultistateDiscreteType", read and write "mode" can, for example, return the following values:	
	OFFLINE (0)	No communication between master and slave.
	STOP (1)	No communication between master and slave except for diagnostics data.
	CLEAR (2)	Parameter assignment and configuration phase
	AUTOCLEAR (3)	Autoclear phase, the DP master can no longer access all slaves
	OPERATE (4)	Productive phase

Example:

CP 5614.mode

Module information of the DP master

Syntax for the module information of the DP master (info)

Namespace URI: DP: (Namespace index: 3)

<DPmaster>. <moduleinformation>

Explanations

<DPmaster>

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

<moduleinformation>

Module information	Description	
ident number	Identification number of the certification Data variable of the OPC UA type "UInt16", read-only	
hardware	Hardware Data variable of the UA type "MultistateDiscreteType", read-only.	
	0	SOFTNET
	1	CP 5613 (electrical PROFIBUS port)
	2	CP 5613 FO (optical PROFIBUS port)
	3	CP 5614 (electrical PROFIBUS port)
	4	CP 5614 FO (optical PROFIBUS port)
	5	CP 5614 FO (optical PROFIBUS port) with connected external power supply
	6	CP 5613 FO (optical PROFIBUS port) with connected external power supply
	7	CP 5613 A2 (electrical PROFIBUS port)
	8	CP 5614 A2 (electrical PROFIBUS port)
	9	CP 5623 (electrical PROFIBUS port)
	10	CP 5624 (electrical PROFIBUS port)
	11	CP 5613 A3 (electrical PROFIBUS port)
	12	CP 5614 A3 (electrical PROFIBUS port)
hardwareversion	Hardware version Data variable of the OPC UA type "Byte", read-only	
firmwareversion	Firmware version Data variable of the OPC UA type "UInt16", read-only	
buspar	Read bus parameters of the master Data variable of the OPC UA type "ByteString", read-only	

Example:

CP 5614.hardware

Watchdog of the DP master

Syntax for the watchdog of the DP master

Namespace URI: DP: (Namespace index: 3)

<DPmaster>.watchdog.<watchdogvariable>

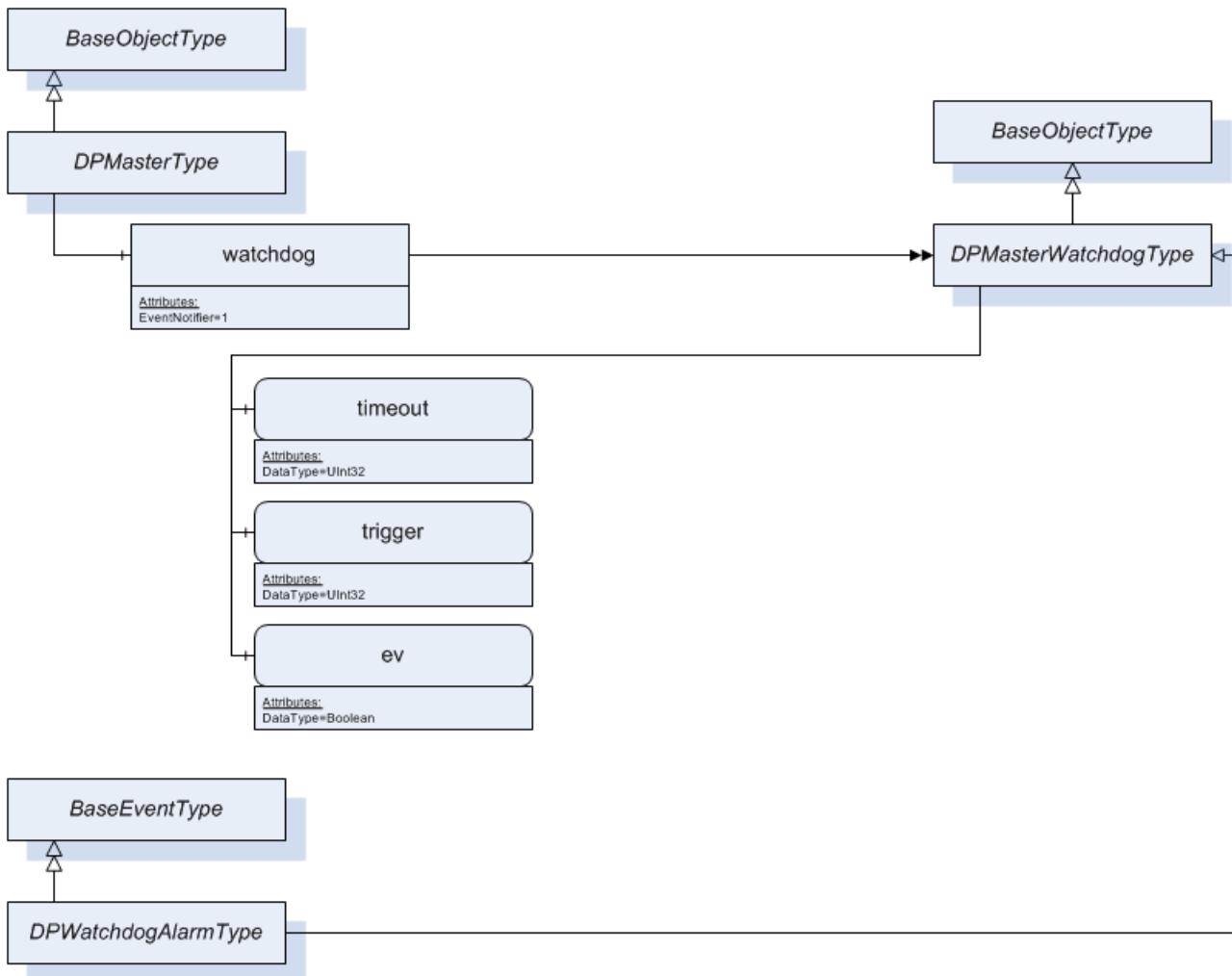


Figure 2-11 Watchdog on the DP master

watchdog.timeout	Read or set, reset or deactivate the watchdog timeout. The OPC UA server automatically rounds up a written value to the next higher value. Data variable of the OPC UA type "UInt32", read and write, the unit is milliseconds.
0	Turn off watchdog

	Range of values: between 20 up to and including 102,000 ms	Set watchdog timeout Granularity in Softnet DP, e.g. CP 5621: 400 ms Granularity in Hardnet DP, e.g. CP 5623: 10 ms
watchdog.trigger	Monitor the watchdog functionality of the OPC UA server. Data variable of the OPC UA type "UInt32", read-only	
watchdog.ev	Trigger monitoring of the watchdog Data variable of the OPC UA type "Boolean", read-only	
	False	Normal operation
	True	DP protocol stack has triggered the watchdog.

Example:

CP 5614.watchdog.ev

CP 5614.watchdog.timeout

DP slave on a DP master**Syntax for DP slave information variables on a DP master**

Namespace URI: DP: (Namespace index: 3)

<DPmaster>.<slave>.<informationvariables>

Explanations**<DPmaster>**

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

<slave>

Symbolic slave name as identifier for access to a DP slave. The symbolic slave name can be assigned in STEP 7 / NCM PC.

Note

If an OPC server < V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier "slv" followed by the configured slave address is used as the symbolic slave name.

If an OPC server >= V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier of the slave is used as the slave name. The consistency check in STEP 7 / NCM PC ensures that unique slave names are used and that the slave name is not empty.

<address>

PROFIBUS address of the DP slave.

Range: 0 ... 126.

<informationvariables>

Information variables	Description				
type	Identify DP slave type. Data variable of the UA type "MultistateDiscreteType", read-only.				
	0	NORM	Standard DP slave		
	1	ET200_U	Nonstandard slave: ET 200 U		
	2	ET200K_B	Nonstandard slave: ET 200 K/B		
	3	ET200_SPM	Nonstandard slave: General SPM station		
dpv	Query the DP protocol version of the DP slave. Data variable of the UA type "MultistateDiscreteType", read-only.				
	0	DP-V0			
	1	DP-V1			
	2	DP-V2			
parcfgdata	With this information variable, the configuration of the IO submodules of the DP slave can be queried by the CP. The parameters originate from the configuration. Data variable of the UA type "ByteString", read-only				
parprmdata	With this information variable, the parameter assignment data of the DP slave can be queried. Data variable of the UA type "ByteString", read-only				
paruserdata	With this information variable, the user parameter assignment data of the DP slave can be queried. Data variable of the UA type "ByteString", read-only				
partype	With this information variable, the general slave parameters such as SI flag, slave type and octet string can be queried. Data variable of the UA type "ByteString", read-only				
paraddtab	With this information variable, the various components of the slave parameters of a DP slave can be read out. Data variable of the UA type "ByteString", read-only				

Examples:

CP 5611.motorcontrol.type

CP 5614.slv17.dpv

Syntax for DP slave IO information variables on a DP master

Namespace URI: DP: (Namespace index: 3)

The available options are as follows:

- <DPmaster>.<slave>.q.<slaveproperties>
- <DPmaster>.<slave>.i.<slaveproperties>

- <DPmaster>.<slave>.q<moduleID>.<slavemoduleproperties>
- <DPmaster>.<slave>.i<moduleID>.<slavemoduleproperties>

Explanations

<DPmaster>

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

<slave>

Symbolic slave name as identifier for access to a DP slave. The symbolic slave name can be assigned in STEP 7 / NCM PC.

Note

If an OPC server < V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier "slv" followed by the configured slave address is used as the symbolic slave name.

If an OPC server >= V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier of the slave is used as the slave name. The consistency check in STEP 7 / NCM PC ensures that unique slave names are used and that the slave name is not empty.

q

Identifier for an output. Outputs can be read and written

i

Identifier for an input. Inputs are read only.

<moduleID>

ID of the submodule containing the input or output area.

<slaveproperties>

IO information variables	Description	
length	Module length in bytes from the DP slave configuration data Property of the type UInt16, read-only	
module type	Module type from the DP slave configuration data. Data variable of the UA type "MultistateDiscreteType", read-only.	
	0	UNKNOWN
	1	I (input)
	2	Q (output)
	3	IQ (input/output)
	the value "3" occurs only with DP slave modules. In this case, there is both an input and output submodule with the same "moduleid".	
slaveid	PROFIBUS address of the corresponding DP slave Property of the type "Byte", read-only	

<slavemodeproperties>

IO information variables	Description	
length	Module length in bytes from the DP slave configuration data Property of the type UInt16, read-only	
module type	Module type from the DP slave configuration data. Data variable of the UA type "MultistateDiscreteType", read-only.	
	0	UNKNOWN
	1	I (input)
	2	Q (output)
	3	IQ (input/output)
	the value "3" occurs only with DP slave modules. In this case, there is both an input and output submodule with the same "moduleid".	
slaveid	PROFIBUS address of the corresponding DP slave Property of the type "Byte", read-only	
moduleid	Module address from the DP slave configuration data. Property of the type "Byte", read-only	
slot	Slot or module identifier of this submodule. Property of the type "Byte", read-only	
dataunitlength	length of the data unit of a submodule from the DP slave configuration data. Data variable of the UA type "MultistateDiscreteType", read-only	
	0	UNKNOWN
	1	BYTE
	2	WORD
consistence	Consistency over all data units of a submodule from the DP slave configuration data. Property of the type "Boolean", read-only	

Examples:

CP 5614.conveyorbelt.q.length

CP 5614.slv17.q.moduletype

CP 5614.slv17.q0.length

Syntax for DP slave diagnostics variables on a DP master

Namespace URI: DP: (Namespace index: 3)

<DPmaster>.<slave>.diagnosis.data{.<diagnosticsvariables>}

Explanations**<DPmaster>**

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

<slave>

Symbolic slave name as identifier for access to a DP slave. The symbolic slave name can be assigned in STEP 7 / NCM PC.

Note

If an OPC server < V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier "slv" followed by the configured slave address is used as the symbolic slave name.

If an OPC server >= V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier of the slave is used as the slave name. The consistency check in STEP 7 / NCM PC ensures that unique slave names are used and that the slave name is not empty.

diagnosis

Diagnostics data of the DP slave.

data

Contains the last diagnostics data of the DP slave. The first 6 bytes contain the standard diagnostics of the DP slave. Data variable of the type "ByteString", that is read-only.

<diagnosticsvariables>

Diagnostics variables	Description
masterlock	Bit 7 from byte 1 of the diagnostics data. The DP slave has already had parameters assigned by a different DP master, in other words, the local DP master cannot currently access this DP slave. Data variable of the type "Boolean", read-only
prmfault	Bit 6 from byte 1 of the diagnostics data. This bit is set by the DP slave if the last parameter assignment frame was incorrect (for example wrong length, wrong ident number, invalid parameters). Data variable of the type "Boolean", read-only
invalidslaveresponse	Bit 5 from byte 1 of the diagnostics data. This bit is set as soon as an implausible response is received from an addressed DP slave. Data variable of the type "Boolean", read-only
notsupported	Bit 4 from byte 1 of the diagnostics data. This bit is set as soon as a function is requested that is not supported by the DP slave (for example operation in the SYNC mode although the DP slave does not support this). Data variable of the type "Boolean", read-only
extdiag	Bit 3 from byte 1 of the diagnostics data. This bit is set by the DP slave. If the bit is set, there must be a diagnostics entry in the slave-specific diagnostics area (Ext_Diag_Data, byte 7-32). If the bit is not set, there may be a status message in the slave-specific diagnostics area (Ext_Diag_Data, byte 7-32). The meaning of the status message must be negotiated with each specific application. Data variable of the type "Boolean", read-only

Diagnostics variables	Description
cfgfault	<p>Bit 2 from byte 1 of the diagnostics data.</p> <p>This bit is set when the configuration data last sent by the DP master does not match the data determined by the DP slave, in other words, when there is a configuration error.</p> <p>Data variable of the type "Boolean", read-only</p>
stationnotready	<p>Bit 1 from byte 1 of the diagnostics data.</p> <p>This bit is set when the DP slave is not yet ready for productive data exchange.</p> <p>Data variable of the type "Boolean", read-only</p>
stationnonexistent	<p>Bit 0 from byte 1 of the diagnostics data.</p> <p>The DP master sets this bit when the DP slave cannot be reached over the bus. If this bit is set, the diagnostics bits contain the status of the last diagnostics message or the initial value. The DP slave always sets this bit to zero.</p> <p>Data variable of the type "Boolean", read-only</p>
deactivated	<p>Bit 7 from byte 2 of the diagnostics data.</p> <p>This bit is set when the DP slave is indicated as being inactive in the local parameter set and has been taken out of cyclic processing.</p> <p>Data variable of the type "Boolean", read-only</p> <p>Note: Bit 6 is reserved.</p>
syncmode	<p>Bit 5 from byte2 of the diagnostics data.</p> <p>This bit is set by the DP slave as soon as it receives the Sync control command.</p> <p>Data variable of the type "Boolean", read-only</p>
freezemode	<p>Bit 4 from byte 2 of the diagnostics data.</p> <p>This bit is set by the DP slave as soon as it receives the Freeze control command.</p> <p>Data variable of the type "Boolean", read-only</p>
wdon	<p>Bit 3 from byte 2 of the diagnostics data.</p> <p>This bit is set by the DP slave. If this bit is set to 1, the watchdog monitoring is activated on the DP slave.</p> <p>Data variable of the type "Boolean", read-only</p>
statdiag	<p>Bit 1 from byte 2 of the diagnostics data.</p> <p>If the DP slave sets this bit, the DP master must receive diagnostics information until the bit is cleared again. The DP slave, for example, sets this bit when it is not capable of providing valid user data.</p> <p>Data variable of the type "Boolean", read-only</p> <p>Note: Bit 2: The DP slave always sets this bit to 1.</p>
prmreq	<p>Bit 0 from byte 2 of the diagnostics data.</p> <p>If the DP slave sets this bit, it must be reassigned parameters and reconfigured. The bit remains set until the parameters are assigned.</p> <p>Data variable of the type "Boolean", read-only.</p> <p>Note: If bit 0 and bit 1 are both set at the same time, bit 0 has the higher priority.</p>

Diagnostics variables	Description
extdiagoverflow	Bit 7 from byte 3 of the diagnostics data. If this bit is set, there is more diagnostic information available than shown in "Ext_Diag_Data". The DP slave sets this bit, for example, when there is more channel diagnostics data than the DP slave can enter in its send buffer; or the DP master sets this bit when the DP slave sends more diagnostics data than the DP master can accommodate in its diagnostics buffer. Data variable of the type "Boolean", read-only
masteraddr	Byte 4 of the diagnostics data. The address of the DP master that assigned parameters to this DP slave is entered in byte 4, the "Diag. Master_Add" byte. If the DP slave has not been assigned parameters by a DP master, the DP slave sets the address 255 in this byte. Data variable of the type "Byte", read-only Note: Bit 6 to bit 0 are reserved.
ident number	Bytes 5 and 6 of the diagnostics data. The vendor ID for a DP slave type is assigned in bytes 5 and 6, the "Ident_Number" byte. This identifier can be used on the one hand for test purposes and on the other for precise identification of the slave. Data variable of the type "Word", read-only
extdiagdata	Byte 7-32 of the diagnostics data. Starting at byte 7, the "Ext_Diag_Data" bytes, the DP slave can store its specific diagnostics information. A block structure with a header byte each for the device- and identifier-related diagnostics is necessary. Data variable of the type "ByteString", read-only

Examples:

CP 5611.motorcontrol.diagnosis.data
 CP 5611.slv17.diagnosis.data.stationnotready
 CP 5611.slv17.diagnosis.data.identnumber

Examples of DP-specific information variables

Here, you will find examples illustrating the syntax of the names of DP-specific information variables.

Information about the hardware of a DP master

Nodeld:

- Namespace URI:
DP: (Namespace index: 3)
- Identifier:
CP 5611 A2.hardware of the type "int"

Possible return value: SOFTNET, CP5613,CP5614 etc.

Watchdog of a DP master

Nodeld:

- Namespace URI:
DP: (Namespace index: 3)
- Identifier:
 - *CP 5624.watchdog.timeout*
Possible return value: 0 = watchdog is deactivated
 - *CP 5624.watchdog.trigger*
Possible return value: 2000 = watchdog functionality of the DP OPC UA server
 - *CP 5624.watchdog.ev*
Possible return value: True = watchdog was activated

2.5.8 DP slave

Note

A PROFIBUS communications processor as DP slave can only be operated via OPC as a DP-V0 slave. With the HARDNET modules named, no data records can therefore be read or written via OPC.

2.5.8.1 Variable services for accessing local DP slave data

The OPC Server for SIMATIC NET can operate in a PROFIBUS DP network not only as a DP master but also in the role of DP slave. The OPC server manages memory areas for the inputs and outputs of this DP slave and maps them to OPC variables. An OPC client can read the outputs set by the master and set values in the inputs that will be fetched by the master in the next cycle.

DP slaves have a modular structure. A DP slave can include several submodules with different input/output areas. The submodules are assigned during configuration.

The variable name identifies an input or output area in the submodule of a slave. The inputs and outputs of the slave are accessed by specifying the submodule number and the input or output area.

Status information of the slaves and the DP master can also be queried using variable names.

2.5.8.2 Syntax of the process variables for the DP slave

Syntax of the process variables

Optimized syntax of the process variables of the DP OPC UA Node ID for reading and writing IO data variables:

Namespace URI: DP: (Namespace index: 3)

Classic syntax

You have two options:

- <DPslave>.i{{<number>}{.<offset>{,<DPtype>{,<quantity>}}}
- <DPslave>.q{{<number>}{.<offset>{,<DPtype>{,<quantity>}}}

Explanations

<DPslave>

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

q

Identifier for an output. Outputs are read only.

i

Identifier for an input. Inputs can be read and written.

<number>

Number of the submodule containing the input or output area.

<offset>

Byte offset in the address space of the slave at which the element to be addressed is located. If a submodule is specified, the offset applies within the submodule. If no submodule is specified, the offset relates to the entire input/output area of the slave.

<DPtype>

A DP data type is converted to the corresponding OPC UA data type on the OPC UA server. The following table lists the type identifier and the corresponding OPC UA data type in which the variable value can be represented.

Data type	OPC UA data type	Note
x<bitaddress>	Boolean	Bit (bool) In addition to the byte offset in the area, the <bitaddress> in the relevant byte must be specified. Range of values 0 to 7
b	Byte	Byte (unsigned) Used as the default if no <DPtype> is specified. OPC UA does not know "Byte[]", but uses the scalar data type "ByteString" for this.
	Byte string	
w	UInt16	Word (unsigned)
dw	UInt32	Double word (unsigned)
lw	UInt64	Long word (unsigned)
c	SByte	Byte (signed)
i	Int16	Word (signed)
di	Int32	Double word (signed)
li	Int64	Long word (signed)
r	Float	Floating point (4 bytes)

Data type	OPC UA data type	Note
lr	Double	Floating point (8 bytes)
s<stringlength>	String	The <stringlength> reserved for the string must be specified. Range of values 1 to 254 When writing, it is also possible to write shorter strings, whereby the transferred data length is always the reserved string length in bytes plus 2 bytes. The unnecessary bytes are filled with the value 0. Reading and writing strings and string arrays is mapped internally to the reading and writing of byte arrays.

<quantity>

Number of elements. The data type of the variable is an array with elements of the specified format. Specifying a number of array elements causes an array of the corresponding type to be formed even when only a single array element is addressed.

2.5.8.3 Examples of the process variables for the DP slave

Here, you will find several examples.

Inputs

- CP 5611.i3.0,b
identifies input byte 0 (offset 0) in submodule 3 of the DP slave
- CP 5611.i3.1,b,3
identifies an array with 3 bytes starting at input byte 1 (offset 1) in submodule 3 of the DP slave
- CP 5614.i3.0,x0
Input bit 0 in byte 0 in submodule 3 of the DP slave
- CP 5614.i3.3,b,8
identifies an array with 8 input bytes starting at offset 3 in submodule 3 of the DP slave

Outputs

- CP 5611.q4.3,w
identifies an output word at address 3 in submodule 4 of the DP slave
- CP 5611.q3.2,dw
identifies an output double word at address 2 in submodule 3 of the DP slave
- CP 5611,q0.3,x7
identifies output bit 7 in byte 3 of the DP slave
- CP 5614.q1.0,w,4
identifies an array with 4 output words on submodule 1 of the DP slave

2.5.8.4 Status of the DP slave

Syntax for the status of the DP slave

Namespace URI: DP: (Namespace index: 3)

<DPslave>.<statusvariables>

Explanations

<DPslave>

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

state

Indicates the status of the DP slave.

<statusvariables>

Status variables	Description	
state.state	Query status of the DP slave. Data variable of the UA type "MultistateDiscreteType", read-only	
	0	NO_DATA_EXCHANGE The DP master has not yet configured and assigned parameters for the DP slave.
	1	DATA_EXCHANGE The DP slave is in cyclic data exchange with the DP master.

2.5.8.5 Mode of the DP slave

Syntax for the mode of the DP slave

<DPslave>.<mode>

Explanations

<DPslave>

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

mode

	Description	
mode	Query or set the mode of the slave. Data variable of the UA type "MultistateDiscreteType", read and write	
	0	OFFLINE The DP slave is not reacting on the bus.

	Description		
1	ONLINE	The DP slave is ready to receive the parameter assignment and configuration frames from the DP master and finally after successful parameter assignment and configuration to change to the "DATA_EXCHANGE" status.	

2.5.8.6 DP slave-specific information variables

There are several predefined information variables for diagnostics purposes on the DP slave.

2.5.8.7 Syntax of the DP slave-specific information variables

Syntax for DP slave information variables on a DP slave

Namespace URI: DP: (Namespace index: 3)

<DPslave>.<informationvariables>

Explanations

<DPslave>

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

<informationvariables>

Information variables	Description		
type	Query slave type Data variable of the UA type "MultistateDiscreteType", read-only	0	NORM Standard DP slave
	The DP slave operated by the OPC server is always a standard DP slave.		
dpv	Query the DP protocol version of the DP slave. Data variable of the UA type "Byte", read-only	0	DP-V0
		1	DP-V1
		2	DP-V2
	Always 0, the DP slave being operated by the OPC server supports only DP-V0.		
parcfgdata	Configuration data of the DP slave. The configuration data contains the information about the configuration of the IO submodules and of the DP slave and is used by the OPC UA server to set up the IO module objects belonging to the DP slave. Data variable of the UA type "ByteString", read-only		
paruserdata	User parameter assignment data of the DP slave. Normally, no data is returned for this. Data variable of the UA type "ByteString", read-only		

Information variables	Description	
hardware	Hardware. Data variable of the UA type "MultistateDiscreteType", read-only	
	0	SOFTNET
	1	CP 5613 (electrical PROFIBUS port)
	2	CP 5613 FO (optical PROFIBUS port)
	3	CP 5614 (electrical PROFIBUS port)
	4	CP 5614 FO (optical PROFIBUS port)
	5	CP 5614 FO (optical PROFIBUS port) with external power supply
	6	CP 5613 FO (optical PROFIBUS port) with external power supply
	7	CP 5613 A2 (electrical PROFIBUS port)
	8	CP 5614 A2 (electrical PROFIBUS port)
	9	CP 5623 (electrical PROFIBUS port)
	10	CP 5624 (electrical PROFIBUS port)
	11	CP 5613 A3 (electrical PROFIBUS port)
	12	CP 5614 A3 (electrical PROFIBUS port)
hardwareversion	Hardware version. Data variable of the UA type "Byte", read-only	
firmwareversion	Firmware version. Data variable of the UA type "UInt16", read-only	
ident number	Identification number of the certification. Data variable of the UA type "UInt16", read-only	

Example:

CP 5611.hardware

Syntax for DP slave IO information variables on a DP slave

There are 4 options:

- <DPslave>.q.<slaveproperties>
- <DPslave>.i.<slaveproperties>
- <DPslave>.q<number>.<slavemoduleproperties>
- <DPslave>.i<number>.<slavemoduleproperties>

Explanations**<DPslave>**

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

q

Identifier for an output. Outputs are read only.

i

Identifier for an input. Inputs can be read and written.

<number>

Number of the submodule containing the input or output area.

<slaveproperties>

IO information variables	Description	
length	Module length in bytes from the DP slave configuration data Property of the type UInt16, read-only	
module type	Module type from the DP slave configuration data. Data variable of the UA type "MultistateDiscreteType", read-only.	
	0	UNKNOWN
	1	I (input)
	2	Q (output)
	3	IQ (input/output)
	the value "3" occurs only with DP slave modules. In this case, there is both an input and output submodule with the same "moduleid".	
slaveid	PROFIBUS address of the corresponding DP slave Property of the type "Byte", read-only	

<slavemodeleproperties>

IO information variables	Description	
length	Module length in bytes from the DP slave configuration data Property of the type UInt16, read-only	
module type	Module type from the DP slave configuration data. Data variable of the UA type "MultistateDiscreteType", read-only.	
	0	UNKNOWN
	1	I (input)
	2	Q (output)
	3	IQ (input/output)
	the value "3" occurs only with DP slave modules. In this case, there is both an input and output submodule with the same "moduleid".	
slaveid	PROFIBUS address of the corresponding DP slave Property of the type "Byte", read-only	
moduleid	Module address from the DP slave configuration data. Property of the type "Byte", read-only	
slot	Slot or module identifier of this submodule. Property of the type "Byte", read-only	
dataunitlength	length of the data unit of a submodule from the DP slave configuration data. Data variable of the UA type "MultistateDiscreteType", read-only	
	0	UNKNOWN
	1	BYTE

IO information variables	Description	
	2	WORD
consistence	Consistency over all data units of a submodule from the DP slave configuration data. Property of the type "Boolean", read-only	

Examples:

CP 5614_s.q.length

CP 5614_s.q.modulename

CP 5614_s.q0.length

2.5.9 DP OPC UA template data variables

2.5.9.1 DP OPC UA template data variables

With the process variables for the DP OPC UA protocol, you have flexible setting options with which you can obtain the process data of your plant in the data formats you require.

The wide variety of addressing options cannot, however, be put together in a fully browsable namespace. Even a data block with the length of a single byte has approximately 40 different data format options - starting with Byte, SByte, arrays with one element of these, individual bits, arrays of bits with up to 8 array elements starting at different bit offsets.

The OPC UA server therefore supports the user with template data variables in the DP namespace. In a typical OPC UA client text input box, these templates can be converted to valid ItemIDs simply by changing a few characters.

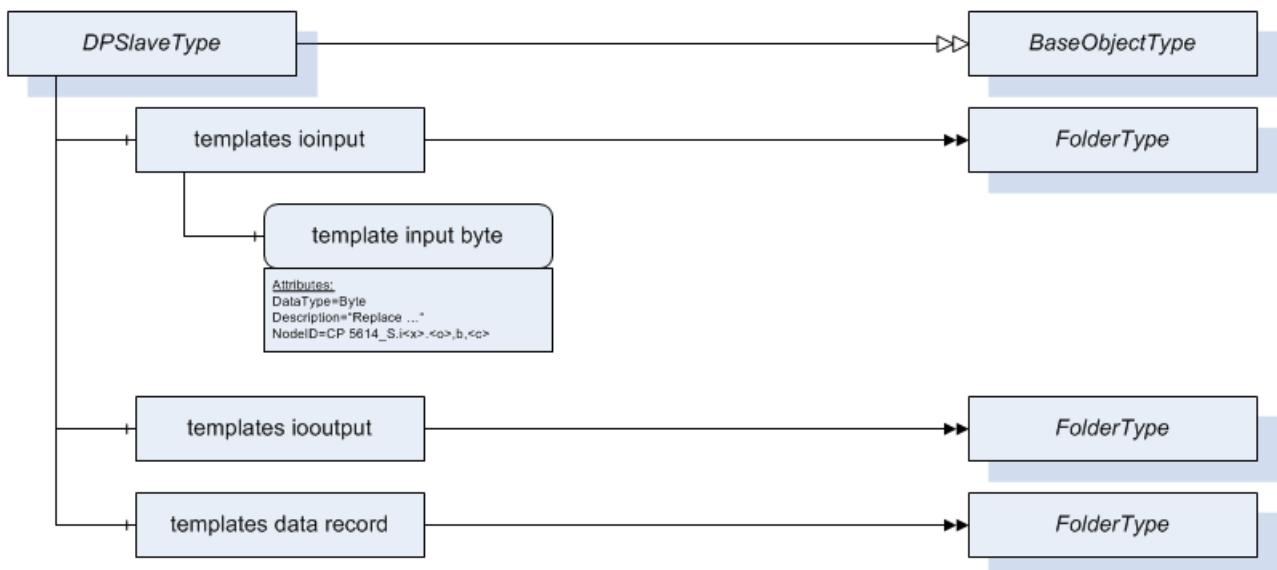


Figure 2-12 Template data variables in DP OPC UA

Note

The usability of DP OPC UA template data variables can be enabled and disabled in the "Communication Settings" configuration program in "OPC protocol selection" > click the arrow symbol beside "DP".

Template data variables within the browse hierarchy

The template data variables are sorted beside the corresponding folders in the namespace representation so that they can be used easily when required.

Syntax of the template data variables

You have two options:

- <DPmaster>.<slave>.i<x>.<o>,<DPtype>,<c>
- <DPmaster>.<slave>.q<x>.<o>,<DPtype>,<c>

Explanations**<DPmaster>**

Protocol-specific connection name. The connection name is specified in the configuration. In the DP protocol, the connection name is the configured name of the communications module.

<slave>

Symbolic slave name as identifier for access to a DP slave. The symbolic slave name can be assigned in STEP 7 / NCM PC.

Note

If an OPC server < V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier "slv" followed by the configured slave address is used as the symbolic slave name.

If an OPC server >= V8.2 is configured in STEP 7 / NCM PC and no symbolic slave name is assigned, the identifier of the slave is used as the slave name. The consistency check in STEP 7 / NCM PC ensures that unique slave names are used and that the slave name is not empty.

i

Identifier for an input. Inputs can be read and written.

q

Identifier for an output. Outputs are read only.

<x>

Placeholder for the number of the submodule containing the input or output area.

<o>

Placeholder for byte offset in the address space of the DP slave.

<DPtypetemplate>

A DP template data type is converted to the corresponding OPC UA data type on the OPC UA server. The following table lists the type identifier and the corresponding OPC data type in which the variable value can be represented.

Data type	OPC UA data type	Note
x<bit>	Boolean	Placeholder for the bit offset (0 to 7)
b	Byte	Placeholder for byte (unsigned)
w	UInt16	Placeholder for word (unsigned)
dw	UInt32	Placeholder for double word (unsigned)
lw	UInt64	Placeholder for long word (unsigned)
c	SByte	Placeholder for byte (signed)
i	Int16	Placeholder for word (signed)
di	Int32	Placeholder for double word (signed)
li	Int64	Placeholder for long word (signed)
r	Float	Placeholder for floating point (4 bytes)
lr	Double	Placeholder for floating point (8 bytes)
s<sl>	String	Placeholder for the length of the string

<c>

Placeholder for the number of elements.

Examples:

NodeID	BrowseName	Description
CP 5624.slv<v>.i<x>.<o>,x<bit>,<c>	template input bit array	<v> PROFIBUS address of the DP slave <x> Address of the submodule <o> Offset within the submodule <bit> bit offset (0 to 7) <c> Size of the array
CP 5624.slv<v>.i<x>.<o>,b,<c>	template input byte array	<v> PROFIBUS address of the DP slave <x> Address of the submodule <o> Offset within the submodule <c> Size of the array
CP 5624.slv<v>.i<x>.<o>,w,<c>	template input word array	<v> PROFIBUS address of the DP slave <x> Address of the submodule <o> Offset within the submodule <c> Size of the array
CP 5624.slv<v>.i<x>.<o>,s<sl>,<c>	template input string array	<v> PROFIBUS address of the DP slave <x> Address of the submodule <o> Offset within the submodule <sl> Length of the string <c> Size of the array

NodeId	BrowseName	Description
CP 5624.slv<v>.q<x>.<o>,x<bit>,<c>	template output bit array	<v> PROFIBUS address of the DP slave <x> Address of the submodule <o> Offset within the submodule <bit> bit offset (0 to 7) <c> Size of the array
CP 5624.slv<v>.dr<dr>,<l>.<o>,b	template data record byte	<v> PROFIBUS address of the DP slave <x> Address of the submodule <dr> Data record number <l> Length of the file <o> Offset within the file
CP 5624_S.i<x>.<o>,b	template input byte	<x> Address of the submodule <o> Offset within the submodule

2.6 S7 communication

The S7 protocol for PROFIBUS and Industrial Ethernet is used for communication between system components within the SIMATIC S7 programmable controller and for communication between SIMATIC S7 system components and programming devices or PCs.

Properties of S7 communication with OPC

The OPC server from SIMATIC NET has the following characteristics:

- Variable services
Access to and monitoring of S7 variables.
- Bufferoriented services
Programcontrolled transfer of larger blocks of data.
- Server functionality
The PC can be used as server for blocks of data and data blocks.
- Block management services
Transfer of a loadable data area from and to S7.
- S7 password functions
Setting a password to access protected blocks.
- Events
Processing S7 messages (S7 alarms) as OPC Alarms & Events.

2.6.1 Powerful SIMATIC NET OPC server for the S7 protocol

Introduction

The section below describes a configuration variant for the S7 protocol that meets requirements for higher performance. To use this variant, all underlying S7 protocol libraries and the COM server as inproc server are loaded on the outproc OPC server. The protocol is handled in the process of the OPC server, additional execution times for changing between processes and multiprotocol mode are avoided. The process change between the OPC client and OPC server is, however, still necessary.

Configuration

This high-speed variant is enabled implicitly by selecting the "S7" protocol as the only protocol in the "Communication Settings" configuration program (if other protocols or the OPC UA interface are selected, the performance advantage described is lost):

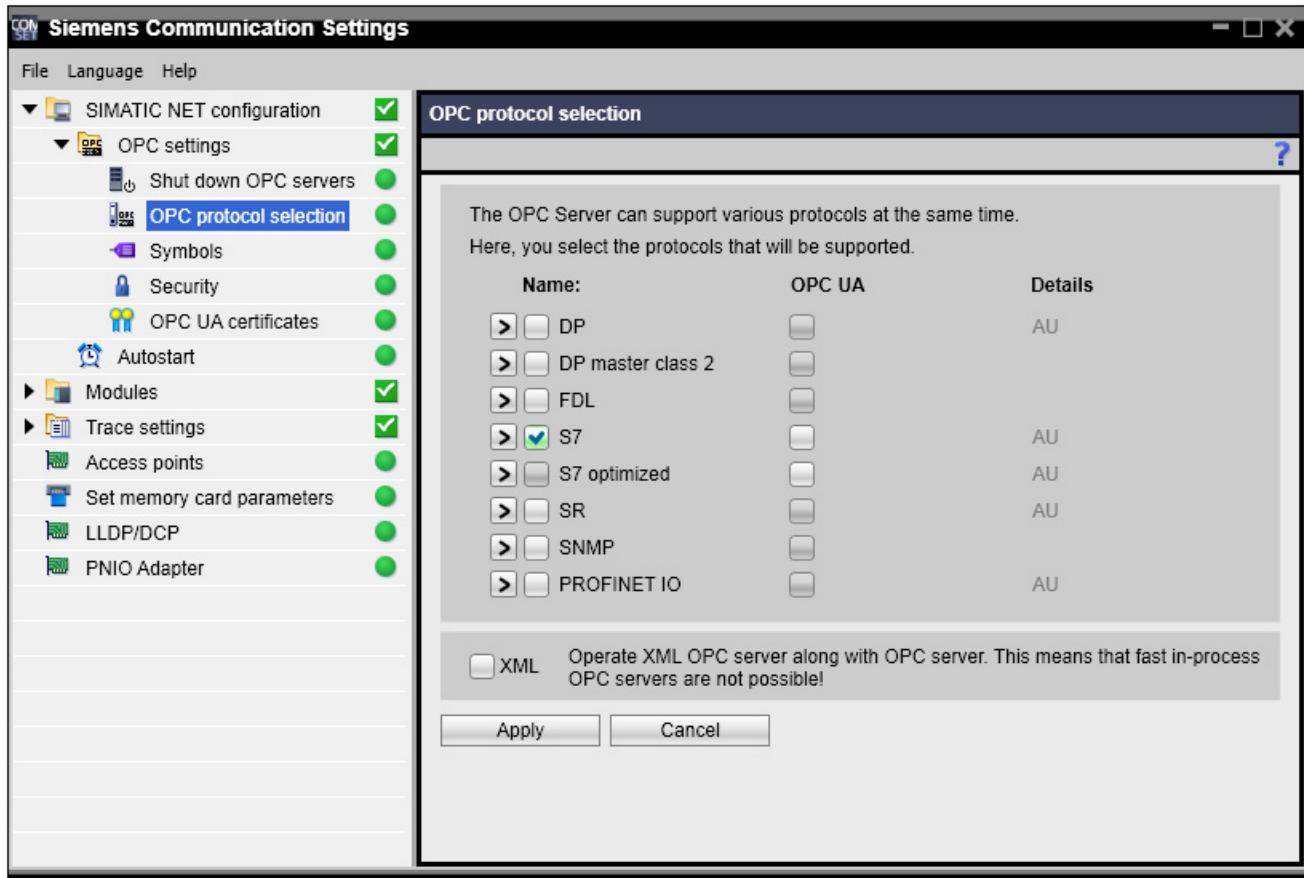


Figure 2-13 Window in the "Communication Settings" configuration program for selecting the S7 protocol

"Symbols" can also be selected.

Note

When creating symbols with STEP 7 or the symbol editor, the use of the following characters is permitted: A-Z, a-z, 0-9, _, -, ^, !, #, \$, %, &, ', /, (,), <, >, =, ?, ~, +, *, ', :, |, @, [,], {, }, ".

When creating symbols with STEP 7, you should also remember that problems can arise when resolving the array if your symbol file includes symbols in the form <symbolname> and <symbolname>[<index>] at the same time.

Advantages / disadvantages

Using the powerful S7 OPC server does, however, have the disadvantage that only the S7 single protocol mode is possible. On the other hand, it does have the following advantages:

- Higher performance than with multiprotocol mode.
- Simple configuration.
- Several clients can use the server at the same time.
- The stability of the OPC server does not depend on the clients.

Notes

Note

Make sure that the number of configured S7 connections of an OPC server does not exceed the number of S7 connections that can be operated simultaneously if the OPC server is configured as a passive connection partner in the connection properties (over configuration).

Background:

If more connections were configured in the PC station than are permitted at the same time, the OPC server cannot handle a connection request for all connections and it will not be possible to establish every S7 connection.

Note

S7 connections can be configured in SIMATIC STEP 7/NCM PC to be established "on demand".

The communication of the OPC functions is established only as necessary; this means that connection establishment may take longer and OPC may signal errors in the start-up phase.

2.6.2

Protocol ID

The protocol ID for the S7 protocol is "S7".

2.6.3 Connection names

STEP 7 connections

The connection name is the name configured in STEP 7 to identify the connection. In STEP 7, this name is known as the "local ID". The local ID is unique within the OPC server.

The OPC server supports the following connection types:

- S7 connection
- Faulttolerant S7 connection

Note

Compatibility with version 2.2 and earlier - please note the following:

In older versions, the connection information was made up of the information for the S7 connection, the VFDs, and the communications processor. These elements continue to be accepted in the ItemID of the variables but no longer exist in the namespace. If several VFDs were used by the OPC server in a configuration, these can be configured in STEP 7 solely for reasons of compatibility.

What characters are permitted for S7 connection names?

For the <connectionname>, you can use numbers "0-9", upper and lower case alphabetic characters "A-z" and the special characters "-+()". The connection name can be 24 characters long. The names are not case-sensitive.

Other printable characters are not permitted.

The connection names "SYSTEM" and "@LOCALSERVER" are reserved and must not be used.

Examples of connection names

Typical examples are:

- S7_connection_1
- S7 OPC connection

Connection objects

The following S7 connection objects exist:

- Productive S7 connections
S7 connections are used to exchange data between programmable controllers and normally configured in STEP 7.
- The DEMO connection
This is used only for testing.
- The @LOCALSERVER connection
This makes the local S7 data blocks for the S7 server functionality available.

2.6.4 Variable services

Variable services allow access to and monitoring of S7 variables on the programmable controller. S7 variables are addressed symbolically. The type of access is oriented on the notation of the S7 tools.

Objects on the programmable controller

The OPC server supports the following objects:

- Data blocks
- Instance data blocks
- Inputs
- Outputs
- Peripheral inputs
- Peripheral outputs
- Memory bits
- Timers
- Counters

Some S7 programmable controllers do not support all object types.

2.6.4.1 Syntax of the process variables for S7 variable services

Syntax

You have three options:

```
S7: [<connectionname>] DB<no>, {<type>}<address>{, <quantity>}
S7: [<connectionname>] DI<no>, {<type>}<address>{, <quantity>}
S7: [<connectionname>]<object>{<type>}<address>{, <quantity>}
```

Explanations

S7

S7 protocol for access to the process variable.

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

DB

Data block. Identifier for an S7 variable from a data block.

DI

Instance data block. Identifier for an S7 variable from an instance data block.

<no>

Number of the data block or instance data block.

<object>

Specifies the block/area type on the S7 PLC.

Possible values are as follows:

I	Input
Q	Output
PI	Peripheral input
PQ	Peripheral output
M	Bit memory
T, TBCD, TDA	Timer
C	Counter

Time base and range of values for S7 timer variables (type T):

The range of values of OPC process variables for S7 of the type timer (T) is decimal coded in ms. The timebase (for writing) is obtained from the value range as shown in the table below:

Possible Values	Time Base	Comment
1,000,000 ms to 9,990,000 ms	10 s	The values must be a multiple of 10,000 ms.
100,000 ms to 999,000 ms	1 s	The values must be a multiple of 1,000 ms.
10,000 ms to 99,900 ms	100 ms	The values must be a multiple of 100 ms.
10 ms to 9,990 ms	10 ms	The values must be a multiple of 10 ms. A smaller timebase is not possible. 0 ms is permitted but has no function.

The number of zeros on the right decides the timebase, the leading digits from 1 to 999 multiplied by the timebase decide the time.

Examples:

The value 90,000 ms has a timebase of 100 ms.

The value 123,000 ms has a timebase of 1 s.

The value 150 ms has a timebase of 10 ms.

The OPC data type of the S7 timer variable of the type T is a word (signed, VT_UI4).

Time base and range of values for S7 timer variables (type TBCD):

The range of values of OPC process variables for S7 of the type TBCD is BCD coded. The timebase (for writing) is obtained from the value range as shown in the table below:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Meaning (symbol)	0	0	x	x	t	t	t	t	t	t	t	t	t	t	t	t
Explanation:																
Meaning "0"	not relevant															
Meaning "x"	Specifies the timebase															
	Bits 13 and 12								Timebase in seconds							
	00								0.01							
	01								0.1							
	10								1							
	11								10							
Meaning "t"	BCD-coded time value (0 to 999)															

Example:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	0	1	0	0	0	0	0	1	1	1	0	1	0	1

Bits 0-11 specify the number 075. Bits 12 and 13 specify the timebase 0.1.
 $75 * 0.1 = 0.75$ seconds

The OPC data type of the S7 timer variable of the type TBCD is a word (unsigned, VT_UI2).

Timebase and range of values for S7 timer variables (type TDA):

With a simple timer variable of the type T, although timers can be operated easily, it is not possible to set all possible combinations of timebases and value ranges because the value ranges can overlap.

In this case, the S7 timer variable of the type TDA (decimal array) can be used.

<object>: TDA

Data type: Field of two words {timebase in ms VT_UI2 / time value VT_UI2}.

Range of values:

Timebase in ms: 10, 100, 1000, 10000.

Time value: 0 to 999; 0 is permitted but has no function.

Time ranges: 10 ms: 0 to 9990 ms; 100 ms: 0 to 99900 ms; 1000 ms: 0 to 999000 ms;
 10,000 ms: 0 to 9990000 ms

It is not possible to enter <quantity> for the TDA object.

Example:

Writing the value {100|50} to timer TDA3 initializes timer 3 with the value $50 * 100\text{ms} = 5000$

ms and this decrements 50 times in 100 ms steps.
This setting is not possible for the type *T*.

Range of values of S7 counter variables (type C):

The range of values of OPC process variables for S7 of the type counter (C) is 0 to 999 decimal coded.

The OPC data type of the S7 counter variable of the type C is a word (unsigned, VT_UI2).

<type>

S7 data type.

An S7 data type is converted to the corresponding OLE data type in the OPC server. All specified OLE data types can be read via the automation interface of OPC. However, certain development tools (such as Visual Basic) only offer a restricted number of data types. The following table lists the corresponding Visual Basic type in which the variable value can be represented.

For objects of the types *T*, *TDA* and *C* no type can be specified.

S7 data type	Description	OLE data type	Visual Basic type
X	Bit (bool). You specify the bit number (0 to 7).	VT_BOOL	Boolean
B	Byte (unsigned)	VT_UI1	Byte
W	Word (unsigned)	VT_UI2	Long
D	Double word (unsigned)	VT_UI4	Double
CHAR	Byte (signed)	VT_I1	Integer
INT	Word (signed)	VT_I2	Integer
DINT	Double word (signed)	VT_I4	Long
REAL	Floating point number	VT_R4	Single
String	String. You specify the string length. The string must be initialized with valid values on the S7.	VT_BSTR	String
DT	Date and time, 8 bytes BCD format (S7 CPU DATE-AND-TIME)	VT_DATE	Date
DATE	Date and time, 8 bytes The time is always 00:00:00, range of values starting at 01.01.1990. Mapping of the CPU data type DATE (unsigned, 16 bits).	VT_DATE	Date
TIME	Time value (signed), IEC format, in ms	VT_I4	Long
TOD	Time of day (unsigned), 0 to 86399999 ms	VT_UI4	Double
S5TIMEBCD	Mapping of the CPU data type S5TIME (unsigned, 16 bits) with a restricted range of values, 0 to 9990000 ms	VT_UI2	Long

Timebase and range of values for the S7 data type S5TIMEBCD:

The range of values of the time variable of data type S5TIMEBCD is BCD-coded. The range of values is according to the following table:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Meaning (symbol)	0	0	x	x	t	t	t	t	t	t	t	t	t	t	t	t
Explanation:																
Meaning "0"	not relevant															
Meaning "x"	Specifies the timebase															
	Bits 13 and 12								Timebase in seconds							
	00								0.01							
	01								0.1							
	10								1							
	11								10							
Meaning "t"	BCD-coded time value (0 to 999)															

Example:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	0	1	0	0	0	0	0	1	1	1	0	1	0	1

Bits 0-11 specify the number 075. Bits 12 and 13 specify the timebase 0.1.
 $75 * 0.1 = 0.75$ seconds

The OPC data type of the time variable of data type S5TIMEBCD is a word (unsigned, VT_UI2). When writing, the range of values is limited accordingly.

<address>

Address of the first variable to be addressed. Possible values are as follows:

- Byte offset
- Byte offset.bit (only for data type X)
 The value range of the bit address then allows 0 to 7.
- Byte offset string length (only for data type string, string length 1 byte to 254 bytes)

The value range for the byte offset is 0 to 65534. The actual usable value of the address may be lower depending on the device and type.

<quantity>

Number of variables of a type to be addressed starting at the offset specified in the *address* parameter. The value range depends on the configuration.

For data type X, the quantity for write access can only be entered in multiples of 8. The bit address must then be zero.

For data type X, the entry of a quantity is unrestricted.

Examples:

S7:[S7-OPC-1]DB1,X10.0,64, access rights RW

S7:[S7-OPC-1]DB1,X10.3,17, access rights R

2.6.4.2 Examples of process variables for S7 variable services

Here you will find examples illustrating the syntax of variable names for variable services.

Data block parameter

S7:[S7-connection-1]DB5,B12

DB5,B12

identifies data byte 12 in data block 5.

Instance data block parameter

S7:[S7-connection-1]DI5,W10,9

DI5,W10,9

identifies 9 data words starting at byte address 10 in instance data block 5.

Objects parameter

S7:[S7connection1]IB0

IB0

identifies input byte 0.

2.6.4.3 Examples of items with an optimum structure

The OPC server provides an optimization algorithm for the S7 variable services:

- OPC items that are read or monitored simultaneously should be arranged consecutively in the name space of the partner device. Small gaps between the relevant sections are processed but make for a poorer data throughput.
- OPC items that are written simultaneously must be arranged consecutively in the name space. For optimum write access, there must be no gaps.

If your OPC client does not need to use individual OPC items for process variables, you can use arrays directly for accessing the relevant data and assign the elements of the arrays read to individual process variables.

Organization of a data block for read access

DB10,W10

DB10,B12

DB10,B13

DB10,DW14

DB10,W20

This is executed as a read access to DB10,B10,12 via the communications system.

Despite the gap at byte 18 and byte 19, this is converted to a single read job. The data read unnecessarily is discarded. Instead of five single variable requests, only one request is transported over the network.

Organization of a data block for write access

DB10,W10

DB10,B12

DB10,B13

DB10,DW14

DB10,W20

This is executed via the communication system as two writes to the DB10,B10,8 array and the individual variable DB10,W20.

2.6.5 Bufferoriented services

Bufferoriented services allow programcontrolled transfer of larger blocks of data. Data transfer is implemented with variables:

- Variables that send blocks of data
- Variables that receive blocks of data

Up to 65534 bytes of payload data can be transferred regardless of the size of the PDU. The segmentation of the data is handled by the functions themselves.

Note

Bufferoriented services can only be used on twoended connections. The created connection configuration must be downloaded to the S7 programmable controller or to the relevant PCs.

Example of using buffer-oriented services

The following schematic illustrates how an S7400 sends a data packet to a PC station with the S7 OPC server.

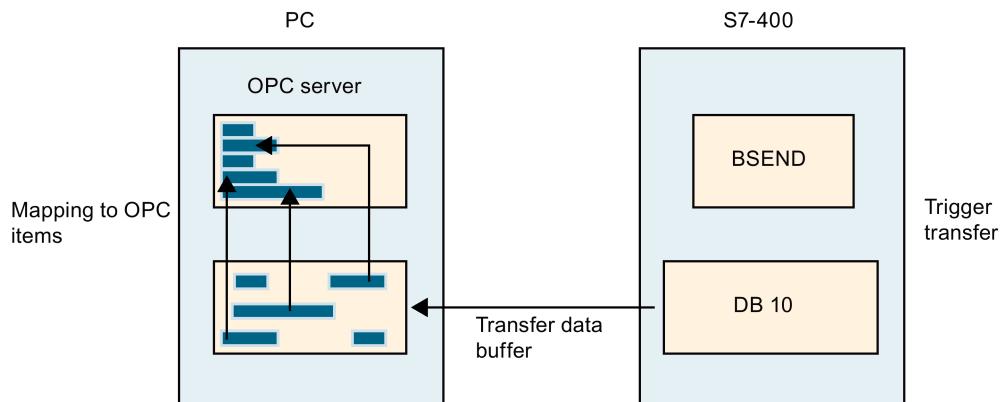


Figure 2-14 Sending a data packet from an S7-400 device to a PC station with S7 OPC server

To allow the OPC server to receive data, one or more OPC items of the type *BRCV* are added to an active group on the PC.

On the S7400, a user program starts the *BSEND* function block. *BSEND* starts the transfer of the entire data area as a buffer to the PC.

On the PC, the received data is transferred to the OPC server. The OPC server now maps the subareas of the block of data to the corresponding OPC items. If these OPC items are monitored, the OPC server sends a callback to the OPC client if the values change.

2.6.5.1 Syntax of the process variables for buffer-oriented services

Syntax

You have two options:

```
S7: [<connectionname>] BRCV,<RID>{,{<type>}<address>{,<quantity>}}  
S7: [<connectionname>] BSEND<length>,<RID>{,{<type>}<address>  
{,<quantity>}}}
```

Explanations

S7

S7 protocol for access to the process variable.

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

BRCV

BRCV contains the last block of data received from the partner. The content and length of the received data are specified by the sending partner.

The variable is readonly.

Use these variables for monitoring. The reception of a block of data by the OPC server is then signaled to an OPC client application.

An explicit job to read these variables returns when the connection times out (time specified in the configuration for the specific connection) or a block of data is received.

OLE data type	Visual Basic type
VT_ARRAY VT_UI1	Byte()

BSEND

BSEND contains the block of data to be sent to a partner device.

The block of data is sent to the partner only when the variable is written. A read gets the content of the last successfully transferred block of data.

OLE data type	Visual Basic type
VT_ARRAY VT_UI1	Byte()

<length>

Length in bytes of the block of data to be sent.

<RID>

ID of the addressing parameter. This is set for a block pair (*BSEND / BRCV*) and uniquely defined within a connection.

You can send several *BSEND* blocks over a connection or receive several *BRCV* blocks but each must have a different ID. The same IDs can be used on other connections. The value range for BSEND/BRCV is 0 to 4294967295.

<type>

S7 data type.

An S7 data type is converted to the corresponding OLE data type in the OPC server.

All specified OLE data types can be read via the automation interface of OPC. However, certain development tools (for example Visual Basic) only offer a restricted number of data types. The following table therefore lists the corresponding Visual Basic type in which the variable value can be represented.

Format identifier	Description	OLE data type	Visual Basic type
X	Bit (bool). You must specify the bit number.	VT_BOOL	Boolean
BYTE or B	Byte (unsigned)	VT_UI1	Byte
WORD or W	Word (unsigned)	VT_UI2	Long
DWORD or D	Double word (unsigned)	VT_UI4	Double
CHAR	Byte (signed)	VT_I1	Integer
INT	Word (signed)	VT_I2	Integer

Format identifier	Description	OLE data type	Visual Basic type
DINT	Double word (signed)	VT_I4	Long
REAL	Floating point number	VT_R4	Single
String	Character string. You need to specify the string length	VT_BSTR	String
DT	Date and time, 8 bytes BCD format	VT_DATE	Date
TIME	Time value (signed), IEC format, in ms	VT_I4	Long
TOD	Time of day (unsigned), 0 to 86399999 ms	VT_UI4	Double

<address>

Address of the first variable to be addressed.

Possible values are as follows:

byte number

bytetenumber.bit (only for X format)

The value range for the byte number is 0 to 65534. The actual usable value of the address may be lower depending on the device and type.

<quantity>

Number of variables of a type to be addressed starting at the address specified in the *address* parameter (range of values 0 to 65535).

Note

Reading (BRCV) and writing (BSEND) individual bits (format X) is possible. It is also possible to read and write arrays of individual bits without any restriction relating to the start address and array length within a block of data.

Example: *S7:[S7-OPC-1]BRCV,1,X10.4,78*

Note

Please note the following:

- By specifying the optional information type, address and quantity, you can define structured access to subareas of blocks of data.
- Variables for send data or receive data of different length or different RID have independent memory areas.
- The send data buffer is allocated and initialized with zero when an item is created for an independent memory area. A write to a BSEND item is written to the internal write buffer and transferred.
- The blocks of data are transferred acyclically. Simultaneous network jobs for the same data are possible. The complete block of send data is always transferred. This also applies to subelement access or when several clients write to this item at the same time. Writing simultaneously to the same block of send data or subareas of the block of send data by several clients can lead to inconsistencies. We therefore recommend,
 - that you always read or write the complete block of data,
 - in NCM S7 that you set the "Maximum number of parallel network jobs" to "one" although this reduces the transmission performance.

2.6.5.2

Examples of process variables for bufferoriented services

Here you will find examples illustrating the syntax of variable names for bufferoriented services.

Receive block of data in the entire buffer

S7:[S7-OPC-1]BRCV,1

BRCV,1

A block of data is received in the receive buffer with RID 1. The complete buffer is mapped on an array of bytes.

Alternate access to received block of data

S7:[S7-OPC-1]BRCV,1,W2,4

BRCV,1,W2,4

From the received block of data, the content is mapped starting at offset 2 on an array of 4 words. In total, 8 bytes of the block of data are relevant.

Transfer double word

S7:[S7-OPC-1]BSEND16,1,D2

BSEND16,1,D2

A double word is addressed in a block of data with a length of 16 with RID 1 starting at offset 2. If a write command is applied to the variable, the written value is entered at the specified position in the block and the block of data is sent.

transfer floatingpoint numbers

S7:[S7-OPC-1]BSEND32,1,REAL20,2

BSEND32,1,REAL20,2

An array with floatingpoint numbers is addressed in a block of data with a length of 32 with RID 1 starting at offset 20. If a write command is applied to the variable, the written value is entered at the specified position in the block and the block of data is sent.

2.6.6 S7specific information variables

There are S7specific variables with which you can obtain information about the S7 communication and the established connections.

The following information can be obtained:

- Attribute of a virtual field device (VFD)
- Status of an S7 connection
- Status of a virtual device
- Configured parameters and delay time parameters of the S7 connection for diagnostics

2.6.6.1 Syntax of the S7-specific information variables

S7-specific information variables

Syntax:

S7: [<connectionname>]<informationvariable>

Explanations

S7

S7 protocol for access to the process variable.

<connectionname>

Protocolspecific connection name. The connection name is specified in the configuration.

<informationparameter>

Possible values are as follows:

&identify()

Vendor attributes of a communication partner.

Return values (as elements of a string field):

- *Vendor*
- *Model*
- *Revision*

OLE data type	Visual Basic type
VT_ARRAY of VT_BSTR	String()

&vfdstate()

State of a virtual field device.

Data type: *VT_ARRAY of VT_VARIANT* with the following components:

Logical status

Supported services

Return Value:

- *S7_STATE_CHANGES_ALLOWED*

All services are permitted

OLE data type	Visual Basic type
VT_BSTR	String

Physical status

Operability of the real device

Return values:

- *S7_OPERATIONAL*
the real device is operational
- *S7_NEEDS COMMISSIONING*

the real device can only be used after local settings have been made

OLE data type	Visual Basic type
VT_BSTR	String

Detailed Information on the local VFD state.

The state is returned as an octet string. For more detailed information on the meaning of the return value, refer to the documentation of the partner device.

OLE data type	Visual Basic type
VT_ARRAY of VT_BSTR	String()

&statepath()

State of a communication connection to a partner device.

Return values:

- *DOWN*
Connection is not established
- *UP*
Connection is established
- *RECOVERY*
Connection is being established
- *ESTABLISH*
(reserved for future expansions)

OLE data type	Visual Basic type
VT_BSTR	String

&statepathval()

State of a communication connection to a partner device.

Return values:

- *1*
Connection is not established
- *2*
Connection is established
- *3*
Connection is being established
- *4*
(reserved for future expansions)

OLE data type	Visual Basic type
VT_UI1	Byte

Examples of S7-specific information variables and return values

Here you will find examples illustrating the syntax of S7-specific information variable names.

Information on the vendor attributes of a virtual device

S7:[S7-OPC-1]&identify()

&identify()

can, for example, return the following values:

- Vendor: *SIEMENS AG*
- Model of the virtual device: *6ES7413-1AE0-0AB0*
- Revision: *V1.0*

Status of a device

S7:[S7-OPC-1]&vfdstate()

&vfdstate()

can, for example, return the following values:

- Logical status: *S7_STATE_CHANGES_ALLOWED*
- All services are permitted.
- Physical status: *S7_OPERATIONAL*
- The real device is operational.
- Detailed information: *02.00.00*
- Detailed information on the local VFD status.

Status of a communications connection as string

S7:[S7-OPC-1]&statepath()

&statepath()

can, for example, return the following values:

- Connection status: *RECOVERY*
- *The connection is currently being established.*

Status of a communications connection as a number

S7:[S7-OPC-1]&statepathval()

&statepathval()

can, for example, return the following values:

- Connection status: *2*
- *The connection is established.*

2.6.6.2 S7-specific diagnostics variables

The following diagnostics variables are available for checking the configuration parameters and current runtime parameters for an S7 connection. In the OPC Scout V10, there is a separate diagnostics view; the variables are also displayed in the namespace in the "Configuration" folder below the relevant S7 connection.

Syntax

S7: [<connectionname>]<diagnosticsvariable>

Diagnostics variable	Description						
&vfd()	Name of the OPC server to which the connection is assigned. This normally has the text "OPC Server" for connections configured in NCM. Data type VT_BSTR, read-only.						
&cp()	Name of the interface parameter assignment to which the connection is assigned. Data type VT_BSTR, read-only.						
&remoteaddress()	Address of the connection partner. Data type VT_BSTR, read-only. The address of the connection partner is a data buffer with a data length dependent on the connection type. To make it more manageable for evaluation by the user, the data buffer is displayed formatted in a string. PROFIBUS address Format: "ddd" (1-3 decimal digits) IP address (ISO-on-TCP) Format: "ddd.ddd.ddd.ddd" (each 1-3 decimal digits) MAC address (ISO) Format: "xx-xx-xx-xx-xx-xx" (each 2 hexadecimal digits)						
&localsap()	Local SAP of the connection. Data type VT_BSTR, read-only. The local SAP of the connection partner is a data buffer with a data length dependent on the connection type. To make it more manageable for evaluation by the user, the data buffer is displayed formatted in a string. Format: "xx.xx" (each 2 hexadecimal digits)						
&remotesap()	Remote SAP of the connection. Data type VT_BSTR, read-only. The remote SAP of the connection partner is a data buffer with a data length dependent on the connection type. To make it more manageable for evaluation by the user, the data buffer is displayed formatted in a string. Format: "xx.xx" (each 2 hexadecimal digits)						
&connect()	Type of connection establishment. Data type VT_UI4, read-only. <table border="1" data-bbox="531 1471 1453 1598"> <tr> <td>0</td><td>Passive, connection is kept established permanently.</td></tr> <tr> <td>1</td><td>Active, establish connection only when necessary, connection termination if not used after wait time.</td></tr> <tr> <td>2</td><td>Active, connection is kept established permanently.</td></tr> </table>	0	Passive, connection is kept established permanently.	1	Active, establish connection only when necessary, connection termination if not used after wait time.	2	Active, connection is kept established permanently.
0	Passive, connection is kept established permanently.						
1	Active, establish connection only when necessary, connection termination if not used after wait time.						
2	Active, connection is kept established permanently.						
&abortconnectionafter()	Automatic connection termination. Delay time for the automatic connection termination: After this time has elapsed, the OPC server automatically terminates the connection if there was no further variable access during this time. This allows the number of necessary connections to be reduced when variables are accessed at long intervals. Data type VT_UI4, read-only. 0: no termination >0: idle time before termination in ms						

Diagnostics variable	Description
&optimizes7read()	Read accesses for S7 block access are optimized. Data type VT_BOOL, read-only. "True": Optimization "False": no optimization Optimization means: Several access jobs to individual variables are converted internally to a single array access to the communications partner.
&optimizes7write()	Write accesses for S7 block access are optimized. Data type VT_BOOL, read-only. "True": Optimization "False": no optimization Optimization means: Several access jobs to individual variables are converted internally to a single array access to the communications partner.
&autopasswordreset()	Automatic resetting of the S7 password for block access Data type VT_BOOL, read-only. "True": Resetting enabled "False": Resetting disabled On an S7, the domain services enabled by password remain active until explicitly reset. Automatically resetting the password when the connection is established makes sure that the password is not enabled for an unnecessarily long time especially in conjunction with automatic termination when a connection is not in use.
&fastconnectionstate-returnenable()	Fast return of a write/read job if the connection is interrupted. Data type VT_BOOL, read-only. "True": enabled "False": disabled
&connecttimeout()	Connection establishment timeout Data type VT_UI4, read-only 0: No timeout >0: Timeout in ms
&timeout()	Job timeout for productive jobs in S7 communication. Data type VT_UI4, read-only 0: No timeout >0: Timeout in ms
&receivecredit()	Maximum number of simultaneous network jobs, receive direction Proposed value for connection establishment Data type VT_UI2, read-only >=1, proposed value for connection establishment
&tradedreceivecredit()	The number of simultaneous protocol jobs in the receive direction negotiated after connection establishment. Data type VT_UI2, read-only If the connection is interrupted, the quality of this variable is "BAD".
&sendcredit()	Maximum number of simultaneous network jobs, send direction Proposed value for connection establishment Data type VT_UI2, read-only >=1, proposed value for connection establishment Is set along with &receivecredit() in the configuration.

Diagnostics variable	Description	
&tradedsendcredit()	<p>The number of simultaneous protocol jobs in the send direction negotiated after connection establishment.</p> <p>Data type VT_UI2, read-only</p> <p>If the connection is interrupted, the quality of this variable is "BAD".</p>	
&pdusize()	<p>Size of the PDU</p> <p>Proposed value for connection establishment</p> <p>Data type VT_UI2, read-only</p> <p>>=1, proposed value for connection establishment</p>	
&tradedpdusize()	<p>Size of the PDU, negotiated after connection establishment.</p> <p>Data type VT_UI2, read-only</p> <p>If the connection is interrupted, the quality of this variable is "BAD".</p>	
&defaultalarmseverity()	<p>Default alarm severity for unconfigured alarm events.</p> <p>Data type VT_UI2, read-only</p> <p>1: low priority</p> <p>...</p> <p>1000: high priority</p> <p>In the configuration, there is only one option for the default priority of alarms for S7 alarms and S7 diagnostics alarms.</p>	
&defaultdiagnosisseverity()	<p>Default alarm severity for unconfigured diagnostics events.</p> <p>Data type VT_UI2, read-only</p> <p>1: low priority</p> <p>...</p> <p>1000: high priority</p> <p>In the configuration, there is only one option for the default priority of alarms for S7 alarms and S7 diagnostics alarms.</p>	
&events()	<p>Registering alarms and events on the connection partner.</p> <p>Data type VT_UI4, read-only.</p> <p>The individual values can be combined.</p>	
	0x000000 01	SCAN item (no longer supported)
	0x000000 02	Simple alarms (no longer supported)
	0x000000 04	Simple symbol-related alarms (no longer supported)
	0x000000 08	Simotion TO alarms
	0x000000 10	Connection monitoring alarms
	0x000000 20	Block-related alarms (as conditional events)
	0x000000 40	Symbol-related alarms (as conditional events)
	0x000000 80	Diagnostics alarms

Diagnostics variable	Description
&connectiontype()	S7 connection type Data type VT_UI2, read-only 2:S7D_STD_TYPE; standard connection 3:S7D_H_TYPE; fault-tolerant connection If the S7 connection is not yet established, the quality "BAD" is reported for this item and the values are invalid.
&connectionstate()	S7 connection state Data type VT_UI2, read-only 0x11:STD_DOWN; standard connection deliberately terminated 0x12:STD_ABORT; standard connection unintentionally terminated (error) 0x13:STD_NOT_USED; standard connection was never established 0x14:STD_OK; standard connection established 0x20:H_OK_RED; fault-tolerant connection established (redundant) 0x21:H_OK_RED_PATH_CHG; fault-tolerant connection established (redundant, there was a failover) 0x22:H_OK_NOT_RED; fault-tolerant connection established not redundant 0x23:H_ABORT; fault-tolerant connection unintentionally terminated (error) 0x24:H_NOT_USED; fault-tolerant connection was never established 0x25:H_DOWN; fault-tolerant connection deliberately terminated The data variable differs from the &statepath() variable in that it reflects the connection status from the perspective of the protocol stack and also delivers additional information about the redundant H connections used (transparent to the OPC server).
&hconnectionwaystate()	State of the H connection paths Data type VT_ARRAY VT_UI2 The values describe the stat of a path of the H connection. 0x30:HW_PROD; path is productive connection 0x31:HW_STBY; path is standby connection 0x32:HW_ABORT; path was unintentionally terminated (error) 0x33:HW_NOT_USED; path was never established 0x34:HW_DOWN; path was deliberately terminated 0x35:HW_CN_BREAK; path could not be established If the S7 connection is not yet established, the quality "BAD" is reported for this item and the values are invalid.

2.6.6.3 Syntax of the systemspecific information variables

S7: [SYSTEM] &version()

Note

The connection name configured in STEP 7 must *not* be "SYSTEM" because this is a reserved name.

&version()

Returns a version ID for the S7 OPC Server, here, for example, the string
SIMATIC NET Core Server S7 V 7.xxxx.yyyy.zzzz Copyright 2012

Data type: VT_BSTR

Access right: readonly

S7: [SYSTEM] &sapiversion ()

&sapiversion()

Returns a version ID of the SAPI S7 user interface.

Data type: VT_BSTR

Access right: readonly

2.6.7 Block management services

Block management services control the transfer and return of data and program elements between the PC and the programmable controller. On the PC, data and program elements are stored in files. Block management services can be protected by a password.

Blocks

Data and program elements are stored on S7 programmable controllers in blocks. These blocks are compiled using STEP 7 and transferred to the S7 device. The following blocks can exist on S7 devices:

- Organization blocks
- Function blocks
- Data blocks (DB / DI)

Block management functions

You can execute the following functions via the SIMATIC NET OPC Server:

- Transfer blocks between PCs and programmable controllers
- Delete blocks
- Link blocks
- Compress the memory of the programmable controller

Note

It is not possible to create blocks with the OPC server, you must use STEP 7.

2.6.7.1 Syntax of the control variables for block management services

Syntax

S7: [<connectionname>]<serviceparameter>

Explanations

S7

S7 protocol for access to the process variable.

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

<serviceparameter>

These variables can only be written. The service is triggered by writing a value.

Possible values are as follows:

&blockread()

A block from a programmable controller is transferred to the PC and stored in a file.

The written value contains the parameter setting for the service. It is represented as an array of variants with the following elements:

Flags

The following hexadecimal numbers are possible:

0x0001	An unlinked block is read. If a target file exists, it is not overwritten, an error message is displayed.
0x0040	A linked block is read. If a target file exists, it is not overwritten, an error message is displayed.
0x1001	An unlinked block is read. An existing destination file is overwritten.
0x1040	A linked block is read. An existing destination file is overwritten.

OLE data type	Visual Basic type
VT-BSTR	String

Block

Block type and number

OB	Organization block
FB	Function block
FC	Function
DB	Data block

OLE data type	Visual Basic type
VT-BSTR	String

File

Full path of the file in which the block will be stored

OLE data type	Visual Basic type
VT-BSTR	String

&blockwrite()

Transfer a block from a PC to a programmable controller. After the transfer, the block on the programmable controller is still in the passive (not linked in) status. The block must be set to the linked state using the *&blocklinkin* function.

The written value contains the parameter settings for this service. It is represented as an array of variants with the following elements:

Flags

The following hexadecimal numbers are possible:

0x1000	An unlinked block of the same name on the automation system is to be overwritten
0x0000	An unlinked block of the same name on the automation system is not to be overwritten

OLE data type	Visual Basic type
VT-BSTR	String

File

Full path of the file in which the block is stored

OLE data type	Visual Basic type
VT-BSTR	String

Note

The domain services *blockread()* and *blockwrite()* do not allow access to source and target files of the blocks on a network drive.

Overwriting (0x1000) a block is only possible after the executable part has been linked in.

&blocklinkin()

Link a block with the passive status into the program sequence of the programmable controller. The executable part of the block is copied to the work memory of the programmable controller.

The block is then accessible to the program on the automation system. By linking a block, an existing, active block is overwritten without an error message.

The written value contains the specification of the block to be linked as the parameter setting for the service.

Block

Block type and number.

The following types are possible:

OB	Organization block
FB	Function block
FC	Function
DB	Data block

OLE data type	Visual Basic type
VT-BSTR	String

Note

The S7 connection must not be terminated between a *blockwrite()* and a *blocklinkin()* otherwise the blocks on the remote CPU will be deleted again immediately. The *blocklinkin()* function then no longer works.

&blockdelete()

Delete a block on the programmable controller. Both the passive blocks and those linked into the program sequence can be removed.

The written value contains the parameter setting for this service. It is represented as an array of variants with the following elements:

Flags

The following hexadecimal numbers are possible:

0x0001	An unlinked block is deleted
0x0040	A linked block is deleted
0x0041	The linked and the unlinked block are deleted

OLE data type	Visual Basic type
VT-BSTR	String

Block

Block type and number.

The following types are possible:

OB	Organization block
FB	Function block
FC	Function
DB	Data block

OLE data type	Visual Basic type
VT-BSTR	String

&blockcompress()

Compress the memory of the programmable controller.

Fragmented memory areas are moved together to create space for new blocks to be transferred.

This variable can only be written. The service is triggered by writing an empty string.

Note

The parameters of some of the block management services are set by the written value. Here, arrays are used as the data type. From the user's point of view, it may be advantageous to transfer the value parameters as a value of the data type *VT_BSTR*. The OPC server converts the values into the required data type automatically. The representation of an array in a string is, for example: *{firstelement/secondelement/thirdelement}*. You should also refer to the examples.

2.6.7.2 Examples of using the block management services

Here you will find examples illustrating the syntax of variable names for block management services.

Reading a block

S7:[S7-OPC-1]&blockread()

&blockread()

For example, to read the block *OB1* in a file "c:\temp\ob1.blk", the item must be written with the following value:

{0x0040|OB1|c:\temp\ob1.blk}

Note

To specify the value as in this example, string must be requested as the data type. The value can then be specified in the converted representation for arrays.

Writing a block

S7:[S7-OPC-1]&blockwrite()

&blockwrite()

For example, to send the block in the file "c:\temp\ob1.blk" to the programmable controller, the item must be written with the following value:

{0x1000/c:ltemplob1.blk}

Note

To specify the value as in this example, string must be requested as the data type. The value can then be specified in the converted representation for arrays.

Example of linking a block

S7:[S7-OPC-1]&blocklinkin()

&blocklinkin()

For example, to link the block *DB1* into program execution on the programmable controller, the item must be written with the following value:

DB1

Example of deleting a block

S7:[S7-OPC-1]&blockdelete()

&blockdelete()

To delete a linked block *DB1*, the item must be written with the following value:

{0x0040/DB1}

Example of compressing the memory of the programmable controller

S7:[S7-OPC-1]&blockcompress()

&blockcompress()

To compress memory, the item must be written with an empty string.

2.6.8**Passwords**

Write and read access by block management services to the CPU can be protected by a password during configuration. The password has higher priority than the keyswitch of the CPU.

Protection levels

There are three protection levels for S7 programmable controllers. These are activated with the aid of STEP 7:

- Protection due to the setting of the keyswitch
- Write protection
- Write and read protection

When the correct password is transferred, all the protection levels for this connection are revoked. When an empty string is transferred as the password, the connection protection is reactivated.

A default protection level can be sent during configuration for twoended connections. If the protection for twoended connections is not activated by the configuration, all services are possible on these connections even when the keyswitch is set to *RUN*.

Connections configured as single-ended are dependent on the keyswitch setting. The following table shows the relationship between password transfer and the protection level on connections configured at one end.

Oneended configured connections	Resulting Active Protection Level				
Configured protection level	Password was <i>not</i> transferred		Correct password was transferred		
	Keyswitch setting		Keyswitch setting		
	RUN	RUN-P, STOP	RUN	RUN-P, STOP	
Keyswitch setting RUN - can be canceled by password	Read	All	All	All	All
Keyswitch setting RUN - can <i>not</i> be canceled by password	Read	All	Read	All	All
Write protection with password	Read	Read	All	All	All
Write/read protection	None	None	All	All	All

Key:

All = all services possible

Read = only read block

None = no block services allowed

2.6.8.1 Syntax of the control variables for passwords

Syntax

S7: [<connectionname>] &password()

Explanations

S7

S7 protocol for access to the process variable.

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

&password()

Identifier for the password service.

The transfer of the password is triggered by writing a value.

The password can be transferred in the following representations:

- *Octet string*
String with the hexadecimal codes of the individual characters of the password separated by a period (maximum of 8 characters)
- *String*
Any alphanumeric string starting with the "&" character (maximum 8 characters)

Writing the password returns the following values:

- *OPC_E_BADRIGHTS*
The password is invalid
- *S_OK*
The password is valid

OLE data type	Visual Basic type
String	String

2.6.8.2 Examples of using passwords

Here, you will find an example illustrating the transfer of passwords.

Example of transferring a password

S7:[S7-OPC-1]&password()

&password()

To transfer the password *SetMeFre* to the programmable controller, the variable must be written with the following value:

&SetMeFre

2.6.9 Server services

Access to DB 1

The PC station with an active OPC server for the S7 protocol provides an S7 data block for reading and writing and therefore becomes the S7 server. With PUT and GET function blocks, S7 stations can write or read the data block of the PC station in their program. Connection establishment is possible from both ends.

The data block (DB 1) is 65535 bytes long. There are no symbolic names nor structuring of the data block according to variables. The number of the data block can be queried both remotely and locally, for example by a SIMATIC NET OPC server with OPC Scout and displayed when browsing the namespace.

Only one data block is available. No bit memory, inputs, outputs, counters, or timers. It is not necessary to configure or download the block.

Over a local S7 connection *@LOCALSERVER*, an OPC client (for example the OPC Scout) can read or write the values of the data block. The OPC client can also monitor the data block and be notified of data changes.

If more than one client attempts to write the same data areas, each job is completed fully one after the other (data consistency despite simultaneous access).

When the PC station restarts, the values are retained in the data block (permanent data).

Note

To operate the S7 server services, a local OPC client must be active with S7 items on the PC station. If connection establishment is configured when necessary in NCM, the S7 items must contain the S7 connections over which the remote partner will read or write data.

Alternatively, permanent connection establishment can be configured. Passive connections require a permanent connection establishment. Any S7 item can then be activated to start the S7 OPC server.

Note

The S7 server services are released only with OPC server V6.3 configurations of the PC station.

2.6.9.1

Example of using server services

An S7 client could, for example, be an S7 300 station that reports status data to the PC station without the status data needing to be polled continuously, see figure below. It then (occasionally) writes status values to the data block.

A local client on the PC station can be informed of changes to the status values, see figure "Monitoring by regular cyclic reading of a value...". This avoids continuous polling of the status values on the S7 station.

The following schematic illustrates how an S7 device writes data to a PC station with S7 OPC Server.

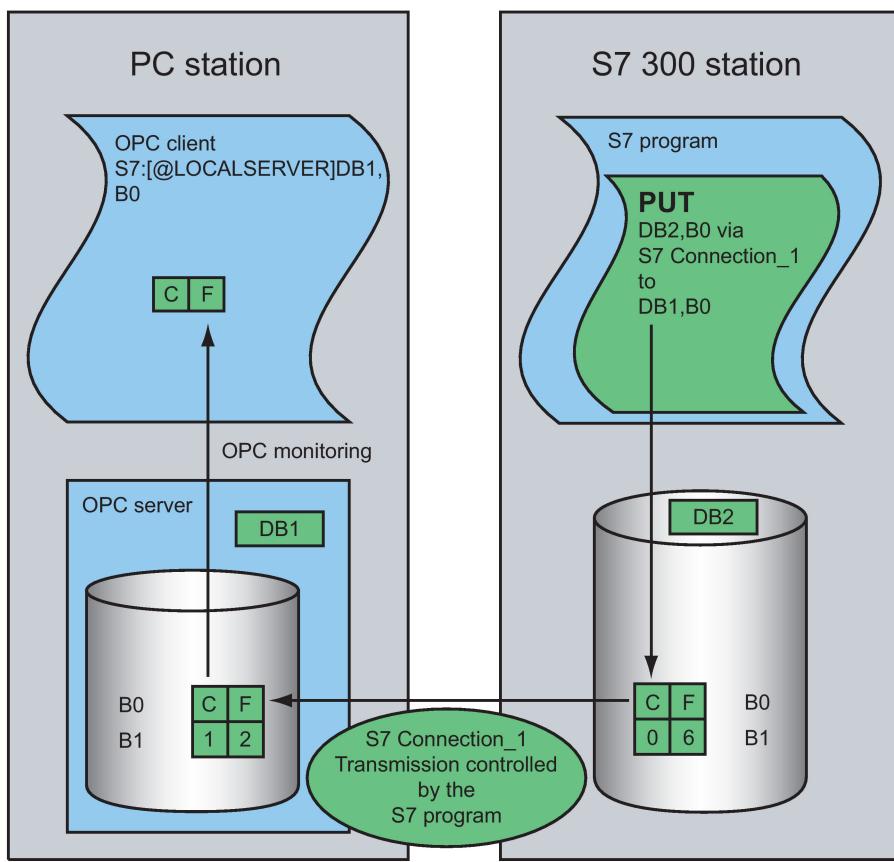


Figure 2-15 Server service: Occasional writing (PUT) of a value to relieve the S7 station

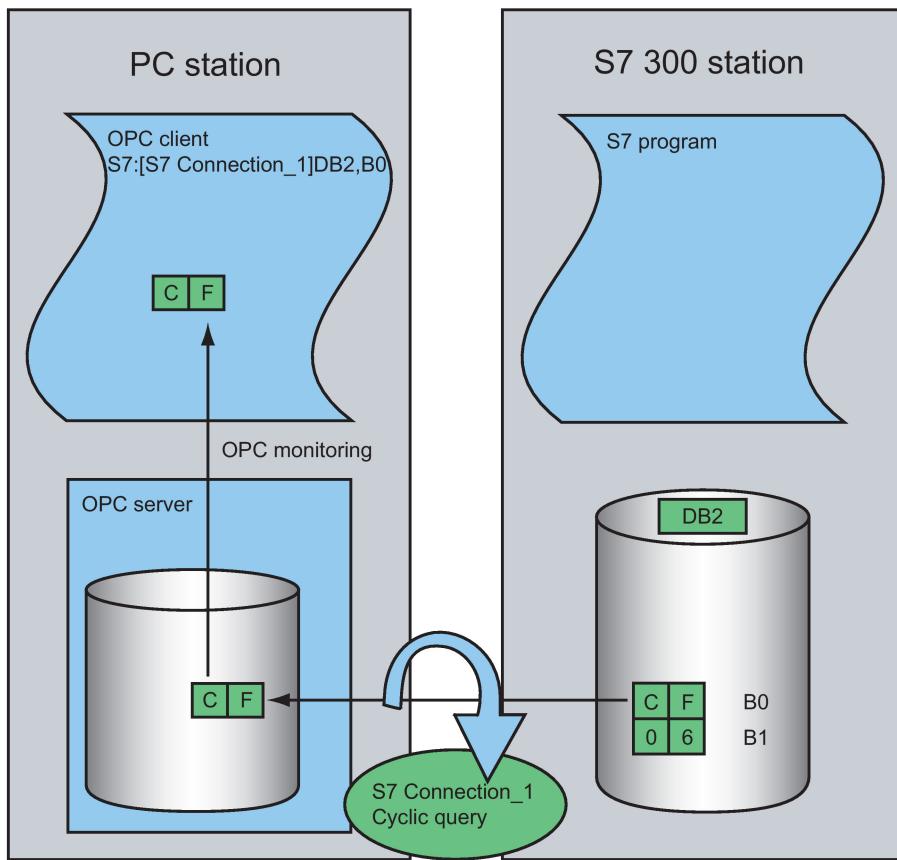


Figure 2-16 Monitoring by regular cyclic reading of a value (polling); extra load on the S7 station

2.6.10 S7 template data variables

With the process variables for the S7 protocol, you have flexible setting options with which you can obtain the process data of the plant in the data formats you require.

The wide variety of addressing options cannot, however, be put together in a fully browsable namespace. Even a data block with the length of a single byte has approximately 40 different data format options - starting with Byte, Char, arrays with one element of these, individual bits, arrays of bits with up to 8 array elements starting at different bit offsets.

The OPC server therefore supports the user with these so-called template data variables in the S7 namespace. In a typical OPC client text input box, these templates can be converted to valid ItemIDs simply by changing a few characters.

Example:

```
S7: [S7-connection_1]DB<db>,DWORD<o> //Template for a DWORD of a data block
```

By replacing <db> with the block address and <o> the offset within the data block, the user obtains a valid item syntax.

```
⇒ S7: [S7-connection_1]DB100,d3
```

Further example:

```
S7:[S7-connection_1]bsend<len>,<rid>,x<o>.<bit>,<c>//Template for a bit array, buffer  
service
```

```
⇒ S7:[S7-connection_1]bsend100,21,x0.0,24
```

Note

The usability of S7 OPC UA template data variables can be enabled and disabled in the "Communication Settings" configuration program in "OPC protocol selection" > click the arrow symbol beside "S7".

2.6.10.1 Template data variables in the namespace

The template data variables are located in separate folders in the namespace.

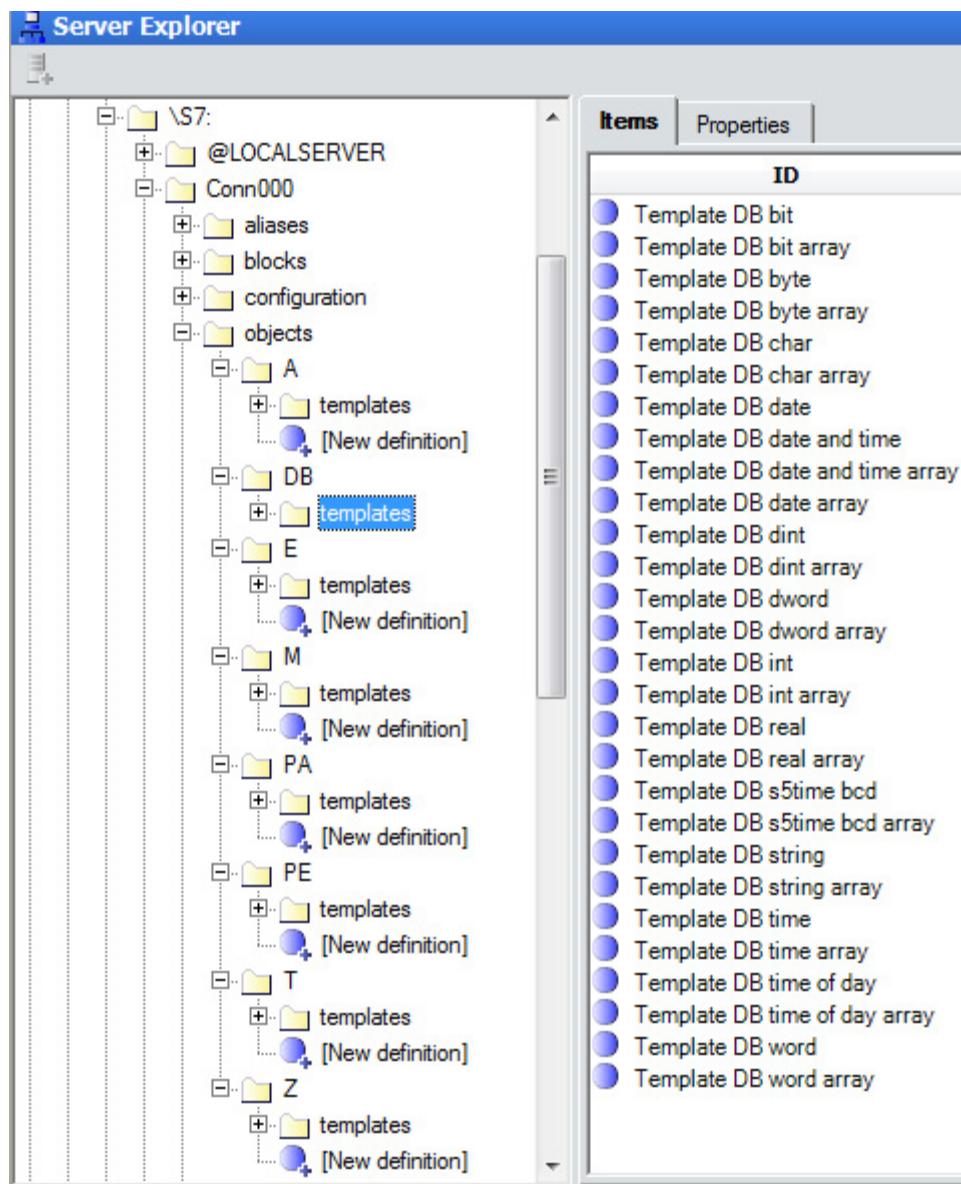


Figure 2-17 Hierarchy of the template data variables

The name of the leaf in the namespace provides considerable help ("Template M Byte").

2.6.10.2 Syntax of the template data variables

The following templates exist:

For bit memory, inputs and outputs

```

S7: [<connectionname>]<object><type><o>,<c>

<object>:=   "I" | // input
              "Q" | // output
              "M" | // memory bit
              "PI" | // peripheral input
              "PQ" // peripheral output

<type><o>:=   "X<o>.<bit>" | // Bit <bit>= template for the bit address
                "B<o>" | // byte (unsigned)
                "W<o>" | // word (unsigned)
                "D<o>" | // double word (unsigned)
                "CHAR<o>" | // byte (signed)
                "INT<o>" | // word (signed)
                "DWORD<o>" | // double word (signed)
                "REAL<o>" | // floating point 4 bytes
                "DT<o>" | // date and time, 8 bytes BCD format
                "DATE<o>" | // date and time, 8 bytes, time always 00:00:00
                "TIME<o>" | // time value (signed), IEC format, in ms
                "S5TIMEBCD<o>" | // time variable (unsigned, 16 bits), 0 to 9990000 ms
                "TOD<o>" | // time of day (unsigned), 0 to 86399999 ms
                "STRING<o>.<len>" // character string. <len> is a template for the string length

```

<o> // Template for the address of the first variable, this is a byte offset in the address range.

<c> // Template for the number of variables of a type to be addressed starting at the offset specified in the address parameter.

For data blocks

```

S7: [<connectionname>]DB<db>,<type><o>,<c>

<db> // template for the number of the data block

```

Buffer send/receive services

```

S7: [<connectionname>]bsend<len>,<rid>,<type><o>,<c>
S7: [<connectionname>]brcv,<rid>,<type><o>,<c>

<len> // template for the length in bytes of the block of data to be sent.

<rid> // template for the ID of the addressing parameter. This is specified for a block pair (BSEND/ BRCV) and defined uniquely within a connection.

```

S7 timers

`S7:[<connectionname>]TDA<i> // S7 timer variable (type TDA):`
`S7:[<connectionname>]TBCD<i>{,<c>} // S7 timer variable (type BCD):`

Counter

`S7:[<connectionname>]C<i>{,<c>} // counter, the address information <i> is a counter number.`

2.6.11 Unconfigured S7 connection

Access to a partner without configuration

Normally, connections to partner devices are defined in a configuration. This is done in the STEP 7 or NCM PC programs.

There are, however, situations in which, for example, data without a configured connection must be read from a partner device or variables written or monitored. It is possible to implement these tasks without configuration.

Requirements

For device access without configuration, all the data of the partner device relevant for communication must be known. These include the connection name, the access point and the station address. The required parameters are listed in the following section. When selecting the connection name, make sure that it is unique and does not already exist among the names already configured.

Service Access Point (SAP)

The service access point is a point at which a layer of the ISO/OSI reference model provides its services to the next higher layer.

The entire exchange of information between two neighboring layers takes place via the service access points. This is the interface between the higher and lower layer. If the SAP is between layer 3 (network layer) and layer 4 (transport layer), the service access point is known as the NSAP, Network Service Access Point, and if the SAP is between layer 4 and layer 5 (session layer), it is known as the TSAP or Transport Service Access Point.

Normally, certain resources and communications partners are assigned to a service access point. and the SAPs required for communication are therefore identified with unique names and numbers.

Parameters for an unconfigured connection

An unconfigured connection has the following syntax:

`S7:[<connectionname>]<VFD>|<accesspoint>|<address_specification><dataelement>`

The individual components have the following significance:

<connectionname>

The connection name must not exist already. If the selected connection name has already been configured and is used for another freely specified S7 connection, the additional information for freely specified connections will be ignored and the item will be assigned to the existing connection.

<VFD>

a VFD name is the same as the application name in the PC station configuration of NetPro. Any VFD name. All connections can be created on the same VFD. The VFD name can be a maximum of 32 characters long. The selected name must not already exist.

<accesspoint>

The access point of the communications module must be configured in advance with the "Communication Settings" program.

<address_specification> = <local TSAP>,<stationaddress>,<remote TSAP>,<mode>

The address specification contains the following information with the individual values separated by commas:

- **Local TSAP (Local Transport Service Access Point)**

For the S7 protocol, the local TSAP consists of exactly two numbers separated by blanks or the period with the following significance:

- The first byte can include a device ID, permitted values are *02* or *03*.
02 OS (operating station control and monitoring)
03 miscellaneous
- The second byte is always 0.

Recommended setting: 02.00

- **stationaddress**

There are three possible representations for the station address:

Transmission technique	Representation of the station address	Example
PROFIBUS	PROFIBUS address, decimal representation	65
TCP/IP	TCP/IP address	192.168.0.7
ISO	MAC address	08-06-05-e4-3a-00

remote TSAP (Remote Transport Service Access Point)

The representation is the same as for the local TSAP, however the second byte has a different meaning:

- First byte: contains a device ID, permitted values are *02* or *03*.
 - 02* OS (operating station control and monitoring)
 - 03* miscellaneous
 Recommended setting: 02
- Second byte: contains the addressing of the SIMATIC S7-CPU, divided into:
 - bits 7 to 5* rack (subsystem) of the S7-CPU
 - bits 4 to 0* slot of the S7-CPU

Note

When setting the "local TSAP" and the "remote TSAP", it is advisable to select the same setting for the first byte.

Mode

S7 connections are actively established either by the OPC server or by the connection partner. On such connections, it is also possible to use optimized write or read access. The following values are possible for mode:

- 1: Active connection establishment of the OPC server with optimization
- 3: Active connection establishment of the OPC server without optimization

<dataelement>

Here, for example, a data block with its number and the type (byte, word etc.) and the address (for example byte offset) can be specified. Which data elements are possible for S7 is described in the section "S7 communication (Page 116)".

Note

Further connection parameters:

The PDU size cannot be specified. A maximum size of 960 bytes is proposed during connection establishment and the maximum possible size of the partner device is negotiated.

Fixed values (in each case, 15,000 ms) I use for the connection establishment timeouts and job timeouts.

The proposed value for the maximum number of parallel network jobs is 2.

Examples

S7:[S7-connection 1/VFD1/S7ONLINE/01.00,192.168.0.7,02.02,1]DB10,B0

S7:[S7_connection 2/VFD2/S7ONLINE/02.00,65,02.02,1]DB10,B0

S7:[S7_connection 3/VFD3/S7ONLINE/03.00,08.06.05.e4.3a.00,02.02,1]MB0

Defining the access point

1. Open the "Communication Settings" configuration program to define an access point and to assign an interface parameter assignment.

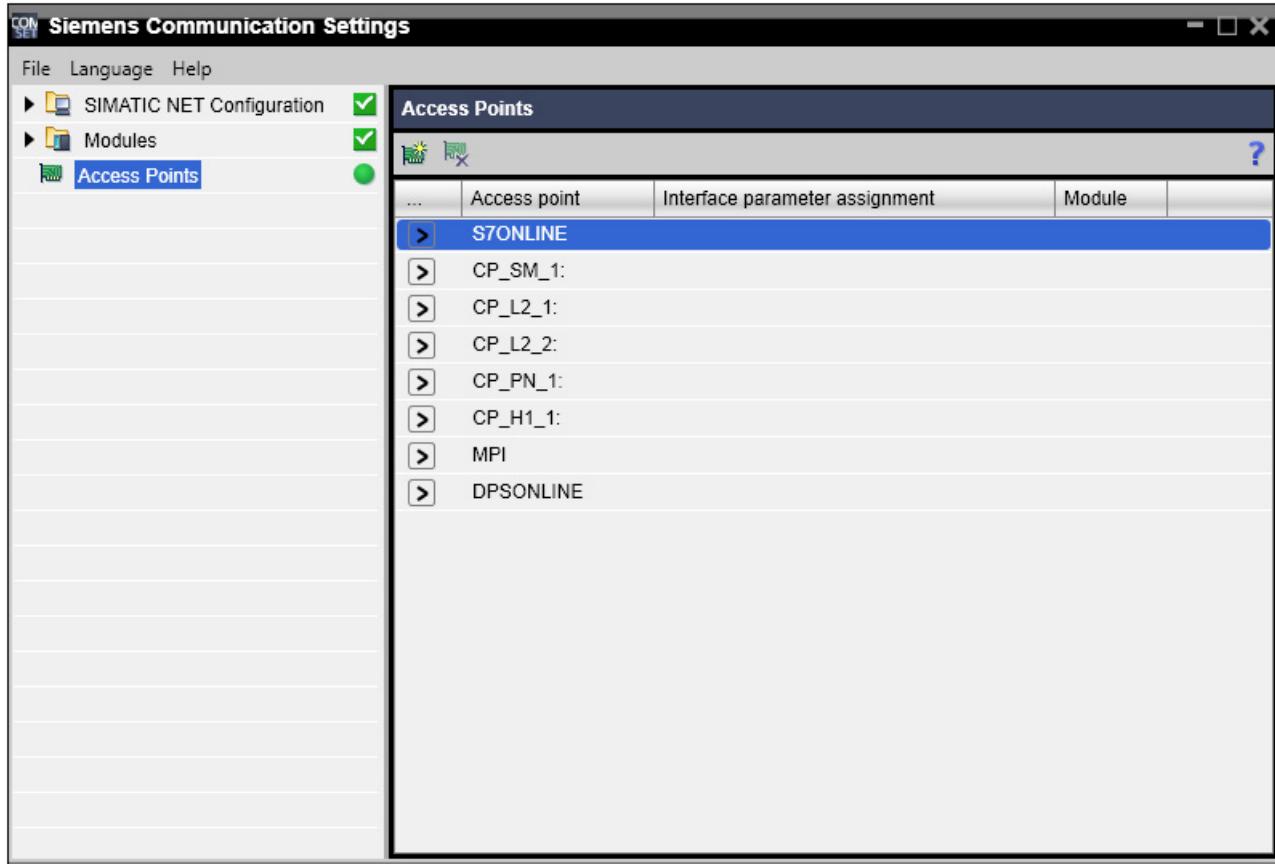


Figure 2-18 Opening the "Communication Settings" configuration program

2. Click on the arrow symbol next to "S7Online".
3. Select the interface parameter assignment for the access point using the "Associated interface parameter assignment" drop-down list.

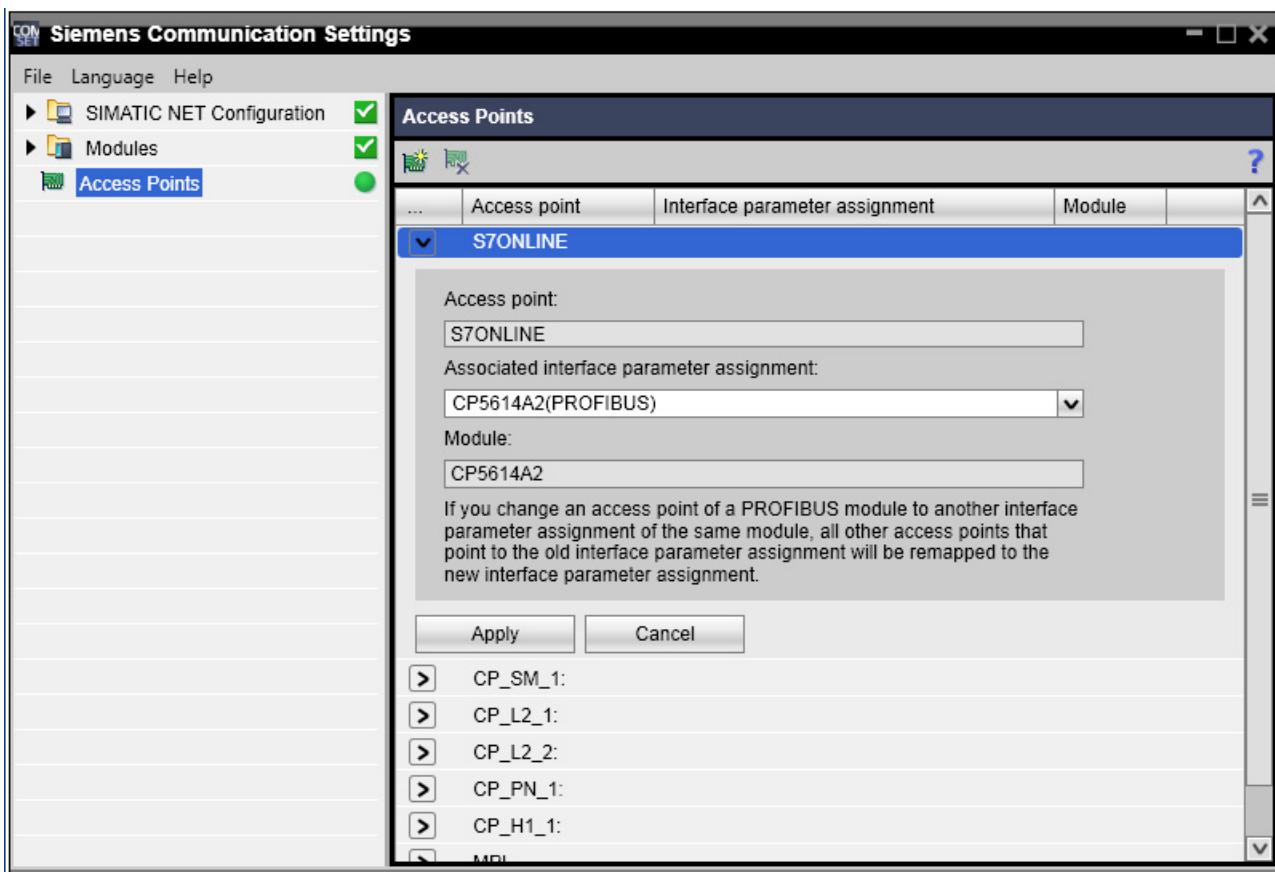


Figure 2-19 Assigning an interface parameter assignment

Add Item

Open the client program and create an item using the syntax shown above. In the OPC Scout V10 program, drag any item from the "S7" folder to the DA view and then click the "..." button at the right-hand edge of the "ID" table column. Now enter the syntax and confirm with "OK".

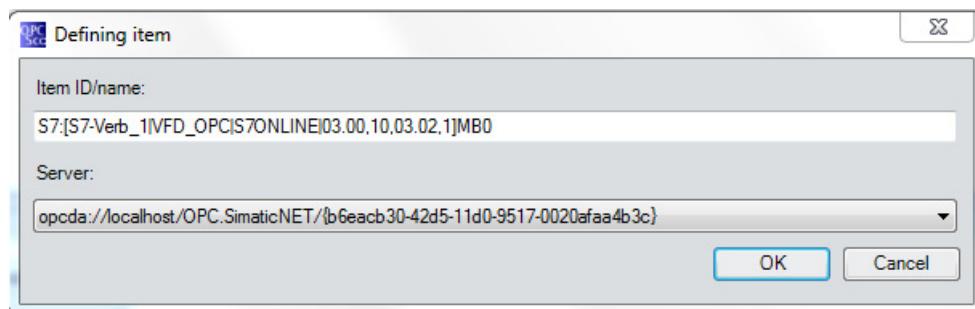


Figure 2-20 OPC Scout V10 - Inserting an item

After adding the item and as long as the item is active, the connection can be used like a configured connection. This means that you can browse in the namespace and also add

further items without using the syntax of the unconfigured connection. All you need to do is specify the connection name, for example S7:[S7-Verb_1]MB1.

2.6.12 COML S7 connection

Access to partner devices with local COML S7 connections

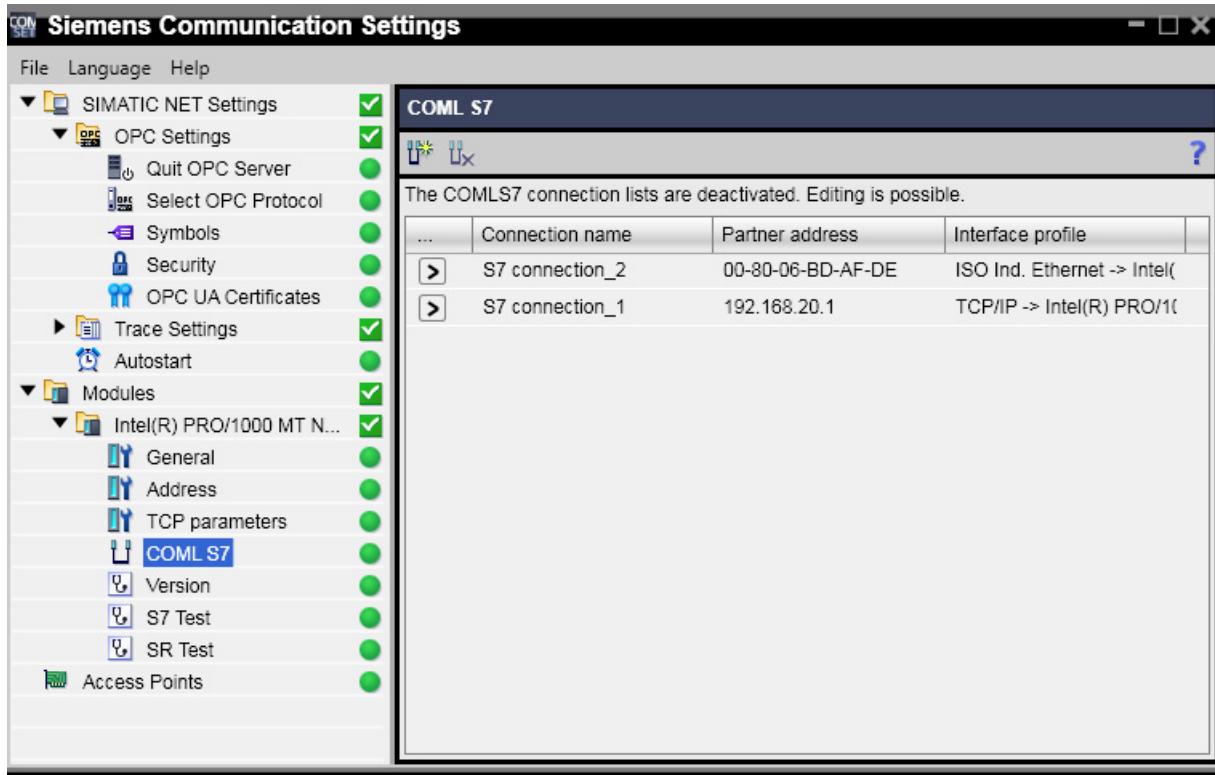


Figure 2-21 COML S7 - Configuration Management local - configuration software for S7 connections

Using the COML S7 configuration software, S7 connections to an S7 CPU or to a PC can be created. Below every module in the "Communication Settings" configuration program, you will find the "COML S7" tab. If you click on this tab, the COML S7-connection list appears on the right-hand side of the dialog window and you can create S7 connections to the partner devices for the selected module. The data resulting from the configuration work is collected in a database for all modules and saved on the PC. It can be exported as a backup and reimported. Following this, the COML S7 configuration must be activated in the shortcut memory of the "Modules" level before you can use the created S7 connections in the COML S7 connection lists for S7 communication.

Requirements for using COML S7 connections

For device access with COML-S7 configuration, all data of the partner device relevant to communication must be known. These include, for example, the partner TSAP and the partner address. You will find more detailed information on the parameters in the online help when creating the COML S7 connections.

Note

The simultaneous operation of configured S7 connections of STEP 7/SIMATIC NCM PC and COML S7 is not possible. They are interlocked.

After enabling COML S7 connections, the loading of S7 connections of STEP 7/SIMATIC NCM PC is rejected.

Note

Simultaneous operation of configured connections and unconfigured connections is not possible. They are interlocked.

After enabling configured connections, the operation of unconfigured connections is rejected.

2.7 S7 communication with OPC UA

2.7.1 Properties of S7 communication with OPC UA

The SIMATIC NET OPC server allows the use of S7 communication via OPC UA.

The S7 OPC UA server from SIMATIC NET has the following characteristics:

- Variable services
 - Access and monitoring of S7 variables
- Buffer-oriented services
 - Program-controlled transfer of larger blocks of data
- Server functionality
 - The PC can be used as a server for data fields and data blocks.
- Block services
 - Transfer of a loadable data area from and to S7
- S7 password functions
 - Setting a password for access to protected blocks
- Events, conditions and alarms
 - Processing S7 alarms and S7 diagnostics events

2.7.2 SIMATIC NET OPC UA server for the S7 protocol

Introduction

The section below describes a configuration variant for the S7 protocol that also supports OPC UA. To do this, the S7 COM OPC Data Access server is set up as an outproc OPC server.

Configuration

The S7 UA server is activated by selecting "S7" and "OPC UA" in the "Communication Settings" configuration program in the "OPC protocol selection" catalog:

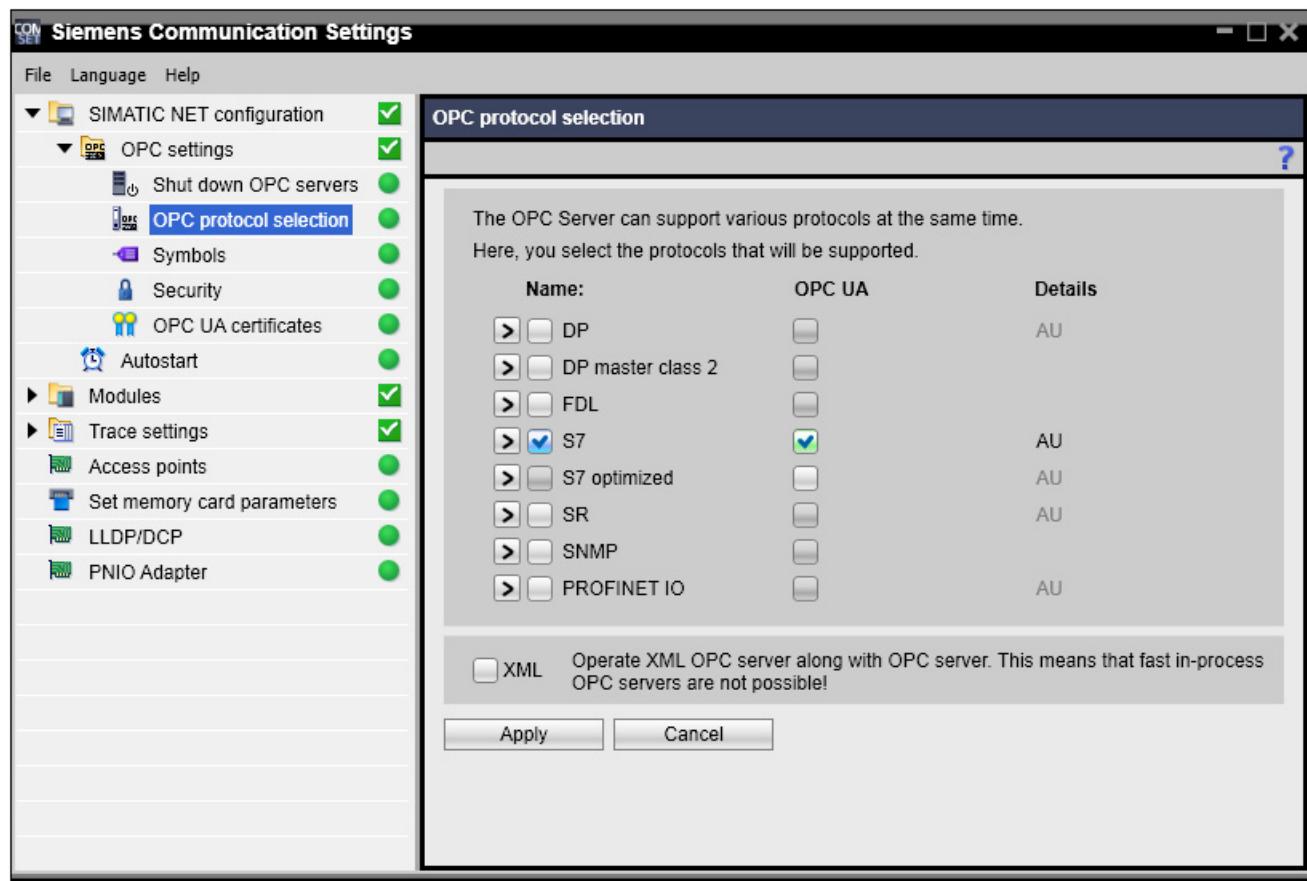


Figure 2-22 Window in the "Communication Settings" configuration program for selecting OPC UA for the S7 protocol

You can also select "Symbols".

Note

When creating symbols with STEP 7 Professional (TIA Portal) or the symbol editor, use of the following characters is permitted: A-Z, a-z, 0-9, _, -, ^, !, #, \$, %, &, ', /, (,), <, >, =, ?, ~, +, *, ',', :, |, @, [,], {, }, ". When creating symbols with STEP 7, you should also remember that problems can arise when resolving the array if your symbol file includes symbols in the form <symbolname> and <symbolname>[<index>] at the same time.

Advantages / disadvantages

When using the S7 OPC UA server, only the outproc mode of S7 is possible. The S7 OPC UA server process must be running to maintain readiness to receive. Even after all OPC UA clients have logged off, the S7 OPC UA server is not and must not be exited.

On the other hand, it does have the following advantages:

- COM/DCOM configuration is no longer necessary.
- High-speed, secure communication
- Only one server is required for events and data access.
- Disadvantage: Simultaneous operation of the high-speed Inproc OPC DA S7 server is not possible.

Notes

Note

If OPC UA is active for the S7 protocol, permanently configured S7 connections are established as soon as the first OPC UA client logs on to the S7 OPC UA server. If no OPC UA client has connected, for example after starting up the computer, no permanent S7 connections are established yet.

Note

S7 connections can be configured in SIMATIC STEP 7/NCM PC to be established "on demand".

The communication of the OPC functions is established only as necessary; this means that connection establishment may take longer and OPC may signal errors in the start-up phase.

2.7.3 How is the S7 OPC UA server addressed?

Server URL

For the native binary TCP protocol, there are two ways for the OPC client to address the server:

- Direct addressing:
 - opc.tcp://<hostname>:55101
 - or
 - opc.tcp://<IP-Adresse>:55101
 - or
 - opc.tcp://localhost:55101

The S7 OPC UA server has port 55101.

The redundant S7 OPC UA server is addressed from outside using the common redundant IP address.

- The URL of the S7 OPC UA server can also be found using the OPC UA Discovery service.

To locate the Discovery server enter the following:

- opc.tcp://<hostname>:4840
 - or
- opc.tcp://<IP-Adresse>:4840
 - or
- http://<hostname>:52601/UADiscovery/
 - or
- http://<IP-Adresse>:52601/UADiscovery/

The Discovery server has port 4840 (for TCP connections) and port 52601 (for HTTP connections).

IPv6 address

An IPv6 address can also be used as the IP address. The address must be in parentheses, for example [fe80:e499:b710:5975:73d8:14]

Endpoints and security modes

The SIMATIC NET S7 OPC UA server supports endpoints with the native binary TCP protocol and requires authentication using certificates and encrypted transfer.

The Discovery service on the addressed host signals the endpoints of the servers, in other words, their security requirements and protocol support.

The server URL "opc.tcp://<hostname>:55101" of the S7 OPC UA server provides the following endpoints:

- Endpoint in "SignAndEncrypt" security mode:

A signature and encryption are required to communicate with the server. Communication is protected by exchanging certificates and entering a password.

In addition to the security mode, the security policy Basic128Rsa15 is also displayed.

- Endpoint in "None" security mode:

In this mode, no security functions are required by the server (security policy "None").

For more detailed information on the security functions, refer to the section "Programming the OPC UA interface (Page 496)".

The security policies "Basic128Rsa18" and "None" are in the UA specification of the OPC Foundation at the following Internet address:

[> "Specifications" > "Part 7"](http://opcfoundation.org/UA)

You will find more detailed information on the following Internet page:

OPC Foundation ([<http://www.opcfoundation.org/profilereporting/index.htm>](http://www.opcfoundation.org/profilereporting/index.htm))
> "Security Category" > "Facets" > "Security Policy"

The OPC UA Discovery of the OPC Scout V10

The OPC Scout V10 allows you to open the OPC UA Discovery dialog to enter UA endpoints in the navigation area of the OPC Scout V10.

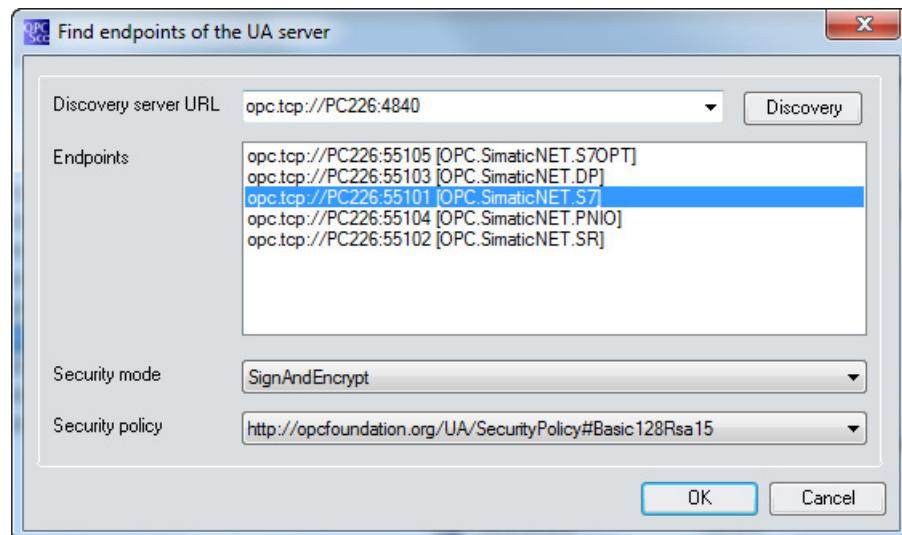


Figure 2-23 The "Find endpoints of the UA server" dialog of OPC Scout V10

The S7 OPC UA server can be found using the OPC UA Discovery service. For the entry, refer to "Server URL" above.

OPC Scout V10 contains a list of the OPC UA endpoints. The Discovery service on the addressed host then signals the registered OPC UA servers and their ports and security modes.

For more detailed information, refer to the online help of OPC Scout V10.

2.7.4 Which namespaces does the S7 OPC UA server provide?

The S7 OPC UA server provides the following namespaces:

Table 2- 2 The namespaces of OPC UA

Namespace index	"Identifier" (namespace URI) / comment
0	"http://opcfoundation.org/UA/" specified by the OPC Foundation
1	"urn:Siemens.Automation.SimaticNET.S7:(GUID)" Unique identifier of the local high-speed S7 OPC UA server.
2	"S7TYPES:" Definitions for S7-specific object types.
3	"S7:" Identifier of the local high-speed S7 OPC UA server with new simplified syntax (browsable and can be used with UA)
4	"S7COM:" Identifier of the server with the old syntax, S7 OPC DA-compatible (can be used with UA but cannot be browsed)
5	"S7SOURCES:" Identifier of the sources of alarms and user diagnostics events.
6	"S7AREAS:" Identifier for areas of an alarm hierarchy.
7	"SYM:" Optional server with ATI-S7 symbols; dependent on the project engineering and the configuration of the PC station (browsable and can be used with UA). As an alternative, a prefix can be used here that is specified in the parameter assignment for symbols ("Communication Settings").

The namespace indexes 0 and 1 are reserved and their significance is specified by the OPC Foundation.

The assignment of the remaining namespace indexes to the identifiers (namespace URI) must be obtained via the data variable "NamespaceArray" at the beginning of an OPC UA session by the client specifying the identifier. The identifiers "S7TYPES:", "S7:", "S7COM:", "S7AREAS:" and "S7SOURCES:" always exist with the S7 OPC UA server.

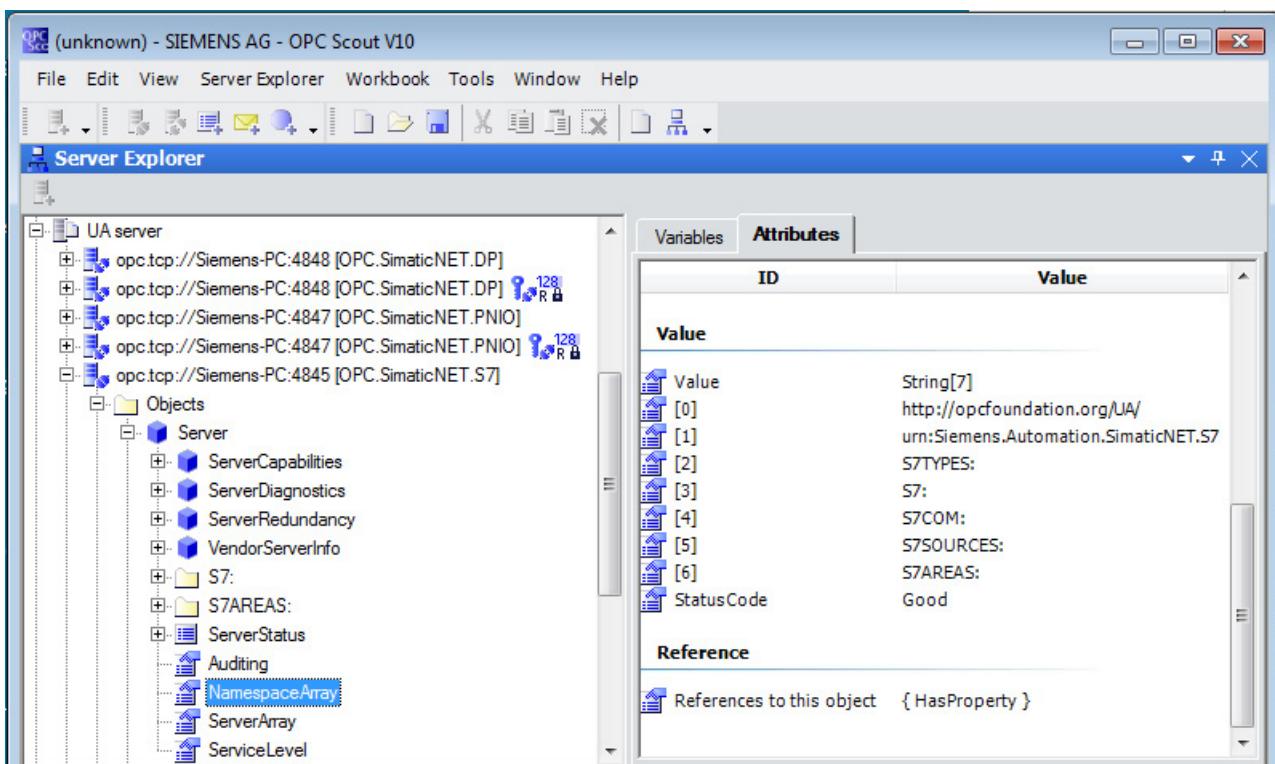


Figure 2-24 Display of the S7 OPC UA namespaces using the browse function of the OPC Scout V10

2.7.5 The Nodeld

Identification of an S7 process variable

The Nodeld identifies an S7 process variable with the aid of the following tuple:

- Namespace index
- Identifier (character string, numeric value)

Examples

- Nodeld:
 - Namespace URI:
S7:
(= namespace index 3) for Siemens.Automation.SimaticNET.S7
 - Identifier:
S7_connection_1.m.10,b

- Nodeld:
 - Namespace URI:
S7COM:
(= namespace index 4) for OPC.SimaticNET; the syntax is S7 OPC DA-compatible
 - Identifier:
S7:[S7_connection_1]mb10

How does the new namespace adapted to OPC UA behave?

The world of the OPC Data Access items of a COM server for reading and writing process variables is self-contained. Alongside it but independent, there is the alarm world.

On the other hand, the OPC UA view of automation objects is also related to various properties of the objects. OPC UA no longer access items alone, but also objects and their subobjects.

- Data variables, methods and, to some extent events, are, for example, subobjects of an S7 connection object. Attributes and properties define the objects in greater detail.
- An OPC Data Access item for block access is the closest to an OPC UA data variable.
- An OPC Data Access item for domain services is the closest to an OPC UA method.

The qualified identifiers of the Nodelds have a greater significance in OPC UA than in OPC Data Access. Each individual access to an object, subobject, property and attribute uses its Nodeld.

OPC UA provides the display name among other things to support local languages. This means that the same objects, for example in different language environments specified by the OPC UA client, can be browsed differently although the same Nodeld is presented every time. Selection of the display name is analogous to the relevant Nodeld. The texts of the entire namespace are in English.

Syntax of the S7 OPC UA data objects

An optimized syntax is introduced with OPC UA. The Nodelds of all OPC UA objects have the following structure:

<connectionobject>".<subobject>".<property>

A subobject can contain further subobjects.

A Nodeld that cannot be interpreted is rejected with an error. The letters "A-Z" are not case-sensitive for any items.

Symbolic object representation

The OPC UA specification recommends a uniform symbolic representation for the hierarchical description of the address space. The following symbols are used in this document:

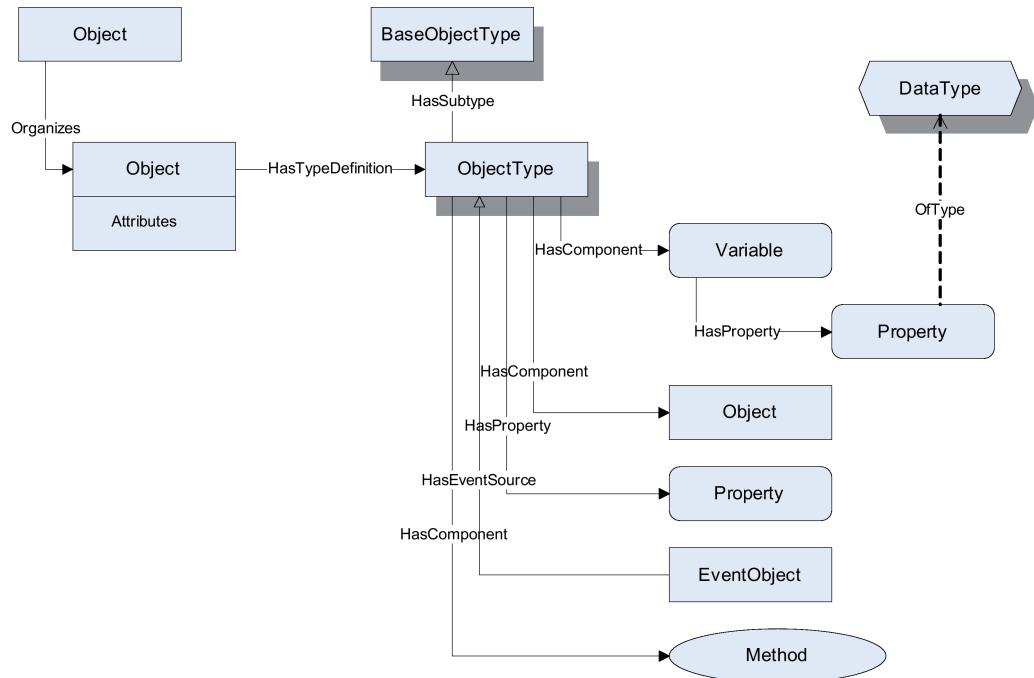


Figure 2-25 Symbols of the OPC UA address space

2.7.6 Connection objects

S7 connection objects

The following S7 connection objects exist:

- Productive S7 connections

S7 connections are used for the data exchange between automation devices and are generally configured with STEP 7.

- The DEMO connection

This is used only for testing.

- The @LOCALSERVER connection

This provides the local S7 data blocks for the S7 server functionality.

Demo connection

```
<connectionobject>:= "DEMO"
```

Under the demonstration connection called DEMO, there are objects that contain a namespace similar to S7 connections. The demonstration connection is intended to familiarize users with the SIMATIC NET OPC system and can be added during configuration.

Note

The demonstration connection with the name DEMO must not be used with an S7 connection with the same name. An S7 connection configured with this name is ignored if the demonstration connection is added to the configuration.

Local server connection

<connectionobject>:= "@LOCALSERVER"

Under the virtual local server connection with the name "@LOCALSERVER", there are data variables that allow access to the local data blocks managed by the server. After installation, only data block DB1 is available.

The syntax of the data variables is identical to that of the S7 variables for data blocks, for example:

@LOCALSERVER.DB1.100,B

The S7 variables for data blocks are described later.

Other data variables and methods found on a normal S7 connection, such as statepath, buffer or domain services do not exist.

Note

The connection name "@LOCALSERVER" is reserved and must never be used as the name of an S7 connection.

2.7.6.1 Connection names

The connection name of an S7 connection

The connection name is the name configured in STEP 7 or COML S7 to identify the connection. In STEP 7, this name is known as the "local ID". The local ID is unique within the OPC server.

Connection types

The OPC server supports the following connection types:

- S7 connection
- Faulttolerant S7 connection

What characters are permitted for S7 connection names?

For the <connectionname>, you can use numbers "0-9", upper and lower case alphabetic characters "A-z" and the special characters "_ -+()'". The connection name can be 24 characters long. The names are not case-sensitive.

Other printable characters are not permitted.

The connection names "SYSTEM" and "@LOCALSERVER" are reserved and must not be used.

Examples of connection names:

Typical examples are:

- S7_connection_1
- S7 OPC connection

2.7.7 Structure and functions of the productive S7 connection object

What are S7 connection objects?

All productive, protocol-specific objects are always assigned to a connection. In S7, these are the connections to the communication partners (S7 connection). Exceptions to this are the system connection and the demo connection.

2.7.7.1 Type definition of the S7 connection object

Type definition of the S7 connection object

A specific OPC UA object type is defined for objects and functionalities that can be used over a productive S7 connection:

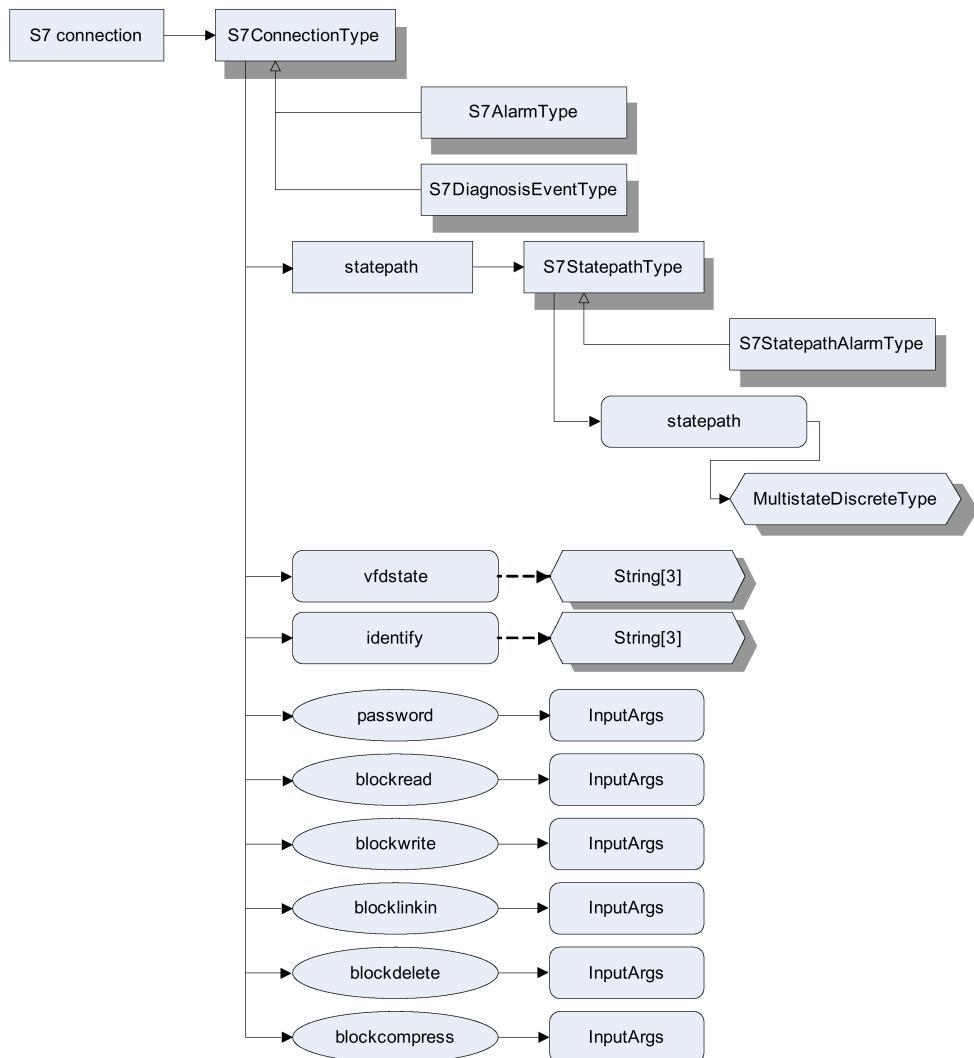


Figure 2-26 The type of the S7 connection object in the name space of OPC UA

Instances of this type are displayed in the OPC UA name space for objects. The type itself can be read out structured under "Types".

2.7.7.2 S7 connection information objects

S7-specific data variables for information

There are S7specific data variables with which you can obtain information about the S7 communication and the established connections.

The following information can be obtained:

- Attribute of a virtual field device (VFD)
- Status of an S7 connection
- Status of a virtual device
- Status of the logon for messages

(see "events" parameter in the section "Diagnostic and configuration information (Page 188)".)

Syntax of the S7-specific information variables

Nodeld:

Namespace index: 3 // for Siemens.Automation.SimaticNET.S7
<connectionobject>.<informationparameter>

Explanations

<informationparameter>:= "identify"|"vfdstate"|"statepath"

identify	Vendor attributes of a communications partner. Data type Array Of String (3 array elements), ReadOnly. "identify" can, for example, return the following values:	
	Vendor	Siemens AG
	Model	6ES7 416-3XR05-0AB0
	Revision	V5.0
	Status of a virtual device Data type Array Of String (3 array elements), ReadOnly.	
vfdstate	Logical status	S7_STATE_CHANGES_ALLOWED All services are permitted.
	Physical status	S7_OPERATIONAL The real device can be used. S7_NEEDS_COMMISSIONING The real device can only be used after local settings have been made.
	Detailed information on the VFD status, device-dependent	<bytestring(3)> (array of 3 bytes) The status is returned as an octet string. For further information on the significance of the return value, refer to the documentation of the partner device.

statepath	Status of a communications connection to the partner device The value of this variable is read out as a number and can be assigned to a text by additionally reading the associated Enumstring {UNKNOWN, DOWN, UP, RECOVERY, ESTABLISH}. Variable of the UA type MultistateDiscreteType, ReadOnly		
	0	UNKNOWN	Reserved for future expansions
	1	DOWN	Connection is not established
	2	UP	Connection is established
	3	RECOVERY	Connection is not established. An attempt is being made to establish the connection.
	4	ESTABLISH	Reserved for future expansions

2.7.7.3 Examples of S7-specific information variables and return values

Here, you will find examples illustrating the syntax of the names of S7-specific information variables.

Information on the vendor attributes of a virtual device

- Nodeld:
 - Namespace URI:
S7: (Namespace index: 3) // for Siemens.Automation.SimaticNET.S7
 - Identifier:
S7_connection_1.identify

Possible return values:

- Vendor: *SIEMENS AG*
- Model of the virtual device: *6ES7413-1AE0-0AB0*
- Revision: *V1.0*

Status of a device

- Nodeld:
 - Namespace URI:
S7: (Namespace index: 3) // for Siemens.Automation.SimaticNET.S7
 - Identifier:
S7_connection_1.vfdstate

Possible return values:

- Logical status: *S7_STATE_CHANGES_ALLOWED*
All services are permitted.
- Physical status: *S7_OPERATIONAL*
The real device can be used.
- Detailed information: *02.00.00*
Detailed information on the local VFD status

Status of a communications connection as string

- Nodeld:
 - Namespace URI:
S7: (Namespace index: 3) // for Siemens.Automation.SimaticNET.S7
 - Identifier:
S7_connection_1.statepath

Possible return values of the type "MultistateDiscreteType":

- Connection status: *RECOVERY*
The connection is currently being established.

2.7.7.4 Methods for the S7 block services

What do the block services do?

Block management services control the transfer and return of data and program elements between the PC and the programmable controller. On the PC, data and program elements are stored in files. Block management services can be protected by a password.

Blocks

Data and program elements are stored on S7 programmable controllers in blocks. These blocks are compiled using STEP 7 and transferred to the S7 device. The following blocks can exist on S7 devices:

- Organization blocks (OB)
- Function blocks (FB)
- Functions (FC)
- Data blocks (DB / DI)

Block management functions

You can execute the following functions via the SIMATIC NET OPC Server:

- Transfer blocks between PCs and programmable controllers
- Delete blocks
- Link blocks
- Compress the memory of the programmable controller

Note

It is not possible to create blocks with the OPC server. You have to use STEP 7 for this.

OPC UA supports product-specific methods. The S7 block services are the same as the execution of methods.

Syntax of the methods for the S7 block services

Nodeld:

*Namespace URI: S7: (Namespace index: 2)
for Siemens.Automation.SimaticNET.S7*

<connectionname>. <DomainMethod>

<DomainMethod>:=

"password()"|"blockread()"|"blockwrite()"|"blocklinkin()"|"blockdelete()"|"blockcompress()"

Parameter	Meaning						
password()	<p>Transfer password for domain services to the partner device InputArgument1: "password"; data type: ByteString; WriteOnly To transfer the password to the connection partner, a suitable string must be transferred as the password argument. There are two possibilities for the string</p> <table border="1"> <tr> <td>"" (empty string)</td><td>Password is reset</td></tr> <tr> <td><bytestring(8)></td><td>Password in byte string representation (array of 8 bytes)</td></tr> </table>	"" (empty string)	Password is reset	<bytestring(8)>	Password in byte string representation (array of 8 bytes)		
"" (empty string)	Password is reset						
<bytestring(8)>	Password in byte string representation (array of 8 bytes)						
blockread()	<p>A block from a programmable controller is transferred to the PC and stored there in a file: InputArgument1: "flags", data type UInt32 InputArgument2: "block", data type String InputArgument3: "filename"; data type: String</p> <table border="1"> <tr> <td>Flags <unsigned32></td><td> <p>Flags The following hexadecimal numbers are possible: 0x0001 A unlinked block of data is read. If a destination file exists, this is not overwritten. An error message is output. 0x0040 A linked in block of data is read. If a destination file exists, this is not overwritten. An error message is output. 0x1001 An unlinked block of data is read. An existing destination file is overwritten. 0x1040 A linked block of data is read. An existing destination file is overwritten.</p> </td></tr> <tr> <td>Block type and number: "OB"<unsigned16>" FB"<unsigned16> "FC"<unsigned16>" DB"<unsigned16></td><td></td></tr> <tr> <td>Full path of the file in which the block is stored: <string(511)></td><td></td></tr> </table>	Flags <unsigned32>	<p>Flags The following hexadecimal numbers are possible: 0x0001 A unlinked block of data is read. If a destination file exists, this is not overwritten. An error message is output. 0x0040 A linked in block of data is read. If a destination file exists, this is not overwritten. An error message is output. 0x1001 An unlinked block of data is read. An existing destination file is overwritten. 0x1040 A linked block of data is read. An existing destination file is overwritten.</p>	Block type and number: "OB"<unsigned16>" FB"<unsigned16> "FC"<unsigned16>" DB"<unsigned16>		Full path of the file in which the block is stored: <string(511)>	
Flags <unsigned32>	<p>Flags The following hexadecimal numbers are possible: 0x0001 A unlinked block of data is read. If a destination file exists, this is not overwritten. An error message is output. 0x0040 A linked in block of data is read. If a destination file exists, this is not overwritten. An error message is output. 0x1001 An unlinked block of data is read. An existing destination file is overwritten. 0x1040 A linked block of data is read. An existing destination file is overwritten.</p>						
Block type and number: "OB"<unsigned16>" FB"<unsigned16> "FC"<unsigned16>" DB"<unsigned16>							
Full path of the file in which the block is stored: <string(511)>							
blockwrite()	<p>A block is transferred from a PC to a programmable controller. After the transfer, the block on the programmable controller is still in the passive (not linked in) status. The block must be changed from the passive to the linked status with the <i>&blocklinkin</i> function.</p> <p>InputArgument1: "flags", data type UInt32 InputArgument2: "filename", data type String</p>						

Parameter	Meaning	
	Flags <unsigned32>	Flags The following hexadecimal numbers are possible: 0x1000 An unlinked block with the same name on the automation system is to be overwritten. 0x0000 An unlinked block with the same name on the automation system is not to be overwritten.
	Full path of the file in which the block is stored. Data type <string(511)>	
blocklinkin()	<p>Link a block with a passive status into the program sequence of the automation system. The executable part of the block is copied to the work memory of the programmable controller.</p> <p>The block is then accessible to the program on the automation system. By linking a block, an existing, active block is overwritten without an error message.</p> <p>The written value contains the specification of the block to be linked as the parameter setting for this service.</p> <p>InputArgument1: "block" data type String</p>	
	<p>Block type and number:</p> <p>"OB"<unsigned16> "FB"<unsigned16> "FC"<unsigned16> "DB"<unsigned16></p>	
blockdelete()	<p>Delete a block on the programmable controller</p> <p>InputArgument1: "flags" data type UInt32</p> <p>InputArgument2: "block" data type String</p>	
	Flags Data type VT_BSTR <unsigned32>	Flags The following hexadecimal numbers are possible: 0x0001 An unlinked block of data is deleted. 0x0040 A linked in block of data is deleted. 0x0041 The linked and the unlinked block of data are deleted.
	<p>Block type and number</p> <p>Data type VT_BSTR</p> <p>"OB"<unsigned16> "FB"<unsigned16> "FC"<unsigned16> "DB"<unsigned16></p>	
blockcompress()	<p>Compression of memory of the programmable controller</p> <p>"" (empty string)</p>	

Note

The "blockread" and "blockwrite" block services do not allow access to source and target files of the blocks on a network drive.

Overwriting (0x1000) a block is only possible after the executable part has been linked in.

Passwords

Write and read access by block management services to the CPU can be protected by a password during configuration. The password has higher priority than the keyswitch of the CPU.

For more information on passwords and protection levels, refer to the section "Passwords (Page 144)".

2.7.8 Variable services

2.7.8.1 Variable services

What do the variable services do?

Variable services allow access to and monitoring of S7 variables on the programmable controller. S7 variables are addressed using the short name of the addressed objects. The type of access is oriented on the notation of the S7 tools.

Objects in the programmable controller of the S7 OPC UA servers support the following objects:

- Data blocks
- Instance data blocks
- Inputs
- Outputs
- I/O inputs
- I/O outputs
- Bit memory
- Timers
- Counters

Not every S7 programmable controller supports all object types.

2.7.8.2 Syntax of the variable services

Syntax of the process variables

Simplified syntax of the process variables of the S7 OPC UA Node ID:

Namespace URI: S7: (Namespace index: 3)

Classic syntax

You have three options:

- <connectionname>.<S7object>.<address>{,<S7type>{,<quantity>}}
- <connectionname>.<S7timerobject>.<address>{,<S7timertype>{,<quantity>}}
- <connectionname>.<S7counterobject>.<address>{,<S7countertype>{,<quantity>}}

Explanations

Namespace URI: S7: (Namespace index: 3)

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

The following separator is the period (*. *).

<S7object>

Specifies the area type on the S7 PLC. Possible values are as follows:

Parameter	Meaning
db<no>	Data block or instance data block. Identifier for an S7 variable from a data block. In S7 communication, no distinction is made between instance data blocks and normal data blocks. As a result, it is not necessary to assign an additional identifier to simplify matters. <no> Number of the data block or instance data block without leading zeros.
m	Bit memory
i	Input Writable and readable To simplify things, there is only this English identifier.
q	Output Writable and readable To simplify things, there is only this English identifier.
pi	I/O input Read only To simplify things, there is only this English identifier.
pq	I/O output Write only To simplify things, there is only this English identifier.

The following separator is the period (*. *).

<S7Timerobject>

Parameter	Meaning
t	Timer. Word (unsigned). The address information that follows is a timer number.

<S7Counterobject>

Parameter	Meaning
c	Counter. Word (unsigned). The address information that follows is a counter number.

<address>

Byte address of the first variable to be addressed. No leading zeros are supported (for example 001).

The value range for the byte address is 0 to 65534. The actual usable value of the address may be lower depending on the device and type.

<S7type>

S7 data type

An S7 data type is converted to the corresponding OPC UA data type on the OPC server. The following table lists the type "identifier" and the corresponding OPC UA data type in which the variable value can be represented.

S7 data type <S7type>	OPC UA data type	Description
b	Byte	Byte (unsigned) Used as the default if no <S7type> is specified.
w	UInt16	Word (unsigned)
c	SByte	Byte (signed)
i	Int16	Word (signed)
di	Int32	Double word (signed)
dw	UInt32	Double word (unsigned)
r	Float	Floating point (4 bytes)
dt	DateTime	Date and time, range of values as of 01.01.1990, (on CPU DATE_AND_TIME)
date	DateTime	Date and time (8 bytes), where the time is always 00:00:00, range of values starting from 01.01.1990. Mapping of the CPU data type DATE (unsigned, 16 bits).
t	Int32	Signed time value in milliseconds
tod	Int32	Time of day, 0 to 86399999 ms starting at midnight

S7 data type <S7type>	OPC UA data type	Description
s5bcd	UInt16	Mapping of the CPU data type S5TIME to UInt 16 (unsigned, 16 bits) with a restricted range of values, 0 to 9990000 ms.*)
x<bitaddress>	Boolean	Bit (bool) In addition to the byte offset in the area, the <bit-address> must also be specified in the byte. Range of values 0 to 7
s<stringlength>	String	The <stringlength> reserved for the string must also be specified. Range of values 1 to 254 When writing, it is also possible to write shorter strings, whereby the transferred data length is always the reserved string length in bytes plus 2 bytes. The unnecessary bytes are filled with the value 0. Reading and writing strings and string arrays is mapped internally to the reading and writing of byte arrays. The string must be initialized with valid values on the S7.

*) See table below "Timebase and range of values for the S7 data type s5bcd"

The data types "c", "i", "di" and "r" can be used in a data block (db) or an instance data block, for bit memory (m), for inputs (i), for outputs (q), for I/O inputs (pi) and for I/O outputs (pq).

Timebase and range of values for the S7 data type s5bcd:

The range of values of the time variable of data type s5bcd is BCD-coded. The range of values is according to the following table:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Meaning (symbol)	0	0	x	x	t	t	t	t	t	t	t	t	t	t	t	t
Explanation:																
Meaning "0"	not relevant															
Meaning "x"	Specifies the timebase															
Bits 13 and 12										Timebase in seconds						
00										0.01						
01										0.1						
10										1						
11										10						
Meaning "t"	BCD-coded time value (0 to 999)															

Example:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	0	1	0	0	0	0	0	1	1	1	0	1	0	1

Bits 0-11 specify the number 075. Bits 12 and 13 specify the timebase 0.1.
 $75 * 0.1 = 0.75$ seconds

The OPC data type of the time variable of data type s5bcd is a word (unsigned, UInt16). When writing, the range of values is limited accordingly.

<S7Timertype>

S7 data type <S7type>	OPC UA data type	Description
tbcd	UInt16	Timer, BCD coded Used as the default if no <S7Timertype> is specified.
tda	UInt16[2]	Timer, decimal timebase and time value

Timebase and range of values for S7 timer variables "t", "tbcd" (<S7Timerobject> = t, <S7Timertype> = tbcd).

The range of values of OPC process variables for S7 of the type timer (t) is BCD coded. The timebase (for writing) is obtained from the value range as shown in the table below:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Meaning (symbol)	0	0	x	x	t	t	t	t	t	t	t	t	t	t	t	t
Explanation:																
Meaning "0"	not relevant															
Meaning "x"	Specifies the timebase															
	Bits 13 and 12								Timebase in seconds							
	00								0.01							
	01								0.1							
	10								1							
	11								10							
Meaning "t"	BCD-coded time value (0 to 999)															

Example:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	0	1	0	0	0	0	0	1	1	1	0	1	0	1

Bits 0-11 specify the number 075. Bits 12 and 13 specify the timebase 0.1.
 $75 * 0.1 = 0.75$ seconds

Timebase and range of values for S7 timer variables "t", "tda" (<S7Timerobject> = t, <S7Timertype> = tda):

Simple timer variables of the type "t" make it easy to control timers. However, not all possible combinations of timebase and value range can be set because the value ranges can overlap. In this case, the S7 timer variable of the type "tda" (decimal array) can be used.

Data type: Field of two words {timebase in milliseconds as UInt16 | time value UInt16}.

Table 2- 3 Possible Values

Timebase [ms]	10, 100, 1 000, 10 000
Time value	0...999 0 is permitted but has no function.
Timebases [ms]	10 ms: 0...9 990 100 ms: 0 to 99 900 ms 1 000 ms: 0...99 9000 10,000 ms: 0...9 990 000

It is not possible to enter <quantity> for the TDA object.

Example:

Writing the value {100|50} to timer "T.3", "tda" initializes timer 3 with the value $50 * 100 \text{ ms} = 5000 \text{ ms}$ and this is clocked down 50 times in 100 ms steps.

With the "t" and "bcd" types, this setting is not possible.

<S7Countertype>

c	UInt16	Counter Range of values for S7: 0 to 999, decimal-coded Used as the default if no <S7Countertype> is specified.
---	--------	---

<quantity>

Number of variables of a type to be addressed starting at the offset specified in the "address" parameter (range of values 1 to 65 535).

Specifying a number of array elements causes an array of the corresponding type to be formed even when only a single array element is addressed.

The separator is a comma (",").

For data type "x", the quantity for write access can only be entered in multiples of 8. The bit address must then be zero.

For data type "x", the entry of a quantity for read access is unrestricted. The value range of the bit address then allows 0 to 7.

It is not possible to enter <quantity> for the TDA object.

Examples:

Namespace URI: S7: (Namespace index: 3)

S7-OPC-1.db1.10,x0,64, access rights RW

S7-OPC-1.db1.10,x3,17, access rights R

2.7.8.3 Examples of process variables for S7 OPC UA variable services

Here you will find examples illustrating the syntax of variable names for variable services.

Data block DB single byte

Namespace URI: S7: (Namespace index: 3) // for Siemens.Automation.SimaticNET.S7

S7_connection-1.db5.12,B

Indicates data byte 12 in data block 5 via S7_connection-1.

Data block DB, array of words

Namespace URI: S7: (Namespace index: 3)

S7_connection-1.db5.10,w,9

Indicates 9 data words as of byte address 10 in data block 5 via S7_connection-1.

Data block DB, array of strings

Namespace URI: S7: (Namespace index: 3)

S7_connection-1.db100.50,s32,3

Indicates 3 strings with a length of 32 as of byte address 50 in instance data block 100 via S7_connection-1.

Input 0

Namespace URI: S7: (Namespace index: 3)

S7_connection-1.i.0

Indicates input byte 0 via S7_connection-1.

Output 0 bit 0

Namespace URI: S7: (Namespace index: 3)

S7_connection-1.q.0,x0

Indicates output address 0, bit 0 via S7_connection-1.

Array of readable memory bits

Namespace URI: S7: (Namespace index: 3)

S7_connection-1.m.3,x4,12 access rights R

Indicates 12 bits as of bit memory address 3 and there as of bit address 4 via S7_connection-1. Read only.

Timer 22 BCD-coded

Namespace URI: S7: (Namespace index: 3)

S7_connection-1.t.22

Indicates timer 22, TBCD default via S7_connection-1.

2.7.9 Buffer-oriented services

What do the buffer-oriented services do?

Buffer-oriented services allow program-controlled transfer of larger blocks of data. The transfer is implemented based on variables:

- Variables that send blocks of data
- Variables that receive blocks of data

The amount of data in the transfer is up to 65 534 bytes regardless of the size of the PDU. The segmentation of the data is handled by the functions themselves.

Note

Buffer-oriented services can only be used on connections configured at both ends. The created configuration connection must be loaded on the S7 programmable controller.

2.7.9.1 Syntax of the buffer-oriented services

Simplified syntax

Simplified syntax of the process variables S7 OPC UA Nodeld:

Namespace URI: *S7*: (Namespace index: 2) for "Siemens.Automation.SimaticNET.S7"

Classic syntax

The available options are as follows:

- <connectionname>.brcv<rid>{.<address>{,<S7type>{,<quantity>}}}
- <connectionname>.brcv<rid>{,<address>{,<S7type>{,<quantity>}}}
- <connectionname>.bsend<rid>.<bufferlength>{.<address>{,<S7type>{,<quantity>}}}
- <connectionname>.bsend<rid>.<bufferlength>{,<address>{,<S7type>{,<quantity>}}}

Explanations

Parameter of the name variables	Meaning
connectionname	Protocol-specific connection name. The connection name is specified in the configuration.
brcv	<p>Data type: Array of bytes</p> <p>brcv contains the last block of data received from the partner. The content and length of the received data is set by the sending partner.</p> <p>The variable is read-only. Set this variable for monitoring. This signals the receipt of a block of data by the OPC server to the OPC client application.</p>
bsend	<p>Data type: Array of bytes</p> <p>BSEND contains the send block of data for transfer to the partner device. The block of data is only sent to the partner device when the variable is written. With read access, the content of the last successfully transferred block of data is returned.</p>
<rid>	<p>ID of the addressing parameter. This is specified for a block pair (BSEND/ BRCV) and defined uniquely within a connection.</p> <p>You can send several BSEND blocks via a connection or receive several BRCV blocks, but always with a different ID. The same IDs can be used for other connections.</p>
<bufferlength>	<p>Length (in bytes) of the block of data to be sent. Any number of write buffers of different lengths can be defined. This makes it possible to send subareas of different lengths to the partner.</p>
<address>,<S7type>	<p>The data buffers can be structured. From these, one or more subareas can be selected. Specify the data type <S7type> and the <address> that are also valid for the S7 data block variable. They are both described at the beginning of the section "Variable services (Page 120)".</p> <p>Read: Since the structure and the length of the block of data received from the partner do not need to be fixed, it is not permitted to define and request variables outside the area. If the relevant area can no longer be filled when data is received, the quality of the variables changes to reflect this.</p> <p>Write: With write access to a subarea of a block of data to be sent to the partner, the entire data buffer is sent; its length is specified in <bufferlength>. If several subareas of the data buffer are written during an OPC UA group call, the data buffer is only sent after all its subareas have been updated. Non-specified subareas are filled from the cache.</p> <p>Note In the data buffer, the data types are interpreted in Motorola format and converted to Intel format for writing.</p>
quantity	<p>Number of field elements</p> <p>Range of values: 1...65 534</p>

Notes on reading and writing

- Reading (brcv) and writing (bsend) individual bits (format x) is possible.
- The reading and writing of fields of individual bits is possible with no restrictions relating to the start bit addresses and with any field length within the block of data.
Example: S7-OPC-1.brcv1.10.x4,78
- Variables for send data or receive data of different length, different addressing parameter ID (RID) or different connection name have independent memory areas.
- If an item is created for an independent memory area, the send data buffer is allocated and initialized with zero. A write to a BSEND item is written to the internal write buffer and transferred.
- The blocks of data are transferred acyclically. Simultaneous network jobs for the same data are possible. The complete block of send data is always transferred. This also applies to access to a subelement or when several clients write to this item at the same time. Parallel writing of the same block of send data or of subareas of the block of send data by more than one client can lead to inconsistencies.

We therefore recommend ...

- that you always read or write the entire block of data or
- set the maximum number of parallel network jobs to 1 in NCM S7. This does, however, reduce the transfer performance.
- If the receipt of blocks of data is read using the OPC UA monitoring service (brcv as MonitoredItem), the response to receipt of blocks of data can be set in the configuration of the monitoring service. Assuming that DataChangeTrigger is set to OpcUa_DataChangeTrigger_StatusValueTimestamp in the monitoring service, newly received blocks of data with the same data are also signaled to the OPC UA client. This allows a client to receive unchanged data buffers from the partner. The DataChange notification is not faster than the negotiated update rate. For this function, you should therefore always set faster update rates than the send rate of the bsend/brcv data.

2.7.9.2

Examples of process variables for buffer-oriented services

Here you will find examples illustrating the syntax of variable names for buffer-oriented services.

Receive block of data in the entire buffer

Namespace URI: S7: (Namespace index: 3)

S7-OPC-1.brcv1

A block of data is received in the receive buffer with RID 1 via S7-OPC-1. The complete buffer is mapped to an array of bytes.

Alternate access to received block of data

Namespace URI: S7: (Namespace index: 3)

S7-OPC-1.brcv1.2,w,4

From the received block of data, the content is mapped to an array of 4 words, starting at address 2. A total of 8 bytes of the block of data are therefore taken into account.

Transfer double word

Namespace URI: S7: (Namespace index: 3)

S7-OPC-1.bsend1.16,2,d

A double word starting at address 2 is addressed via S7-OPC-1 in a block of data with a length of 16 and RID 1. If a write command is used to access the variable, the written value is entered at the specified position in the block and block of data is sent.

Transfer floating-point numbers

Namespace URI: S7: (Namespace index: 3)

S7S7-OPC-1.bsend1.32,20,r,2

A field with floating-point numbers starting at offset 20 is addressed via S7-OPC-1 in a block of data with a length of 32 and RID 1. If a write command is used to access the variable, the written value is entered at the specified position in the block and block of data is sent.

2.7.10 Block information objects of an S7 connection

2.7.10.1 Length information

The block objects under an S7 connection object

The type and size of the block structure in an S7 programmable controller is calculated during runtime. Under an S7 connection object, there are the following block objects that provide this information to an OPC-UA client.

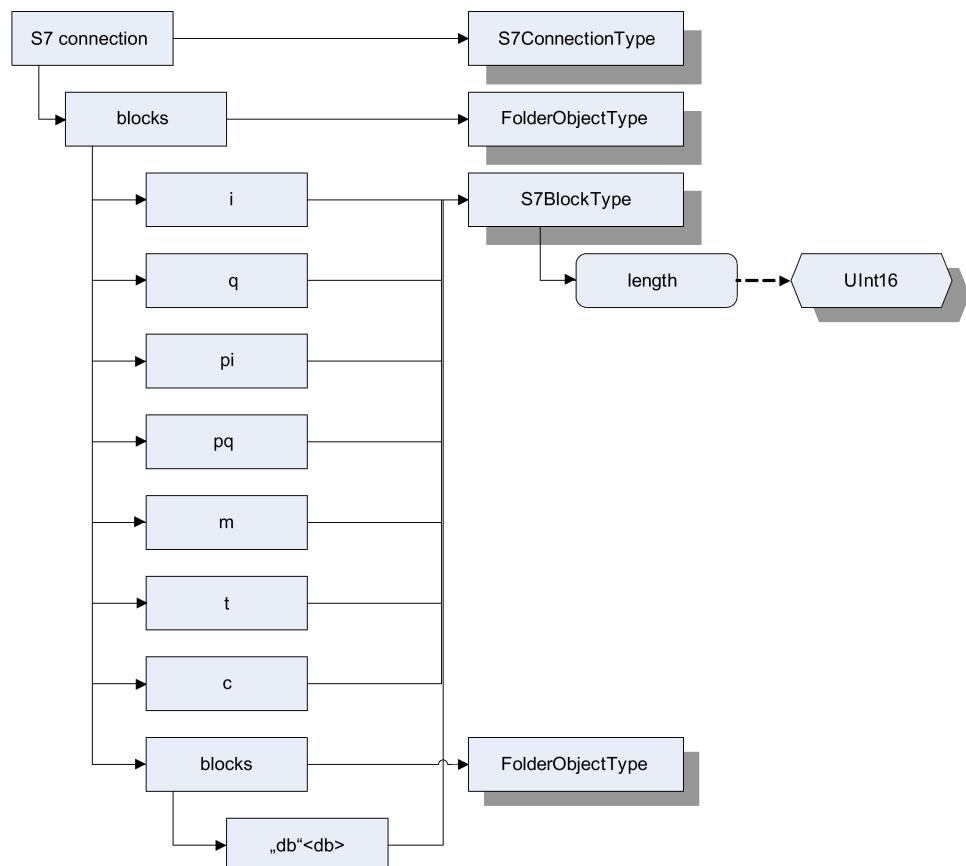


Figure 2-27 Block objects under an S7 connection object

Each block object has an OPC UA property with length/size information about the block.

Example:

Namespace URI: S7: (→Namespace index: 3)

S7-connection_1.db10.length //Length property of DB10 block (length in bytes)

2.7.10.2 Template objects

The template object of a block object

For every block object displayed when browsing, a template object is shown whose NodeID can be used as a template for further user-defined data objects. The template object has the standard data type B (or c or tbcd) for the relevant block object and always begins at address 0. If this item is not accessible on the connection partner, this is indicated by access results and quality codes to this effect.

Example:

Namespace URI: S7: (Namespace index: 3)

S7_connection_1.db10.0,b

S7_connection_1.m.0,b

2.7.10.3 Diagnostic and configuration information

The properties of an S7 connection object

In general, the properties of an S7 connection are configured the STEP 7 configuration tool. During runtime, it may be useful to evaluate some of the configuration parameters.

Some configuration parameters are provided for OPC UA as properties for the S7 connection object:

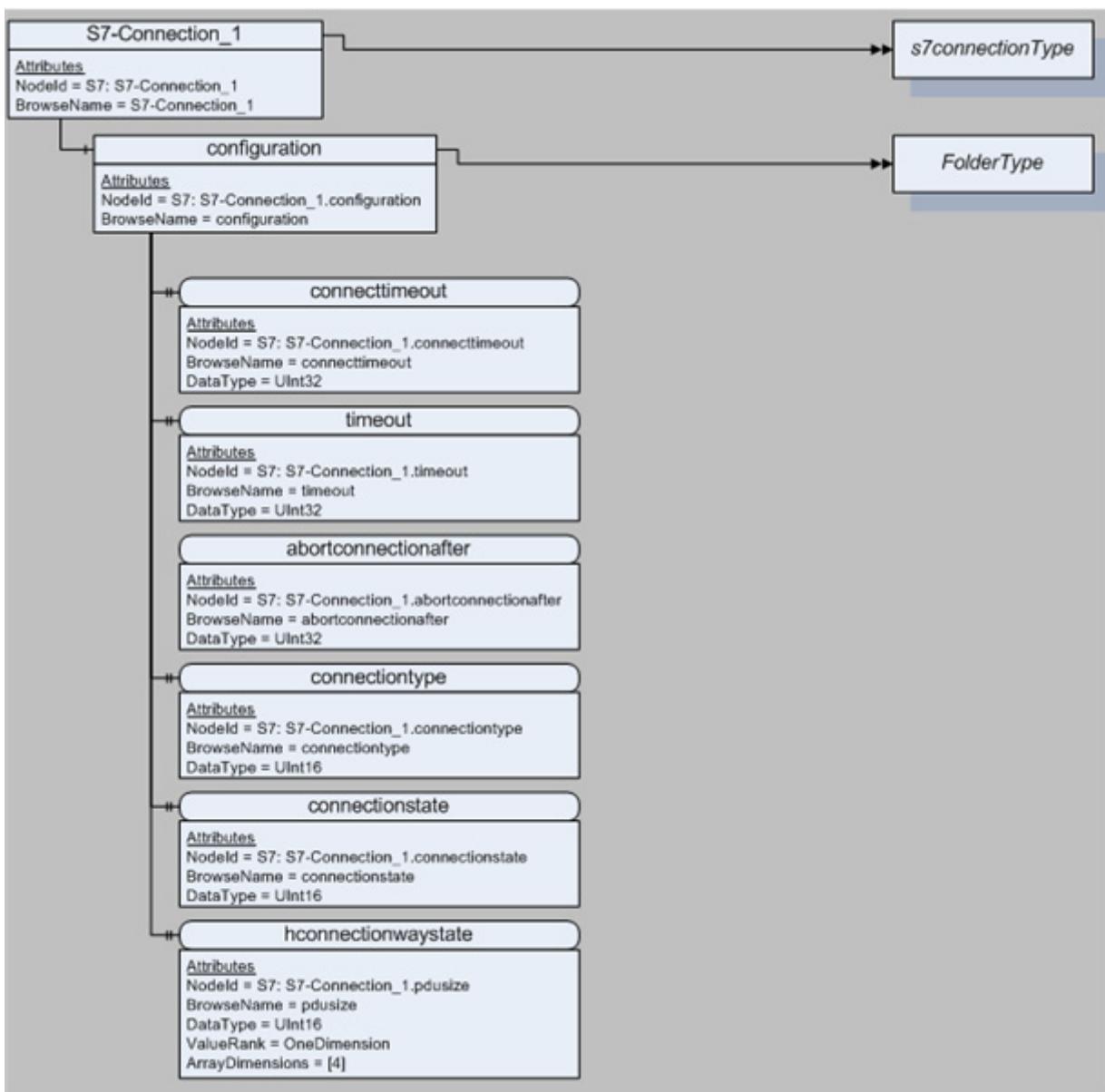


Figure 2-28 Properties of an S7 connection object (part 1)

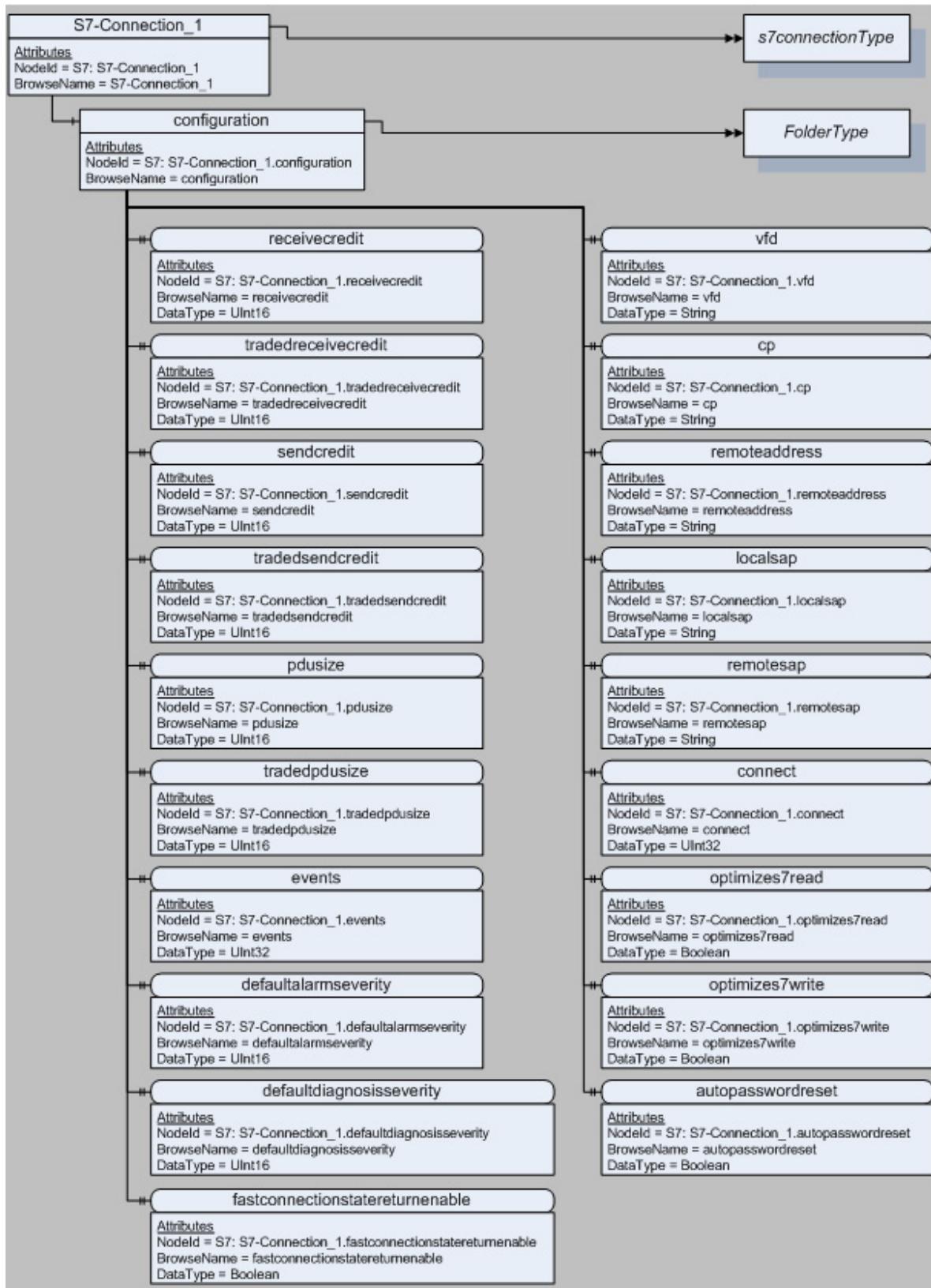


Figure 2-29 Properties of an S7 connection object (part 2)

Syntax for diagnostics and configuration information

```

Namespace URI: S7: (-->Namespace index: 3)
<connectionname>. <S7connectionproperty>

<S7connectionproperty>:= "vfd"|"cp"|"remoteaddress"|"localsap"|"remotesap"|
"con-
nect"|"autopasswordreset"|"fastconnectionstatereturnenable"|
"sendcredit"|"receivecredit"|"pdusize"|
"tradedsendcredit"|"tradedreceivecredit"|"tradedpdusize"|
"connecttimeout"|"timeout"|"abortconnectionafter"|
"optimizes7read"|"optimizes7write"|"defaultalarmseverity"|
"defaultdiagnosisseverity"|"events"|"connectiontype"|
"connectionstate"|"hconnectionwaystate"

```

S7 connection diagnostics property	Meaning
vfd	Name of the OPC server to which the connection is assigned. This normally has the text "OPC Server" for connections configured in NCM. Data type String, read-only.
cp	Name of the interface parameter assignment to which the connection is assigned. Data type String, read-only.
remoteaddress	Address of the connection partner. Data type String, read-only. The address of the connection partner is a data buffer with a data length dependent on the connection type. To make it more manageable for evaluation by the user, the data buffer is displayed formatted in a string. PROFIBUS address Format: "ddd" (1-3 decimal digits) IP address (ISO-on-TCP) Format: "ddd.ddd.ddd.ddd" (each 1-3 decimal digits) MAC address (ISO) Format: "xx-xx-xx-xx-xx-xx" (each 2 hexadecimal digits)
localsap	Local SAP of the connection. Data type String, read-only. The local SAP of the connection partner is a data buffer with a data length dependent on the connection type. To make it more manageable for evaluation by the user, the data buffer is displayed formatted in a string. Format: "xx.xx " (each 2 hexadecimal digits)
remotesap	Remote SAP of the connection. Data type String, read-only. The remote SAP of the connection partner is a data buffer with a data length dependent on the connection type. To make it more manageable for evaluation by the user, the data buffer is displayed formatted in a string. Format: "xx.xx " (each 2 hexadecimal digits)
connect	Type of connection establishment. Data type UInt32, read-only. 0 Passive, connection is kept established permanently.

S7 connection diagnostics property	Meaning	
	1	Active, establish connection only when necessary, connection termination if not used after wait time.
	2	Active, connection is kept established permanently.
autopasswordreset	Automatic resetting of the S7 password for block access Data type Boolean, read-only. True: Resetting enabled False: Resetting disabled On an S7, the domain services enabled by password remain active until explicitly reset. Automatically resetting the password when the connection is established makes sure that the password is not enabled for an unnecessarily long time especially in conjunction with automatic termination when a connection is not in use.	
fastconnectionstatereturnable	Fast return of a write/read job if the connection is interrupted. Data type Boolean, read-only. True: enabled False: disabled	
sendcredit	Maximum number of simultaneous network jobs, send direction Proposed value for connection establishment. Data type UInt16, read-only. >=1, proposed value for connection establishment Is set along with &receivecredit() in the configuration.	
receivecredit	Maximum number of simultaneous network jobs, receive direction Proposed value for connection establishment. Data type UInt16, read-only. >=1, proposed value for connection establishment	
pdu size	Size of the PDU Proposed value for connection establishment Data type UInt16, read-only. >=1, proposed value for connection establishment	
tradedsendcredit	The number of simultaneous protocol jobs in the send direction negotiated after connection establishment. Data type UInt16, read-only. If the connection is interrupted, the quality of this property is "BAD".	
tradedreceivecredit	The number of simultaneous protocol jobs in the receive direction negotiated after connection establishment. Data type UInt16, read-only. If the connection is interrupted, the quality of this property is "BAD".	
tradedpdu size	Size of the PDU, negotiated after connection establishment. Data type UInt16, read-only. If the connection is interrupted, the quality of this property is "BAD".	
connecttimeout	Connection establishment timeout Data type UInt32, read-only. 0: no timeout >0: timeout in ms	

S7 connection diagnostics property	Meaning
timeout	Job timeout for the productive traffic in ms. Data type UInt32, read-only. 0:no timeout >0:timeout in ms
abortconnectionafter	Automatic connection termination. Delay time for the automatic connection termination: After this time has elapsed, the OPC server automatically terminates the connection if there was no further variable access during this time. This allows the number of necessary connections to be reduced when variables are accessed at long intervals. Data type UInt32, read-only. 0:no termination >0:idle time before termination in ms
optimizes7read	Optimization of S7 read access to blocks. Data type Boolean, read and write. True: Optimization False: No optimization Optimization means: Several access jobs to individual variables are converted internally to a single array access to the communications partner.
optimizes7write	Optimization of S7 write access to blocks. Data type Boolean, read-only. True: Optimization False: No optimization Optimization means: Several access jobs to individual variables are converted internally to a single array access to the communications partner.
defaultalarmseverity	Default priority for unconfigured alarm events. Data type UInt16, read-only. 1:low priority ... 1000:high priority In the configuration, there is only one option for the default priority of alarms for S7 alarms and S7 diagnostics alarms.
defaultdiagnosisseverity	Default priority for unconfigured diagnostics events. Data type UInt16, read-only. 1:low priority ... 1000:high priority In the configuration, there is only one option for the default priority of alarms for S7 alarms and S7 diagnostics alarms.
events	Registering alarms and events on the connection partner. Data type UInt32, read-only. The individual values can be combined
	0x00000001 SCAN item (no longer supported)
	0x00000002 Simple alarms (no longer supported)
	0x00000004 Simple symbol-related alarms (no longer supported)

S7 connection diagnostics property	Meaning
	0x00000008 Simotion TO alarms 0x00000010 Connection monitoring alarms 0x00000020 Block-related alarms (as conditional events) 0x00000040 Symbol-related alarms (as conditional events) 0x00000080 Diagnostics alarms
connectiontype	S7 connection type Data type UInt16, read-only. 2:S7D_STD_TYPE; standard connection 3:S7D_H_TYPE; fault-tolerant connection If the S7 connection is not yet established, the quality "BAD" is reported for this item and the values are invalid.
connectionstate	S7 connection state Data type UInt16, read-only. 0x11:STD_DOWN; standard connection deliberately terminated 0x12:STD_ABORT; standard connection unintentionally terminated (error) 0x13:STD_NOT_USED; standard connection was never established 0x14:STD_OK; standard connection established 0x20:H_OK_RED; fault-tolerant connection established (redundant) 0x21:H_OK_RED_PATH_CHG; fault-tolerant connection established (redundant, there was a failover) 0x22:H_OK_NOT_RED; fault-tolerant connection established not redundant 0x23:H_ABORT; fault-tolerant connection unintentionally terminated (error) 0x24:H_NOT_USED; fault-tolerant connection was never established 0x25:H_DOWN; fault-tolerant connection deliberately terminated If the S7 connection is not yet established, the quality "BAD" is reported for this item and the values are invalid.
hconnectionwaystate	State of the H connection paths Data type Arrays Of UInt16, 4 array elements, read-only. 0x30:HW_PROD; path is productive connection 0x31:HW_STBY; path is standby connection 0x32:HW_ABORT; path was unintentionally terminated (error) 0x33:HW_NOT_USED; path was never established 0x34:HW_DOWN; path was deliberately terminated 0x35:HW_CN_BREAK; path could not be established If the S7 connection is not yet established, the quality "BAD" is reported for this item and the values are invalid.

2.7.11 S7 OPC UA template data variables

With the process variables for the S7 protocol with OPC UA, you have flexible setting options with which you can obtain the process data of your plant in the data formats you require.

The wide variety of addressing options cannot, however, be put together in a fully browsable namespace. Even a data block with the length of a single byte has approximately 40 different

data format options - starting with Byte, SByte, arrays with one element of these, individual bits, arrays of bits with up to 8 array elements starting at different bit offsets.

The OPC UA server therefore supports the user with so-called template data variables in the S7 namespace. In a typical OPC client text input box, these templates can be converted to valid ItemIDs simply by changing a few characters.

Example:

```
S7_connection1.db<db>.<o>,dw
```

By replacing <db> with the data block number and <o> the offset within the data block, you obtain a valid Nodeld.

```
-> S7_connection1.db10.4,dw
```

Further example:

```
S7_connection1.bsend<rid>.<l>,<o>,b,<c>
```

```
-> S7_connection1.bsend43.1000,0,b,100
```

The advantage of this concept is that it can be used by practically all OPC UA clients without any adaptation of the client being necessary.

Note

The usability of S7 OPC UA template data variables can be enabled and disabled in the "Communication Settings" configuration program in "OPC protocol selection" > click the arrow symbol beside "S7".

Template data variables within the browse hierarchy

The template data variables are sorted beside the corresponding folders in the namespace representation so that they can be used easily when required.

Special use of some of the attributes of the template data variables

The use of the OPC UA attributes is stipulated in the UA specification and requires no further explanation.

Example of the template of a data block byte variable:

Nodeld:	S7_connection1.db<db>.<o>,d
Browse name:	Template byte
Description:	<db>Address of the data block number <o> Offset within the file

2.7.12 Events, conditions and alarms

2.7.12.1 What alarms are there?

This section describes the mapping of S7 alarms and S7 diagnostics events to OPC UA events, conditions and alarms.

The following event and alarm types exist for OPC UA and S7:

- Statepath alarm
Alarms relating to the S7 connection status
- Symbol-related alarms (SCANs)
These allow the monitoring of bits in the areas I, Q, M and DB of the CPU asynchronous to the PLC user program.
- Block-related alarms (alarm SFB, alarm SFC)
Acknowledgeable alarm SFBs are: ALARM (SFB 33), ALARM_8 (SFB 34) and ALARM_8P (SFB 35)
The following SFBs are not acknowledgeable: NOTIFY (SFB 36) and NOTIFY_8P (SFB 31)
Acknowledgeable alarm SFCs are: ALARM_SQ (SFC 17) and ALARM_DQ (SFC 107)
The following SFCs are not acknowledgeable: ALARM_S (SFC 18) and ALARM_D (SFC 108)
- Diagnostics alarms
System diagnostics (ID 0x1000-0x79FF, 0xC000-0xEFFF, 0xF900-0xF9FF) user diagnostics (ID 0x8000-0xB9FF) with WR_USMSG (SFC 52)

2.7.12.2 What are events?

A plant is characterized by the states of its hardware and software components. With UA Alarming, you have the option of reporting status changes selected from all the states to the registered user as events. The information about the event is stored in its properties. Which properties an event has is defined by the event type.

The event type has its own properties or properties inherited from another event type (that may also in turn have inherited properties). The option of simple inheritance of properties leads to an event type hierarchy. The UA Alarming Specification includes numerous predefined event types structured in a predefined type hierarchy. The UA Alarming Specification also includes stipulations about the type of properties and the semantics of these properties. All predefined events are located in the namespace ns="http://opcfoundation.org/UA/".

2.7.12.3 Which events of the S7 UA Alarming server are there?

The S7 UA Alarming server is based on predefined event types and derives its own event types from the predefined event types. Separate event types are defined for S7 events. All S7 event types are located in the namespace ns="S7TYPES:".

S7 UA Alarming is used to represent S7 alarms. The table below shows which event types an S7 UA Alarming server reports. With the exception of the first event types, events with the specified event type occur only after receiving a mapped S7 alarm.

Table 2- 4 Event types of S7 UA Alarming and their use

Nodeld of the S7 event type	Display name	With OPC server 7.0, maps the following in the S7 system:	For OPC server 8.0, maps the following in the S7 system:
ns=S7Types, i=14	"S7StatepathAlarmType"	1)	1)
ns=S7Types, i=40	"S7ExclusiveLimitAlarmType"	-	All configured block- and symbol-related alarms with alarm class: "Alarm - high", "Alarm low", "Warning - high" or "Warning - low".
ns=S7Types, i=41	"S7ExclusiveDeviationAlarmType"	-	All configured block- and symbol-related alarms with alarm class: "Tolerance - high" or "Tolerance - low".
Ns=S7Types, i=43	"OffNormalAlarmType"	Block- and symbol-related alarms	<ol style="list-style-type: none"> 1. All unconfigured block- and symbol-related alarms and 2. configured alarms with other alarm classes other than "Tolerance - high", "Tolerance - low", "Alarm - high", "Alarm - low", "Warning - high" and "Warning - low").
Ns=S7Types, i=60	"S7DiagnosisEventType"	Diagnostics alarms	Configured or unconfigured diagnostics alarms

Note on 1): The event type ns="S7TYPES:", i=14 with the display name "S7StatepathAlarmType" maps the inverse state of an S7 connection ("inactive" if the S7 connection is established ("Up"); and "active" if the S7 connection is not established ("Down")). The status is obtained only at the PC end and, as a result, can also be reported when there is no physical connection to the S7 device.

2.7.12.4 Event type hierarchy of S7 UA Alarming servers

The event type hierarchy of the S7 UA alarming server consists of the standard event type hierarchy with some event types being derived from S7 event types.

The event type hierarchy can be browsed with the OPC Scout V10. When browsing, however, it is not the Nodelds of the elements that are displayed but their display names.

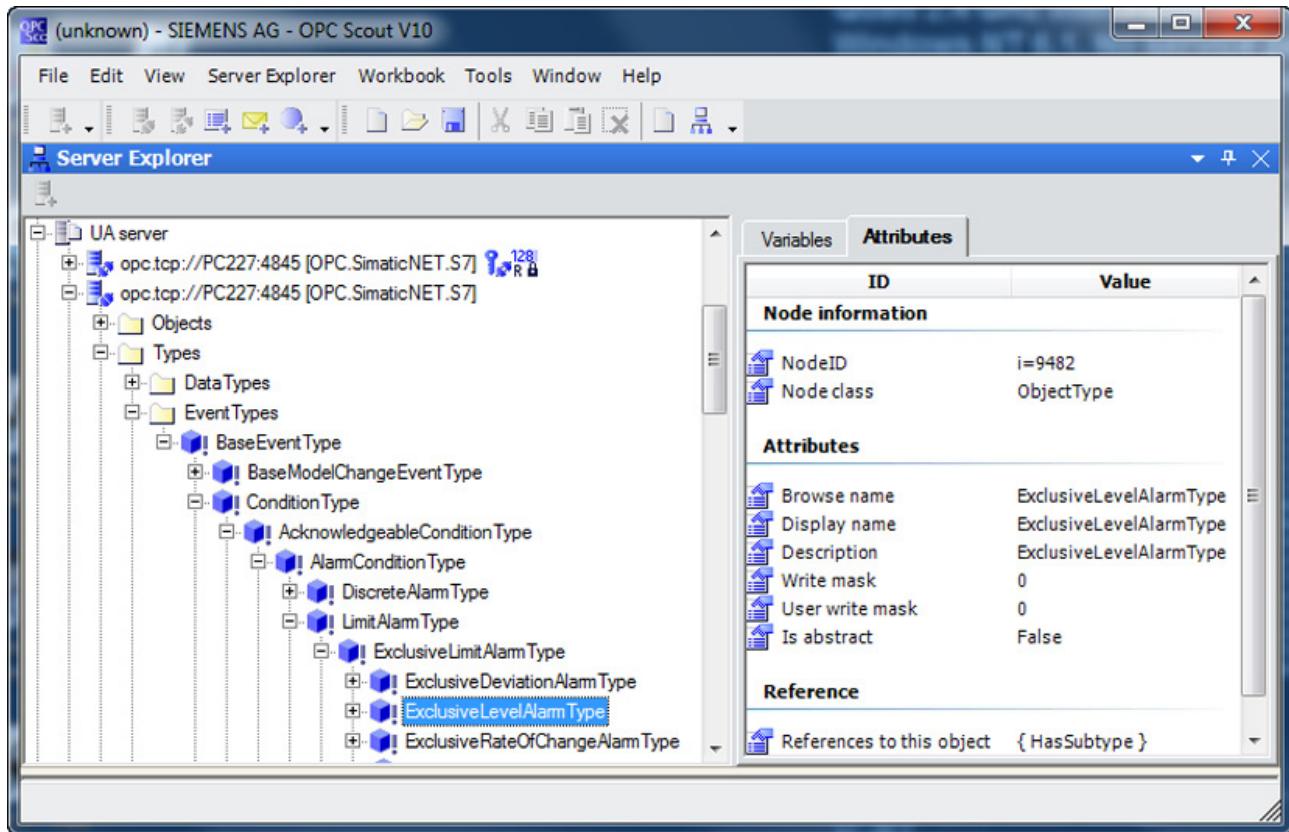


Figure 2-30 Mapping the event type hierarchy

With event types, inheritance is as follows:

Event type with display name	inherits from ...
"ExclusiveLevelAlarmType"	"ExclusiveLimitAlarmType"
"ExclusiveLimitAlarmType"	"LimitAlarmType"
"LimitAlarmType"	"AlarmConditionType"
"AlarmConditionType"	"AcknowledgeableConditionType"
"AcknowledgeableConditionType"	"ConditionType"

The event type with the display name "BaseEventType" is the type from which all event types are derived. This type defines all properties used in all events as well as their behavior.

An event type is a numeric NodId (for example ns="http://opcfoundation.org/UA/", i=2041 and display name ="BaseEventType").

In S7 UA Alarming, event types derived from the ConditionType are instantiated. An S7 alarm is therefore mapped with all its properties to an alarm object: the condition instance. The properties of the condition instance can also be read or monitored with Data Access.

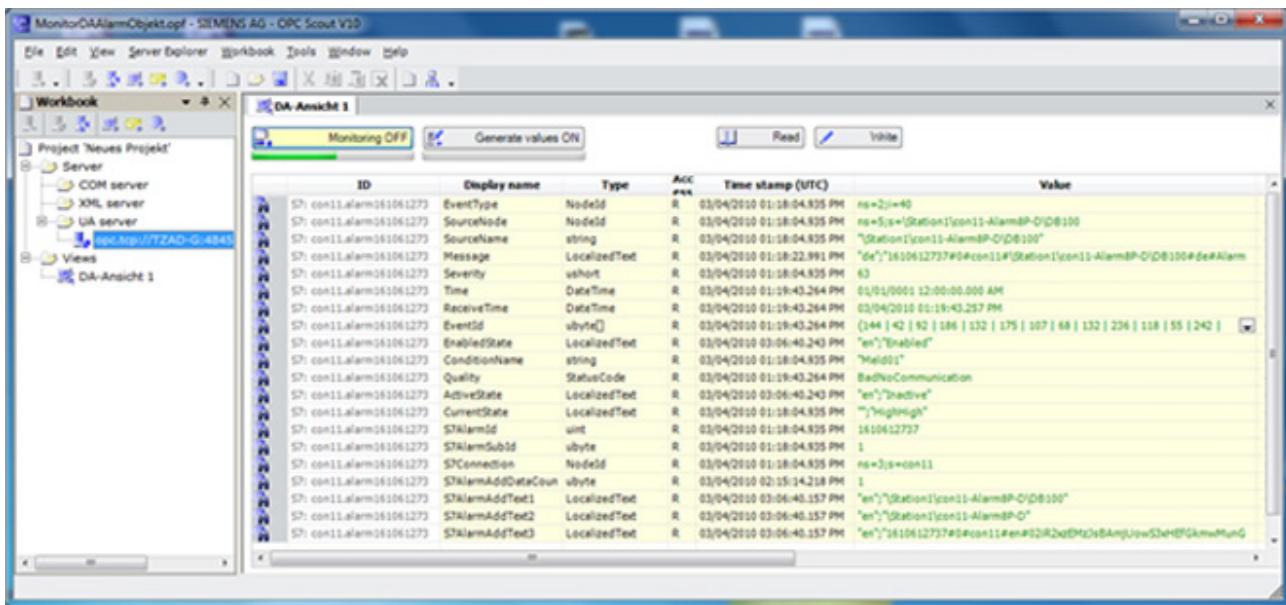


Figure 2-31 Monitoring alarm objects with Data Access

The following sections describe the properties relevant to the user when programming a UA application. They are grouped according to the event types in which they are defined. When referencing a property, the data type "QualifiedName" is used. "QualifiedName" contains a namespace and a browse name (sometimes a browse path).

Since, however, in the context of the document, only properties will be referenced whose "QualifiedNames" have a namespace that is identical to the namespace of the defining types, the specification of the namespace will be omitted. Instead of "QualifiedName", the browse name of the property will be specified. In addition to this, the requested attribute separated by ":" from the data type is shown in parentheses. In most cases, the requested attribute is numeric 13 which specifies the "Value". In a few cases, the numeric value 1 occurs and this specifies the NodId.

2.7.13 Standard event types

2.7.13.1 Standard event type with the display name "BaseEventType"

NodId: ns="http://opcfoundation.org/UA/", i=2041

Derived from: Is not derived from any other EventType.

Browse name of the relevant properties

- "EventType" (13|NodId)
- "SourceNode" (13|NodId)
- "SourceName" (13|String)
- "Message" (13|LocalizedText)
- "Severity" (13|UInt16)
- "Time" (13|DateTime)
- "ReceiveTime" (13|DateTime)
- "EventId" (13|ByteString)

S7 UA event types directly derived from this type: ns="S7TYPES:", i =60, display name="S7DiagnosisEventType". It has no condition instance. An event of this event type is uniquely identified by the source name.

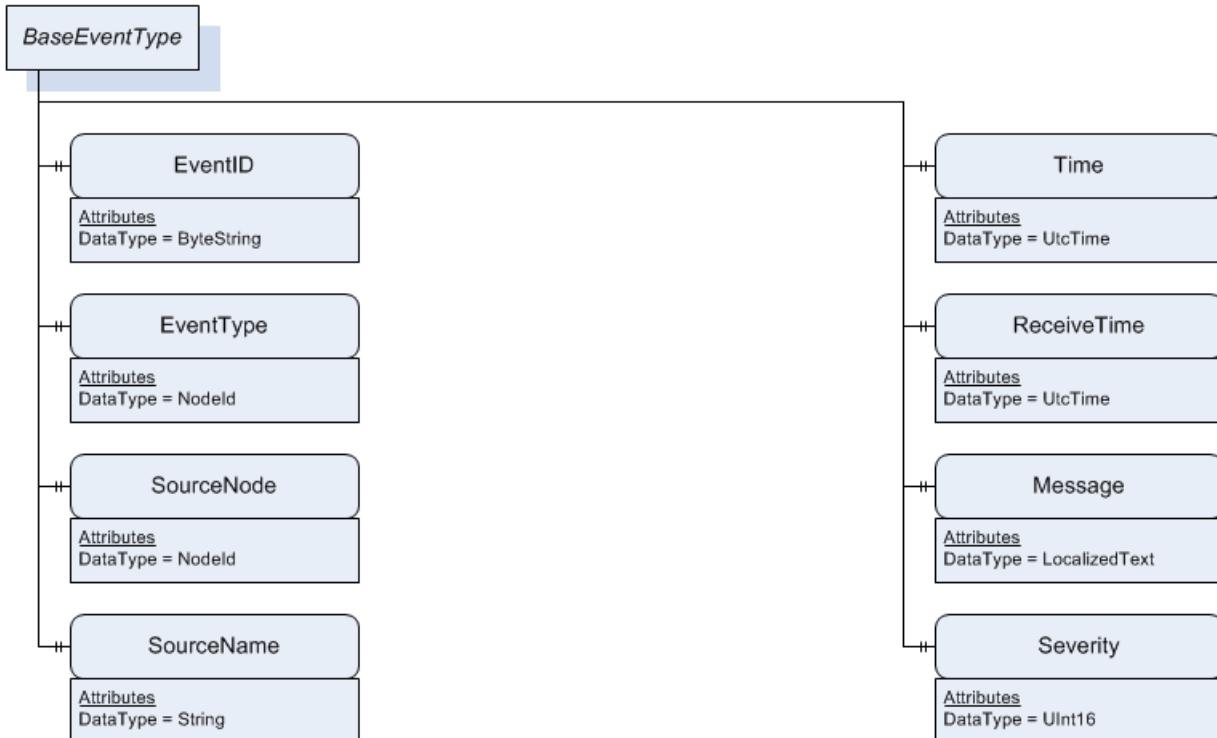


Figure 2-32 Example of the representation of `BaseEventType`

EventType

is always one of the S7 event types listed earlier.

SourceNode

Source node of an event; in S7 UA Alarming, an object specified by a `Nodeld` in `ns="S7SOURCES:"` or the S7 connection object in `ns="S7:"`. See also section "Area tree and source space (Page 219)".

SourceName

non-localized name of the source. Refer to the following table "Formation of the source name".

Message

Refer to the following table "Formation of the message".

Severity

For priority, refer to the following table "Formation of Severity".

Time

Time stamp as close to the process as possible. This is decided by the configuration and the following options are available:

- CPU time stamp
- CPU time + offset
- PC time (UTC).

With PC time(UTC), "Time" is identical to "ReceiveTIme".

ReceiveTime

Time stamp of the PC.

EventId

An identifier (opaque Id) that uniquely identifies and references an event. The client requires the EventId, for example to acknowledge alarms.

Formation of the source name

Nodeld of the S7 event type	Display name	Formation rule for OPC server 7.0:	Formation rule for OPC server 8.0:
ns=S7TYPES:, i=14	"S7StatepathAlarmType"	S7 connection name + ".statepath". Example con13.statepath	S7 connection name + ".statepath" Example: con13.statepath. A corresponding SourceNode exists.
ns= S7TYPES:, i=40	"S7ExclusiveLimitAlarmType"	S7 connection name Example: con13	<ol style="list-style-type: none"> 1. Unconfigured S7 alarms: S7 connection name Example: con13 2. Configured alarms: Additional text 1 A corresponding SourceNode exists. 3. Configured alarms without additional text 1: Plant_path + "\" + block (or symbol name with SCAN) . Example: Station1\con13-Scan-Notify\S7_program(3)\DB604. A corresponding SourceNode exists.

Nodeld of the S7 event type	Display name	Formation rule for OPC server 7.0:	Formation rule for OPC server 8.0:
ns= S7TYPES:, i=41	"S7ExclusiveDeviationAlarmType"	S7 connection name example: con13	<ol style="list-style-type: none"> 1. Unconfigured S7 alarms: S7 connection name Example: con13 2. Configured alarms: Additional text 1 A corresponding source node exists. 3. Configured alarms without additional text 1: Plant_path + "\" + block (or symbol name with SCAN) . Example: Station1\con13-Scan-Notify\S7_program(3)\DB604. A corresponding SourceNode exists.
ns= S7TYPES:, i=43	"OffNormalAlarmType"	S7 connection name Example: con13	<ol style="list-style-type: none"> 1. Unconfigured S7 alarms: S7 connection name Example: con13 2. Configured alarms: Additional text 1. A corresponding SourceNode exists. 3. Configured alarms without additional text 1: Plant_path + "\" + block (or symbol name with SCAN) . Example: Station1\con13-Scan-Notify\S7_program(3)\DB604. A corresponding SourceNode exists.
ns= S7TYPES:, i=60	"S7DiagnosisEventType"	S7 connection name + ".diagnosis" + alarm number as hex value Example: Con13.diagnosis0xA001	<ol style="list-style-type: none"> 1. Unconfigured alarms: S7 connection name + ".diagnosis" + alarm number as hex value Example: Con13.diagnosis0xA001 2. Configured alarms: Plant-path + "\" + alarm identifier Example: Station1\con13-Scan-Notify\S7_program(3)\DB604 3. Configured alarms: Plant-path + "\" + alarm identifier. Example: Station1\con13-Scan-Notify\S7_program(3)\WR_USMS G (1). A corresponding SourceNode exists.

Forming the alarm

NodeID of the S7 event type	Display name	Formation rule for OPC server 7.0:	Formation rule for OPC server 8.0:
ns= S7TYPES:, i=14	"S7StatepathAlarmType"	"statepath"	"statepath"
ns= S7TYPES:, i=40	"S7ExclusiveLimitAlarmType"	Connection name + ".alarm" + alarm number Example: con13.alarm1	1. Unconfigured S7 alarms: S7 connection name + ".alarm"+ alarm number Example: con13.alarm1 2. Configured alarms: Alarm text
ns= S7TYPES:, i=41	"S7ExclusiveDeviationAlarmType"	Connection name + ".alarm" + alarm number Example: con13.alarm1	1. Unconfigured S7 alarms: S7 connection name + ".alarm"+ alarm number Example: con13.alarm1 2. Configured alarms: Alarm text
ns= S7TYPES:, i=43	"OffNormalAlarmType"	Connection name + ".alarm" + alarm number Example: con13.alarm1	1. Unconfigured S7 alarms: S7 connection name + ".alarm"+ alarm number Example: con13.alarm1 2. Configured alarms: Alarm text
ns= S7TYPES:, i=60	"S7DiagnosisEventType"	S7 Connection name + ".diagnosis" + hex value Example: con13.diagnosis0xA001	1. Unconfigured alarms: S7 Connection name + ".diagnosis" + hex value. Example: con13.diagnosis0xA001 2. Configured alarms: Alarm text entering state and Alarm text exiting state

Formation of Severity

NodeID of the S7 event type	Display name	Formation rule for OPC server 7.0:	Formation rule for OPC server 8.0:
ns= S7TYPES:, i=14	"S7StatepathAlarmType"	Default priority for alarms for every S7 connection.	Default priority for alarms for every S7 connection.
ns= S7TYPES:, i=40	"S7ExclusiveLimitAlarmType"	-	1. The priority is detected for each separate alarm on every S7 connection if it exists or it is derived from the S7 alarm priority (0 to 16) of the blocks and symbols.

Nodeld of the S7 event type	Display name	Formation rule for OPC server 7.0:	Formation rule for OPC server 8.0:
ns= S7TYPES; i=41	"S7ExclusiveDeviationAlarmType"	-	1. The priority is detected for each separate alarm on every S7 connection if it exists or it is derived from the S7 alarm priority (0 to 16) of the blocks and symbols.
ns= S7TYPES; i=43	"OffNormalAlarmType"	Default priority for alarms for every S7 connection. Priority detected according to alarm number if it exists.	1. Unconfigured S7 alarms: Default priority for alarms for every S7 connection or priority obtained individually from alarm number. 2. Configured alarms: The priority is detected for each S7 connection if it exists or it is derived from the S7 alarm priority (0 to 16) of the blocks and symbols.
ns= S7TYPES; i=60	"S7DiagnosisEvent Type"	Default priority for alarms for every S7 connection or Priority based on alarm number.	1. Default priority for alarms for every S7 connection or 2. Priority based on alarm number if it exists.

Table 2- 5 Conversion table S7 alarm priority severity

S7 alarm priority	Severity
0	1
1	63
2	125
3	188
4	250
5	313
6	375
7	438
8	500
9	563
10	625
11	688
12	750
13	813
14	875
15	938
16	1000

2.7.13.2 Standard event type with the display name "ConditionType"

Nodeld: ns:"<http://opcfoundation.org/UA/>", i=2782

Derived from: ns="<http://opcfoundation.org/UA>", i=2041 with display name "BaseEventType"

Browse name of the relevant properties

"" (1|Nodeld)

"ConditionName" (13|String)

"EnableState|ID" (13|Boolean)(Browse-Path)

"EnableState" (13|LocalizedText)

"Quality" (13|StatusCode)

S7 event types directly derived from this type: None

Event types derived directly or indirectly from this type have a condition instance.

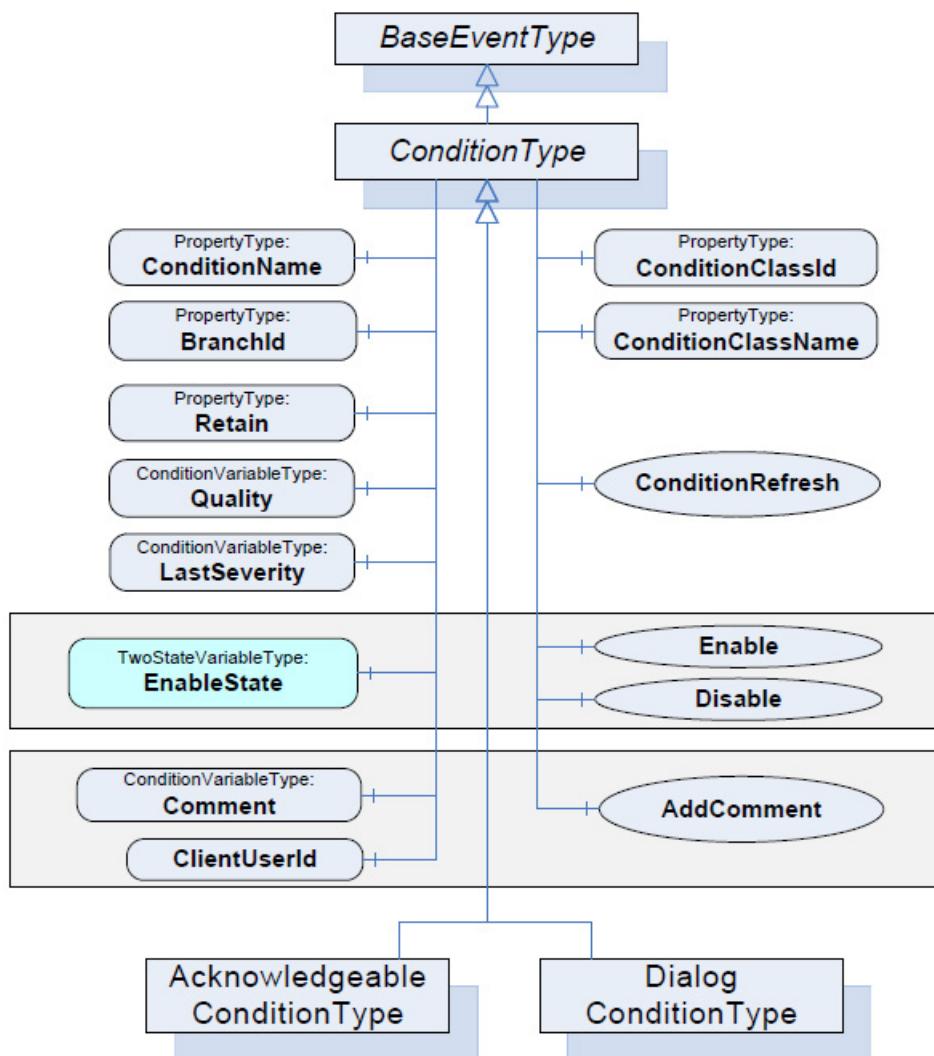


Figure 2-33 Example of the representation of ConditionType

ConditionName

An event of a condition is identified uniquely by the combination of "SourceName" and "ConditionName". See below, section "Forming the condition name".

EnableState

Is part of a state machine. Possible values according to UA Alarming Specification for EnableState (LocalizedText) are "de";"Enabled" and "de";"Disabled". S7 UA Alarming does not support disabling, so "en","Enabled" always appears.

Quality

For the possible quality values, refer to the UA Specification.

Formation of the condition name

Nodeld of the S7 event type	Display name	Formation rule for OPC server 7.0:	Formation rule for OPC server 8.0:
ns=S7TYPES:, i=14	"S7StatepathAlarmType"	"statepath"	"statepath"
ns= S7TYPES:, i=40	"S7ExclusiveLimitAlarmType"	"alarm" + Alarm number. Example: alarm1	<p>1. Unconfigured S7 alarms: "alarm"+ alarm number Example: alarm1</p> <p>2. Configured alarms:</p> <p>Block-related alarms: Name of the input variable of the generating function block marked with the S7_a_type attribute. With the signals 2 to 8 followed by "," and the signal number. Example: Alar01, or Alar01,</p> <p>symbol-related alarms: Name of the symbol Example: Scan01</p>

NodeID of the S7 event type	Display name	Formation rule for OPC server 7.0:	Formation rule for OPC server 8.0:
ns= S7TYPES:, i=41	"S7ExclusiveDeviationAlarmType"	<p>"alarm" + Alarm number. Example: alarm1</p>	<p>1. Unconfigured S7 alarms: "alarm"+ alarm number Example: alarm1</p> <p>2. Configured alarms: Block-related alarms: Name of the input variable of the generating function block marked with the S7_a_type attribute. With the signals 2 to 8 followed by "," and the signal num- ber. Example: Alar01, or Alar01, symbol-related alarms: Name of the symbol Example: Scan01</p>
ns= S7TYPES:, i=43	"OffNormalAlarmType"	<p>"alarm" + Alarm number. Example: alarm1</p>	<p>1. Unconfigured S7 alarms: "alarm"+ alarm number Example: alarm1</p> <p>2. Configured alarms: Block-related alarms: Name of the input variable of the generating function block marked with the S7_a_type attribute. With the signals 2 to 8 followed by "," and the signal num- ber. Example: Alar01, or Alar01, symbol-related alarms: Name of the symbol Example: Scan01</p>

2.7.13.3 Standard event type with the display name "AcknowledgeableConditionType"

NodeId: ns="http://opcfoundation.org/UA/", i=2881

Derived from: ns="http://opcfoundation.org/UA", i=2782 with display name "ConditionType"

Browse name of the relevant properties

"AckedState|Id" (13|Boolean)

"AckedState" (13|LocalizedText)

S7 event types directly derived from this type: None

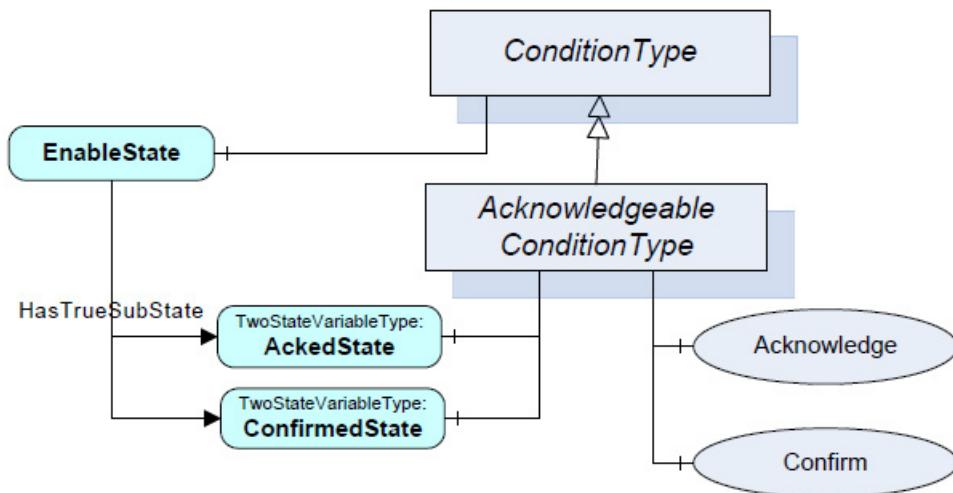


Figure 2-34 Example of the representation of AcknowledgeableConditionType

AckedState

Is part of a state machine. Possible values are "en", "Acknowledged" and "en", "Unacknowledged". Each value change of AckedState is reported with an event. Events reported with the AckedState "en", "Unacknowledged" must be acknowledged with the acknowledge function. Refer to the STEP 7 description of the block interrupts that require an acknowledgment.

2.7.13.4 Standard event type with the display name "AlarmConditionType"

NodeId: ns="http://opcfoundation.org/UA/", i=2915

Derived from: ns="http://opcfoundation.org/UA", i=2881 with display name

"AcknowledgeableConditionType"

Browse name of the relevant properties

"ActiveState|Id" (13|Boolean)

"ActiveState" (13|LocalizedText)

S7 event types directly derived from this type: None

Possible values according to UA Alarming Specification for ActiveState (LocalizedText) are "en"; "Active" and "en"; "Inactive".

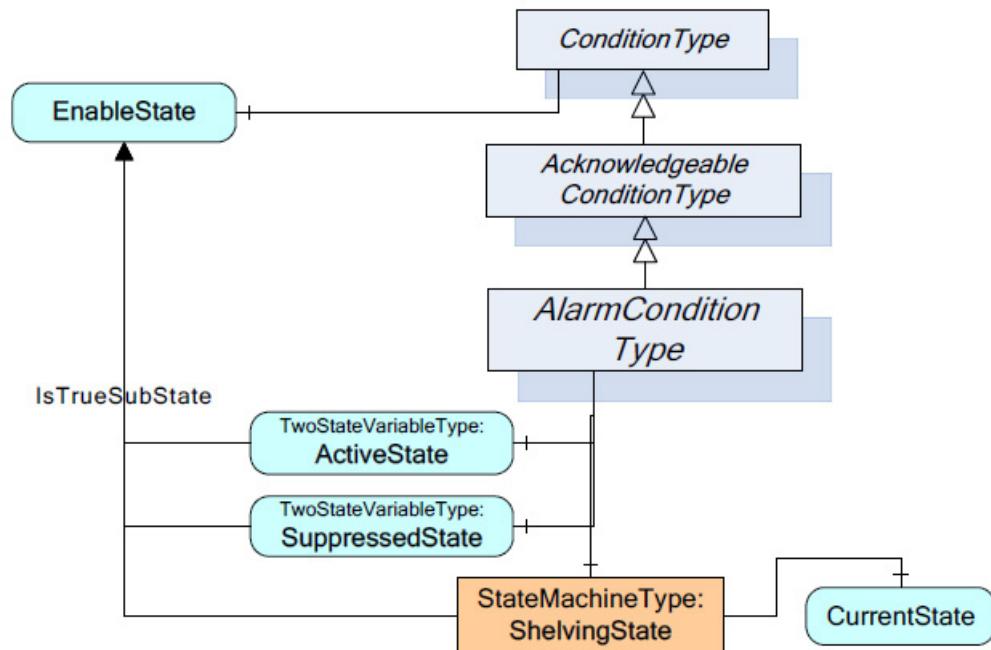


Figure 2-35 Example of the representation of AlarmConditionType

ActiveState

Is part of a state machine. In S7 UA Alarming, the ActiveState is controlled by block- and symbol-related alarms. If the alarm triggering signal has the value "1", AckedState "Active" is reported, if it has the value "0", "Inactive" is reported. Each value change of ActiveState is reported by an event.

ActiveState with event type with the display name = "S7StatepathAlarmType"

Here, the "Active" state is reached when the S7 connection is not established. The "Inactive" state is reached when the S7 connection is established.

2.7.13.5 Standard event type with the display name "ExclusiveLimitAlarmType"

NodId: ns="http://opcfoundation.org/UA/", i=9341

Derived indirectly from: ns="http://opcfoundation.org/UA/", i=2915 with display name "AlarmConditionType"

Browse name of the relevant properties:

"LimitState|CurrentState" (13|LocalizedText)

S7 UA event types directly derived from this type: None

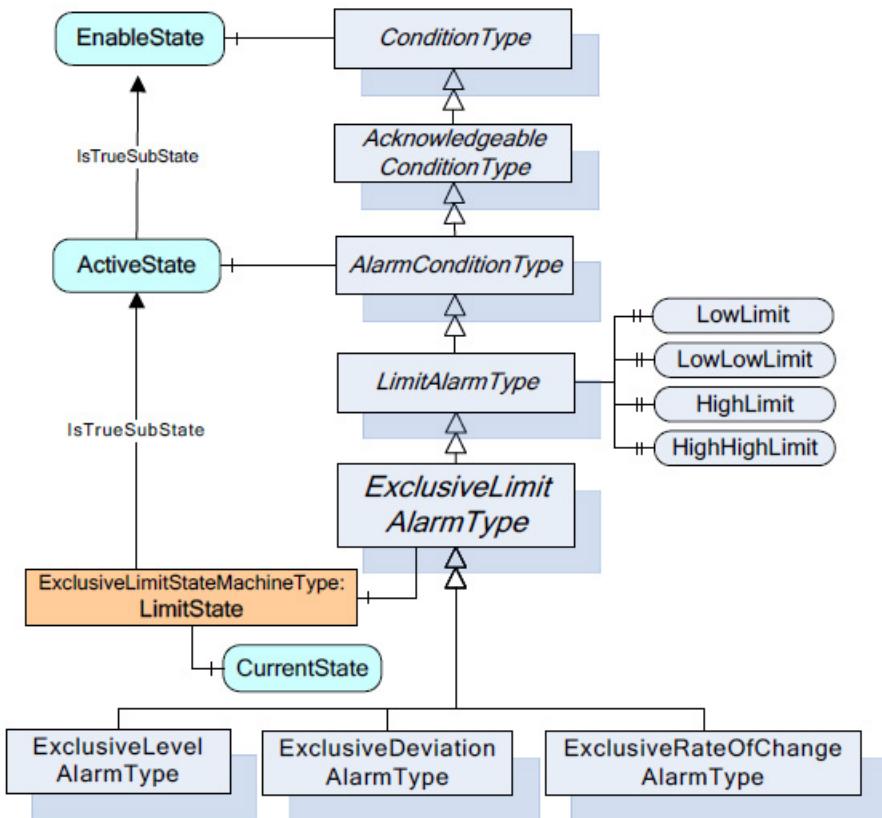


Figure 2-36 Example of the representation of ExclusiveLimitAlarmType

LimitState|CurrentState

Possible values are "High", "HighHigh", "Low" and "LowLow".

2.7.13.6 Standard event type with the display name "ExclusiveLevelAlarmType"

NodId: ns="http://opcfoundation.org/UA/", i=9482

Derived from: ns="http://opcfoundation.org/UA/", i=9341 display name "ExclusiveLimitAlarmType"

Browse name of the relevant properties: None

S7 UA event type directly derived from this type:

ns="S7TYPES:", i =40, display name="S7ExclusiveLevelAlarm Type"

The event type with the display name "S7ExclusiveLevelAlarm Type" is generated when an OPC server 8.0 is selected and the following is set for the alarm class of the alarms

"Alarm - high" (LimitState|CurrentState="HighHigh"),
 "Alarm – low" (LimitState|CurrentState="LowLow"),
 "Warning - high" (LimitState|CurrentState="High") and
 "Warning - low" (LimitState|CurrentState="Low").

2.7.13.7 Standard event type with the display name "ExclusiveDeviationAlarmType"

NodeID: ns="http://opcfoundation.org/UA/", i=9764

Derived from: ns="http://opcfoundation.org/UA/", i=9341 display name "ExclusiveLimitAlarm Type"

Browse name of the relevant properties: None

S7 UA event type directly derived from this type:

ns=S7Types, i =41, display name="S7ExclusiveDeviationAlarm Type"

The event type with the display name "S7ExclusiveDeviationAlarm Type" is generated when an OPC server 8.0 is selected and "Tolerance - high" or "Tolerance - low" is set for the alarm class of the alarms.

2.7.13.8 Standard event type with the display name "OffNormalAlarmType"

NodeID: ns="http://opcfoundation.org/UA/", i=10637

Indirectly derived from: ns="http://opcfoundation.org/UA/", i=2915 with display name "AlarmCondition Type"

Browse name of the relevant properties: None

S7 UA event types directly derived from this type:

ns=S7TYPES:, i = 43, display name="S7OffNormalAlarm Type"

ns=S7TYPES:, i =14, display name="S7StatepathAlarm Type"

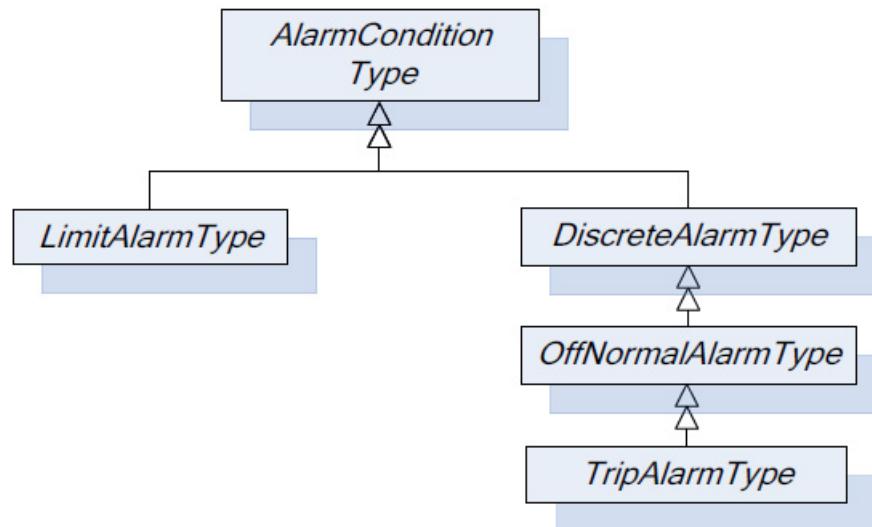


Figure 2-37 Example of the representation of OffNormalAlarm Type

The event type with the display name "S7OffNormalAlarm Type" is generated when an OPC server 7.0 is selected or OPC server 8.0 and alarm classes other than "Tolerance - high", "Tolerance - low", "Alarm - high", "Alarm - low", "Warning - high" and "Warning - low" are set for the alarms.

2.7.14 S7 event types

2.7.14.1 S7 event type with the display name "S7StatepathIAlarmType"

NodId: ns="S7TYPES:", i=14

Indirectly derived from: ns="http://opcfoundation.org/UA/", i=2915 with display name "AlarmCondition Type"

Browse name of the relevant properties:
"S7Connection" (13|NodId)

This event type maps the inverse state of an S7 connection ("inactive" if the S7 connection is established ("Up"); and "active" if the S7 connection is not established ("Down")). The status is obtained only at the PC end and, as a result, can also be reported when there is no physical connection to the S7 device.

S7Connection

Specifies the NodId of the connection via which an S7 alarm is received.

Example: ns="S7:", s="connectionname".

2.7.14.2 S7 event type with the display name "S7ExclusiveLevelAlarmType"

Nodeld: ns="S7TYPES.", i=40
 Indirectly derived from: ns="http://opcfoundation.org/UA/", i=9482 with display name "ExclusiveLevelAlarm Type"
 Browse name of the relevant properties:
 "S7AlarmId" (13|UInt32)
 "S7AlarmSubId" (13|Byte)
 "S7Connection" (13|Nodeld)
 "S7Time" (13|DateTime)
 "S7AlarmAddDataCount" (13|Byte)
 "S7AlarmAddData1|DataType" (13|Byte)
 "S7AlarmAddData1|Data" (13|ByteString)
 "S7AlarmAddData2|DataType" (13|Byte)
 "S7AlarmAddData2|Data" (13|ByteString)
 "S7AlarmAddData3|DataType" (13|Byte)
 "S7AlarmAddData3|Data" (13|ByteString)
 "S7AlarmAddData4|DataType" (13|Byte)
 "S7AlarmAddData4|Data" (13|ByteString)
 "S7AlarmAddData5|DataType" (13|Byte)
 "S7AlarmAddData5|Data" (13|ByteString)
 "S7AlarmAddData6|DataType" (13|Byte)
 "S7AlarmAddData6|Data" (13|ByteString)
 "S7AlarmAddData7|DataType" (13|Byte)
 "S7AlarmAddData7|Data" (13|ByteString)
 "S7AlarmAddData8|DataType" (13|Byte)
 "S7AlarmAddData8|Data" (13|ByteString)
 "S7AlarmAddData9|DataType" (13|Byte)
 "S7AlarmAddData9|Data" (13|ByteString)
 "S7AlarmAddData10|DataType3|Byte)
 "S7AlarmAddData10|Data" (13|ByteString)
 "S7AlarmAddText1" (13|LocalizedText)
 "S7AlarmAddText2" (13|LocalizedText)
 "S7AlarmAddText3" (13|LocalizedText)
 "S7AlarmAddText4" (13|LocalizedText)
 "S7AlarmAddText5" (13|LocalizedText)
 "S7AlarmAddText6" (13|LocalizedText)
 "S7AlarmAddText7" (13|LocalizedText)
 "S7AlarmAddText8" (13|LocalizedText)

With OPC servers 8.0, this S7 event type maps all configured block- and symbol-related alarms with the alarm classes "Alarm - high", "Alarm low", "Warning - high" or "Warning - low". This S7 event type is not used with OPC servers 7.0.

S7AlarmId

Alarm number of the S7 alarm.

S7AlarmSubId

The number of the triggering signal of an S7 alarm. The range of values is 1 to 8.

S7Connection

Specifies the Nodeld of the connection via which the S7 alarm is received.
Example: ns="S7:", s="connectionname".

S7AlarmAddData|Data

Are the associated values of the generating S7 alarms.
Alarms shown in ByteString format. $1 \leq n \leq 10$

S7AlarmAddData8|DataType

Are the S7 data types of the associated values. Mapping the S7 data types to UA data types is not possible because they cannot be mapped unequivocally.

S7AlarmAddDataCount

Is the actual number of associated values of the S7 alarms.

S7AlarmAddTextn

$1 \leq n \leq 8$. Additional texts from the configuration.

2.7.14.3 S7 event type with the display name "S7ExclusiveDeviationAlarmType"

Nodeld: ns="S7TYPES.", i=41

Indirectly derived from: ns="http://opcfoundation.org/UA/", i=9764 with display name "ExclusiveDeviationAlarm Type"

Browse name of the relevant properties:

"S7AlarmId" (13|UInt32)
"S7AlarmSubId" (13|Byte)
"S7Connection" (13|Nodeld)
"S7AlarmAddDataCount" (13|Byte)
"S7Time" (13|DateTime)
"S7AlarmAddData1|DataType" (13|Byte)
"S7AlarmAddData1|Data" (13|ByteString)
"S7AlarmAddData2|DataType" (13|Byte)
"S7AlarmAddData2|Data" (13|ByteString)
"S7AlarmAddData3|DataType" (13|Byte)
"S7AlarmAddData3|Data" (13|ByteString)
"S7AlarmAddData4|DataType" (13|Byte)
"S7AlarmAddData4|Data" (13|ByteString)
"S7AlarmAddData5|DataType" (13|Byte)
"S7AlarmAddData5|Data" (13|ByteString)
"S7AlarmAddData6|DataType" (13|Byte)
"S7AlarmAddData6|Data" (13|ByteString)
"S7AlarmAddData7|DataType" (13|Byte)
"S7AlarmAddData7|Data" (13|ByteString)
"S7AlarmAddData8|DataType" (13|Byte)
"S7AlarmAddData8|Data" (13|ByteString)
"S7AlarmAddData9|DataType" (13|Byte)
"S7AlarmAddData9|Data" (13|ByteString)
"S7AlarmAddData10|DataType" (13|Byte)
"S7AlarmAddData10|Data" (13|ByteString)
"S7AlarmAddText2" (13|LocalizedText)
"S7AlarmAddText3" (13|LocalizedText)
"S7AlarmAddText4" (13|LocalizedText)
"S7AlarmAddText5" (13|LocalizedText)
"S7AlarmAddText6" (13|LocalizedText)
"S7AlarmAddText7" (13|LocalizedText)
"S7AlarmAddText8" (13|LocalizedText)

With OPC servers 8.0, this S7 event type maps all configured block- and symbol-related alarms with the alarm classes "Tolerance - high" or "Tolerance - low". Refer to the notes on "S7ExclusiveLevelAlarm Type".

2.7.14.4 S7 event type with the display name "S7OffNormalAlarmType"

NodId: ns="S7TYPES:", i=43
Indirectly derived from: .ns="http://opcfoundation.org/UA/", i=10637 with display name "OffNormalAlarm Type"
Browse name of the relevant properties:
"S7AlarmId" (13|UInt32)
"S7AlarmSubId" (13|Byte)
"S7Connection" (13|NodId)
"S7AlarmAddDataCount" (13|Byte)
"S7Time" (13|DateTime)
"S7AlarmAddData1|DataType" (13|Byte)
"S7AlarmAddData1|Data" (13|ByteString)
"S7AlarmAddData2|DataType" (13|Byte)
"S7AlarmAddData2|Data" (13|ByteString)
"S7AlarmAddData3|DataType" (13|Byte)
"S7AlarmAddData3|Data" (13|ByteString)
"S7AlarmAddData4|DataType" (13|Byte)
"S7AlarmAddData4|Data" (13|ByteString)
"S7AlarmAddData5|DataType" (13|Byte)
"S7AlarmAddData5|Data" (13|ByteString)
"S7AlarmAddData6|DataType" (13|Byte)
"S7AlarmAddData6|Data" (13|ByteString)
"S7AlarmAddData7|DataType" (13|Byte)
"S7AlarmAddData7|Data" (13|ByteString)
"S7AlarmAddData8|DataType" (13|Byte)
"S7AlarmAddData8|Data" (13|ByteString)
"S7AlarmAddData9|DataType" (13|Byte)
"S7AlarmAddData9|Data" (13|ByteString)
"S7AlarmAddData10|DataType" (13|Byte)
"S7AlarmAddData10|Data" (13|ByteString)
"S7AlarmAddText2" (13|LocalizedText)
"S7AlarmAddText3" (13|LocalizedText)
"S7AlarmAddText4" (13|LocalizedText)
"S7AlarmAddText5" (13|LocalizedText)
"S7AlarmAddText6" (13|LocalizedText)
"S7AlarmAddText7" (13|LocalizedText)
"S7AlarmAddText8" (13|LocalizedText)

This event type maps block-and symbol-related alarms with OPC servers 7.0. With OPC servers 8.0, it maps all unconfigured block- and symbol-related alarms and configured alarms with other alarm classes other than "Tolerance - high", "Tolerance - low", "Alarm - high", "Alarm - low", "Warning - high" and "Warning - low"). Refer to the notes on "S7ExclusiveLevelAlarm Type".

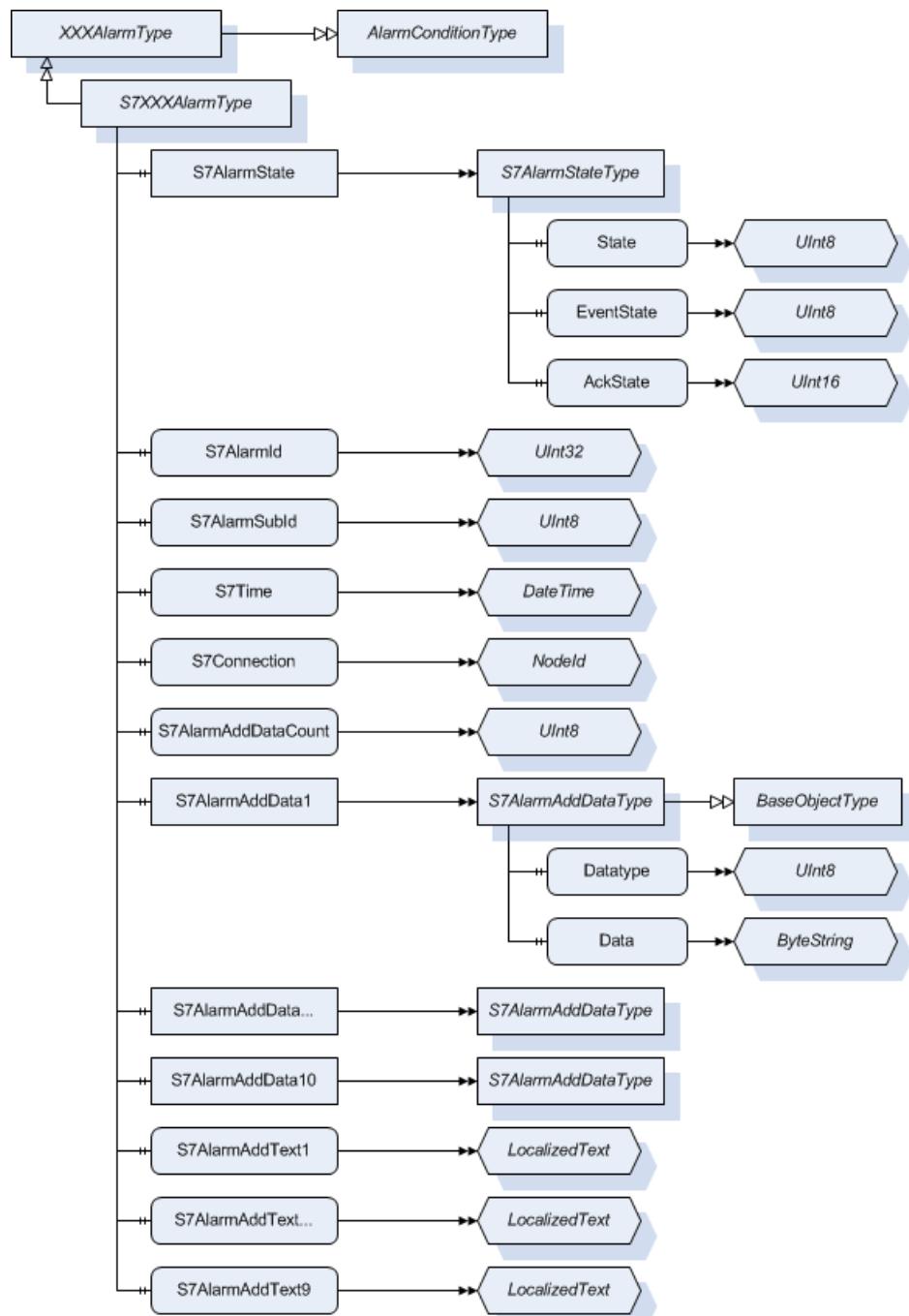


Figure 2-38 Example of the representation of S7-specific properties for the S7OffNormalAlarm Type and the other S7XXXAlarm Types

2.7.14.5 S7 event type with the display name "S7DiagnosisEventType"

Nodeld: ns="S7TYPES:", i=60

Indirectly derived from: .ns="http://opcfoundation.org/UA/", i=2041 with display name "BaseEvent Type"

Browse name of the relevant properties:

"S7DiagnosisId" (13|UInt32)

"S7Connection" (13|Nodeld)

"S7DiagnosisData" (13|ByteString)

"S7Time" (13|DateTime)

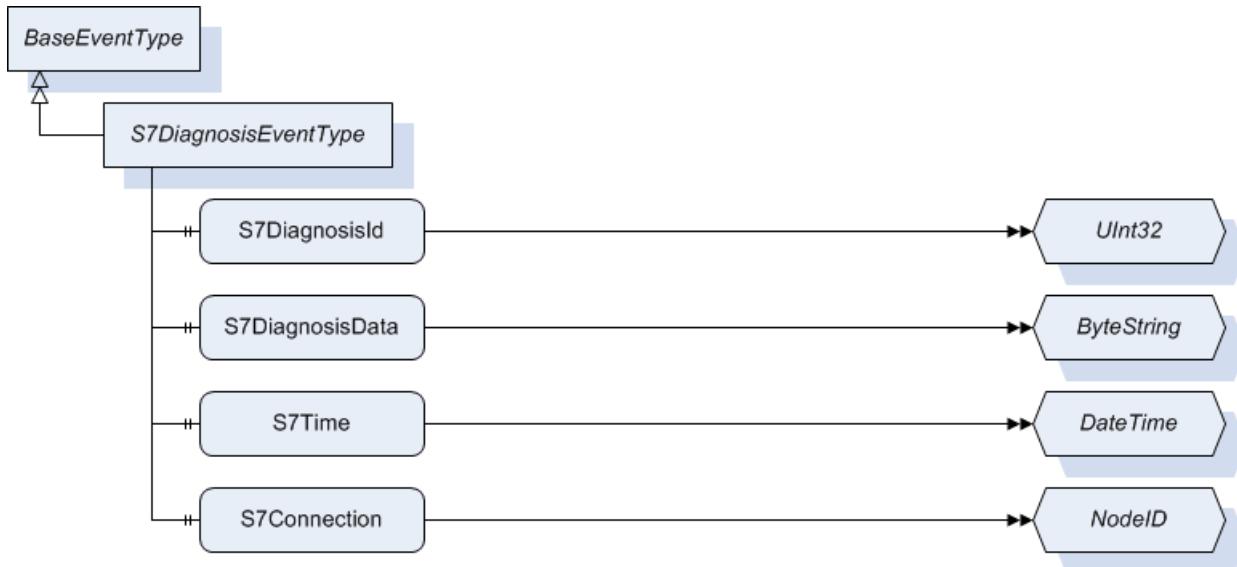


Figure 2-39 Example of the representation of S7DiagnosisEventType

S7DiagnosisId

Alarm ID of the diagnostics alarm.

S7Connection

Specifies the Nodeld of the connection via which the diagnostics alarm is received.
Example: ns="S7:", s="connectionname".

S7DiagnosisData

Diagnostics data

Note

You will find a detailed description of SFC 52 (S7 diagnostics data) in the document "STEP 7 System and Standard Functions for S7-300/400".

2.7.15 Area tree and source space

With the OPC server 7.0, there is no area tree and no source space. With the OPC server 8.0, the area tree and the source space contain the SourceNodes of configured S7 alarms. Unconfigured S7 alarms have no SourceNodes.

S7 UA Alarming sets up the area tree to assign the SourceNodes to areas. Nodes of the area tree are special UA folders that map areas of the plant. These special UA folders are known as area nodes.

Areas of a plant are defined as "additional text 2" of all block- and symbol-related messages, for example the additional text "machineroom\tank" defines the higher-level area "machineroom" and the lower-level area "tank" with the Nodelds of the corresponding UA folders

- ns="S7AREAS:", s="machineroom".
- ns="S7AREAS:", s="machineroom\tank".

The "additional text 1" of the block- and symbol-related alarm generates a source node. Assuming, for example, the "additional text 1" is "overpressure sensor", the source node with the following Nodeld corresponds to this additional text

- ns="S7SOURCES:", s="overpressure sensor".

The "additional text 1" can include "\", this, however, does not lead to several SourceNodes.

Specifying an "additional text 1" and "additional text 2" is not necessary. If these do not exist, the path in the plant tree in STEP 7 is used as a substitute for "additional text 2" and the path in the plant tree in STEP 7 + the alarm instance block or symbol name (with SCAN) is used as a substitute for "additional text 1". For example, in the alarm instance data block DB404 in "S7_program(1)" of the CPU "CPU 416-3" of the AS device "Station1", the alarm generates the areas "Station1" with subordinate areas "CPU 416-3" and "S7_program(1)" with the Nodelds of the corresponding UA folder

- ns="S7AREAS:", s="Station"
- ns="S7AREAS:", s="Station1\CPU 416-3"
- ns="S7AREAS:", s="Station1\CPU 416-3\S7-program(1)"

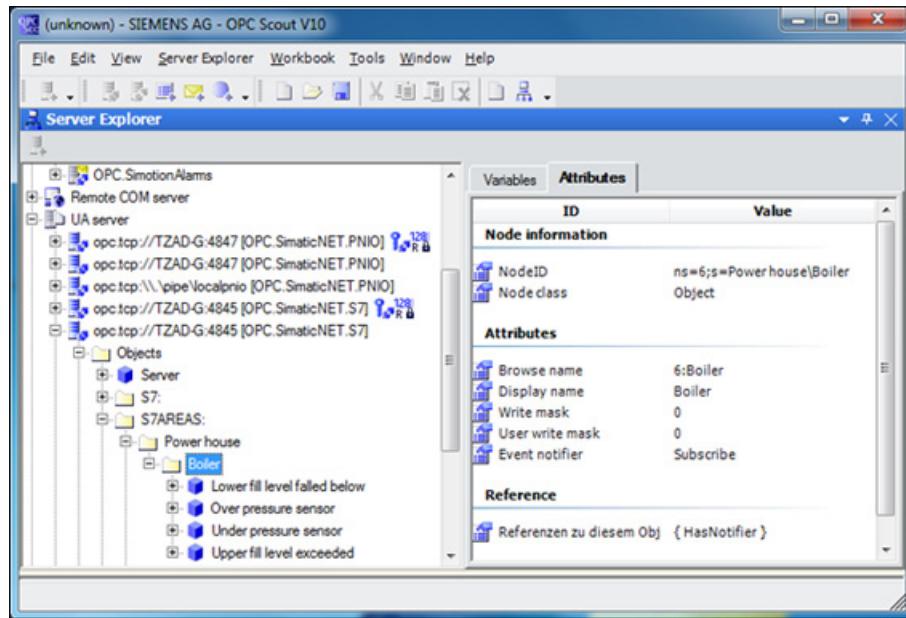
and the SourceNode

- ns="S7SOURCES:", "Station1\CPU 416-3\S7-program(1)\DB404"

Diagnostics alarms have no "Additional text 1" and "Additional text 2". Analogous to the alarms, the path in the plant tree in STEP 7 is used as the substitute for additional text 2 and the path in the plant tree + alarm identifier in STEP 7 is used as the substitute for additional text 1. For example the diagnostics alarm with alarm identifier in "S7:program(1)" shown above generates not only the existing UA folders but also the SourceNode

- ns="S7SOURCES:", "Station1\CPU 416-3\S7-program(1)\WR_USMSG (1)"

The area tree and source space can be browsed with the OPC Scout V10.



The area node with the display name "con13 Scan Notify" (Nodeld ns="S7AREAS:", s="con13-Scan-Notify") is selected in the left window. The right-hand window displays its attributes; from these, it can be seen that the area node has the reference "HasNotifier".

2.7.16 Receiving events

EventItems and EventNotifier

The only possibility of receiving current events from a server is to generate one or more monitored event items in a subscription. With an event, the EventNotifier attribute (12) is monitored. EventNotifiers only reference objects that have the "HasNotifier" reference. In S7 UA Alarming with OPC server 8.0, these are the server node ns="http://opcfoundation.org/UA/", i=2253, all nodes of the S7 connections, for example, ns="S7:", s="S7_connection_1" and all area nodes in ns="S7AREAS:". The EventNotifiers of the server node and the nodes of the S7 connections are the only ones that report events for unconfigured S7 alarms.

When a monitored event item is generated, the properties to be returned must be specified. If no properties are specified, it is known that an event has occurred, but not which. To include a property in the list of properties to be returned, the following needs to be specified for each property:

- the Nodeld of the defined event type (Typeld),
- the browse name and possibly browse path of the property and
- the Attributeld

You will find the NodId, browse name and, if applicable, the browse path and the AttributId of the defined event type in the section "Event type hierarchy of S7 UA Alarming servers (Page 197)".

Note

Not all status changes of an alarm are forwarded by an event on the OPC UA interface. If there is a fast change in the alarm status, it is possible that some status changes are not reported by an event and only the last and now current status is sent. This depends on OPC parameters such as "PublishingInterval" and "SamplingInterval" as well as the configuration and the computer performance.

Selection of the properties

An event does not need to have all properties to be returned. "Zero" is returned for properties that an event does not have. S7 OPC UA Alarming cannot distinguish between properties that the event does not have or such that do not have a valid property due to bad specification, for example, of the browse name.

Using filter conditions

If an EventNotifier of an area node is monitored, all events are reported that originate from this area. Depending on the particular system (the set of potentially available EventSources can be taken from the source space combined with the area space), the number of reported events can be extremely high. This makes filtering of the events a necessity.

2.7.17 Methods of UA alarms**Enable()/Disable()**

No events are generated on clients for an alarm that is disabled. Regardless of this, the alarms to the S7 controller are always monitored because alarms for the controller cannot be individually enabled or disabled.

AddComment()

A comment can be stored for each alarm instance.

ConditionRefresh()

All active or unacknowledged alarms are reported.

Acknowledge()

The acknowledgment of an alarm is transported to the S7 controller. Regardless, the change in the acknowledgment status is returned by the S7 controller to the OPC server explicitly, only then does the alarm change to the acknowledged status at the OPC end and triggers appropriate events.

2.8 Optimized S7 communication with OPC UA

2.8.1 Properties of optimized S7 communication with OPC UA

The SIMATIC NET OPC server allows the use of S7 communication via OPC UA and Industrial Ethernet with the new S7-1200 and S7-1500 stations.

The S7OPT OPC UA server from SIMATIC NET has the following characteristics:

- Variable services
Access and monitoring of S7 variables using standard access and optimized access. The optimized access is achieved using symbolic addressing.
- S7 CPU protection level concept
Setting a password for protected connection establishment and access to the S7-1200 and S7-1500 stations
- OPC UA events, conditions and alarms
Processing PLC alarms

2.8.2 SIMATIC NET OPC UA server for the optimized S7 protocol

Introduction

The following section describes the configuration for the optimized S7 protocol via Industrial Ethernet and OPC UA.

Configuration

The S7OPT OPC UA server is activated by selecting "OPC UA" (in the "S7 optimized" row) in the "Communication Settings" configuration program in the "OPC protocol selection" catalog. For "S7 optimized", no check mark is shown since the optimized S7 protocol is only available for OPC UA. For this reason, enabling the S7OPT UA server does not influence the inproc/outproc configuration of the other S7 COM OPC Data Access servers.

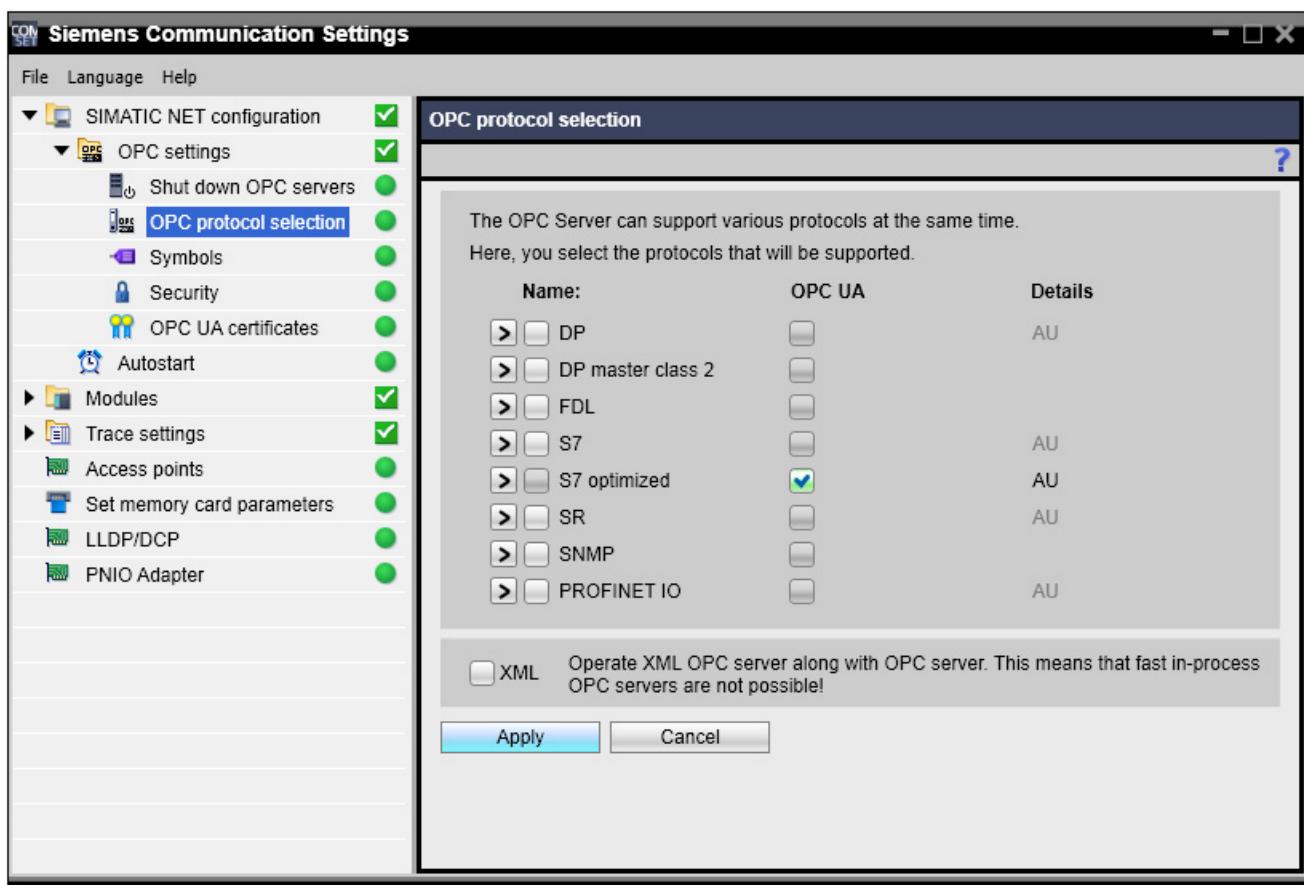


Figure 2-40 Window in the "Communication Settings" configuration program for selecting OPC UA for the optimized S7 protocol

Note

Optimized access to S7 variables of the S7-1200 and S7-1500 stations is possible only with symbolic addressing.

Note

When creating symbols with STEP 7 Professional (TIA Portal) or the symbol editor, use of the following characters is permitted: A-Z, a-z, 0-9, _, -, ^, !, #, \$, %, &, ', /, (,), <, >, =, ?, ~, +, *, ',', :, |, @, [,], { , }, ". When creating symbols with STEP 7, you should also remember that problems can arise when resolving the array if your symbol file includes symbols in the form <symbolname> and <symbolname>[<index>] at the same time.

Advantages / disadvantages

When using the S7OPT OPC UA server, only the outproc mode of "S7 optimized" is possible. The S7OPT OPC UA server process must be running to maintain readiness to receive. Even after all OPC UA clients have logged off, the S7OPT OPC UA server is not and must not be exited.

This is offset by the following advantages:

- COM/DCOM configuration is no longer necessary.
- High-speed, secure communication
- Only OPC one server is required for PLC alarms and data access.

Note

If the S7OPT OPC UA server is active for the S7 protocol, S7 connections configured as permanent are established as soon as the first OPC UA client logs on to the S7OPT OPC UA server. If no OPC UA client has connected, for example after starting up the computer, no permanent S7 connections are established yet.

Note

S7 connections for the "S7 optimized" protocol can be configured with the setting "Establish connection on demand" in SIMATIC STEP 7 Professional (TIA Portal). The first time a variable is accessed that requires this connection, the connection is first established and only then does the actual access take place. This means that the first access to the variable may take longer and various errors may be signaled during connection establishment.

2.8.3 How is the S7OPT OPC UA server addressed?

Server URL

For the native binary TCP protocol, there are two ways for the OPC client to address the server:

- Direct addressing:
 - opc.tcp://<hostname>:55105
 - or
 - opc.tcp://<IP-Adresse>:55105
 - or
 - opc.tcp://localhost:55105

The S7OPT OPC UA server has port 55105.

- The URL of the S7OPT OPC UA server can also be found using the OPC UA Discovery service.

The entry for locating the Discovery server is as follows:

– opc.tcp://<hostname>:4840

or

– opc.tcp://<IPaddress>:4840

The Discovery server uses port 4840 (for TCP connections).

IPv6 address

An IPv6 address can also be used as the IP address. The address must be in parentheses, for example [fe80:e499:b710:5975:73d8:14]

Endpoints and security modes

The SIMATIC NET S7OPT OPC UA server supports endpoints with the native binary TCP protocol and allows authentication using certificates and encrypted transfer.

The Discovery service on the addressed host signals the endpoints of the servers, in other words, their security requirements and protocol support.

The server URL "opc.tcp://<hostname>:55105" of the S7OPT OPC UA server provides the following endpoints:

- Endpoint in Security mode "SignAndEncrypt":
A signature and encryption are required to communicate with the server. Communication is protected by the exchange of certificates and entry of a password.
In addition to the security mode, the security policy Basic128Rsa15 is also shown.
- Endpoint in Security mode "none"
In this mode, no security functions are required by the server (security policy "None").

For more detailed information on the security functions, refer to the section "Programming the OPC UA interface (Page 496)".

The security policies "Basic128Rsa15" and "None" are in the UA specification of the OPC Foundation at the following Internet address:

<http://opcfoundation.org/UA> (<http://opcfoundation.org/UA>) > "Specifications" > "Part 7"

You will find more detailed information on the following Internet page:

OPC Foundation (<http://www.opcfoundation.org/profilereporting/index.htm>)
> "Security Category" > "Facets" > "Security Policy"

The OPC UA Discovery of the OPC Scout V10

The OPC Scout V10 allows you to open the OPC UA Discovery dialog to enter OPC UA endpoints in the navigation area of the OPC Scout V10.

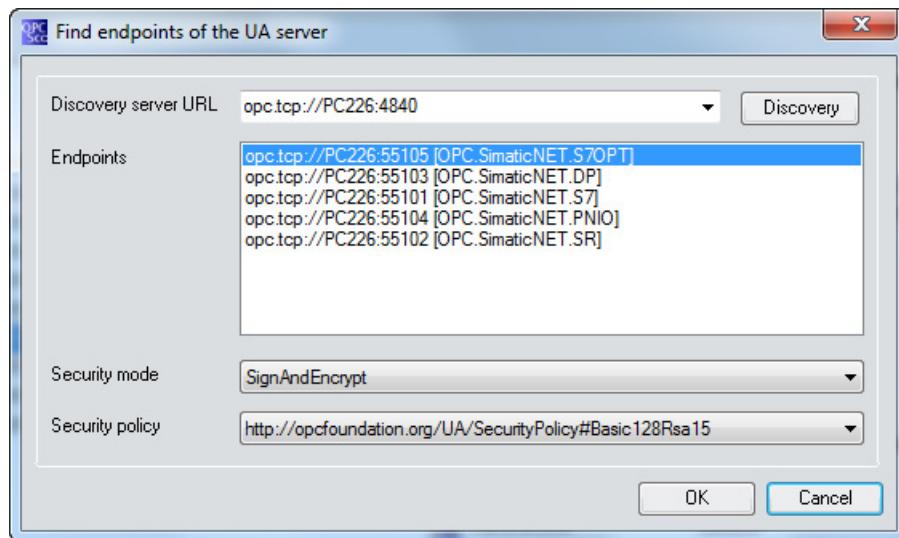


Figure 2-41 The "Find endpoints of the UA server" dialog of OPC Scout V10

The S7OPT OPC UA server can be found using the OPC UA Discovery service. For the entry, refer to "Server URL" above.

OPC Scout V10 contains a list of the OPC UA endpoints. The Discovery service on the addressed host then signals the registered OPC UA servers and their ports and security modes.

For more detailed information, refer to the online help of OPC Scout V10.

2.8.4 Which namespaces does the S7OPT OPC UA server provide?

The S7OPT OPC UA server provides the following namespaces:

Namespace index	"Identifier" (namespace URI) / comment
0	" http://opcfoundation.org/UA/ " specified by the OPC Foundation
1	"urn:Siemens.Automation.SimaticNET.S7OPT:(GUID)" Unique identifier of the local high-speed S7OPT OPC UA server.
2	"S7OPTTYPES:" Definitions for S7OPT-specific object types.
3	"S7OPT:" Name of the local high-speed S7OPT OPC UA server.
4	"S7OPTSOURCES:" Identifier of the sources of alarm events.

Namespace index	"Identifier" (namespace URI) / comment
5	"S7OPTAREAS:" Identifier for areas of an alarm hierarchy.
6	"SYM:" Optional server with ATI-S7OPT symbols; dependent on the project engineering and the configuration of the PC station (browsable and can be used with OPC UA). As an alternative, a prefix can be used here that is specified in the parameter assignment for symbols ("Communication Settings").

The namespace indexes 0 and 1 are reserved and their significance is specified by the OPC Foundation.

The assignment of the remaining namespace indexes to the identifiers (namespace URI) must be obtained via the data variable "NamespaceArray" at the beginning of an OPC UA session by the client specifying the identifier. The identifiers "S7OPTTYPES:", "S7OPT:", "S7OPTAREAS:" and "S7OPTSOURCES:" always exist with the S7OPT OPC UA server.

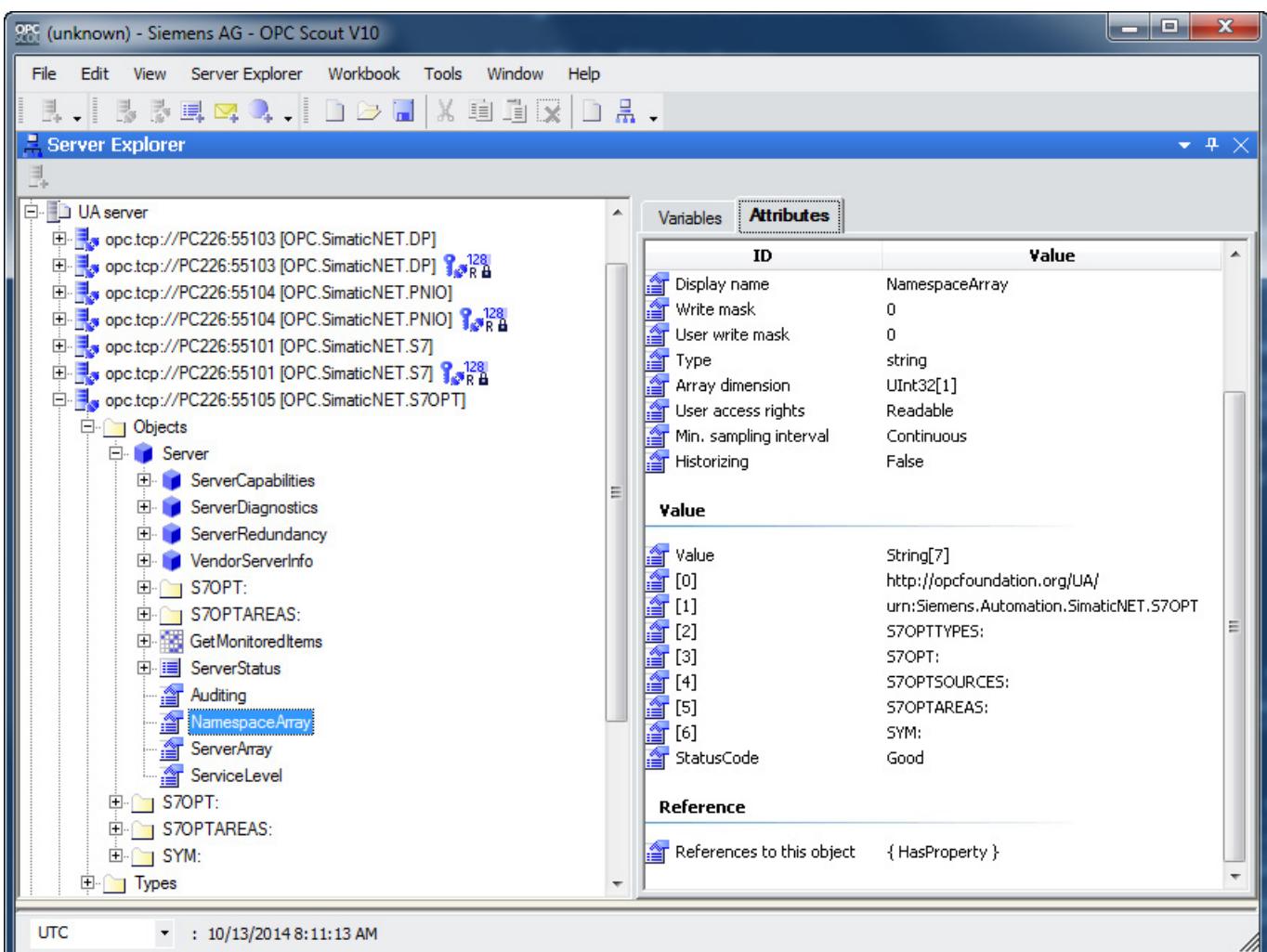


Figure 2-42 Display of the S7OPT OPC UA namespaces using the browse function of the OPC Scout V10

2.8.5 Die Nodeld

Identification of an S7OPT process variable

The Nodeld identifies an S7OPT process variable with the aid of the following tuple:

- Namespace index
- Identifier (character string)

Examples

- Nodeld:
 - Namespace URI:
S7OPT:
(= namespace index 3)
 - Identifier:
S7_connection_1.m.10,b

How does the OPC UA namespace behave?

The OPC UA view relates to automation objects also to various properties of the objects. OPC UA accesses objects and their subobjects.

- Data variables, methods and, to some extent events, are, for example, subobjects of an "S7 optimized" connection object. Attributes and properties define the objects in greater detail.

The qualified identifiers of the Nodelds are very important in OPC UA. Each individual access to an object, subobject, property and attribute uses its Nodeld.

OPC UA provides the display name among other things to support local languages. This means that the same objects, for example in different language environments specified by the OPC UA client, can be browsed differently although the same Nodeld is presented every time. Selection of the display name is analogous to the relevant Nodeld. The texts of the entire namespace are in English.

Syntax of the S7OPT process variables

An optimized syntax is introduced with OPC UA. The Nodelds of all OPC UA objects have the following structure:

<connectionobject>"."<subobject>"."<property>

A subobject can contain further subobjects.

A Nodeld that cannot be interpreted is rejected with an error. The letters "A-Z" are not case-sensitive for any Nodelds.

Symbolic object representation

The OPC UA specification recommends a uniform symbolic representation for the hierarchical description of the address space. The following symbols are used in this document:

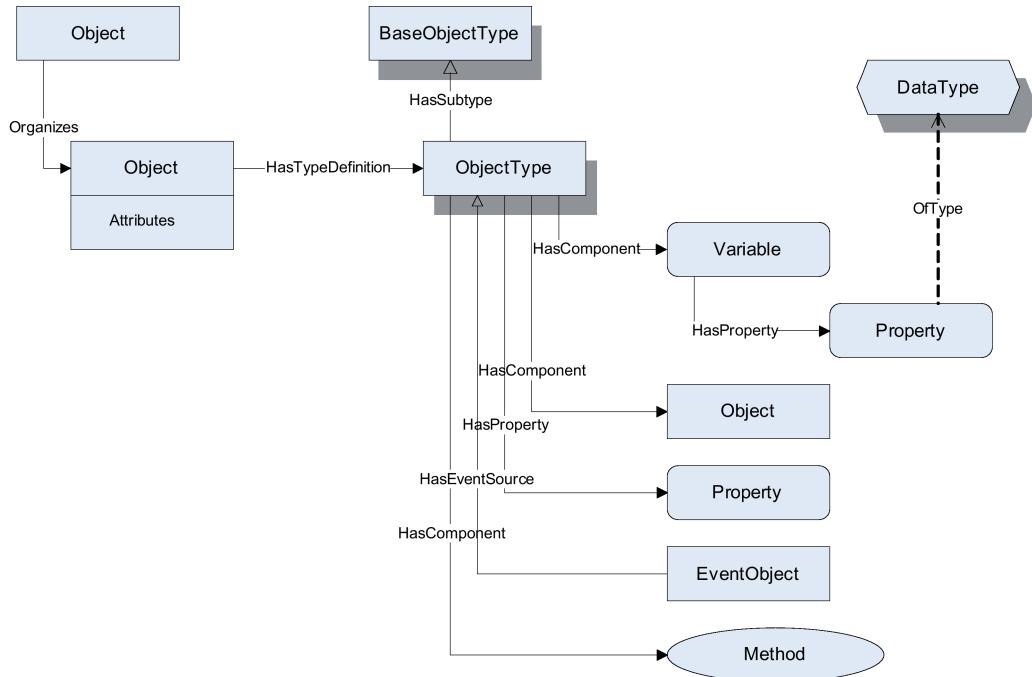


Figure 2-43 Symbols of the OPC UA address space

2.8.6 Connection objects

S7OPT connection objects

The following S7OPT connection objects exist:

- Productive S7 connections
S7 connections are used for the optimized S7 communication to exchange data between programmable controllers and are configured in STEP 7 Professional (TIA Portal).
- The CpuSubscription object
This is used to subscribe to cyclic monitoring of data changes directly on the CPU without polling being necessary. The S7-1200 / S7-1500 stations support a limited number of CpuSubscriptions.

Note

You will find information about the configuration limits of the CPUs in the relevant device manuals in the section "Technical specifications".

CpuSubscription object

```
<Connection object>= "cpusubscription"
```

Note

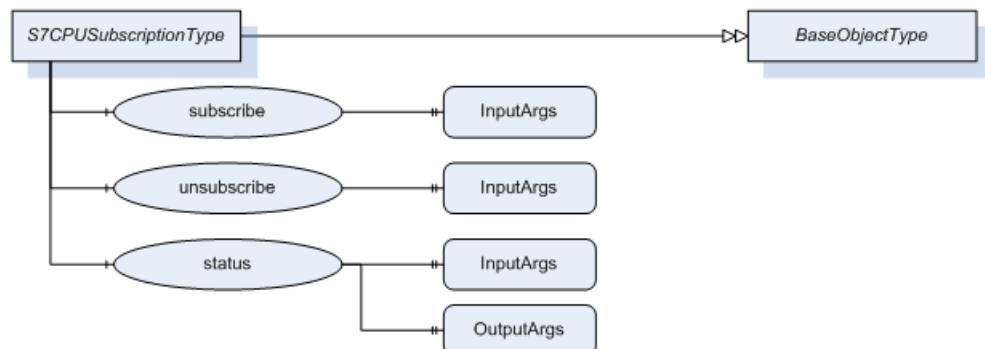
The connection name "CpuSubscription" is reserved and must never be used as the name of an S7 connection. STEP 7 Professional (TIA Portal) prevents this during creation of the connection configuration.

For in the S7-1200 and S7-1500 station, the S7OPT OPC UA server allows monitoring of data changes directly on the CPU without requiring cyclic read jobs on the S7 connection. By using these CpuSubscriptions you can considerably reduce the communications load of your CPU.

The number of CpuSubscriptions is strictly limited and depends on the type of S7 station being used. This is in the range of a several hundred objects with a few thousand bytes and cannot be queried while the S7OPT OPC UA server is running. This is also not exclusively available to the S7OPT OPC UA server but can also be used by other applications.

For this reason, the S7OPT OPC UA server provides its UA clients with the option of using OPC UA methods to query its preferred data variables for CpuSubscriptions and to obtain the current status of these CpuSubscriptions.

These methods are not restricted to a single S7 connection. They assume OPCUA_NodeIds as the parameter for the data variables based on which the connection path to the CPU can be identified. Data variables of the symbols whose OPCUA_NodeIds do not normally contain a connection name can also be included in the CpuSubscriptions.



Syntax of CpuSubscription methods

Namespace URI: S7OPT: (Namespace index: 3)

```
cpusubscription.subscribe(OpcUa_UInt32 cycletime, OpcUa_NodeId[] nodeids)  
cpusubscription.unsubscribe(OpcUa_NodeId[] nodeids)  
cpusubscription.status(OpcUa_NodeId[] nodeids, OpcUa_Status[] status)
```

Examples:

```

cpusubscription.subscribe(100, {S7OPT_1.db10.1,b|PCStation1.PLC1500.optDB.Byte})

cpusubscription.unsubscribe({S7OPT_1.db10.1,b|PCStation1.PLC1500.optDB.Byte})

cpusubscription.status({S7OPT_1.db10.1,b|PCStation1.PLC1500.optDB.Byte}, Status)

```

Explanation of the OPC UA methods**• subscribe()**

Creates the CpuSubscription for multiple data variables. (If errors occurred, corresponding OPC UA error codes are available, see also section "Messages of the OPC UA server (Page 513)")

- InputArgument1: "cycletime"; interval that signals the data changes of the PLC to the S7OPT OPC UA server.
- Data variable of the OPC UA data type "UInt32", specified in ms, at least 100 ms, only whole multiples of 100 ms
- InputArgument2: "nodeids"; specifies one or more Nodelds to be created using the CpuSubscription access option.
- Data variable of the OPC UA data type "OpcUa_Nodeld[]"

Note

If a CpuSubscription has already been created for a Nodeld, this is modified with the changed cycle time.

• unsubscribe()

Removes the CpuSubscription. (If errors occurred, corresponding OPC UA error codes are available, see also section "Messages of the OPC UA server (Page 513)")

- InputArgument1: "nodeids"; list of the Nodelds can be displayed
- Data variable of the OPC UA data type "OpcUa_Nodeld[]"

• status()

Returns the CpuSubscription status. (If errors occurred, corresponding OPC UA error codes are available, see also section "Messages of the OPC UA server (Page 513)")

- InputArgument1: "nodeids"; list of the Nodelds can be displayed
- Data variable of the OPC UA data type "OpcUa_Nodeld[]"
- OutputArgument1: "status"
- Data variable of the OPC UA data type "OpcUa_Status[]"

The CpuSubscription method call adds or removes the transferred Nodelds of the internal CpuSubscription management of the S7OPT OPC UA server, regardless of the connection status of the connection partner. The (successful) completion of a method call therefore does not indicate that the CpuSubscription could actually be generated on the CPU. If incorrect Nodelds have been used, the functions return a parameter error and none of the transferred Nodelds has been transferred to the CpuSubscription management.

Note

If the number of available CpuSubscriptions or data variables has been exceeded, the evaluation of the method return parameter cannot provide any information about the successful creation of the CpuSubscription on the CPU. For this reason, check the successful completion of a Subscribe or Unsubscribe method by calling the Status method and if there has been an error, react by adapting or reducing the number of data variables/CpuSubscriptions.

After successful connection establishment or after adding or removing a CpuSubscription, the S7OPT OPC UA server optimizes all the CpuSubscriptions registered up to that time on the relevant connection and registers or deregisters them again with the connection partner.

If the registration with the connection partner was successful, this automatically returns the changed data following a data change. Monitored Items with the same or a higher update rate set up on the S7OPT OPC UA server that can be mapped to the use CpuSubscriptions are now no longer polled.

If the registration with the connection partner was unsuccessful, the MonitoredItems mapped to this CpuSubscription will be polled in keeping with the standard procedure without the use of CpuSubscriptions. The status of this CpuSubscription is set to "Bad" and this can be read out with the Status method, see also section "Messages of the OPC UA server (Page 513)".

The CpuSubscriptions are effective for all UA clients, this means that a UA client can remove a CpuSubscription set up by another client. Once the last UA client has logged off, all the CpuSubscriptions still existing at this point in time are removed.

For optimum and full use of the CpuSubscription services, you should remember the following when setting up and using the CpuSubscriptions:

- Register the CpuSubscriptions only for the most important data variables
- Register larger, contiguous areas if you want to monitor a lot of small data variables in this data area.
- Where possible, register data variables that change seldom, but whose changes require a fast response
- Match the cycle time of the CpuSubscription to the update time of your MonitoredItems.

For the OPC client, the use of CpuSubscriptions is transparent. Whatever the case (without CpuSubscription, successfully registered CpuSubscription, unsuccessfully registered CpuSubscription) data variables registered as MonitoredItems will be monitored for data changes either by polling or by the CpuSubscriptions.

The S7OPT OPC UA server has no influence on the order in which the acknowledgements are processed or the signaling of data changes by the connection partner. Both acknowledgements of read jobs as well as data change messages can be on their way at the same time. The S7OPT OPC UA server therefore always updates its memory with the last value to arrive.

2.8.7 Connection names

The connection name of an S7 connection

The connection name is the name configured in STEP 7 Professional (TIA Portal) to identify the connection. In STEP 7 Professional (TIA Portal), this name is known as the "local ID". The local ID is unique within the OPC server.

Connection types

The OPC server supports the S7 connection as a connection type.

What characters are permitted for S7 connection names?

For the <connectionname>, you can use numbers "0-9", upper and lower case alphabetic characters "A-z" and special characters. Spaces at the start or end of the connection name and the special characters ".", "|", "\", "/", "[", "]" are not permitted. The connection name can be a maximum of 24 characters long. The names are not case-sensitive.

Other printable characters with ASCII code > 126 and ASCII code < 30 are not permitted.

The connection names "SYSTEM" and "cpusubscription" are reserved and must not be used.

Examples of S7 connection names:

Typical examples are:

- S7_connection_1
- S7OPT OPC connection

2.8.8 Structure and functions of the productive S7OPT connection object

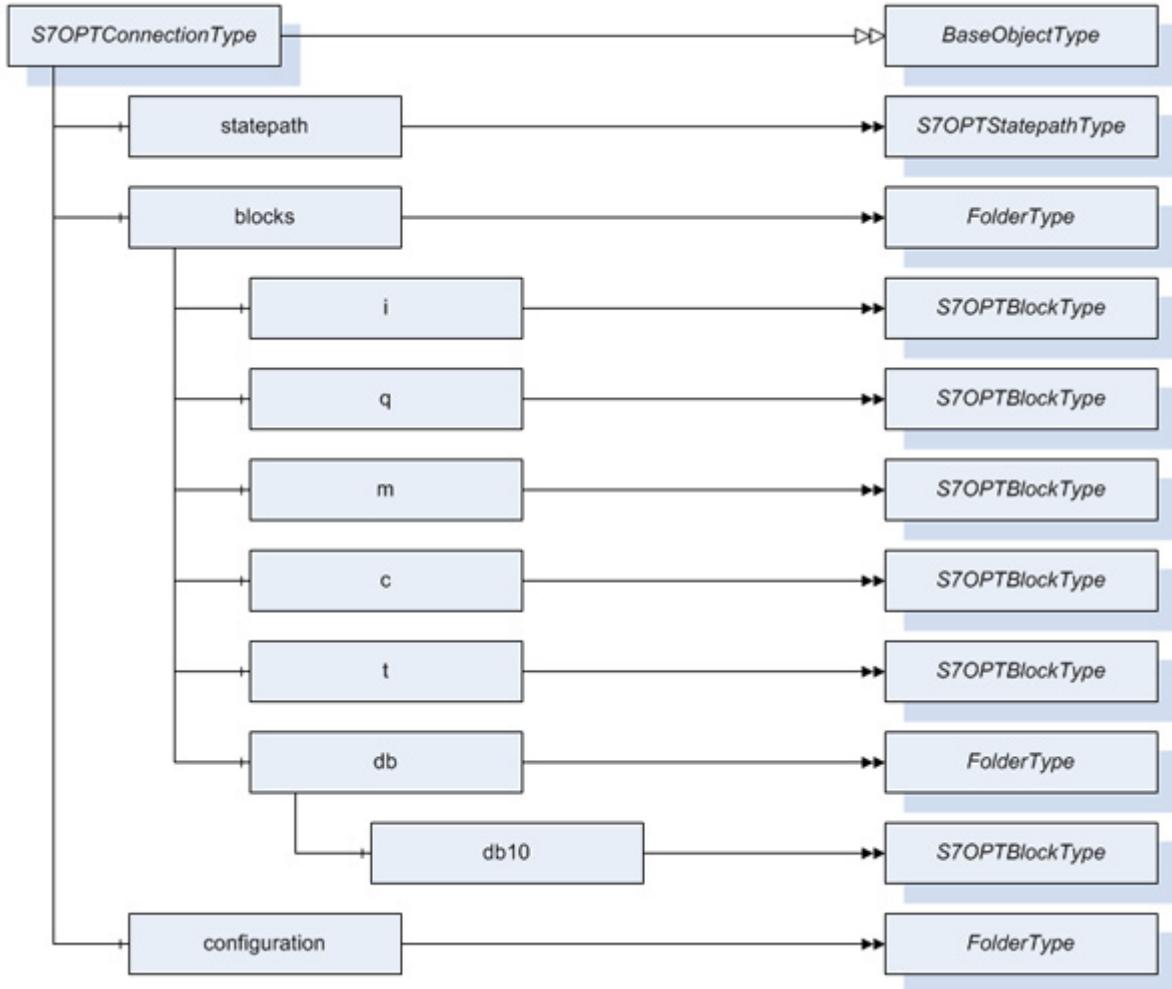
What are S7OPT connection objects?

All productive protocol-specific objects are always assigned to a connection. With "S7 optimized", these are connections to communications partners (S7 connection). The CpuSubscription is an exception to this.

2.8.9 Type definition of the S7OPT connection object

Type definition of the S7OPT connection object

For the objects and functionalities that can be used via a productive S7 connection, a specific OPC UA object type is defined:



Instances of this type are displayed in the OPC UA namespace for objects. The type itself can be read out structured under "Types".

2.8.10 S7OPT connection information objects

S7OPT-specific data variables for information

There are S7OPT-specific data variables with which you can obtain information about the optimized S7 communication and the established connections.

The following information can be obtained:

- Status of an S7 connection with optimized access

Syntax of the S7OPT-specific information variables

Nodeld:

Namespace index: 3

<connectionobject>. <informationparameter>

Explanations

<informationparameter>:= "statepath"

alarm	Instance of the statepath alarm belonging to the statepath object.		
statepath	Status of a communications connection to the partner device The value of this variable is read out as a number and can be assigned to a text by additionally reading the associated Enumstring {UNKNOWN, DOWN, UP, RECOVERY, ESTABLISH}. Variable of the UA type "MultistateDiscreteType", read-only		
0	UNKNOWN	Reserved for future expansions	
1	DOWN	Connection is not established	
2	UP	Connection is established	
3	RECOVERY	Connection is not established. An attempt is being made to establish the connection.	
4	ESTABLISH	Reserved for future expansions	

2.8.11 Examples of S7OPT-specific information variables and return values

Here, you will find examples illustrating the syntax of the names of S7OPT-specific information variables.

Priority level of a statepath alarm

- Nodeld:
 - Namespace URI:
S7OPT: (Namespace index: 3)
 - Identifier:
S7_connection_1.statepath.alarm.Severity

Possible return values: 500

Possible status texts of a communications connection

- Nodeld:
 - Namespace URI:
S7OPT: (Namespace index: 3)
 - Identifier:
S7_connection_1.statepath.statepath.EnumStrings

Possible return values: { UNKNOWN | DOWN | UP | RECOVERY | ESTABLISH }

Status of a communications connection as a value

- Nodeld:
 - Namespace URI:
S7OPT: (Namespace index: 3)
 - Identifier:
S7_connection_1.statepath.statepath

Possible return values: 3 (RECOVERY) -The connection is currently being established.

2.8.12 Variable services

2.8.12.1 Standard access

What do variable services do that are called using standard access?

Variable services called using standard access allow access to and monitoring of S7 variables on the programmable controller. The addressing of the S7 variables takes the form of a short name of the address objects. The type of access is oriented on the notation of the S7 tools.

Objects in the programmable controller

The S7OPT OPC UA server supports the following objects:

- Data blocks (standard access)
- Instance data blocks and multi-instance data blocks (standard access)
- Inputs
- Outputs
- Bit memory
- Timers (S7-1500 only)
- Counters (S7-1500 only)
- UDTs

Not every S7 programmable controller supports every object type.

Note

The standard access to variable services can also be achieved using symbols. You will find all the necessary information on this in the "Commissioning PC Stations" manual

Syntax of the variable services

Syntax of the process variables

Simplified syntax of the process variables of the S7OPT OPC UA NodeId:

Namespace URI: S7OPT: (Namespace index: 3)

Classic syntax

You have three options:

```
<connectionname>.<S7OPTobject>.<address>{,<S7OPTtype>{,<quantity>} }  
<connectionname>.<S7OPTtimerobject>.<address>  
{,<S7OPTtimertype>{,<quantity>} }  
<connectionname>.<S7OPTcounterobject>.<address>  
{,<S7OPTcountertype>{,<quantity>} }
```

Explanations

Namespace URI: S7OPT: (Namespace index: 3)

<connectionname>

Protocol-specific connection name. The connection name is specified during configuration. The following separator is the period (".").

<S7OPTobject>

Specifies the area type on the S7. Possible values are as follows:

Parameter	Meaning
db<no>	Data block or instance data block. Identifier for an S7OPT variable from a data block. In S7 communication, no distinction is made between instance data blocks and normal data blocks. As a result, it is not necessary to assign an additional identifier to simplify matters. <no> Number of the data block or instance data block without leading zeros. Only the non-optimized data blocks on the S7-1200 /S7-1500 can be addressed using these data variables. For data variables in optimized data blocks, no address information can be created during runtime. These optimized data variables are accessed using symbolic access.
m	Bit memory
i	Input Readable and writable To simplify things, there is only this English identifier.
q	Output Readable and writable To simplify things, there is only this English identifier.

The following separator is the period (*. *).

<S7OPTtimerobject>

Type identifier	S7 data type	OPC UA data type	Description
t	TIMER	UInt16	The address information that follows is a timer number.

<S7OPTcounterobject>

Type identifier	S7 data type	OPC UA data type	Description
c	COUNTER	UInt16	The address information that follows is a counter number.

<address>

Byte address of the first variable to be addressed. No leading zeros are supported (for example 001).

The value range for the byte address is 0 to 65534. The actual usable value of the address may be lower depending on the device and type.

<S7OPTtype>

An S7 data type is converted to the corresponding OPC UA data type on the OPC UA

server. The following table lists the type identifier and the corresponding OPC UA data type in which the variable value can be represented.

Type identifier	S7 data type	OPC UA data type	Description
b	BYTE	Byte	8 bits (bit string)
	USINT	Byte	8 bits (unsigned)
c	CHAR	SByte	8 bits (character)
	SINT	SByte	8 bits (signed)
w	WORD	UInt16	16 bits (bit string)
	UINT	UInt16	16 bits (unsigned)
dw	DWORD	UInt32	32 bits (bit string)
	UDINT	UInt32	32 bits (unsigned)
lw	LWORD	UInt64	64 bits (bit string); available only for S7-1500
	ULINT	UInt64	64 bits (unsigned); available only for S7-1500
i	INT	Int16	16 bits (signed)
di	DINT	Int32	32 bits (signed)
li	LINT	Int64	64 bits (signed); available only for S7-1500
r	REAL	Float	Floating point (4 bytes) IEEE 754
lr	LREAL	Double	Floating point (8 bytes) IEEE 754
dt	DATE_TIME	UtcTime	Date and time of day, Range of values, starting at 01.01.1990; available only for S7-1500
ldt	LDT	UtcTime	Date and time of day, accurate to the nano-second Range of values, starting at 01.01.1990; available only for S7-1500
dtl	DTL	UtcTime	Date and time of day, accurate to the nano-second Range of values 01.01.1970 – 31.12.2553; available only for S7-1500
date	DATE	UtcTime	Date and time (8 bytes), where the time is always 00:00:00, range of values starting from 01.01.1990. Mapping of the CPU data type "DATE" (unsigned, 16 bits).
t	TIME	Int32	Signed time value in milliseconds
lt	LTIME	Int64	Signed time value in nanoseconds; available only for S7-1500
tod	TOD	UInt32	Time of day, 0 to 86399999 ms starting at midnight
ltod	LTOD	UInt64	Time of day, 0 to 863999999999 ns starting at midnight; available only for S7-1500

Type identifier	S7 data type	OPC UA data type	Description
s5bcd	S5TIME	UInt16	Mapping of the CPU data type "S5TIME" to UInt 16 (unsigned, 16 bits) with a restricted range of values, 0 to 9990000 ms.*); available only for S7-1500
x<bitaddress>	BOOL	Boolean	Bit (bool) In addition to the byte offset in the area, the <bitaddress> must also be specified in the byte. Range of values 0 to 7
s<stringlength>	STRING	String	The <stringlength> reserved for the string must also be specified. Range of values 1 to 254 When writing, it is also possible to write shorter strings, whereby the transferred data length is always the reserved string length in bytes plus 2 bytes. The unnecessary bytes are filled with the value 0. Reading and writing strings and string arrays is mapped internally to the reading and writing of byte arrays. On the S7, the string must be initialized with valid values.

*) See table below "Timebase and range of values for the S7 data type "s5bcd""

Note

The type identifier "" is used by the S7OPT OPC UA server as the default value if no "<S7OPTTyp>" is specified.

Note

The "DTL" S7 data type can only be read by the SIMATIC NET OPC UA server.

Note

The S7 data type "Byte[]" is mapped to a byte string by the SIMATIC NET OPC UA server.

Timebase and range of values for the S7 data type "s5btcd"

The range of values of the time variable of the data type "s5btcd" is BCD-coded. The range of values is according to the following table:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Meaning (symbol)	0	0	x	x	t	t	t	t	t	t	t	t	t	t	t	t
Explanation:																
Meaning "0"	not relevant															
Meaning "x"	Specifies the timebase															
	Bits 13 and 12										Timebase in milliseconds					
	00										10					
	01										100					
	10										1000					
	11										10000					
Meaning "t"	BCD-coded time value (0 to 999)															

Example:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	0	1	0	0	0	0	0	1	1	1	0	1	0	1

Bits 0-11 specify the number 075. Bits 12 and 13 specify the timebase 100 ms.

$75 * 100 \text{ ms} = 7500 \text{ ms}$. This clocks down 75 times in steps of 100 ms.

The OPC data type of the time variable of data type "s5btcd" is a word (UInt16). When writing, the range of values is limited accordingly.

<S7OPTTimertype>

An S7 timer data type is converted to the corresponding OPC UA data type on the OPC UA server. The following table lists the type identifier and the corresponding OPC UA data type in which the variable value can be represented.

Type identifier	S7 data type	OPC UA data type	Description
tbcd	TIMER	UInt16	BCD coded Used as the default if no <S7timertype> is specified.
tda	TIMER	UInt16[2]	Decimal timebase and time value

Timebase and range of values of S7OPTtimerobject "t" and S7OPTtimertype " tbcd":

The range of values of S7 timer variables of the type "tbcd" is BCD coded. The timebase (for writing) is obtained from the value range as shown in the table below:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Meaning (symbol)	0	0	x	x	t	t	t	t	t	t	t	t	t	t	t	t
Explanation:																
Meaning "0"	not relevant															

Meaning "x"	Specifies the timebase	
	Bits 13 and 12	Timebase in milliseconds
	00	10
	01	100
	10	1000
	11	10000
Meaning "t"	BCD-coded time value (0 to 999)	

Example:

Bit no.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	0	1	0	0	0	0	0	1	1	1	0	1	0	1

Bits 0-11 specify the number 075. Bits 12 and 13 specify the timebase 100.
 $75 * 100 = 7500$ ms. This clocks down 75 times in steps of 100 ms.

Timebase and range of values for the S7 timer data type "tda":

Data type: Field of two words {timebase in milliseconds as UInt16 | time value UInt16}

Timebase [ms]	10, 100, 1 000, 10 000
Time value	0 ... 999
Timebases [ms]	10 ms: 0 ... 9 990 100 ms: 0 ... 99 900 ms 1 000 ms: 0 ... 99 9000 10,000 ms: 0 ... 9 990 000

It is not possible to enter <quantity> for the "TDA" object.

Example:

Writing the value {100|50} to timer "t.3,tda", "tda" initializes timer 3 with the value
 $50 * 100$ ms = 5000 ms and this is clocked down 50 times in 100 ms steps.

<S7OPTcountertype>

Type identifier	S7 data type	OPC UA data type	Description
c	COUNTER	UInt16	Range of values for S7: 0 to 999, decimal-coded. Used as the default if no <S7OPTcountertype> is specified.

<quantity>

Number of variables of a type to be addressed starting at the offset specified in the "address" parameter (range of values 1 to 65 535).

Specifying a number of array elements causes an array of the corresponding type to be formed even when only a single array element is addressed.

The separator is a comma (",").

For data type "x", the quantity for write access can only be entered in multiples of 8. The bit address must then be zero.

For data type "x", the entry of a quantity for read access is unrestricted. The value range of the bit address then allows 0 to 7.

Examples:

Namespace URI: S7OPT: (Namespace index: 3)

S7-OPC-1.db1.10,x0,64, access rights RW

S7-OPC-1.db1.10,x3,17, access rights R

Examples of process variables for non-optimized S7OPT OPC UA variable services

Here you will find examples illustrating the syntax of variable names for variable services.

Non-optimized data block DB single byte

Namespace URI: S7OPT: (Namespace index: 3)

S7_connection-1.db5.12,b

Indicates data byte 12 in data block 5 via S7_connection-1.

Non-optimized data block DB, array of words

Namespace URI: S7OPT: (Namespace index: 3)

S7_connection-1.db5.10,w,9

Indicates 9 data words as of byte address 10 in data block 5 via S7_connection-1.

Non-optimized data block DB, array of strings

Namespace URI: S7OPT: (Namespace index: 3)

S7_connection-1.db100.50,s32,3

Indicates 3 strings with a length of 32 as of byte address 50 in data block 100 via S7_connection-1.

Input 0

Namespace URI: S7OPT: (Namespace index: 3)

S7_connection-1.i.0

Indicates input byte 0 via S7_connection-1.

Output 0 bit 0

Namespace URI: S7OPT: (Namespace index: 3)

S7_connection-1.q.0,x0

Indicates output address 0, bit 0 via S7_connection-1.

Array of readable memory bits

Namespace URI: S7OPT: (Namespace index: 3)

S7_connection-1.m.3,x4,12 //access rights R

Indicates 12 bits starting at bit memory address 3 and there starting at bit address 4 via S7_connection-1. Read only.

Timer 22 BCD-coded

Namespace URI: S7OPT: (Namespace index: 3)

S7_connection-1.t.22

Indicates timer 22, TBCD default via S7_connection-1.

2.8.12.2 Optimized access

What do variable services do that are called using optimized access?

Variable services called using optimized access allow optimized access to and monitoring of S7 variables on the programmable controller. S7 variables are addressed symbolically. The type of access is oriented on the notation of the S7 symbols.

Objects in the programmable controller

The S7OPT OPC UA server supports the following objects:

- Data blocks (optimized access)
- Instance data blocks (optimized access)

Note

The optimized S7 communication with OPC UA works with the S7-1200 (as of V4.0) and the S7-1500 stations.

Syntax of the optimized variable services

Syntax of the symbolic addressing

Simplified syntax of the process variables of the S7OPT OPC UA NodeId:

Namespace URI: SYM: (Namespace index: 4)

Symbol syntax

<stationname>.<CPUname>{.<folder>}.<symbol>

Explanations

Namespace URI: SYM: (Namespace index: 4)

<stationname>

The station name is specified during configuration in STEP 7 Professional (TIA Portal). The following separator is the period (*. *).

<CPU_name>

The name of the S7 CPU is specified in STEP 7 Professional (TIA Portal). The following separator is the period (*. *).

<folder>

Name of the data block or a data structure on the S7 CPU. The name is specified in STEP 7 Professional (TIA Portal). The following separator is the period (*. *). With data structures in data blocks, the folder structure is expanded accordingly.

<symbol>

Name of the variable on the S7 CPU. The variable name is specified in STEP 7 Professional (TIA Portal) in the data block or in the symbol table.

S7 data type	OPC UA data type	Description
BYTE	Byte	8 bits (bit string)
USINT	Byte	8 bits (unsigned)
CHAR	SByte	8 bits (character)
SINT	SByte	8 bits (signed)
WORD	UInt16	16 bits (bit string)
UINT	UInt16	16 bits (unsigned)
DWORD	UInt32	32 bits (bit string)
UDINT	UInt32	32 bits (unsigned)
LWORD	UInt64	64 bits (bit string); available only for S7-1500
ULINT	UInt64	64 bits (unsigned); available only for S7-1500
INT	Int16	16 bits (signed)
DINT	Int32	32 bits (signed)
LINT	Int64	64 bits (signed); available only for S7-1500
REAL	Float	Floating point (4 bytes) IEEE 754
LREAL	Double	Floating point (8 bytes) IEEE 754
DATE_TIME	UtcTime	Date and time of day, Range of values, starting at 01.01.1990; available only for S7-1500

S7 data type	OPC UA data type	Description
LDT	UtcTime	Date and time of day, accurate to the nanosecond Range of values, starting at 01.01.1990; available only for S7-1500
DTL	UtcTime	Date and time of day, accurate to the nanosecond Range of values 01.01.1970 – 31.12.2553; available only for S7-1500
DATE	UtcTime	Date and time (8 bytes), where the time is always 00:00:00, range of values starting from 01.01.1990. Mapping of the CPU data type "DATE" (unsigned, 16 bits).
TIME	Int32	Signed time value in milliseconds
LTIME	Int64	Signed time value in nanoseconds; available only for S7-1500
TOD	UInt32	Time of day, 0 to 86399999 ms starting at midnight
LTOD	UInt64	Time of day, 0 to 8639999999999 ns starting at midnight; available only for S7-1500
S5TIME	UInt16	Mapping of the CPU data type "S5TIME" to UInt 16 (unsigned, 16 bits) with a restricted range of values, 0 to 9990000 ms.*); available only for S7-1500
BOOL	Boolean	Bit (bool) In addition to the byte offset in the area, the <bitaddress> must also be specified in the byte. Range of values 0 to 7
STRING	String	The <stringlength> reserved for the string must also be specified. Range of values 1 to 254 When writing, it is also possible to write shorter strings, whereby the transferred data length is always the reserved string length in bytes plus 2 bytes. The unnecessary bytes are filled with the value 0. Reading and writing strings and string arrays is mapped internally to the reading and writing of byte arrays. On the S7, the string must be initialized with valid values.

Note

The "DTL" S7 data type can only be read by the SIMATIC NET OPC UA server.

Note

With user-defined data types (UDT) the SIMATIC NET OPC UA server only supports access to the subelements of the UDT. The whole UDT cannot be accessed.

Note

The S7 data type "Byte[]" is mapped to a byte string by the SIMATIC NET OPC UA server.

Example of a process variable for optimized S7OPT OPC UA variable services

Here you will find an example illustrating the syntax of variable names for variable services.

Optimized data block DB single byte

Namespace URI: SYM: (Namespace index: 4)

S7-1500-Station_1.S7-1500.datatypes_optimized.byte

Indicates an S7 data variable "byte" in the data block "datatypes_optimized" on the CPU "S7-1500" of the station "S7-1500-Station_1".

The syntax rules are set by the user in the STEP 7 Professional project by assigning unique names for stations, CPUs, data blocks and variables. The selection of the symbols available in the OPC address space is made in the "Symbol configuration" dialog box of the OPC server in the TIA Portal.

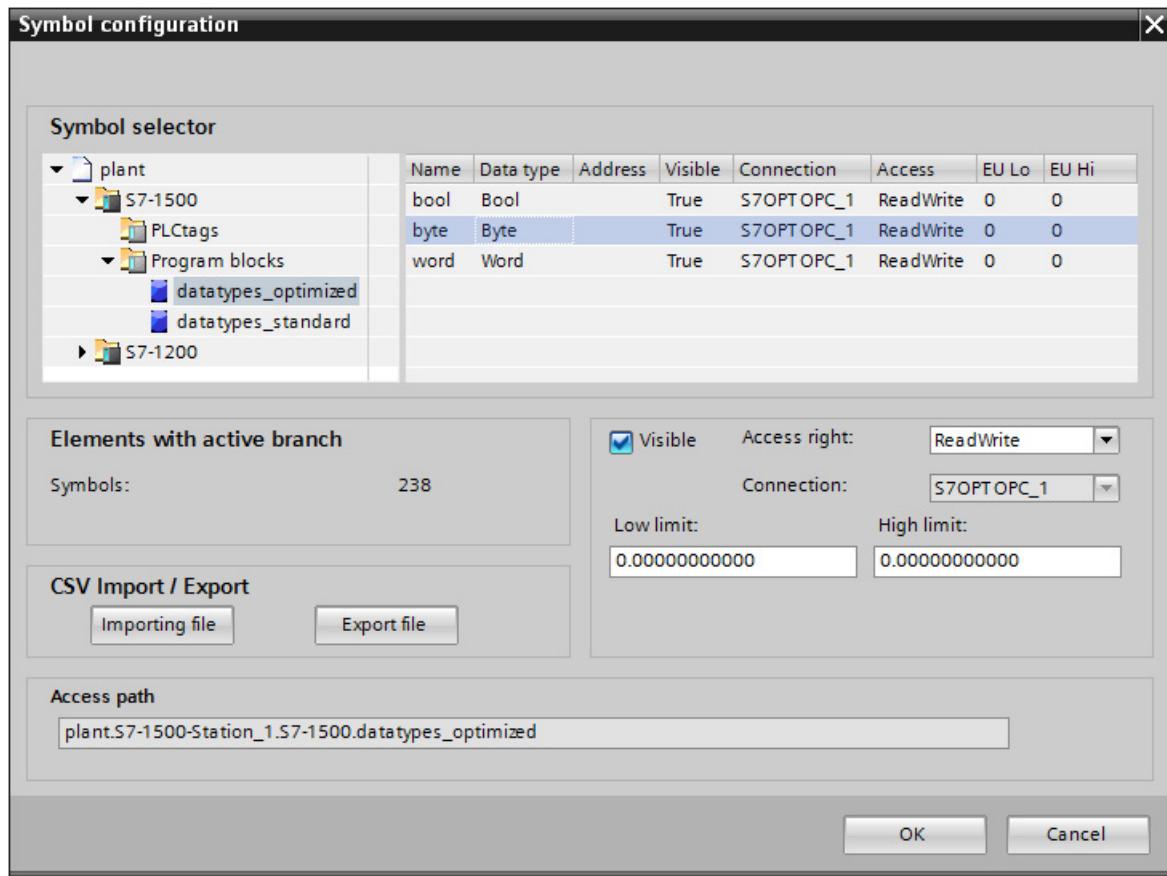


Figure 2-44 Symbol configuration of the OPC server in STEP 7 Professional (TIA Portal)

2.8.13 Block information objects of an S7 connection

2.8.13.1 Length information

The block objects under an S7OPT connection object

The type and size of the block structure on an S7 programmable controller is calculated during runtime. It is only possible to obtain this detailed information for the non-optimized blocks of the S7-1200 /S7-1500 station. Under an S7 connection object, there are the following block objects that provide this information to an OPC UA client:

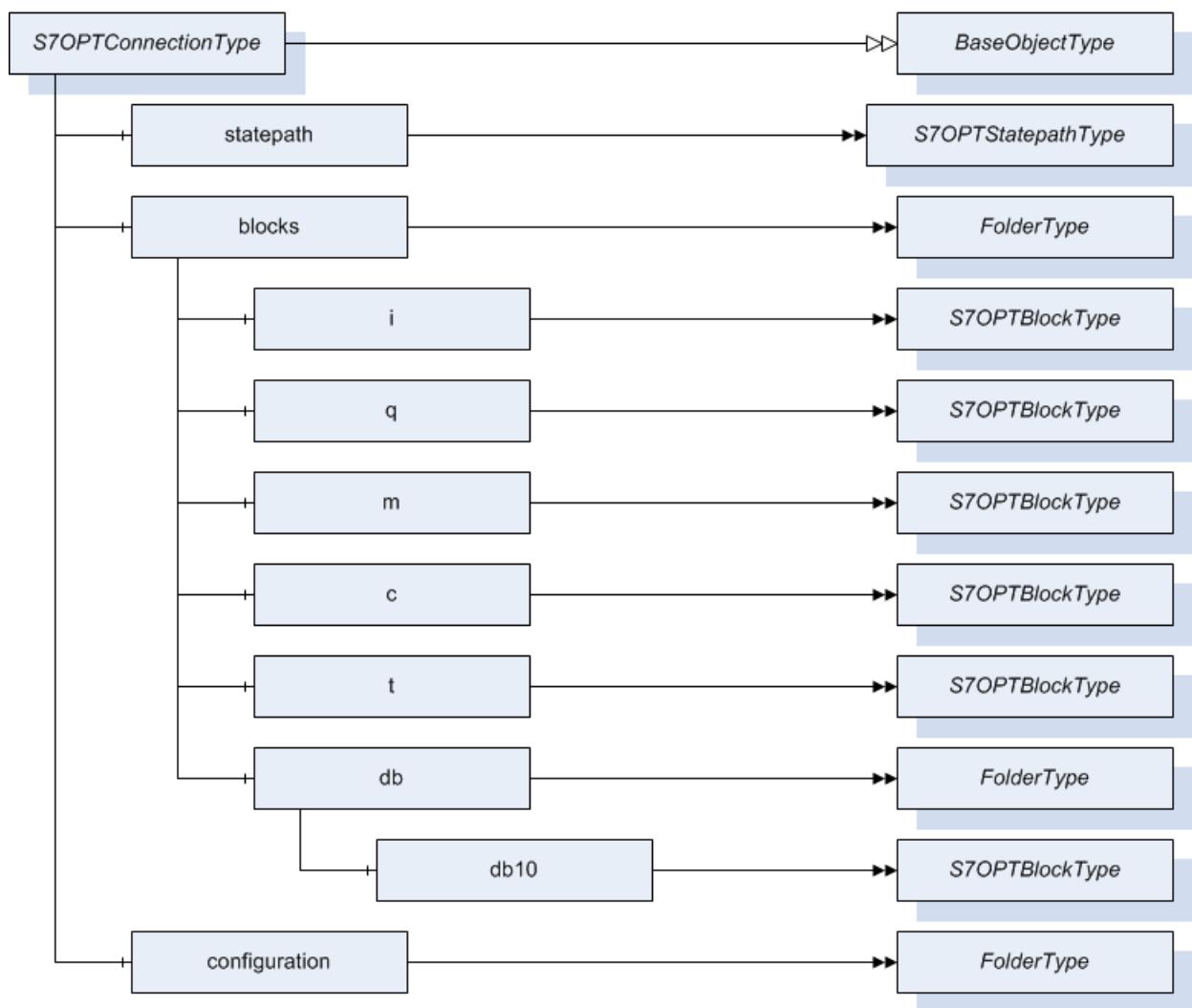


Figure 2-45 Block objects under an S7 connection object

Each block object has an OPC UA property with length/size information about the block.

Example:

Namespace URI: S7OPT: (-->Namespace index: 3)

S7-connection_1.db10.length //Length property of DB10 block (length in bytes)

Note

The length information field is available to you with data blocks that are called using standard access.

2.8.13.2 Template objects

The template object of a block object

For every block object displayed when browsing, a template data variable is generated whose Nodeld can be used as a template for further user-defined data objects. The template data variable has the standard data type B (or c or tbcd) for the relevant block object and always starts at address 0 (with data blocks for standard access). If this item is not accessible on the connection partner, this is indicated by corresponding access results and quality codes.

Example:

Namespace URI: *S7OPT:* (Namespace index: 3)

S7_connection_1.db10.0,b

S7_connection_1.m.0,b

2.8.13.3 Diagnostics and configuration information

The properties of an S7OPT connection object

In general, the properties of an S7 connection are configured the STEP 7 Professional (TIA Portal) configuration tool. During runtime, it may be useful to evaluate some of the configuration parameters.

Some configuration parameters are provided for OPC UA as properties for the S7OPT connection object:

Syntax for diagnostics and configuration information

Namespace URI: *S7OPT:* (–>Namespace index: 3)

<connectionname>.<S7OPTconnectionproperty>

<S7OPTconnectionproperty>:= "vfd"|"cp"|"remoteaddress"|"connect"|"modelversionid"|"fastconnectionstatereturnable"|"connecttimeout"|"timeout"|"abortconnectionafter"|"optimizebuffer"|"defaultalarmseverity"

S7 connection diagnostics property (read-only)	Description				
vfd	Name of the OPC server to which the connection is assigned. This normally has the text "OPC Server_1" for STEP 7 connections configured in the TIA Portal. Data type String, read-only.				
cp	Name of the interface parameter assignment to which the connection is assigned. Data type String, read-only.				
remoteaddress	Address of the connection partner. Data type String, read-only. The address of the connection partner is a data buffer with a data length dependent on the connection type. To make it more manageable for evaluation by the user, the data buffer is displayed formatted in a string. IP address (ISO-on-TCP) Format: "ddd.ddd.ddd.ddd" (each 1-3 decimal digits) or using the MAC address				
modelversionid	Type and version of the connection partner Data type UInt32, read only				
connect	Type of connection establishment. Data type UInt32, read-only. <table border="1" data-bbox="928 1143 1453 1291"> <tr> <td>1</td><td>Active, establish connection only when necessary, connection termination if not used after wait time.</td></tr> <tr> <td>2</td><td>Active, connection is kept established permanently.</td></tr> </table>	1	Active, establish connection only when necessary, connection termination if not used after wait time.	2	Active, connection is kept established permanently.
1	Active, establish connection only when necessary, connection termination if not used after wait time.				
2	Active, connection is kept established permanently.				
fastconnectionstatereturnenable	Fast return of a write/read job if the connection is interrupted. Data type Boolean, read and write. True: Enabled False: disabled				
connecttimeout	Connection establishment timeout in ms Data type UInt32, read and write. 0:no timeout >0:timeout in ms				
timeout	Job timeout for the productive traffic in ms. Data type UInt32, read and write. 0:no timeout >0:timeout in ms				

S7 connection diagnostics property (read-only)	Description
abortconnectionafter	<p>Automatic connection termination.</p> <p>Delay time for the automatic connection termination: After this time has elapsed, the OPC server automatically terminates the connection if there was no further variable access during this time. This allows the number of necessary connections to be reduced when variables are accessed at long intervals.</p> <p>Data type UInt32, read and write.</p> <p>0: no termination >0: idle time before termination in ms</p>
optimizebuffer	<p>Size of the optimization buffer for block communication with data blocks using standard access.</p> <p>Data type UInt16, read and write.</p> <p>0: No optimization >0: Size of the optimization buffer in bytes</p>
defaultalarmseverity	<p>Default priority for the statepath alarm.</p> <p>Data type UInt16, read and write.</p> <p>1: low priority ... 1000: high priority</p>

2.8.14 S7OPT OPC UA template data variables

With the process variables for the OPC UA S7OPT protocol, you have flexible setting options with which you can obtain the "non-optimized" process data of the plant in the data formats you require.

The wide variety of addressing options cannot, however, be put together in a fully browsable namespace. Even a data block called with standard access with the length of a single byte has approximately 40 different data format options - starting with Byte, SByte, arrays with one element of these, individual bits, arrays of bits with up to 8 array elements starting at different bit offsets.

The OPC UA server therefore supports the user with so-called template data variables in the S7OPT namespace. In a typical OPC client text input box, these templates can be converted to valid ItemIDs simply by changing a few characters.

Example:

```
S7_connection1.db<db>.<o>,dw
```

By replacing <db> with the data block number and <o> the offset within the data block, you obtain a valid Nodeld.

```
-> S7_connection1.db10.4,dw
```

The advantage of this concept is that it can be used by practically all OPC UA clients without any adaptation of the client being necessary.

Note

The usability of OPC UA S7OPT template data variables can be enabled and disabled in the "Communication Settings" configuration program in "OPC protocol selection" > click the arrow symbol beside "S7 optimized".

Template data variables within the browse hierarchy

The template data variables are sorted beside the corresponding folders in the namespace representation so that they can be used easily when required.

Special use of some of the attributes of the template data variables

The use of the OPC UA attributes is stipulated in the UA specification and requires no further explanation.

Example of the template of a data block byte variable:

Nodeld:	S7_connection1.db<db>.<o>,b
Browse name:	Template byte
Description:	<db>Address of the data block <o> offset within the data block

2.8.15 OPC UA events, conditions and alarms

2.8.15.1 What OPC UA alarms are there?

The following OPC UA event and alarm types are supported by the S7OPT OPC UA server:

- Statepath alarm
Alarms relating to the S7 connection status
- Consistency alarm
Alarm relating to inconsistent alarm configuration
- Program alarms (block-related PLC alarms)
From a signal change, the "Program_Alarm" alarm block generates a PLC alarm with or without mandatory acknowledgment with up to ten associated values.

Note

Program alarms are supported by the S7OPT OPC UA server only for the S7-1500.

Note

Before program alarms can be output, they need to be configured in SIMATIC STEP 7 Professional (TIA Portal). They are created in the programming editor and processed in the alarm editor. Refer to the system manual "SIMATIC STEP 7 Professional" or the information system in SIMATIC STEP 7 Professional (TIA Portal).

2.8.15.2 What are OPC UA events?

A plant is characterized by the states of its hardware and software components. With UA Alarming, you have the option of reporting status changes selected from the states to the registered user as events. The information about the event is stored in its properties. Which properties an event has is defined by the event type.

The event type has its own properties or properties inherited from another event type (that may also in turn have inherited properties). The option of simple inheritance of properties leads to an event type hierarchy. The UA Alarming Specification includes numerous predefined event types structured in a predefined type hierarchy. The UA Alarming Specification also includes stipulations about the type of properties and the semantics of these properties. All predefined events are located in the namespace ns="http://opcfoundation.org/UA/".

2.8.15.3 Which S7OPT event types are supported by the S7OPT OPC UA server?

The S7OPT OPC UA server is based on predefined event types and derives its own event types from the predefined event types. Separate event types are defined for S7OPT events. All S7OPT event types are located in the namespace ns="S7OPTTYPES:".

S7OPT UA Alarming is used to represent PLC alarms. The table below shows which event types an S7OPT OPC UA server reports. With the exception of the first two S7OPT event types, events with the specified event type occur only after receiving a configured program alarm.

NodId and display name of the S7OPT event type	Meaning of the S7OPT event type
ns= S7OPTTYPES:, i=14 "S7OPTStatepathAlarmType"	"S7OPTStatepathAlarmType" maps the inverse state of an S7 connection ("Inactive" if the S7 connection is established ("UP") and "Active" if the S7 connection is not established ("Down")). The status is obtained only at the PC end and, as a result, can also be reported when there is no physical connection to the PLC.
ns= S7OPTTYPES:, i=15 "S7OPTConsistencyAlarmType"	"S7OPTConsistencyAlarmType" shows an inconsistent alarm configuration.

NodeID and display name of the S7OPT event type	Meaning of the S7OPT event type
ns= S7OPTTYPES; i=43 "S7OPTOffNormalAlarmType"	Configured program alarms with or without mandatory acknowledgement. The program alarms of this type have a status or condition. This can adopt the status "coming" or "going".
ns= S7OPTTYPES; i=61 "S7OPTInfoReportEventType"	Configured program alarms that only serve as information. To use these, the check box in the "Information only" column in the alarm editor of SIMATIC STEP 7 Professional (TIA Portal) must be selected.

2.8.15.4 Event type hierarchy of S7OPT OPC UA servers

The event type hierarchy of the S7OPT OPC UA server consists of the standard event type hierarchy with some event types being derived from S7OPT event types.

The event type hierarchy can be browsed with the OPC Scout V10.

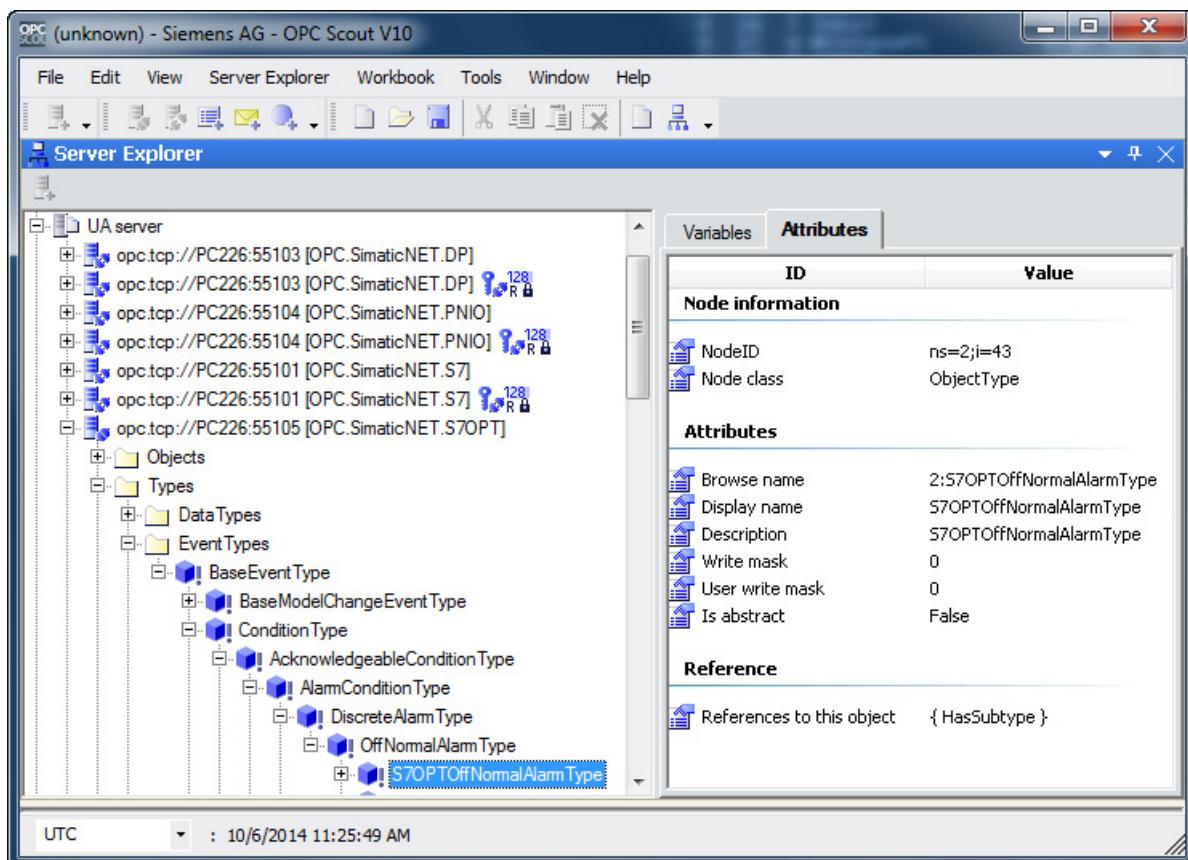


Figure 2-46 Mapping the standard event type hierarchy

With standard event types, inheritance is as follows:

Standard event type with display name	inherits from ...
"ConditionType"	"BaseEventType"
"AcknowledgeableConditionType"	"ConditionType"
"AlarmConditionType"	"AcknowledgeableConditionType"
"DiscreteAlarmType"	"AlarmConditionType"
"OffNormalAlarmType"	"DiscreteAlarmType"
"SystemOffNormalAlarmType"	"OffNormalAlarmType"

The standard event type with the display name "BaseEventType" is the type from which all event types are derived. This type defines all properties used in all events as well as their behavior.

An event type has a numeric NodId (for example ns="http://opcfoundation.org/UA/", i=2041 and display name ="BaseEventType").

The inheritance of the S7OPT event types from the standard event types is as follows:

S7OPT event type with display name	inherits from ...
"S7OPTStatepathAlarmType"	"SystemOffNormalAlarmType"
"S7OPTConsistencyAlarmType"	"SystemOffNormalAlarmType"
"S7OPTOffNormalAlarmType"	"OffNormalAlarmType"
"S7OPTInfoReportEventType"	"BaseEventType"

With S7OPT–UA Alarming, the S7OPT event types "S7OPTStatepathAlarmType", "S7OPTConsistencyAlarmType" and "S7OPTOffNormalAlarmType" are instantiated. A PLC alarm is therefore mapped with all its properties to an alarm object. The properties of the relevant instance can also be read or monitored with Data Access.

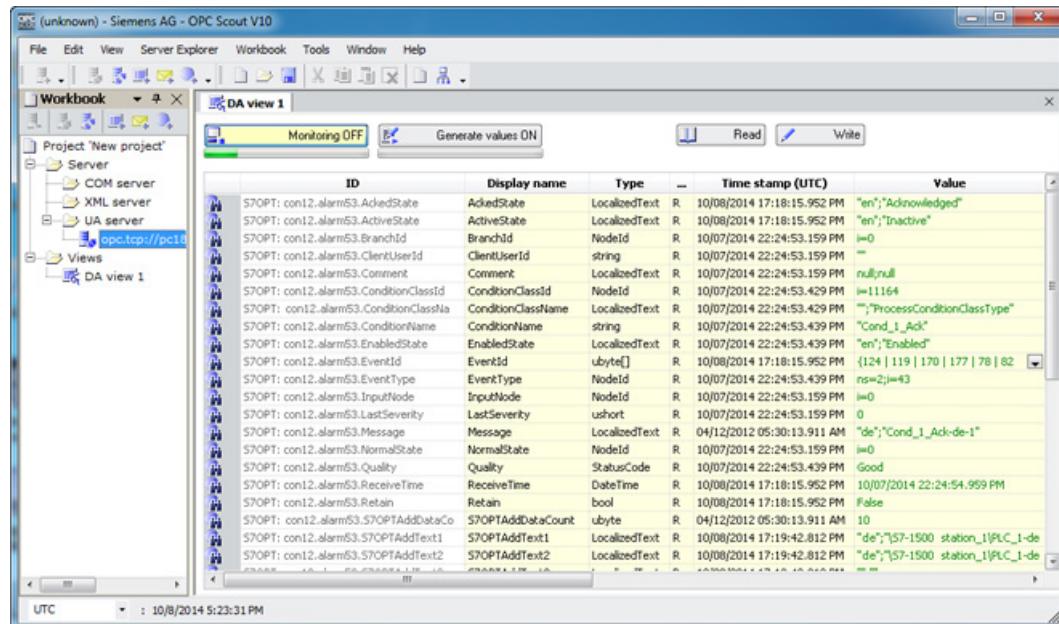


Figure 2-47 Monitoring alarm objects with Data Access

The sections "S7OPT event types (Page 266)" and "Area tree and source space (Page 272)" describe the properties relevant to the user when programming a UA application. They are grouped according to the event types in which they are defined. When referencing a property, the data type "QualifiedName" is used. "QualifiedName" contains a namespace and a browse name (sometimes a browse path).

Since, however, in the context of the document, only properties will be referenced whose "QualifiedNames" have a namespace that is identical to the namespace of the defining types, the specification of the namespace will be omitted. Instead of "QualifiedName", the browse name of the property will be specified. In addition to this, the requested attribute separated by ":" from the data type is shown in parentheses. In most cases, the requested attribute is numeric 13 which specifies the "Value". Occasionally, the numeric value 1 occurs and this specifies the Nodeld.

2.8.16 Standard event types and the use of their properties

2.8.16.1 Standard event type with the display name "BaseEventType"

Nodeld: ns="http://opcfoundation.org/UA/", i=2041

Derived from: is not derived from any other event type.

Browse name of the relevant properties

- "EventId" (13|ByteString)
- "EventType" (13|Nodeld)
- "SourceNode" (13|Nodeld)
- "SourceName" (13|String)
- "Time" (13|DateTime)
- "ReceiveTime" (13|DateTime)
- "Message" (13|LocalizedText)
- "Severity" (13|UInt16)

S7OPT event types directly derived from this type: ns="S7OPTTYPES:", i =61, display name="S7OPTInfoReportEventType".

An event of this event type is uniquely identified by the SourceName.

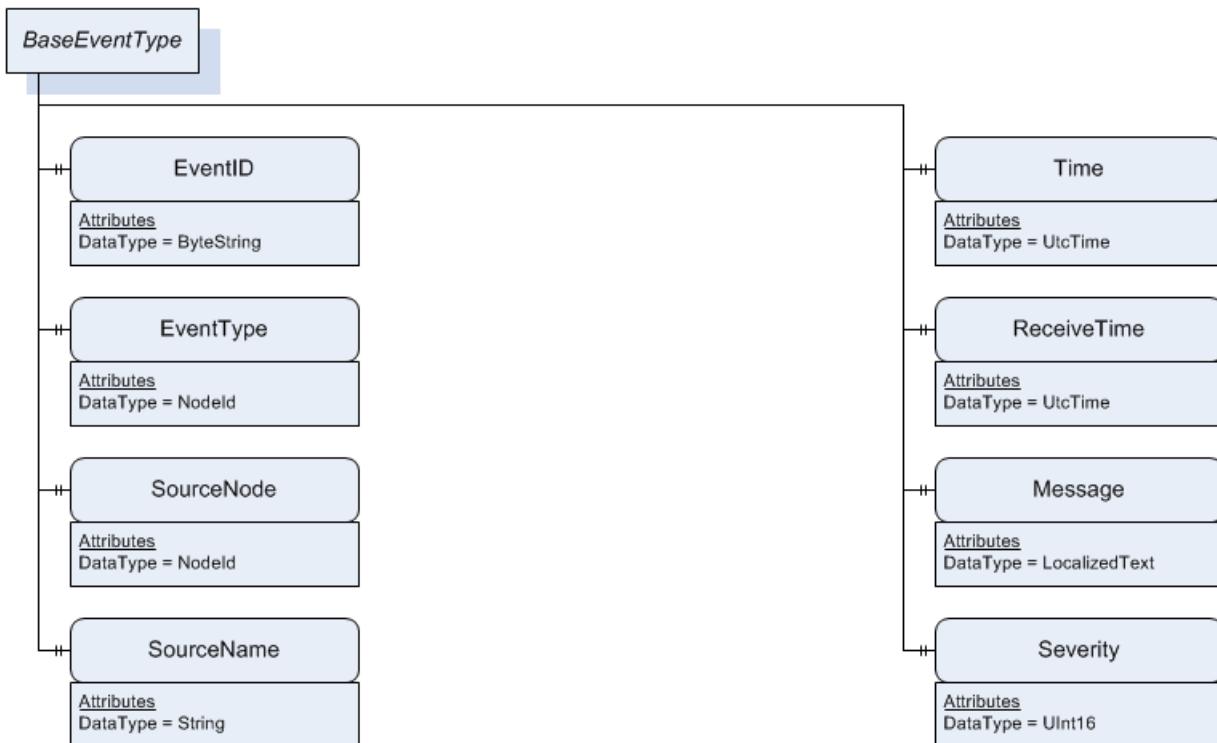


Figure 2-48 Representation of the BaseEventType

EventId

An identifier that uniquely identifies and references an event. The client requires the EventId, for example to acknowledge alarms.

EventType

is always one of the S7OPT event types listed earlier.

SourceNode

Source node of an event; in S7OPT Alarming, an object specified by a Nodeld in ns="S7OPTSOURCES:" or the S7 connection object in ns="S7OPT:". See also section "Area tree and source space (Page 272)".

SourceName

See section "Forming the SourceName, Message and Severity (Page 259)".

Time

Time stamp as close to the process as possible. This is decided by the configuration and the following options are available:

- PLC time stamp
- PLC time + offset
- PC time (UTC)

With PC time(UTC), "Time" is identical to "ReceiveTime".

ReceiveTime

Time stamp of the PC.

Message

See section "Forming the SourceName, Message and Severity (Page 259)".

Severity

See section "Forming the SourceName, Message and Severity (Page 259)".

2.8.16.2 Forming the SourceName, Message and Severity

Some of the names of the properties are set by the STEP 7 project.

The names used in the STEP 7 project, e.g. S7 connection name, station name, PLC name are used to form the properties.

Formation of SourceName

Nodeld and display name of the S7OPT event type	Rule for formation of SourceName
ns= S7OPTTYPES; i=14 "S7OPTStatepathAlarmType"	S7 connection name + ".statepath" Example: connection2.statepath A corresponding SourceNode exists.
ns= S7OPTTYPES; i=15 "S7OPTConsistencyAlarmType"	S7 connection name + ".statepath" Example: connection2.statepath A corresponding SourceNode exists.

Nodeld and display name of the S7OPT event type	Rule for formation of SourceName
ns= S7OPTTYPES:, i=43 "S7OPTOffNormalAlarmType"	Configured program alarms: SourceName consists of Station name (e.g. "station_1") + "\" PLC name (e.g. "PLC_1516") + "\" Symbolic name of the instance data block of the PLC alarm (e.g. "motor1") Example: "station_1\PLC_1516\motor1"
ns= S7OPTTYPES:, i=61 "S7OPTInfoReportEventType"	Configured program alarms: SourceName consists of Station name (e.g. "station_1") + "\" PLC name (e.g. "PLC_1516") + "\" Symbolic name of the instance data block of the PLC alarm (e.g. "motor1") Example: "station_1\PLC_1516\motor1"

Formation of Message

Nodeld of the S7OPT event type	Rule for formation of Message
ns= S7OPTTYPES:, i=14 "S7OPTStatepathAlarmType"	"statepath"
ns= S7OPTTYPES:, i=15 "S7OPTConsistencyAlarmType"	"consistency"
ns= S7OPTTYPES:, i=43 "S7OPTOffNormalAlarmType"	Configured program alarms: Alarm text of the program alarms. The alarm text can include dynamic parameters (variables and text lists).
ns= S7OPTTYPES:, i=61 "S7OPTInfoReportEventType"	Configured program alarms: Alarm text of the program alarms. The alarm text can include dynamic parameters (variables and text lists).

Formation of Severity

Nodeld and display name of the S7OPT event type	Rule for formation of Severity
ns= S7OPTTYPES:, i=14 "S7OPTStatepathAlarmType"	Default priority detected for every S7 connection.
ns= S7OPTTYPES:, i=15 "S7OPTConsistencyAlarmType"	Default priority detected for every S7 connection.

Nodeld and display name of the S7OPT event type	Rule for formation of Severity
ns= S7OPTTYPES; i=43 "S7OPTOffNormalAlarmType"	Configured program alarms: Is derived from the PLC alarm priority (0 to 16) of the program alarms.
ns= S7OPTTYPES; i=61 "S7OPTInfoReportEventType"	Configured program alarms: Is derived from the PLC alarm priority (0 to 16) of the program alarms.

Table 2- 6 Conversion table PLC alarm priority - Severity

PLC alarm priority	Severity
0	1
1	63
2	125
3	188
4	250
5	313
6	375
7	438
8	500
9	563
10	625
11	688
12	750
13	813
14	875
15	938
16	1000

2.8.16.3 Standard event type with the display name "ConditionType"

NodeID: ns="http://opcfoundation.org/UA/", i=2782

Derived from: ns="http://opcfoundation.org/UA", i=2041 with display name "BaseEventType"

Browse name of the relevant properties

"" (1|NodeID)

"ConditionName" (13|String)

"EnableState|ID" (13|Boolean)(Browse-Path)

"EnableState" (13|LocalizedText)

"Quality" (13|StatusCode)

S7OPT event types directly derived from this type: None

Event types derived directly or indirectly from this type have a condition instance.

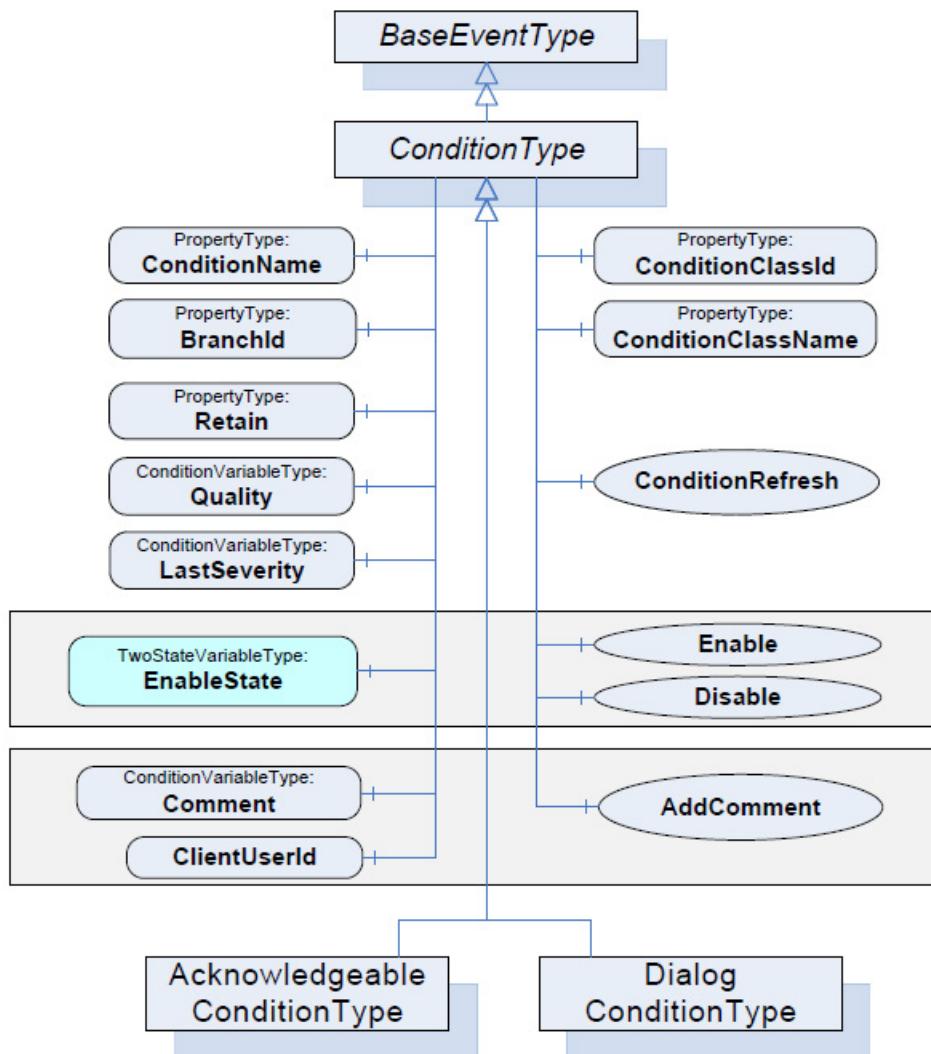


Figure 2-49 Representation of the ConditionType

ConditionName

An event of a condition is identified uniquely by the combination of "SourceName" and "ConditionName". See below, section "Formation of ConditionName".

EnableState

Is part of a state machine. Possible values according to the UA Alarming specification for EnableState are "en";"Enabled" and "en";"Disabled".

Quality

For the possible quality values, refer to the UA Specification.

2.8.16.4 Formation of ConditionName

Some of the names of the properties are set by the STEP 7 project.

Nodeld and display name of the S7OPT event type	Rule for formation of ConditionName
ns=S7OPTTYPES:, i=14 "S7OPTStatepathAlarmType"	"statepath"
ns= S7OPTTYPES:, i=15 "S7OPTConsistencyAlarmType"	"consistency"
ns= S7OPTTYPES:, i=43 "S7OPTOffNormalAlarmType"	Configured program alarms: Name of the program alarms Example: "MyAlarm"
ns= S7OPTTYPES:, i=61 "S7OPTInfoReportEventType"	Configured program alarms: Name of the program alarms Example: "MyAlarm"

2.8.16.5 Standard event type with the display name "AcknowledgeableConditionType"

NodeId: ns="http://opcfoundation.org/UA/", i=2881

Derived from: ns="http://opcfoundation.org/UA", i=2782 with display name "ConditionType"

Browse name of the relevant properties

"AckedState|Id" (13|Boolean)

"AckedState" (13|LocalizedText)

S7OPT event types directly derived from this type: None

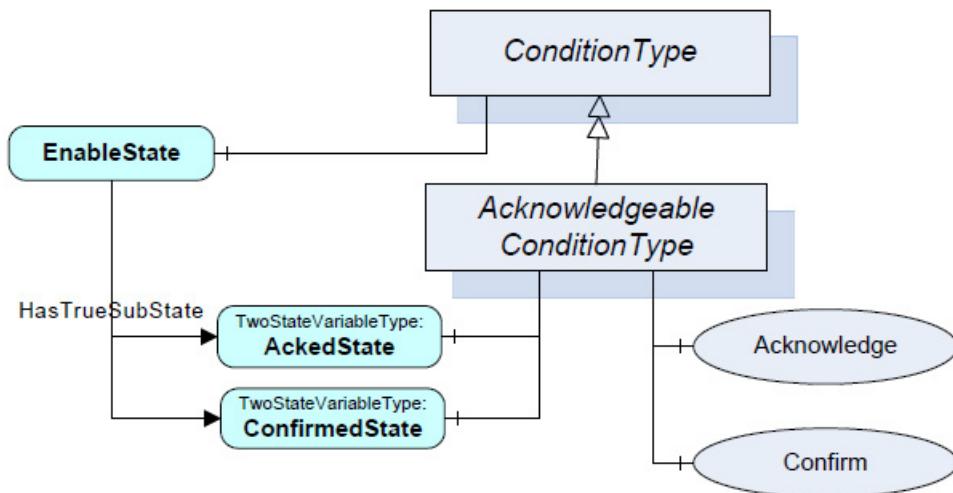


Figure 2-50 Representation of the AcknowledgeableConditionType

AckedState

Is part of a state machine. Possible values are "en";"Acknowledged" and "en";"Unacknowledged". Each value change of AckedState is reported with an event. Events reported with the AckedState "en", "Unacknowledged" must be acknowledged with the acknowledge function. Refer to the system manual "SIMATIC STEP 7 Professional" or the information system in SIMATIC STEP 7 Professional (TIA Portal).

2.8.16.6 Standard event type with the display name "AlarmConditionType"

NodeId: ns="http://opcfoundation.org/UA/", i=2915

Derived from: ns="http://opcfoundation.org/UA", i=2881 with display name

"AcknowledgeableConditionType"

Browse name of the relevant properties

"ActiveState|Id" (13|Boolean)

"ActiveState" (13|LocalizedText)

S7OPT event types directly derived from this type: None

Possible values according to the UA Alarming specification for ActiveState are "en";"Active" and "en";"Inactive".

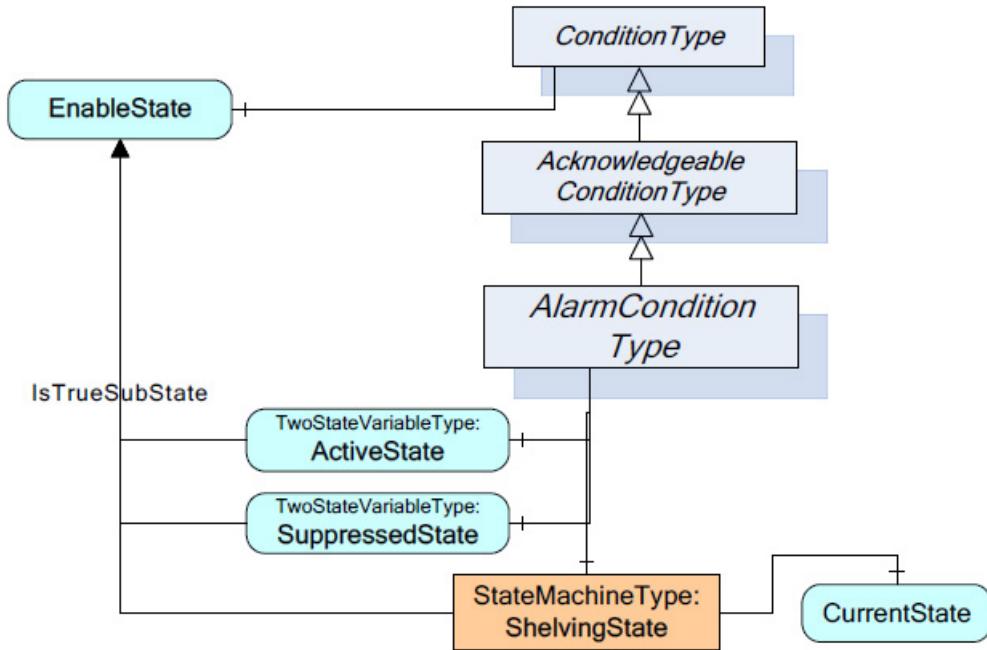


Figure 2-51 Representation of the **AlarmConditionType**

ActiveState

Is part of a state machine. In S7OPT UA Alarming, the ActiveState is controlled by PLC alarms. If the alarm triggering signal has the value "1", ActiveState "Active" is reported, if it has the value "0", "Inactive" is reported. Each value change of ActiveState is reported by an event.

ActiveState with event type with the display name = "S7OPTStatepathAlarmType"

Here, the "Active" state is reached when the S7 connection is not established. The "Inactive" state is reached when the S7 connection is established.

2.8.16.7 Standard event type with the display name "OffNormalAlarmType"

NodeID: ns="http://opcfoundation.org/UA/", i=10637

Indirectly derived from: ns="http://opcfoundation.org/UA/", i=2915 with display name "AlarmConditionType"

Browse name of the relevant properties: None

S7OPT event types directly derived from this type:

ns=S7OPTTYPES:, i =14, display name="S7OPTStatepathAlarmType"

ns=S7OPTTYPES:, i =15, display name="S7OPTConsistencyAlarmType"

ns=S7OPTTYPES:, i =43, display name="S7OPTOffNormalAlarmType"

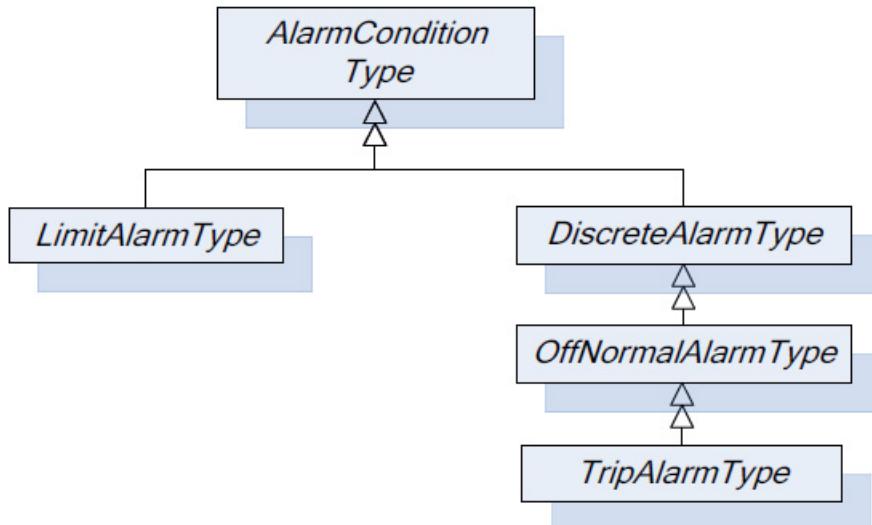


Figure 2-52 Placement of OffNormalAlarmType in the alarm type hierarchy

2.8.17 S7OPT event types

2.8.17.1 S7OPT event type with the display name "S7OPTStatepathAlarmType"

NodId: ns="S7OPTTYPES:", i=14

Indirectly derived from: ns="http://opcfoundation.org/UA/", i=2915 with display name "AlarmConditionType"

Browse name of the relevant properties:
"S7OPTConnection" (13|NodId)

This S7OPT event type maps the inverse state of an S7 connection ("inactive" if the S7 connection is established ("UP"); and "active" if the S7 connection is not established ("DOWN" or "RECOVERY")).

The status is obtained only at the PC end and, as a result, can also be reported when there is no physical connection to the PLC.

In the "Active" status, no further PLC alarms can be received.

S7OPTConnection

Specifies the NodId of the connection via which a PLC alarm is received.

Example: ns="S7OPT:", s="connectionname".

2.8.17.2 S7OPT event type with the display name "S7OPTConsistencyAlarmType"

NodId: ns="S7OPTTYPES:", i=15

Indirectly derived from: ns="http://opcfoundation.org/UA/", i=2915 with display name "AlarmConditionType"

Browse name of the relevant properties:
"S7OPTConnection" (13|NodId)

The S7OPT event type "S7OPTConsistencyAlarmType" does not need to be acknowledged. It shows an inconsistent alarm configuration between the PLC and the PC station. Such inconsistent PLC alarms are not reported to the OPC client. In this case, the condition instance "consistency" is set to "Active" once for this PLC. It is advisable to update the alarm configuration first on the PLC and then on the PC station.

S7OPTConnection

Specifies the Nodeld of the connection via which a PLC alarm is received.
Example: ns="S7OPT:", s="connectionname".

2.8.17.3 S7OPT event type with the display name "S7OPTOffNormalAlarmType"

Nodeld: ns="S7OPTTYPES:", i=43
 Indirectly derived from: .ns="http://opcfoundation.org/UA/", i=10637 with display name "OffNormalAlarmType"
 Browse name of the relevant properties:
 "S7OPTAlarmId" (13|UInt32)
 "S7OPTConnection" (13|Nodeld)
 "S7OPTTime" (13|DateTime)
 "S7OPTDisplayClass" (13|UInt16)
 "S7OPTAlarmClass" (13|String)
 "S7OPTInfoText" (13|LocalizedText)
 "S7OPTAddDataCount" (13|Byte)
 "S7OPTAddData1|Datavalue" (13|Variant)
 "S7OPTAddData2|Datavalue" (13|Variant)
 "S7OPTAddData3|Datavalue" (13|Variant)
 "S7OPTAddData4|Datavalue" (13|Variant)
 "S7OPTAddData5|Datavalue" (13|Variant)
 "S7OPTAddData6|Datavalue" (13|Variant)
 "S7OPTAddData7|Datavalue" (13|Variant)
 "S7OPTAddData8|Datavalue" (13|Variant)
 "S7OPTAddData9|Datavalue" (13|Variant)
 "S7OPTAddData10|Datavalue" (13|Variant)
 "S7OPTAddText1" (13|LocalizedText)
 "S7OPTAddText2" (13|LocalizedText)
 "S7OPTAddText3" (13|LocalizedText)
 "S7OPTAddText4" (13|LocalizedText)
 "S7OPTAddText5" (13|LocalizedText)
 "S7OPTAddText6" (13|LocalizedText)
 "S7OPTAddText7" (13|LocalizedText)
 "S7OPTAddText8" (13|LocalizedText)
 "S7OPTAddText9" (13|LocalizedText)

This S7OPT event type maps the configured "program alarms" with or without mandatory acknowledgment with up to ten associated values.

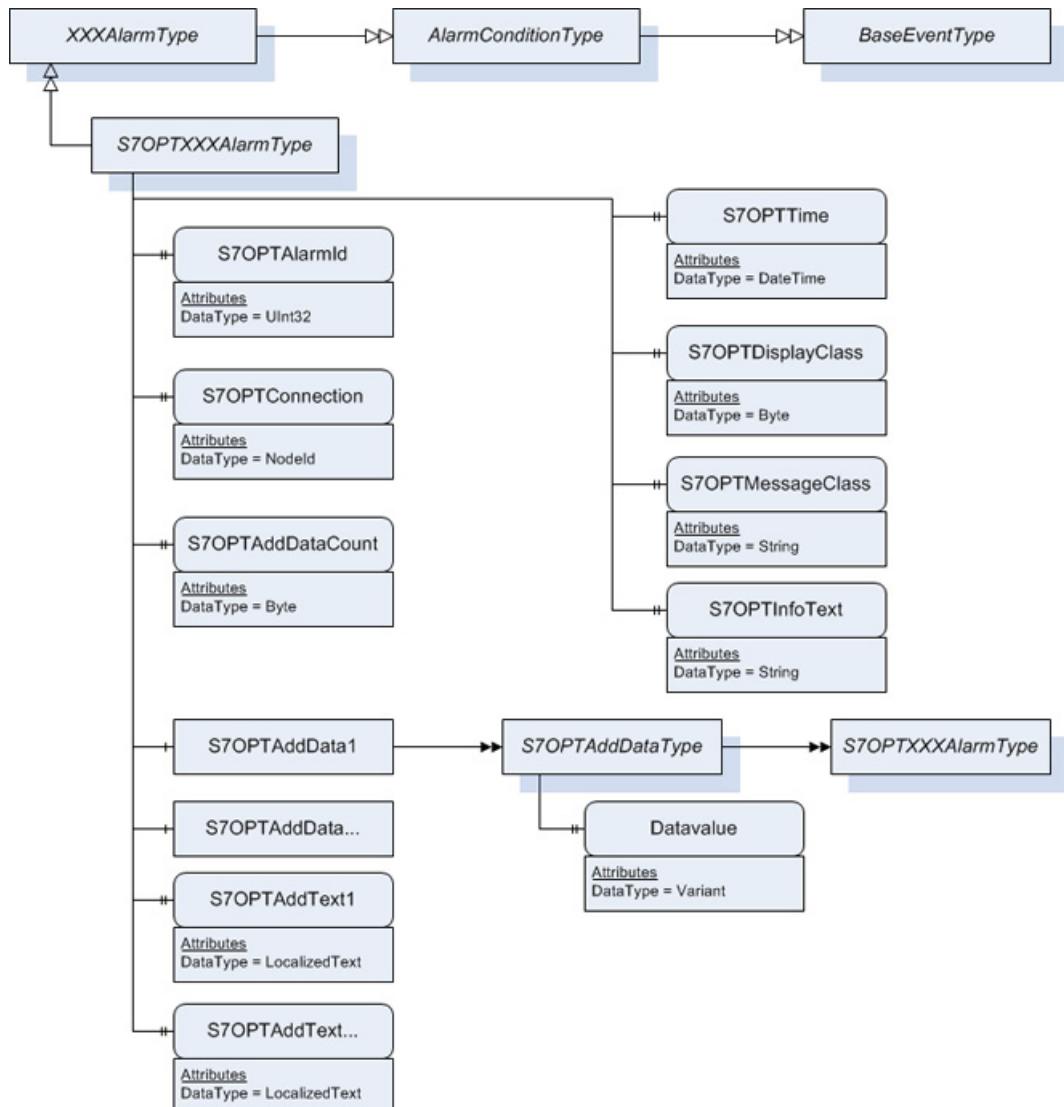


Figure 2-53 Representation of the S7-specific properties for the S7OPTOffNormalAlarmType

S7OPTAlarmId

Alarm number of the PLC alarm. This is unique throughout the PLC.

S7OPTConnection

Specifies the Nodeld of the connection via which the PLC alarm is received.

Example: ns="S7OPT:", s="connectionname".

S7OPTTime

The time stamp indicates when the PLC alarm occurred on the PLC (PLC time).

S7OPTDisplayClass

Display class of the PLC alarm.

S7OPTAlarmClass

Alarm class of the PLC alarm. The alarm class decides whether the alarm needs to be acknowledged or not.

S7OPTInfoText

Infotext of the PLC alarms from the configuration. The infotext can include dynamic parameters (variables and text lists).

S7OPTAddDataCount

Is the actual number of associated values of the PLC alarms.

S7OPTAddData[n]||Datavalue

Are the associated values of the generating PLC alarms.
 $1 \leq n \leq 10$.

S7OPTAddText[n]

$1 \leq n \leq 9$. Additional texts of the PLC alarms from the configuration.

2.8.17.4 S7OPT event type with the display name "S7OPTInfoReportEventType"

NodId: ns="S7OPTTYPES:", i=61
Indirectly derived from: .ns="http://opcfoundation.org/UA/", i=2041 with display name "BaseEventType"
Browse name of the relevant properties:
"S7OPTInfoReportId" (13|UInt32)
"S7OPTInfoReportName" (13|String)
"S7OPTConnection" (13|NodId)
"S7OPTTTime" (13|DateTime)
"S7OPTDisplayClass" (13|UInt16)
"S7OPTAlarmClass" (13|String)
"S7OPTInfoText" (13|LocalizedText)
"S7OPTAddDataCount" (13|Byte)
"S7OPTAddData1|Datavalue" (13|Variant)
"S7OPTAddData2|Datavalue" (13|Variant)
"S7OPTAddData3|Datavalue" (13|Variant)
"S7OPTAddData4|Datavalue" (13|Variant)
"S7OPTAddData5|Datavalue" (13|Variant)
"S7OPTAddData6|Datavalue" (13|Variant)
"S7OPTAddData7|Datavalue" (13|Variant)
"S7OPTAddData8|Datavalue" (13|Variant)
"S7OPTAddData9|Datavalue" (13|Variant)
"S7OPTAddData10|Datavalue" (13|Variant)
"S7OPTAddText1" (13|LocalizedText)
"S7OPTAddText2" (13|LocalizedText)
"S7OPTAddText3" (13|LocalizedText)
"S7OPTAddText4" (13|LocalizedText)
"S7OPTAddText5" (13|LocalizedText)
"S7OPTAddText6" (13|LocalizedText)
"S7OPTAddText7" (13|LocalizedText)
"S7OPTAddText8" (13|LocalizedText)
"S7OPTAddText9" (13|LocalizedText)

This S7OPT event type maps the configured program alarms that only serve as information with up to ten associated values.

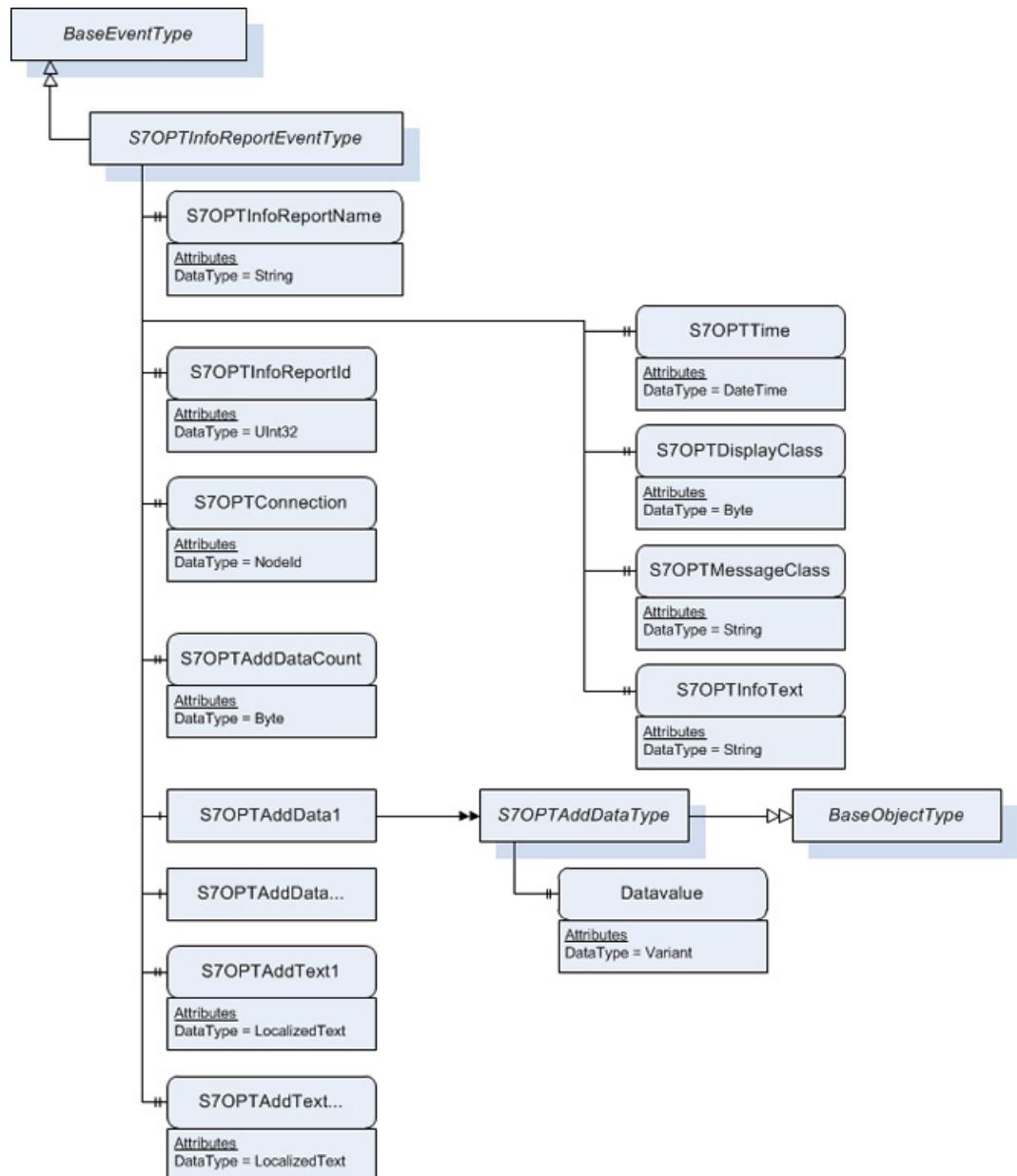


Figure 2-54 Representation of the S7-specific properties for the S7OPTInfoReportEventType

S7OPTInfoReportId

Alarm number of the PLC alarm. This is unique throughout the PLC.

S7OPTInfoReportName

Is made up of "inforeport" + alarm number of the PLC alarm.
Example: inforeport56

S7OPTConnection

Specifies the Nodeld of the connection via which the PLC alarm is received.
Example: ns="S7OPT:", s="connectionname".

S7OPTTime

The time stamp indicates when the PLC alarm occurred on the PLC.

S7OPTDisplayClass

Display class of the PLC alarm.

S7OPTAlarmClass

Alarm class of the PLC alarm. The alarm class decides whether the alarm needs to be acknowledged or not.

S7OPTInfoText

Infotext of the PLC alarms from the configuration. The infotext can include dynamic parameters (variables and text lists).

S7OPTAddDataCount

Is the actual number of associated values of the PLC alarms.

S7OPTAddData[n]|Datavalue

Are the associated values of the generating PLC alarms.
 $1 \leq n \leq 10$.

S7OPTAddText[n]

$1 \leq n \leq 9$. Additional texts of the PLC alarms from the configuration.

2.8.18 Area tree and source space

With the S7OPT OPC UA server, the area tree and the source space contain the SourceNodes of configured PLC alarms.

S7 UA Alarming sets up the area tree to assign the SourceNodes to areas. Nodes of the area tree are special UA folders that map areas of the plant. These special UA folders are known as area nodes.

The path name from the STEP 7 project is used as the area tree. Is made up of the following components within the project navigation:

- Station name
- PLC name
- Groups

The user-defined groups allow topological mapping of the existing customer plant. They can be inserted in the project tree of the STEP 7 project.

The station name and the PLC name are permanently linked and cannot be separated. Groups can only be inserted before or after.

The station name is inserted automatically before the station name.

Example:

```

plant1 (group)
  station_1 (station name)
    PLC_1516 (PLC name)
      house1 (group)
        motor1 (instance data block of the PLC alarm)
      house2 (group)
        motor2 (instance data block of the PLC alarm)

```

Based on the example, the area tree and the SourceNodes are derived as follows:

Nodes of the area tree:

- ns="S7OPTAREAS:", s="plant1".
- ns="S7OPTAREAS:", s="plant1\station_1".
- ns="S7OPTAREAS:", s="plant1\station_1\PLC_1516".
- ns="S7OPTAREAS:", s="plant1\station_1\PLC_1516\house1".
- ns="S7OPTAREAS:", s="plant1\station_1\PLC_1516\house2".

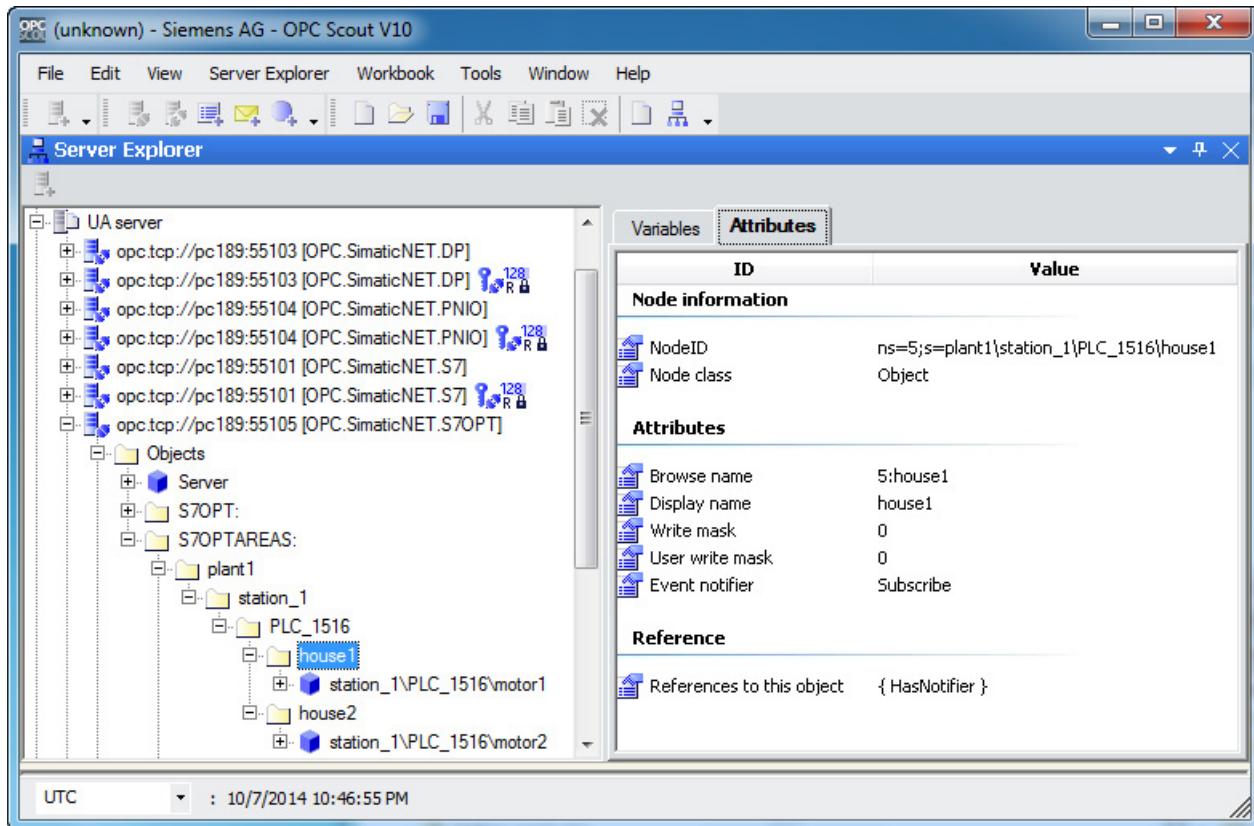
The SourceName of the SourceNode is made up as follows:

- Station name + „\“
- PLC name (name of the configured PLC) + "\"
- Symbolic name of the instance data block of the PLC alarm

According to the example:

- ns="S7OPTSOURCES:", s="station_1\PLC_1516\motor1".
- ns="S7OPTSOURCES:", s="station_1\PLC_1516\motor2".

The area tree and source space can be browsed with the OPC client (e.g. OPC Scout V10).



The area node with the display name "house1" (NodeID ns="S7OPTAREAS:", s="plant1\station_1\PLC_1516\house1") is selected in the left window. The right-hand window displays its attributes; from these, it can be seen that the area node has the reference "HasNotifier".

2.8.19 Receiving events

EventItems and EventNotifier

The only possibility of receiving current events from a server is to generate one or more monitored event items in a subscription. With an event, the EventNotifier attribute (12) is monitored. EventNotifiers only reference objects that have the "HasNotifier" reference. In S7OPT OPC UA Alarming, these are the server node ns="http://opcfoundation.org/UA/", i=2253, all nodes of the S7 connections, for example, ns="S7OPT:", s="S7_connection_1" and all area nodes in ns="S7OPTAREAS:".

If an EventNotifier of an area node is monitored, all events are reported that originate from this area. Depending on the particular system (the set of potentially available EventSources can be taken from the source space combined with the area space), the number of reported events can be extremely high. This makes it necessary to select of the events.

Filtering of events

Apart from the suitable selection of the event item, on the OPC UA interface it is also possible to filter events with certain properties and their values.

Note

The selection of the event item and the setting of the filter conditions decide whether an event can be received by an OPC client.

Selection of the properties

When a monitored event item is generated, the properties to be returned must be specified. If no properties are specified, it is known that an event has occurred, but not which. To include a property in the list of properties to be returned, the following needs to be specified for each property:

- the Nodeld of the defined event type (Typeld),
- the browse name and possibly browse path of the property and
- the Attributeld

You will find the Nodeld, browse name and, if applicable, the browse path and the Attributeld of the defined event type in the section "Event type hierarchy of S7OPT OPC UA servers (Page 255)".

An event does not need to have all properties to be returned. "Zero" is returned for properties that an event does not have. S7OPT OPC UA Alarming cannot distinguish between properties that the event does not have or such that do not have a valid property due to bad specification, for example, of the browse name.

Note

Not all status changes of an alarm are forwarded by an event on the OPC UA interface. If there is a fast change in the alarm status, it is possible that some status changes are not reported by an event and only the last and now current status is sent. This depends on OPC parameters such as "PublishingInterval" as well as the configuration and the computer performance.

2.8.20 Methods of UA alarms

Enable()/Disable()

No events are generated on clients for an alarm that is "disabled". Regardless of this, the alarms to the PLC are always monitored because alarms for the controller cannot be individually "enabled" or "disabled".

NOTICE

Deactivating events

When using these methods, note that a client can disable the events for other clients.

AddComment()

A comment can be stored for each alarm instance.

ConditionRefresh()

All active or unacknowledged alarms are reported (property "Retain" = true).

Acknowledge()

The acknowledgment of an alarm is transferred to the PLC. Regardless, the change in the acknowledgment status is returned by the PLC to the OPC server explicitly, only then does the alarm change to the acknowledged status at the OPC end and triggers appropriate events.

2.9 Open communication services (SEND/ RECEIVE)

Depending on the communications network, there are two versions of the communications network (SEND/RECEIVE):

- Open communication services (SEND/ RECEIVE) via Industrial Ethernet
Protocol ID: SR
- Open communication services (SEND/ RECEIVE) via PROFIBUS
Protocol ID: FDL

2.9.1 Open communication services (SEND/ RECEIVE) via Industrial Ethernet

The open communications service (SEND/RECEIVE) over Industrial Ethernet is also known as the SEND/RECEIVE protocol. It allows communication with S5 and S7 devices as well as with third-party devices.

Properties of open communication services (SEND/ RECEIVE) over Industrial Ethernet

- Communication using SIMATIC S5 handling blocks and S7 function blocks
- Link between two PC stations possible.

- Support of the WRITE and FETCH function for access to objects of the partner device.
- Fast access to large data packets with the modes SEND and RECEIVE.
- Alternate access within a data packet.
- Display and monitoring of the connection state

2.9.1.1 Powerful SIMATIC NET OPC server for the SR protocol

Introduction

The section below describes a configuration variant for the SR protocol that meets requirements for higher performance. To use this variant, all underlying SR protocol libraries and the COM server as inproc server are loaded on the outproc OPC server. The protocol is handled in the process of the OPC server, additional execution times for changing between processes and multiprotocol mode are avoided. The process change between the OPC client and OPC server is, however, still necessary.

Configuration

This high-speed variant is enabled implicitly by selecting the "SR" protocol as the only protocol in the "Communication Settings" configuration program (if other protocols or the OPC UA interface are selected, the performance advantage described is lost):

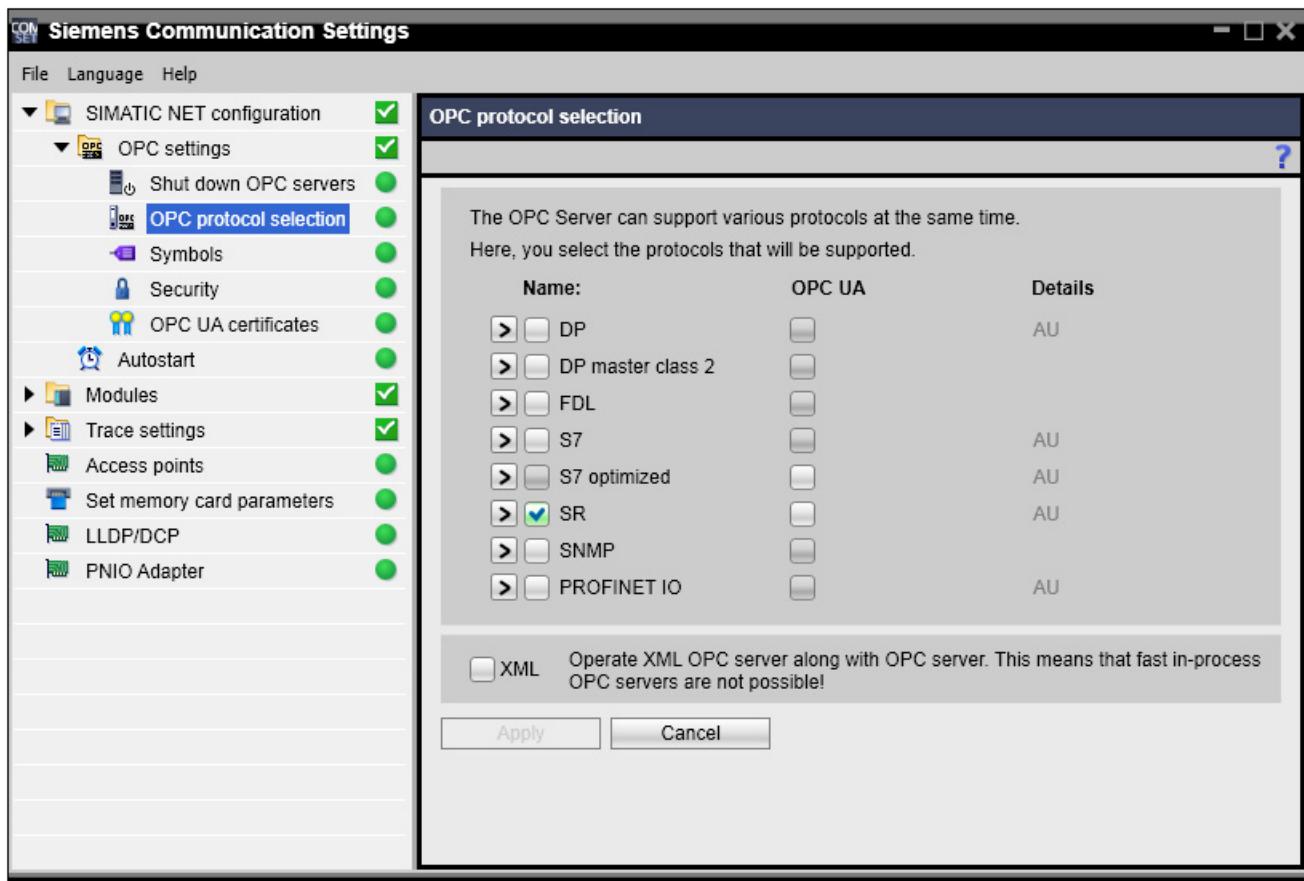


Figure 2-55 Window in the "Communication Settings" configuration program for selecting the SR protocol

"Symbols" can also be selected.

Note

When creating symbols with the symbol editor, the use of the following characters is permitted: A-Z, a-z, 0-9, _, -, ^, !, #, \$, %, &, ', /, (,), <, >, =, ?, ~, +, *, ',', :, |, @, [,], {, }, ". When creating symbols with STEP 7, you should also remember that problems can arise when resolving the array if your symbol file includes symbols in the form <symbolname> and <symbolname>[<index>] at the same time.

Advantages / disadvantages

Using the powerful SR OPC server does, however, have the disadvantage that only the SR single protocol mode is possible. On the other hand, it does have the following advantages:

- Higher performance than with multiprotocol mode.
- Simple configuration.
- Several clients can use the server at the same time.
- The stability of the OPC server does not depend on the clients.

2.9.1.2 Protocol ID

The protocol ID for the SEND/RECEIVE protocol is "SR".

2.9.1.3 Connection names

The connection name is the name configured in STEP 7 to identify the connection.

The following connection types are supported by the OPC server:

- ISO transport connection
- ISOOnTCP connection
- TCP connection

Examples of connection names

A typical example is:

SR_Connection

2.9.1.4 Variable services

Variable services allow direct access to and monitoring of variables on the programmable controller. The variables are addressed symbolically, the notation of the variable names is oriented on the programming tools. For direct read access, the OPC server transfers the required address information to the receiver and this sends the requested data back. During write access, the OPC server transfers the address information along with the value to be written.

Note

The requested variable must be consistent with the configuration of the partner device. Otherwise, the access to certain areas leads to communication errors. The OPC server can only check the syntax when a variable registers. Based on the configuration of the partner, it is not possible to determine whether the variable is valid on the partner device.

Syntax of the process variables for SEND/RECEIVE variable services

SR: [<connectionname>]<area>{,}<type><address>{,}<quantity>}

Explanations

SR

SEND/RECEIVE protocol for access to the process variable.

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

<area>

Object to be addressed.

DB nn	Data block no. nn
Q	Output
I	Input
M	Bit memory
P	Periphery (for S5 CPUs)
PII	Periphery (for S7 CPUs)
RS	System area
IA	Absolute start address
DX nn	Extended data block
DE nn	Data block in external memory
QB	Extended I/O

<type>

Data type

The data type (format identifier) is converted to the corresponding OLE data type on the OPC server.

Format identifier	Description	OLE data type	Visual Basic type
X	Bit (Boolean). You specify the bit number (0 to 7).	VT_BOOL	Boolean
	Note: Only with the objects I, Q, M, P, PII, DB and QB (read access only)		
B or BYTE	Byte (unsigned8)	VT_UI1	Byte
	Note: Only available for the objects I, Q, M, P, PII, and QB		
CHAR	Byte (signed8)	VT_I1	Integer
	Note: Only available for the objects I, Q, M, P, PII, and QB		
W or WORD	Word (unsigned16)	VT_UI2	Long
INT	Word (signed16)	VT_I2	Integer
D or DWORD	Double word (unsigned 32)	VT_UI4	Double

Format identifier	Description	OLE data type	Visual Basic type
DINT	Double word (signed 32)	VT_I4	Long
REAL	Floatingpoint, IEEE representation	VT_R4	Single
S5REAL	Floatingpoint, S5 representation	VT_R4	Single

<address>

Address of the variables in the area.

Depending on the area, the address specified must be a byte or a word address. When accessing the following areas, the specified address is interpreted as a word address: DB nn

Word addresses:

- DX nn
- DE nn

For the other areas, the address is a byte address. For data type X, the syntax of the <address> is = <byte-offset.bit>

The range of values of bit is 0 to 7.

The value range for the byte offset of byte addresses is 0 to 65534 and of word addresses 0 to 32767. The actual usable value of the address may be lower depending on the device and type.

Word and byte addresses are addressed as follows:

- Word addresses: <wordnumber>{.bitnumber}
- Byte addresses: <bytelenumber>{.bitnumber}

<quantity>

Number of variables of a type addressed starting at the address specified in the *address* parameter (range of values 0 to 65535).

For data type X, the entry of a quantity for

- Write access (WRITE): only multiples of 8 starting at bit number 0.
- Read access (FETCH): any number starting at any offset.

Example:

SR:[ISO-WRITE-1]DB1,X10.0,16, access rights W, offset 10, starting at bit number 0, quantity 16 bits

SR:[ISO-FETCH-1]DB1,X10.4,9, access rights R, offset 10, starting at bit number 4, quantity 9 bits

Syntax for timers and counters

SR: [<connectionname>]<number>

- T nn
Timer
- C nn
Counter

2.9.1.5 Bufferoriented services

Buffer-oriented services allow program-controlled transfer of larger blocks of data. These services are also known as SEND/RECEIVE services. Data transfer with the OPC server is implemented with variables:

- Variables that send blocks of data
- Variables that receive blocks of data

A default size for the blocks of data is specified in the configuration, when sending variables, the length can be restricted. Alternate access within the blocks of data is possible.

Fixed variable names

The following fixed variable names are specified for each connection:

- *RECEIVE*
- *SEND*

Syntax of the process variables for buffer-oriented services

The available options are as follows:

```
SR: [<connectionname>]receive{,<type><address>{,<quantity>}}  
SR: [<connectionname>]send{<n>} {,<type><address>{,<quantity>}}
```

As of SIMATIC NET PC Software Edition 2006, there is also a second option for receiving data. When an item is activated, data is signaled even if it has not changed:

```
SR: [<connectionname>]receivedata{,<type><address>{,<quantity>}}
```

This item does not conform to the OPC specification.

Explanations

SR

SEND/RECEIVE protocol for access to the process variable.

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

receive

Last data buffer received from partner.

The structure of the data buffer is not fixed. For this reason, the buffer is always delivered as an array of bytes.

OLE data type	Visual Basic type
VT_ARRAY VT_UI1	Byte

Note

The RECEIVE variable corresponds to a receive buffer. The variable can therefore only be read. With read access to the DEVICE, a receive buffer is prepared explicitly in the communications system. If no data field is received for this buffer within a specified time, a timeout is reported. You should therefore only monitor these variables or read them from the cache.

receivedata

The "receivedata" item does not conform to OPC.

Functionally, "receivedata" corresponds to the "receive" item described above. However, there is one difference as follows:

When an item of this type is activated, data is signaled even if its content has not changed. This allows a client to receive unchanged data buffers from the partner. The OnDataChange callback is also not faster than the negotiated update rate. For this function, you should therefore always set faster update rates than the send rate of the SEND/RECEIVE data.

send

A send buffer that can be transferred to the connection partner.

A default value for the size of the send buffer is specified during configuration. The buffer is always delivered as an array of bytes.

Write access to these variables causes the send buffer to be transferred to the partner.

OLE data type	Visual Basic type
VT_ARRAY VT_UI1	Byte

Note

This variable and variables derived from it must not be read and not activated. Read access to these variables may well result in the partner terminating the connection.

Note

With acknowledged writing to send variables, a result "S_OK" only means that the send buffer was transferred successfully to the communications system and is ready to be fetched by the receive partner. It does not indicate whether the buffer was actually accepted and processed by a partner application.

<n>

Size of the send buffer in bytes.

You can use *n* if send buffers of different sizes are used on a connection. If you omit *n*, the buffer size entered in the configuration is used.

If the miniprotoocol is deactivated in TCP/IP native, (see SIMATIC STEP 7 or SIMATIC NCM PC configuration tool), it is not possible to specify the size of the send buffer. In this case, the buffer size entered in the defined configuration is used.

The size of the send buffer depends on the configuration. Note that the <quantity> multiplied by the size of the <type> in bytes of variables must not be selected higher than the send buffer <n>.

<type>

Data type

The data type (format identifier) is converted to the corresponding OLE data type on the OPC server.

Format identifier	Description	OLE data type	Visual Basic type
X	Bit (Boolean)	VT_BOOL	Boolean
B or BYTE	Byte (unsigned8)	VT_UI1	Byte
CHAR	Byte (signed8)	VT_I1	Integer
W or WORD	Word (unsigned16)	VT_UI2	Long
INT	Word (signed16)	VT_I2	Integer
D or DWORD	Double word (unsigned 32)	VT_UI4	Double
DINT	Double word (signed 32)	VT_I4	Long
REAL	Floatingpoint, IEEE representation	VT_R4	Single
S5REAL	Floatingpoint, S5 representation	VT_R4	Single

<address>

Byte address of the variables in the area *byte number*.

For the *bit* data type, the requested bit is addressed by *byte number.bit number*.

<quantity>

Number of variables of a type to be addressed starting at the address specified in the *address* parameter.

Note

Please note the following:

- By specifying the optional information type, address and quantity, you can define structured access to subareas of blocks of data.
- Variables for send data or receive data of different length or different connection name have independent memory areas.
- The send data buffer is allocated and initialized with zero when an item is created for an independent memory area. A write to a send item is written to the internal write buffer and transferred.
- With acknowledged writing to send variables, a result "S_OK" only means that the send buffer was transferred successfully to the communications system and is ready to be fetched by the receive partner. It does not indicate whether the buffer was actually accepted and processed by a partner application.
- The blocks of data are transferred acyclically. The complete block of send data is always transferred. This also applies to subelement access or when several clients write to this item at the same time.

Examples of process variables for buffer-oriented services

Here you will find examples illustrating the syntax of variable names for buffer-oriented services.

Receive variables

SR:[MyConnection]receive,w4,6
receive,w4,6
6 data words starting at offset 4 in the receive buffer.

SR:[MyConnection]receive,dword7
receive,dword7
A double word starting at offset 7 in the receive buffer

SR:[MyConnection]receive,REAL0,2
receive,REAL0,2
An array with 2 floating-point values starting at offset 0 in the receive buffer.

Example of transferring data that has not necessarily changed (see "receivedata"):

SR:[MyConnection]receivedata,w4,6
receivedata,w4,6
6 data words starting at offset 4 in the receive buffer.

Send variables

SR:[MyConnection]send30,dword7

send30,dword7

A double word starting at byte 7 in a 30byte long send buffer. If the default for the size of the send buffer is not 30, the variable accesses a separate buffer.

SR:[MyConnection]send,B20,6

send,B20,6

An array with 20 bytes starting at offset 6 in a send buffer with default size. The default size of the send buffer is specified in the configuration.

SR:[MyConnection]send8,DINT0

send8,DINT0

A signed double word starting at address 0 in a send buffer with the size 8.

2.9.1.6 SEND/RECEIVEspecific information variables

With the SEND/RECEIVEspecific information variables, you can query information about the connection state.

The following information can be obtained:

Status of a connection

Syntax of information variables for open communications services (SEND/RECEIVE)

SR: [<connectionname>]&<informationparameter>()

Explanations

SR

SEND/RECEIVE protocol for access to the process variable.

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

<informationparameter>

The available options are as follows:

- statepath
- State of a communication connection to a partner device.

The result is represented as a string.

Return values:

DOWN

Connection is not established

UP

Connection is established

RECOVERY

Connection being established

ESTABLISH

Reserved for future expansions

OLE data type	Visual Basic type
VT_BSTR	String

statepathval

State of a communication connection to a partner device.

The result is represented as a number.

Return values:

1

Connection is not established

2

Connection is established

3

Connection being established

4

Reserved for future expansions

OLE data type	Visual Basic type
VT_UI1	Byte

Examples of SEND/RECEIVE specific information variables

Here, you will find several examples of return values of information variables for open communication (SEND/RECEIVE):

*SR:[SR_CONNECTION]&statepath()***&statepath()**

can, for example, return the following values:

UP

Connection is established

*SR:[SR_CONNECTION]&statepathval()***&statepathval()**

can, for example, return the following values:

2

The connection is established.

2.9.1.7 Syntax of the systemspecific information variables*SR:[SYSTEM]&version()*

&version()

Returns a version ID for the SR OPC Server, here, for example, the string
SIMATIC NET Core Server SR V 7.xxxx.yyyy.zzzz Copyright 2012

Data type: VT_BSTR

Access right: readonly

S7:[SYSTEM]&sapiversion()

&winsockversion()

Version ID of the Winsocket interface.

Data type: VT_BSTR

Access right: readonly

2.9.2 Open communication services (SEND/ RECEIVE) over PROFIBUS

The ISO/OSI reference model

The open communications service (SEND/RECEIVE) over PROFIBUS is a simple, frame-oriented protocol for SIMATIC devices. It uses the services of the Fieldbus Data Link (FDL) of layer 2 of the ISO/OSI reference model in PROFIBUS.

Properties of open communication services (SEND/ RECEIVE) over PROFIBUS

The OPC server of SIMATIC NET has the following characteristics:

- Communication using SIMATIC S5 and SIMATIC S7 handling blocks on a programmable controller
- Link between two PC stations possible
- Fast transfer of data packets
- Structured access within a data packet
- Display and monitoring of the connection state

2.9.2.1 Protocol ID

The protocol ID for the FDL protocol is FDL.

2.9.2.2 Connection names

The connection name is the name configured in STEP 7 to identify the connection. Select a unique FDL connection name for the OPC server.

Examples of connection names

A typical name might be:

*FDL_Connection***2.9.2.3 Bufferoriented services**

Bufferoriented services allow programcontrolled transfer of blocks of data. The received data buffers and the data buffers to be sent are mapped on OPC variables.

Fixed variable names

The following fixed variable names are specified for each connection:

- *RECEIVE*
- *SEND*
- *SendSDA*
- *SendSDN*

Syntax of the process variables for buffer-oriented services (FDL)

You have two options:

```
FDL:[<connectionname>] receive{,<type><address>{,<quantity>}}  
FDL:[<connectionname>] send{<n>} {,<type><address>{,<quantity>}}
```

Explanations:**FDL**

FDL protocol for access to the process variable.

<connectionname>

Protocolspecific connection name. The connection name is specified in the configuration.

receive

Last data buffer received from partner.

The structure of the data buffer is not fixed. For this reason, the buffer is always delivered as an array of bytes.

Note

The RECEIVE variable corresponds to a receive buffer. The variable can therefore only be read. With read access to the DEVICE, a receive buffer is prepared explicitly in the communications system. If no data field is received for this buffer within a specified time, a timeout is reported. You should therefore only monitor these variables or read them from the cache.

OLE data type	Visual Basic type
VT_ARRAY VT_UI1	Array with elements of the type byte

send

Send buffer that is always present and can be transferred to the connection partner. A default value for the size of the send buffer is specified during configuration. The buffer is always delivered as an array of bytes.

Write access to these and derived variables causes the send buffer to be transferred to the partner.

This variable and variables derived from it must not be read and not activated.

Depending on the combination of the station address and the SAPs specified in the connection configuration, a special FDL service is used when writing a send item:

Local SAP	Remote station	Remote SAP	Meaning / Service Used with Send
0...62, 255	0...126	0...62, 255	Sending and Receiving / SDA
0...62, 255	0...126	63	Send only / SDA
63	0...126	0...62, 255	Receive only (no send)
0...62, 255	127	63	Broadcast: Only send to all / SDN

Local SAP	Remote station	Remote SAP	Meaning / Service Used with Send
63	127	0...62, 255	Broadcast Receive (no send)
0..62, 255	127	0...62, 255	Multicast: Only send to all those that activated the remote SAP / SDN

OLE data type	Visual Basic type
VT_ARRAY VT_UI1	Array with elements of the type byte

Note

With acknowledged writing to send variables, a result "S_OK" only means that the send buffer was transferred successfully to the communications system and is ready to be fetched by the receive partner. It does not indicate whether the buffer was actually accepted and processed by a partner application.

<n>

Size of the send buffer.

With buffer-oriented services, you can send and receive data buffers. The received data buffers and the data buffers to be sent are mapped on OPC variables. A suitably large array can, for example, contain the entire receive buffer. It is also possible to assign subsections of the buffers to individual variables.

The blocks of data are transferred acyclically. A send job is written to an internal write buffer initialized with "zero". The complete block of data from the write buffer is always transferred. This also applies to alternate access or when several clients write to this item at the same time.

<type>

Data type

The data type (format identifier) is converted to the corresponding OLE data type on the OPC server.

Format identifier	Description	OLE data type	Visual Basic type
X	Bit (Boolean)	VT_BOOL	Boolean
B or BYTE	Byte (unsigned8)	VT_UI1	Byte
CHAR	Byte (signed8)	VT_I1	Integer
W or WORD	Word (unsigned16)	VT_UI2	Long
INT	Word (signed16)	VT_I2	Integer
D or DWORD	Double word (unsigned 32)	VT_UI4	Double
DINT	Double word (signed 32)	VT_I4	Long
REAL	Floatingpoint, IEEE representation	VT_R4	Single
S5REAL	Floatingpoint, S5 representation	VT_R4	Single

<address>

Byte address of the variables in the area *byte number*. The range of values depends on the configuration.

For the *bit* data type, the requested bit is addressed by *byte number.bit number*.

<bytetenumber>{.<bitnumber>}

If you enter nothing for <bitnumber>, bit is addressed with 0.

<quantity>

Number of variables of a type to be addressed starting at the address specified in the *address* parameter. The value range depends on the configuration.

Syntax of the process variables for bufferoriented services using SDA and SDN (FDL)

It can be an advantage when the service used is not dependent on the combination of the address of the partner station and the SAPs.

By using the names "SendSDA" and "SendSDN", you can specify that only the relevant services "SDA" or "SDN" are used for sending. These special names are not displayed in the OPC Browser.

Syntax:

You have two options:

```
FDL: [<connectionname>] SendSDA{<n>} {,<type><address>{,<quantity>}}
```

```
FDL: [<connectionname>] SendSDN{<n>} {,<type><address>{,<quantity>}}
```

Explanations

FDL

FDL protocol for access to the process variable.

<connectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

SendSDA

SendSDN

Only the services SDA or SDN are used to send.

<n>

Size of the send buffer.

With buffer-oriented services, you can send and receive data buffers. The received data buffers and the data buffers to be sent are mapped on OPC variables. A suitably large array can, for example, contain the entire receive buffer. It is also possible to assign subsections of the buffers to individual variables.

<type>

Data type

The data type (format identifier) is converted to the corresponding OLE data type on the OPC server.

Format identifier	Description	OLE data type	Visual Basic type
X	Bit (Boolean)	VT_BOOL	Boolean
B or BYTE	Byte (unsigned8)	VT_UI1	Byte
CHAR	Byte (signed8)	VT_I1	Integer
W or WORD	Word (unsigned16)	VT_UI2	Long
INT	Word (signed16)	VT_I2	Integer
D or DWORD	Double word (unsigned 32)	VT_UI4	Double
DINT	Double word (signed 32)	VT_I4	Long
REAL	Floatingpoint, IEEE representation	VT_R4	Single
S5REAL	Floatingpoint, S5 representation	VT_R4	Single

<address>

Byte address of the variables in the area *byte number*. The range of values depends on the configuration.

For the *bit* data type, the requested bit is addressed by *byte number.bit number*.

<quantity>

Number of variables of a type to be addressed starting at the address specified in the *address* parameter. The value range depends on the configuration.

Note

Please note the following:

- By specifying the optional information type, address and quantity, you can define structured access to subareas of blocks of data.
- Variables for send data or receive data of different length or different connection name have independent memory areas.
- The send data buffer is allocated and initialized with zero when an item is created for an independent memory area. A write to a send item is written to the internal write buffer and transferred.
- The blocks of data are transferred acyclically. Simultaneous network jobs for the same data are possible. The complete block of send data is always transferred. This also applies to subelement access or when several clients write to this item at the same time. Writing simultaneously to the same block of send data or subareas of the block of send data by several clients can lead to inconsistencies. We therefore recommend:
 - that you always read or write the complete block of data.
 - that you set the Maximum number of parallel network jobs to one although this reduces the transmission performance.

Examples of process variables for bufferoriented services (FDL)

Here you will find examples illustrating the syntax of variable names for FDL variables.

Read variable

FDL:[MyConnection]receive,w0,6

6 data words starting at byte 0 in the receive buffer.

FDL:[MyConnection]receive,dword7

One double word starting at byte 7.

FDL:[MyConnection]Receive,REAL0,2

Two real values starting at byte 0 in the receive buffer

Write variable

FDL:[MyConnection]send30,dword7

One double word starting at byte 7 in a 30byte long send buffer.

FDL:[MyConnection]SendSDN,B5,20

An array with 20 bytes starting at offset 5 in a send buffer with default size. The default size is set during configuration. Regardless of which SAP is specified in the configuration, the PROFIBUS FDL SDN service is used.

FDL:[MyConnection]Send8,DINT0

A signed double word from starting at address 0 in a send buffer with the size 8 bytes.

2.9.2.4 FDL-specific information variables

The OPC server for open communications services (SEND/RECEIVE) over PROFIBUS (FDL) provides variables with which information can be queried about the status and extent of the communication network.

Syntax of the FDL-specific information variables

```
FDL: [ | | <CPname> ] &<informationparameter>()
```

Explanations

FDL

FDL protocol for access to the process variable.

<CPname>

The CP name is specified during configuration. Only those CP names are supported over which an FDL connection is configured.

<informationparameter>

The five options are as follows:

- busparameter
- defaultsap
- identify
- ts
- lifelist

These five options are described below.

busparameter

Gets the bus parameters of the PROFIBUS network operated on the specified connection. The values are returned as a byte array and correspond to the results returned by the FDL service FDL_READ_VALUE. For more detailed information, refer to the manual on the FDL programming interface.

Content	Description	Data type	Length (bytes)
HSA	Highest PROFIBUS address on the bus, 2...126	BYTE	1
TS	PROFIBUS address of the local station, 0...hsa or 126.	BYTE	1
Station_Type	Type of local station	Integer	2
Transmission rate	Transmission rate	Integer	2
Medium_red	Redundancy	Integer	2
Retry_Ctr	Number of repeated calls to nonresponding station (remote), 0...7.	UWORD	2
Default_SAP	Number of the default SAP of the station (local), 0...63	BYTE	1
network _con_SAP	reserved	BYTE	1

Content	Description	Data type	Length (bytes)
TSL	SLOT time	UWORD	2
TQUI	Quiet time for modulator / repeater switchover time	UWORD	2
TSET	Setup time	UWORD	2
MIN_TSDR	Minimum station delay responder	UWORD	2
MAX_TSDR	Maximum station delay responder	UWORD	2
TTR	Target rotation time	DWORD	4
GAP	GAP update factor	BYTE	1
in_Ring_desired	Request to enter ring	BOOLEAN	1
physical_layer	Selectable physical bus characteristics	Integer	2
ident	Vendor name, controller type, hardware and software versions	String	211

OLE data type	Visual Basic type
VT_ARRAY VT_UI1	Byte()

defaultsap

Returns the value for the default SAP (SAP = Service Access Point). Whenever an SAP is not explicitly specified, the default SAP is used for addressing.

OLE data type	Visual Basic type
VT_UI1	Byte

identify

Gets the station identifier of the specified connection as an array with four strings:

Elements of the return value:

Vendor

Controller

Hardware version

Software version

OLE data type	Visual Basic type
VT_ARRAY VT_BSTR	String()

ts

Returns the local station address of the specified module.

OLE data type	Visual Basic type
VT_UI1	Byte

lifelist

Information on accessible nodes on the bus.

The array with its 127 elements includes information about every possible station address. Each station address is assigned an index of the array.

Return values of the array entries:

FDL_STATION_NON_EXISTENT

No node exists (value 0x10)

FDL_STATION_PASSIVE

Passive node (value 0x00)

FDL_STATION_READY_FOR_RING

Node ready for entry in the token ring of PROFIBUS (value 0x30)

FDL_STATION_ACTIVE

Active node (value 0x20)

OLE data type	Visual Basic type
VT_ARRAY of VT_UI1	Array with 127 elements of the type byte()

Note

This information is obtained using the FDL service *FDL_LIFE_LIST_CREATE_REMOTE*. This service creates a significant load on the bus so that whenever possible this variable should not be monitored actively.

2.9.2.5 Syntax of the systemspecific information variables

FDL:[SYSTEM]&version()

&version()

Returns a version ID for the FDL OPC Server, here, for example, the string

SIMATIC NET Core Server FDL V 7.xxxx.yyyy.zzzz Copyright 2012

Data type: VT_BSTR

Access right: readonly

2.10 Open communication services (SEND/ RECEIVE) with OPC UA via Industrial Ethernet

2.10.1 Properties of SR communication with OPC UA

The SIMATIC NET OPC server allows the use of SR communication via OPC UA.

The SIMATIC NET OPC UA server for the open communications services (SEND / RECEIVE) via Industrial Ethernet is released for communication with S7 devices. It also allows users to communicate with third-party devices.

The SR OPC UA server from SIMATIC NET has the following characteristics:

- Communication using SIMATIC S5 handling blocks and S7 function blocks
- Link between two PC stations using send and receive is possible.
- Support of the WRITE and FETCH function for access to objects of the partner device.
- Fast access to large data packets with the modes SEND and RECEIVE.
- Alternate access within a data packet.
- Display and monitoring of the connection state

2.10.2 SIMATIC NET OPC UA server for the SR protocol

Introduction

The section below describes a configuration variant for the SR protocol that also supports OPC UA. To do this, the SR COM OPC Data Access server is set up as an outproc OPC server.

Configuration

The SR OPC UA server is activated by selecting "SR" and "OPC UA" in the "Communication Settings" configuration program in the "OPC protocol selection" catalog:

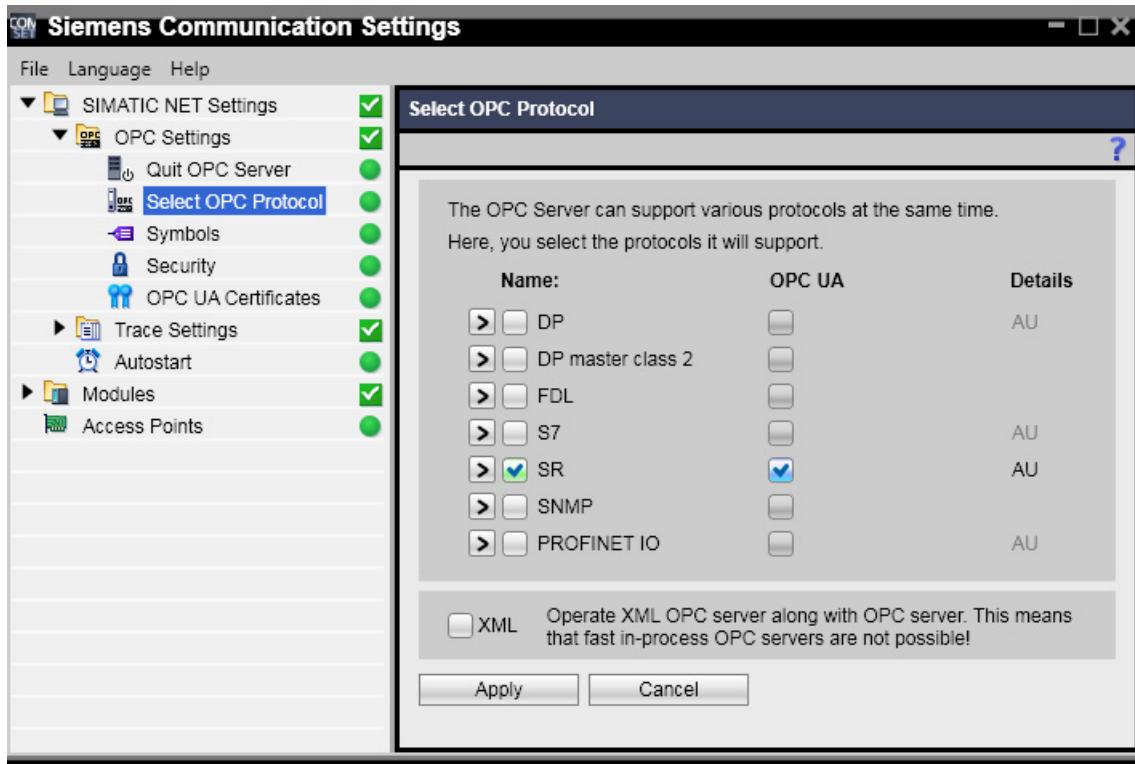


Figure 2-56 Window in the "Communication Settings" configuration program for selecting OPC UA for the SR protocol

Advantages / disadvantages

When using the SR OPC UA server, only the outproc mode of SR is possible. The SR OPC UA server process must be running to maintain readiness to receive. Even after all OPC UA clients have logged off, the SR OPC UA server is not exited. Simultaneous operation of the high-speed Inproc OPC DA SR server is not possible.

On the other hand, it does have the following advantages:

- COM/DCOM configuration is no longer necessary.
- High-speed, secure communication
- Only one server is required for events and data access.

2.10.3 How is the SR OPC UA server addressed?

Server URL

For the TCP protocol, there are two ways for the OPC client to address the server:

- Direct addressing:
 - opc.tcp://<hostname>:55102
 - or
 - opc.tcp://<IP-Adresse>:55102
 - or
 - opc.tcp://localhost:55102
- The URL of the SR OPC UA server can also be found using the OPC UA Discovery service.
To locate the Discovery server enter the following:
 - opc.tcp://<hostname>:4840
 - or
 - opc.tcp://<IP-Adresse>:4840
 - or
 - http://<hostname>:52601/UADiscovery/
 - or
 - http://<IP-Adresse>:52601/UADiscovery/

The Discovery server has port 4840 (for TCP connections) and port 52601 (for HTTP connections).

IPv6 address

An IPv6 address can also be used as the IP address. The address must be in parentheses, for example [fe80:e499:b710:5975:73d8:14]

Endpoints and security modes

The SIMATIC NET SR OPC UA server supports endpoints with the native binary TCP protocol and allows authentication using certificates and encrypted transfer.

The Discovery service on the addressed host signals the endpoints of the servers, in other words, their security requirements and protocol support.

The server URL "opc.tcp://<hostname>:55102" of the SR OPC UA server provides the following endpoints:

- Endpoint in "SignAndEncrypt" security mode:

A signature and encryption are required to communicate with the server. Communication is protected by exchanging certificates and entering a password.

In addition to the security mode, the security policy Basic128Rsa15 is also displayed.

- Endpoint in "None" security mode:

In this mode, no security functions are required by the server (security policy "None").

For more detailed information on the security functions, refer to the section "Programming the OPC UA interface (Page 496)".

The security policies "Basic128Rsa18" and "None" are in the UA specification of the OPC Foundation at the following Internet address:

[> "Specifications" > "Part 7"](http://opcfoundation.org/UA)

You will find more detailed information on the following Internet page:

OPC Foundation ([> "Security Category" > "Facets" > "Security Policy"](http://www.opcfoundation.org/profilereporting/index.htm))

The OPC UA Discovery of the OPC Scout V10

The OPC Scout V10 allows you to open the OPC UA Discovery dialog to enter UA endpoints in the navigation area of the OPC Scout V10.

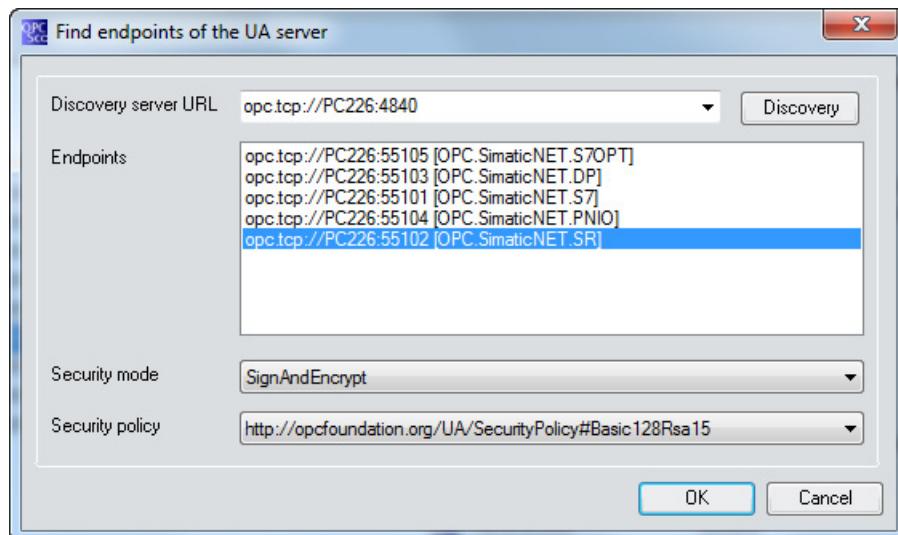


Figure 2-57 The "Find endpoints of the UA server" dialog of OPC Scout V10

The SR OPC UA server can be found using the OPC UA Discovery service. For the entry, refer to "Server URL" above.

OPC Scout V10 contains a list of the OPC UA endpoints. The Discovery service on the addressed host then signals the registered OPC UA servers and their ports and security modes.

For more detailed information, refer to the online help of OPC Scout V10.

2.10.4 Protocol ID

The protocol ID for the OPC UA SEND/RECEIVE protocol in OPC UA is "SR".

2.10.5 Which namespaces does the SR OPC UA server provide?

The SR OPC UA server provides the following namespaces:

Namespace index	"Identifier" (namespace URI) / comment
0	"http://opcfoundation.org/UA/" specified by the OPC Foundation
1	"urn:Siemens.Automation.SimaticNET.SR:(GUID)" Unique identifier of the local high-speed SR OPC UA server.
2	"SRTYPES:" Definitions for SR-specific object types.
3	"SR:" Identifier of the local high-speed SR OPC UA server with new simplified syntax (Browsable and can be used with UA)
4	"SRCOM:" Identifier of the server with the old syntax, SR OPC DA-compatible (can be used with UA but cannot be browsed)

The namespace indexes 0 and 1 are reserved and their significance is specified by the OPC Foundation.

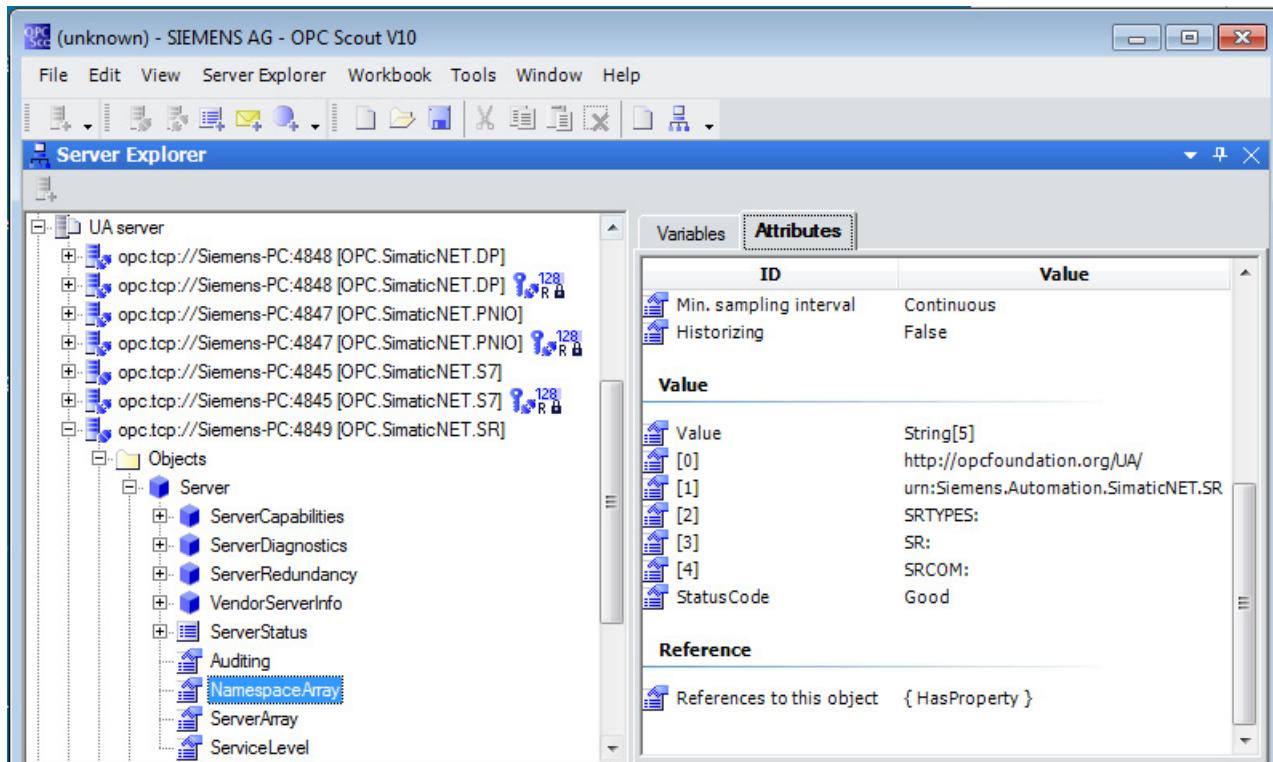


Figure 2-58 Display of the SR OPC UA namespaces using the browse function of the OPC Scout V10

2.10.6 Connection names

The connection name is the name configured in STEP 7 to identify the connection.

The following connection types are supported by the OPC UA server:

- ISO transport connection
- ISOOnTCP connection
- TCP connection

Type of connection

The type of SR access options possible via an SR connection is set in STEP 7. The connection can either be:

- only Fetch,
- only Write or
- only Send/Receive

Examples of connection names

Typical examples are:

- ISOonTCP conn-1
- ISO-on-TCP connection1

2.10.7 The Nodeld

Identification of an SR process variable

The Nodeld identifies an SR process variable with the aid of the following tuple:

- Namespace index
- Identifier (character string, numeric value)

Examples

- Nodeld:
 - Namespace URI:
SR:
(= namespace index 3) for Siemens.Automation.SimaticNET.SR
 - Identifier:
SRconnectionname.send,100.0,b,100
- Nodeld:
 - Namespace URI:
SRCOM:
(= namespace index 4) for OPC.SimaticNET; the syntax is SR OPC DA-compatible
 - Identifier:
SR:[SRconnectionname]send,b0,100

How does the new namespace adapted to OPC UA behave?

The OPC UA view relates to automation objects is also to various properties of the objects. OPC UA no longer access items alone, but also objects and their subobjects.

- Data variables are, for example, subobjects of an SR connection object. Attributes and properties define the objects in greater detail.
- An OPC Data Access item for block access is the closest to an OPC UA data variable.

The qualified identifiers of the Nodelds have a greater significance in OPC UA than in OPC Data Access. Each individual access to an object, subobject, property and attribute uses its Nodeld.

OPC UA provides the display name among other things to support local languages. This means that the same objects, for example in different language environments specified by the OPC UA client, can be browsed differently although the same Nodeld is presented every

time. Selection of the display name is analogous to the relevant Nodeld. The texts of the entire namespace are in English.

Syntax of the SR OPC UA data objects

Normally, the Nodelds of the OPC UA objects on the SR UA server have the following structure:

<connectionobject>."<subobject>".<property>

A subobject can contain further subobjects.

A Nodeld that cannot be interpreted is rejected with an error. The letters "A-Z" are not case-sensitive for any items.

Symbolic object representation

The OPC UA specification recommends a uniform symbolic representation for the hierarchical description of the address space. The following symbols are used in this document:

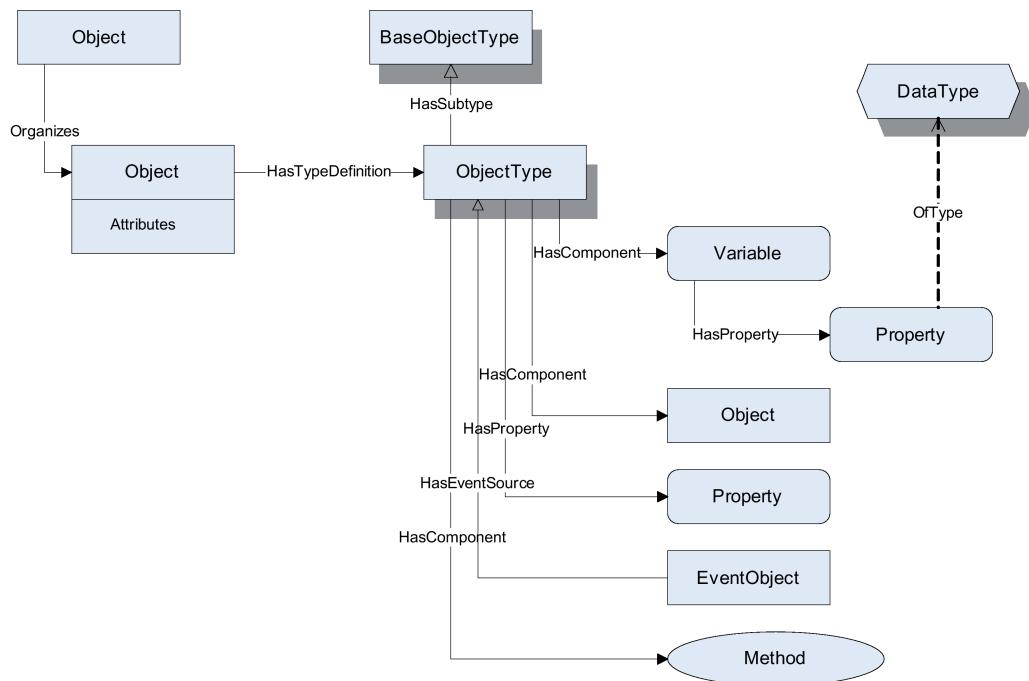


Figure 2-59 Symbols of the OPC UA address space

2.10.8 Data variables for S5 data blocks and areas (S5-compatible communication)

The reading and writing of data variables for S5 data blocks and areas (S5-compatible communication) requires a Fetch or Write connection to be configured.

The data variables on a Fetch connection are only read. On a Write connection, the data variables are only written. If data blocks of a communications partner need to be both read and written, two connections need to be configured, however these are managed completely independently by the OPC UA server.

Syntax of the data variables for S5 data blocks and areas

Simplified syntax of the process variables of the SR OPC UA Node ID for reading and writing data variables:

Namespace URI: SR: (Namespace index: 3)

Syntax

You have two options:

```
<FETCHconnectionname>.<datavariable>.{<offset>{,<SRtype>{,<quantity>}}}
```

```
<WRITEconnectionname>.<datavariable>.{<offset>{,<SRtype>{,<quantity>}}}
```

Explanations

<FETCHconnectionname> and <WRITEconnectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

<datavariable>

Data variable for S5-compatible communication.

Data variable		Note
DB<db>	unsigned8	Data block
DX<dx>	unsigned8	Extended data block
DE<de>	unsigned8	Data block in external memory
I		Input
Q		Output
P		Input and output for peripheral I/O
QB		Extended I/O
M		Bit memory
C		Counter
T		Timer

- Information about the areas of the data variables "db", "dx" and "de"

Following the "DB" or "DX" and "DE" identifiers, the block number also needs to be specified, for example DB10; the protocol-specific addressing, however, only supports block numbers from 0-255 (8 bits). In these areas, the protocol only allows the addressing of entire words (16 bits). The default data type of these areas is therefore "w".

- On Fetch connections, you can also address areas that do not correspond to entire words. This means, for example, that you can address bytes or bits (including arrays) as well as 16- and 32-bit data types with an odd byte offset. The OPC UA server

nevertheless always requests entire words from the communications partner and extracts the relevant data from these.

- On Write connections, for reasons of consistency, you can only address areas that correspond to entire words. This means that you need to specify even and whole number block offsets and only areas that correspond to a multiple of 16 bits (whole number). With S7 communications partners, you can also specify odd block offsets.
Examples:

Example of the syntax	FETCH	WRITE
SRconnectionname.db10.1,b	readonly	not available
SRconnectionname.db10.4, b,8	readonly	write-only
SRconnectionname.db10.3,x1	readonly	not available
SRconnectionname.db10.5,x4,6	readonly	not available
SRconnectionname.db10.6,x0,32	readonly	write-only
SRconnectionname.db10.7,w	readonly	write-only
SRconnectionname.db10.8,w	readonly	write-only

- **Information about the areas of the data variables "m", "i", "q", "p" and "qb"**

In these areas, the protocol only allows addressing of entire bytes (8 bits). For most data types there is therefore no restriction, exception: Bits.

- On Fetch connections, you can also address areas that do not correspond to entire bytes. This means, in particular, that you can address bits (including arrays). The OPC UA server nevertheless always requests entire bytes from the communications partner and extracts the relevant data from these.
- On Write connections, for reasons of consistency, you can only address areas that correspond to entire bytes. This means that you need to specify even and whole number block offsets for bits and only areas that correspond to a multiple of 8 bits (whole number).

Examples:

Example of the syntax	FETCH	WRITE
SRconnectionname.m.1,b	readonly	write-only
SRconnectionname.m.4,b,7	readonly	write-only
SRconnectionname.m.3,x1	readonly	not available
SRconnectionname.m.5,x4,6	readonly	not available
SRconnectionname.m.6,x0,32	readonly	write-only
SRconnectionname.m.7,w	readonly	write-only
SRconnectionname.m.8,w	readonly	write-only

- **Information about the areas of the data variables "c"**

The address information consists of the counter number.

- **Information about the areas of the data variables "t"**

The address information consists of the timer number.

<offset>

Byte address in the data record for the element to be addressed. With OPC UA, this offset is always a byte offset.

<SRtype>

Data type.

The data type is converted to the corresponding OPC UA data type on the OPC UA server.

Table 2- 7 Data type description

Data type	OPC UA data type	Note S7 data type
x<bitaddress>	Boolean	Bit (bool) In addition to the byte offset in the area, the <bitaddress> in the relevant byte must be specified. Range of values 0 to 7
b	Byte Byte string	Byte (unsigned) Used as the default if no <SRtype> is specified. OPC UA does not know "Byte[]", but uses the scalar data type "ByteString" for this.
w	UInt16	Word (unsigned)
dw	UInt32	Double word (unsigned)
c	SByte	Byte (signed)
i	Int16	Word (signed)
di	Int32	Double word (signed)
r	Float	Floating point (4 bytes)
s5r	Float	S5-coded real
c	UInt16	Only in counter area
t	UInt16	Only in timer area

<quantity>

Number of elements. The data type of the variable is an array with elements of the specified format. Specifying a number of array elements causes an array of the corresponding type to be formed even when only a single array element is addressed.

Examples of data variables for S5 data blocks and areas

- **FETCHconnectionname.db10.10,w**
identifies a data word of data block 10 starting at byte address 10.
- **WRITEconnectionname.q.3**
identifies an output byte starting at byte address 3 (BYTE default).
- **FETCHconnectionname.i.0,x0**
identifies a input bit starting at byte address 0, bit 0.
- **WRITEconnectionname.m.3,x4,16**
identifies an array of memory bits starting at byte address 3, bit 4 (read-only)
- **FETCH/WRITEconnectionname.t.22**
identifies timer 22 (default)

2.10.9 Buffer-oriented services

Buffer-oriented services allow program-controlled transfer of larger blocks of data. These services are also known as SEND/RECEIVE services. Data transfer with the OPC UA server is implemented with variables:

- Variables that send blocks of data
- Variables that receive blocks of data

A default size for the blocks of data is specified in the configuration, when sending variables, the length can be restricted. Alternate access within the blocks of data is possible.

Fixed variable names

The following fixed variable names are specified for each connection:

- *receive*
- *send*

Syntax of the process variables for buffer-oriented services

Simplified syntax of the process variables of the SR OPC UA Node ID for reading and writing data variables:

Namespace URI: SR: (Namespace index: 3)

Syntax

The available options are as follows:

```
<SRconnectionname>.send{,<block>}{.<offset>{,<SRtype>{,<quantity>}}}
<SRconnectionname>.receive{.<offset>{,<SRtype>{,<quantity>}}}
```

Explanations

<SRconnectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

send

A send buffer that can be transferred to the connection partner.

A default value for the size of the send buffer is specified during configuration. The buffer is always delivered as an array of bytes (OPC UA data type "Bytestring").

Write access to these variables causes the send buffer to be transferred to the partner.

Note

This variable and variables derived from it must not be read and not activated. Read access to these variables may well result in the partner terminating the connection.

receive

Last data buffer received from partner.

The structure of the data buffer is not fixed. For this reason, the buffer is always delivered as an array of bytes (OPC UA data type "Bytestring").

Note

If the receipt of blocks of data is read using the OPC UA monitoring service (receive as MonitoredItem), the response to receipt of blocks of data can be set in the configuration of the monitoring service. Assuming that DataChangeTrigger is set to OpcUa_DataChangeTrigger_StatusValueTimestamp in the monitoring service, newly received blocks of data with the same data are also signaled to the OPC UA client. This allows an OPC UA client to receive unchanged data buffers from the partner. The DataChange notification is not faster than the negotiated update rate. For this function, you should therefore always set faster update rates than the send rate of the send/receive data.

Note

The RECEIVE variable corresponds to a receive buffer. The variable can therefore only be read. With read access to this nodeld, a receive buffer is prepared explicitly in the communications system. If no data field is received for this buffer within a specified time, a timeout is reported. You should therefore only monitor these variables (access via UA subscription).

<block>

Size of the send buffer in bytes.

You can use <bl> if send buffers of different sizes are used on a connection. If you omit <bl>, the buffer size entered in the configuration is used.

If the miniprotocol is deactivated in TCP/IP native, (see SIMATIC STEP 7 or SIMATIC NCM PC configuration tool), it is not possible to specify the size of the send buffer. In this case, the buffer size entered in the defined configuration is used.

The size of the send buffer depends on the configuration. Note that the <quantity> multiplied by the size of the <SRtype> in bytes of variables must not be selected higher than the send buffer <bl>.

<SRtype>

A DP data type is converted to the corresponding OPC UA data type on the OPC UA server. The following table lists the type identifier and the corresponding OPC data type in which the variable value can be represented.

Data type	OPC UA data type	Note
x<bitaddress>	Boolean	Bit (bool) In addition to the byte offset in the area, the <bitaddress> in the relevant byte must be specified. Range of values 0 to 7
b	Byte	Byte (unsigned)
	Byte string	Used as the default if no <DPtype> is specified. OPC UA does not know "Byte[]", but uses the scalar data type "ByteString" for this.

Data type	OPC UA data type	Note
char	Byte	Integer
w	UInt16	Word (unsigned)
int	Integer	
dw	UInt32	Double word (unsigned)
di	Int32	Double word (signed)
r	Float	Floating point (4 bytes)
s5r	Floatingpoint, S5 representation	Single

<offset>

Byte address in the data record for the element to be addressed.

<quantity>

Number of elements. The data type of the variable is an array with elements of the specified format. Specifying a number of array elements causes an array of the corresponding type to be formed even when only a single array element is addressed.

Note

Please note the following:

- By specifying the optional information type, address and quantity, you can define structured access to subareas of blocks of data.
 - Variables for send data or receive data of different length or different connection name have independent memory areas.
 - The send data buffer is allocated and initialized with zero when an item is created for an independent memory area. A write to a send item is written to the internal write buffer and transferred.
 - The blocks of data are transferred acyclically. The complete block of send data is always transferred. This also applies to subelement access or when several clients write to this item at the same time.
-

Examples of process variables for buffer-oriented services

Here you will find examples illustrating the syntax of variable names for buffer-oriented services.

Receive variables

- **SRconnectionname.receive.4,w,6**
identifies 6 data words starting at offset 4 in the receive buffer.
- **SRconnectionname.receive.7,dw**
identifies a double word starting at offset 7 in the receive buffer
- **SRconnectionname.receive.0,r,2**
identifies an array with 2 floating-point values starting at offset 0 in the receive buffer.

Send variables

- **SRconnectionname.send,30,7,dw**
identifies a double word starting at byte 7 in a 30 byte long send buffer. If the default for the size of the send buffer is not 30, the variable accesses a separate buffer.
- **SRconnectionname.send,256,6,b,20**
identifies an array with 20 bytes starting at offset 6 in a send buffer with the standard size. The default size of the send buffer is specified in the configuration.
- **SRconnectionname.send,8,0,di**
identifies a signed double word starting at address 0 in a send buffer with size 8.

2.10.10 SR-specific information variables

With the SEND/RECEIVE specific information variables, you can query information about the status of the connection.

Syntax of information variables for open communications services (SEND/RECEIVE)

Simplified syntax of the process variables of the SR OPC UA Node ID for reading and writing SR-specific information variables:

Namespace URI: SR: (Namespace index: 3)

Classic syntax

<SRconnectionname>.statepath.statepath

Explanations

<SRconnectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

statepath.statepath

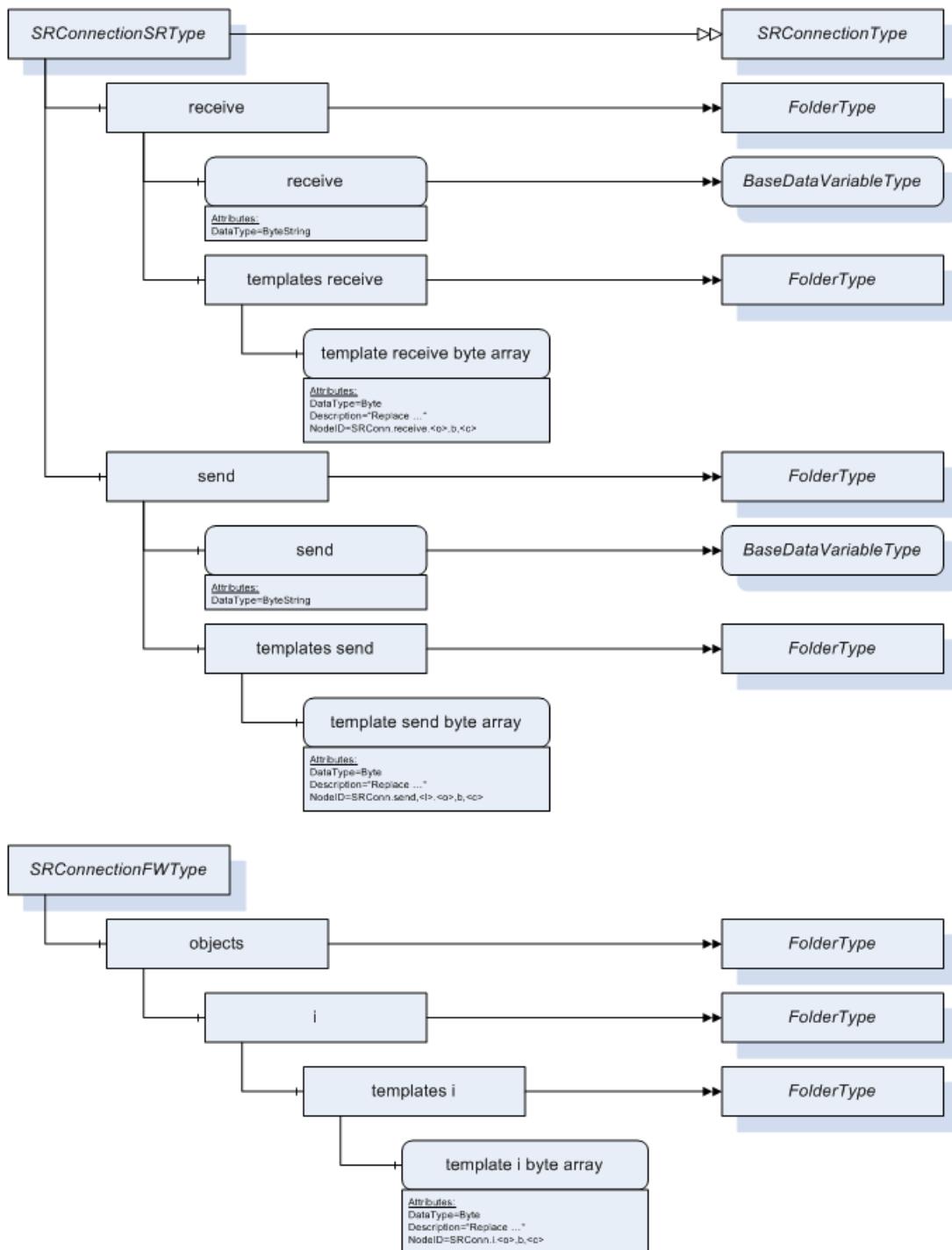
statepath	Status of a communications connection to the partner device The value of this variable is read out as a number and can be assigned to a text by additionally reading the associated Enumstring {UNKNOWN, DOWN, UP, RECOVERY, ESTABLISH}. Variable of the UA type "MultistateDiscreteType", read-only	
	1	DOWN
	2	UP
	3	RECOVERY
	4	ESTABLISH
	0	UNKNOWN

2.10.11 SR OPC UA template data variables

With the process variables for the OPC UA SR protocol, you have flexible setting options with which you can obtain the process data of the plant in the data formats you require.

The wide variety of addressing options cannot, however, be put together in a fully browsable namespace. Even a data block with the length of a single byte has approximately 40 different data format options - starting with Byte, SByte, arrays with one element of these, individual bits, arrays of bits with up to 8 array elements starting at different bit offsets.

The OPC UA server therefore supports the user with template data variables in the SR namespace. In a typical OPC UA client text input box, these templates can be converted to valid ItemIDs simply by changing a few characters.



Note

The usability of OPC UA SR template data variables can be enabled and disabled in the "Communication Settings" configuration program in "OPC protocol selection" > click the arrow symbol beside "SR".

Template data variables within the browse hierarchy

The template data variables are sorted beside the corresponding folders in the namespace representation so that they can be used easily when required.

Syntax of the template data variables

The available options are as follows:

- <SRconnectionname>send,<bl>.<o>,<SRtypetemplate>,<c>
- <SRconnectionname>receive.<o>,<SRtypetemplate>,<c>
- <FETCHconnectionname>.<datavariable>.<o>,<SRtypetemplate>,<c>
- <WRITEconnectionname>.<datavariable>.<o>,<SRtypetemplate>,<c>

Explanations

<SRconnectionname>, <FETCHconnectionname> and <WRITEconnectionname>

Protocol-specific connection name. The connection name is specified in the configuration.

<bl>

Placeholder for the length of the send buffer.

<o>

The placeholder for the byte offset.

<SRtypetemplate>

An SR template data type is converted to the corresponding OPC UA data type on the OPC UA server. The following table lists the data type and the corresponding OPC UA data type in which the variable value can be represented.

Data type	OPC UA data type	Note
x<bitaddress>	Bit (Boolean)	VT_BOOL
b	Byte (unsigned8)	VT_UI1
char	Byte (signed8)	VT_I1
w	Word (unsigned16)	VT_UI2
int	Word (signed16)	VT_I2
dw	Double word (unsigned 32)	VT_UI4
di	Double word (signed 32)	VT_I4

Data type	OPC UA data type	Note
r	Floatingpoint, IEEE representation	VT_R4
s5r	Floatingpoint, S5 representation	VT_R4

<datavariable>

Data variable for S5-compatible communication.

Data variable		Note
db<db>	unsigned8	Data block
dx<dx>	unsigned8	extended data block
de<de>	unsigned8	extended data block
i		Input
q		Output
p		Input and output for peripheral I/O
qb		extended I/O
m		Bit memory
c		Counter
t		Timer

<c>

Placeholder for the number of elements.

Examples:

NodeId	BrowseName	Description
SRconnectionname.send,<bl>.<o>,b	template send byte	<bl> Length of the send buffer <o> Offset
SRconnectionname.receive.<o>,b	template receive byte	<o> Offset
FETCHconnectionname.i.<o>,x<bit>,<c>	template i bit array	<o> Offset <bit> bit offset (0 to 7) <c> Size of the array
WRITEconnectionname.db<db>.<o>,x<bit>,<c>	template db bit array	<db> Number of the data block <o> Offset <bit> bit offset (0 to 7) <c> Size of the array

2.11 SNMP Communication over Industrial Ethernet

Process variables for SNMP

SNMP data are primarily diagnostics variables (with write access also device parameters).

The SNMP OPC Server of SIMATIC NET provides the following variables:

- Process variables
- Information variables
- Trap variables

Further information on the SNMP OPC server:

(<http://www.automation.siemens.com/mcms/industrial-communication/en/ie/software/network-management/snmp-opc-server/Pages/snmp-opc-server.aspx>)

2.11.1 Protocol ID

Protocol ID

The protocol ID for the SNMP protocol is "SNMP".

2.11.2 Data types of the SNMP protocol

Mapping on available data types

In the declaration of the MIB objects, the SNMP protocol uses its own data types. The SNMP OPC Server maps the data types of the SNMP protocol to the available data types on the OPC interface as follows:

SNMP Data Type Code	Canonical OLE Data Type	Visual Basic type	Meaning
ASN_INTEGER 02h	VT_I4	Long	signed long (4 bytes)
ASN_INTEGER32< 02h	VT_I4	Long	signed long (4 bytes)
ASN_UNSIGNED32 47h	VT_UI4	Double	unsigned long (4 bytes)
ASN_OCTETSTRING 04h	VT_BSTR	String	If not a printable character, notation: <i>ostring:xx.xx.xx.xx</i>

SNMP Data Type Code	Canonical OLE Data Type	Visual Basic type	Meaning
ASN_OBJECTIDENTIFIER 06h	VT_BSTR	String	In MIB notation: .a.b.c.d.e.
ASN_IPADDRESS 40h	VT_BSTR	String	In decimal notation: 129.168.0.4
ASN_COUNTER32 41h	VT_UI4	Double	unsigned long (4 bytes)
ASN_GAUGE32 42h	VT_UI4	Double	unsigned long (4 bytes)
ASN_TIMETICKS 43h	VT_UI4	Double	unsigned long (4 bytes)
ASN_OPAQUE 44h	VT_ARRAY of VT_UI1	VT_ARRAY of VT_UI1	unsigned byte

2.11.3 Process variables for SNMP variable services

Variables and device profiles

SNMP variables are the variables on the partner device. These variables are made known to the OPC server based on device profiles. These device profiles can be created with the MIB compiler and the MIB files of the device. Standard device profiles ship with the configuration tool. The defined configuration is then downloaded to the PC with the OPC server so that the variables are available to the OPC server.

Syntax

Note

In contrast to the syntax of the other protocols, the following syntax is used with SNMP.

SNMP: [<devicename>]<objectname>

Explanations

SNMP

SNMP protocol for access to the process variables (MIB objects) and trap variables.

<devicename>

The device name is specified during configuration of the system and is unique.

<objectname>

Symbolic name for the MIB object of the partner device.

Example

SNMP:[OSM]sysName

Queries the MIB object sysName. The device name was configured in the system configuration in this example as OSM.

2.11.4 SNMPspecific information variables

Introduction

The following sections describe the variables provided by the SNMP OPC Server:

- Variables for the communications system and connection statuses
- Variables for the SNMP OPC Server

Information on the SNMP device

The OPC server provides variables with which you can get information on the SNMP device (for example port, switch).

You can query the following information:

- The comment entered during configuration
- The configured IP address of the partner device
- The status of the connection to the partner device
- Additional information for non SNMPcompliant devices

Syntax

SNMP : [<devicename>]<informationparameter>

Explanations

SNMP

SNMP protocol for access to the process variables (MIB objects) and trap variables.

<devicename>

The device name is specified during configuration of the system.

<informationparameter>

The following information parameters are defined:

&description()	The text entered in the comment field during configuration. VT_BSTR, readonly	
&ipaddress()	The IP address of the partner device. VT_BSTR, read only.	
&statepath()	Status of connection to partner device. Value in text form (VT_BSTR, readonly):	
	DOWN	The device cannot be reached with a ping or there are no active read or write jobs.
	UP	Jobs could be executed successfully with this device.
	RECOVERY	The device cannot be reached with a ping or the last job was terminated with an error that suggests a communication abort and there are still active jobs for the device.
&statepathval()	Status of the connection to the partner device. Value as integer. VT_UI1, read-only.	
	1	The device cannot be reached with a ping or there are no active read or write jobs.
	2	Jobs could be executed successfully with this device.
	3	The device cannot be reached with a ping or the last job was terminated with an error that suggests a communication abort and there are still active jobs for the device.
&ping()	Status of connection to partner device. VT_UI1, read-only.	
	0	The device cannot be reached with a ping.
	1	The device can be reached with a ping.
&alarmagentmib2()	Status of connection to SNMP agent partner device (Item sysUpTime from the MIB2 profile can be reached). VT_UI1, read-only.	
	0	"sysUpTime" cannot be read.
	1	"sysUpTime" can be read.

The following information parameters are visible only when you select "No SNMP" during configuration because the device does not support SNMP but only the ping.

&syscontact()	The text entered in the sysContact field during configuration. VT_BSTR, readonly.
&syslocation()	The text entered in the sysLocation field during configuration. VT_BSTR, readonly.
&sysname()	The text entered in the sysName field during configuration. VT_BSTR, readonly.

Example

SNMP:[OSM]&ipaddress()

The configured IP address of the node name with the name OSM is returned.

Information on the SNMP OPC server

The OPC server provides variables with which information about the SIMATIC NET SNMP OPC Server can be queried.

You can query the following information:

- Version of the SIMATIC NET SNMP OPC Server
- Version of the Winsocket
- Information on trap reception

Syntax

```
SNMP:[SYSTEM]&<informationparameter>()
```

Explanations

SNMP

SNMP protocol for access to the information variables for the local system.

SYSTEM

Identifier of the local system; the name is fixed.

<informationparameter>

The following information parameters are defined:

&version()	Version identifier of the SIMATIC NET SNMP OPC Server. VT_BSTR, Readonly.
&winsockversion()	Version identifier of the Winsocket. VT_BSTR, Readonly.
&traplisten()	Indicates whether the SIMATIC NET SNMP OPC Server was able to log on to receive traps. VT_BOOL; Readonly. <i>FALSE</i> The SNMP OPC server could not log on to receive traps. <i>TRUE</i> The SNMP OPC server logged on successfully to receive traps.

Example

```
SNMP:[SYSTEM]&version()
```

The version of the SIMATIC NET SNMP OPC server is returned, for example *SIMATIC NET Core Server SNMP V6.1.1000.2815 Copyright ©SIEMENS AG*

2.11.5 SNMPspecific traps

How the OPC server handles traps

Traps are events sent spontaneously by the device to the OPC SNMP Server. The OPC server processes these events as described below:

- The trap is mapped as a simple event on the Alarms & Events interface.
- For each configured trap, two variables are created on the Data Access interface: One variable is incremented each time the relevant trap occurs, the other variable stores a description of the trap.

Syntax

First variable for the number of the occurrences of the event:

`SNMP:[<devicename>]<trapname>`

Second variable for describing the trap:

`SNMP:[<devicename>]<trapname>_description`

Explanations

SNMP

SNMP protocol for access to the process variables (MIB objects) and trap variables.

<devicename>

The device name is specified during configuration of the system.

<trapname>

Name of the trap.

Example

The first variable returns the number of cold start traps triggered by the partner device:

`SNMP:[OSM]coldStart`

The second variable returns a description of the trap:

`SNMP:[OSM]coldStart_description`

2.12 PROFINET IO communication over Industrial Ethernet

The system model of a PROFINET IO device assigned to the PROFINET IO controller is module-oriented. A PROFINET IO device can contain several modules, each module can contain several submodules. Generally, a submodule contains the physical terminals or driver blocks (channels) for the plant hardware to be connected, for example conveyor belts or sensors whose input data or manipulated variable will be addressed with the IO data. Reading and writing IO data is the most important application for PROFINET IO.

Both devices and modules/submodules can provide data records and can generate interrupts. The use of data records and interrupts is device-dependent.

2.12.1 Powerful SIMATIC NET OPC server for the PROFINET IO protocol

Introduction

This configuration variant for the PROFINET IO protocol meets higher performance requirements. The lower-level PROFINET IO COM server as inproc server is loaded on the outproc OPC server. The protocol is handled in the process of the OPC server; additional execution times for changing between processes and multiprotocol mode are avoided. The process change between the OPC client and OPC server is still necessary.

Configuration

This high-speed variant is enabled implicitly by selecting the "PROFINET IO" protocol as the only protocol in the "Communication Settings" configuration program (if other protocols or the OPC UA interface are selected, the performance advantage described is lost):

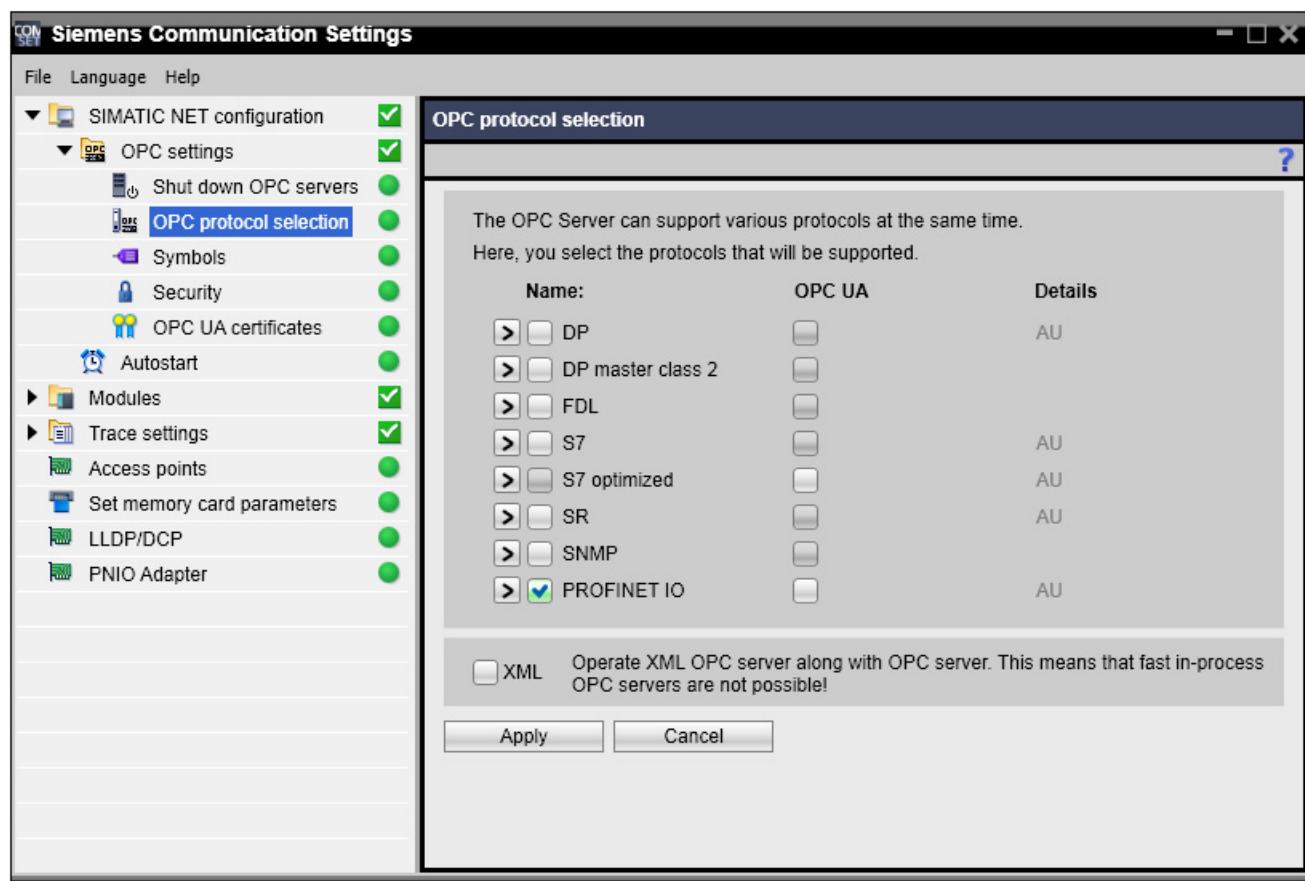


Figure 2-60 Window in the "Communication Settings" configuration program for selecting the PROFINET IO protocol

"Symbols" can also be selected.

Note

When creating symbols with STEP 7 or the symbol editor, the use of the following characters is permitted: A-Z, a-z, 0-9, _, -, ^, !, #, \$, %, &, ', /, (,), <, >, =, ?, ~, +, *, ', :, |, @, [,], {, }, ". When creating symbols with STEP 7, you should also remember that problems can arise when resolving the array if your symbol file includes symbols in the form <symbolname> and <symbolname>[<index>] at the same time.

Advantages/disadvantages

Using the powerful SIMATIC NET OPC Server does, however, have the disadvantage that only the PROFINET IO single protocol mode is possible.

On the other hand, it does have the following advantages:

- Higher performance than with multiprotocol mode
- Simple configuration
- Access via the ProgID "OPC.SimaticNET"
- Several clients can use the server at the same time
- The stability of the OPC server does not depend on the clients.

Note

The configured cycle time of the server is not automatically the time with which the underlying PROFINET IO protocol interface is called. Depending on the update time of an OPC client, the call rate can also be slower.

Note

If you have enabled the "Lock..." button in the "Security" dialog of the "Communication Settings" program, to restore remote communication via the PROFINET IO OPC server, you need to click the "Allow" button (remote basic and OPC communication).

2.12.2 How can IO data be addressed?

Generally, no addressing for IO data involving more than one submodule, module, or device is supported. No internal group calls are possible; in other words, each PROFINET IO address must be read or written individually.

Addressing

Logical addressing is used.

The logical address references an image of the IO areas of all devices in the process images managed by the controller. These process images are 4 gigabytes each

long for the I and Q area ($2^{32}-1$) The process images are assigned with the PROFINET IO configuration tools. The logical address specifies the byte offset of the IO area of a device in the process image.

Logical addressing is ideal for reading and writing IO data. If the user configures carefully, the IO images of the devices are packed tightly together. Continuous areas of several devices can be read and written at the same time.

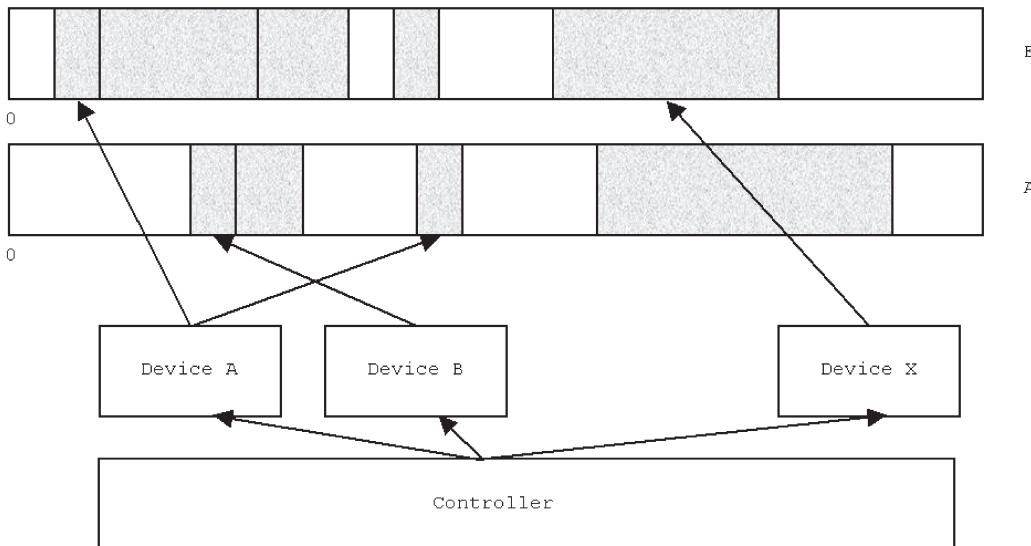


Figure 2-61 Reading and writing the IO data of devices in separate or continuous areas by the controller

Example

Example of addressing a controller CP 1612 PROFINET IO with an OPC server application and an IM 151.

The addresses are:

- Input 0, length 4 bits
- Output 0 and 1, length 2 bits

The diagnostics address of the IM151 is 16382.

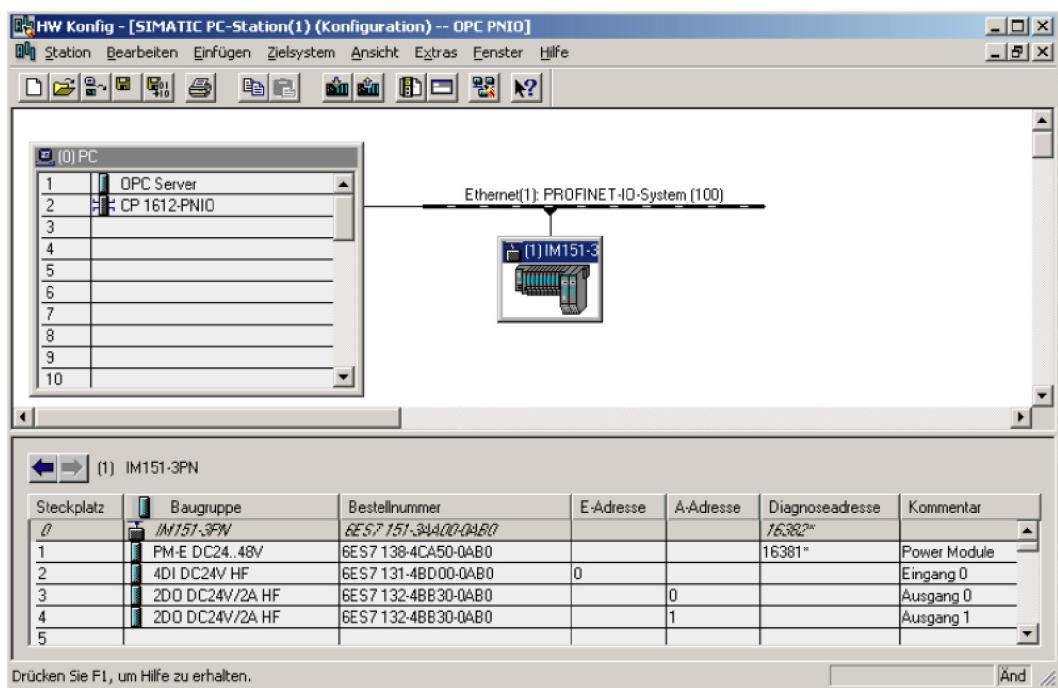


Figure 2-62 Display of the addresses of the modules of a PC station in the station window of "HW Config"

How can the addressing be expanded in practical terms?

If you only want to read a single input bit of a submodule with, for example, a width of 64 bits, all 8 bytes must be read completely. The same applies to writing and output bit; all 8 bytes must be written completely.

The structure of the plant; in other words all the information on the physically existing devices and their modules and submodules, is configured and therefore known to the user. However, it must also be possible to remove or insert modules and submodules during operation.

To provide the OPC user with a degree of convenience when converting the IO data, expanded addressing was introduced. The user, who presumably knows the plant well, can supply the PROFINET IO OPC server locally with the necessary configuration information with the aid of the item syntax.

Example:

Let us assume that a PROFINET IO device controls 64 electric switches via an IO submodule. The submodule is mapped to the logical base address 400 with the configuration software and has a value range of 64 bits, in other words 8 bytes. This information is predefined. With the following item, the user can read or write bit 5 of the submodule starting out offset 4.

PNIO:[ctrl1]QB400,8,X4.5

Note

Note that with simple addressing, reading and writing PROFINET IO data requires the exact configured length information (with advanced addressing, partial addressing is possible).

Example:

An IO module with a length of 4 bytes cannot be partially read using an OPC item PNIO:[CTRL1]IWORD0 (2 bytes).

Here, for example, the following are allowed: "PNIO:[CTRL1]IB0,4" or "PNIO:[CTRL1]IDWORD0".

2.12.3 How can the configured PROFINET IO namespace be browsed?

The configured PROFINET IO devices of the OPC server can be displayed with the browsing functions of OPC DA.

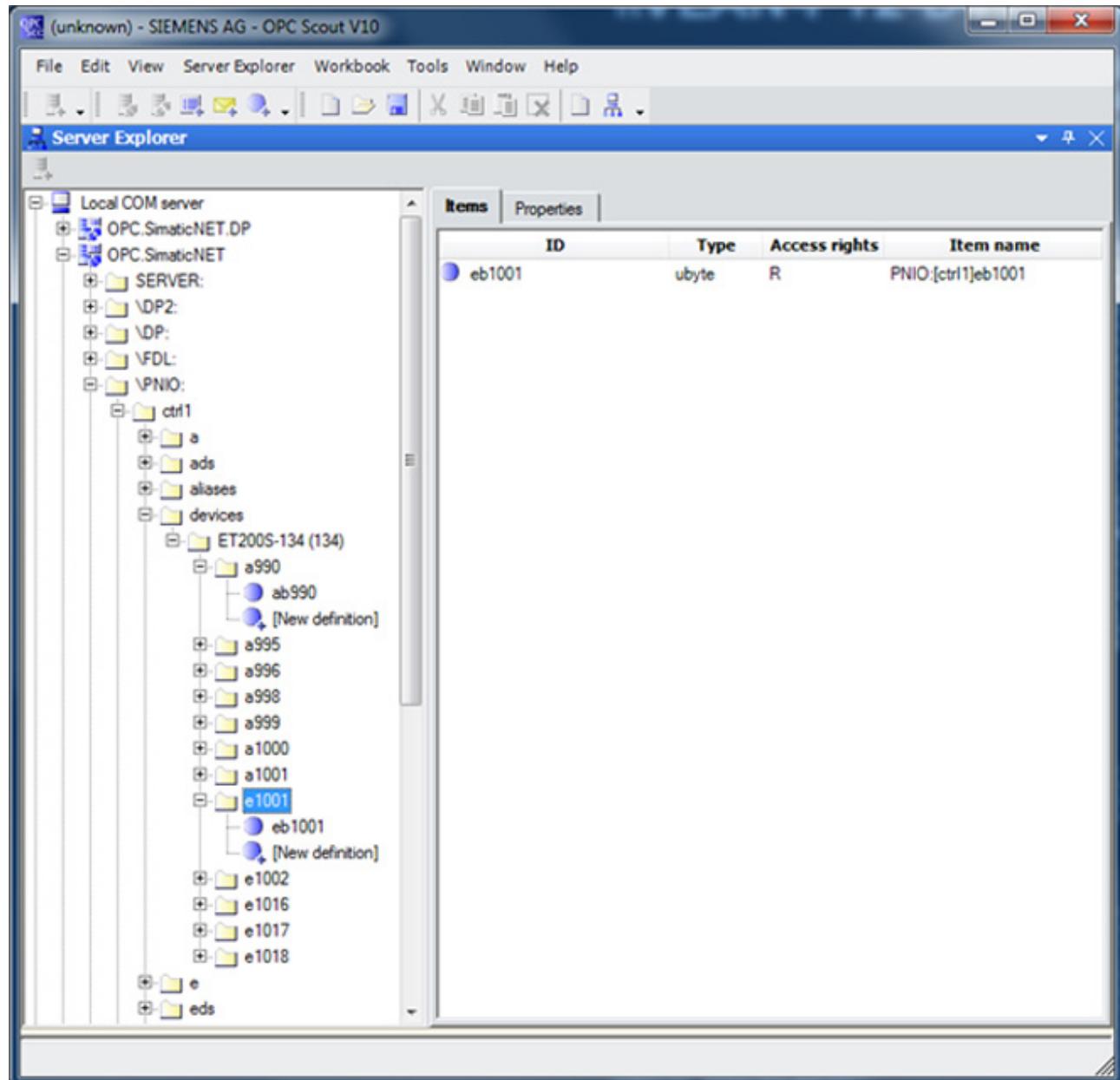


Figure 2-63 Browsing the configured namespace in OPC Scout

The existing inputs and outputs and their addresses are displayed. Since the configured data length is known, a predefined item is available.

In the plant view, all configured devices and their names (with their submodules below them) are listed in the "devices" branch. All existing inputs and outputs and their addresses are displayed for each submodule. Predefined items already exist.

2.12.4 Which OPC variables are available for PROFINET IO?

The PROFINET IO OPC server of SIMATIC NET provides the following variable types:

- Process variables - Reading/writing/monitoring data
- Information variables - status queries
- Control variables - Enable/disable devices

2.12.5 Protocol ID

The protocol ID for the PROFINET IO protocol is "PNIO".

2.12.6 Controller name

The connection name specifies the communication access by specifying the communications processor attached to the PROFINET Ethernet network.

With the PROFINET IO protocol, the controller name is the name of the communications module including the slot (index) in the PC station (Station Configuration Editor).

Controller name = *CTRL<Index>*

Examples of controller names:

- CTRL3
- CTRL5

2.12.7 PROFINET IO process variables

Reading and writing items for PROFINET IO process variables is the most common application. Simple access is possible. The user must specify the controller name and an address from the project engineering.

The configured address space can be browsed in the OPC namespace.

The addresses are checked when items are added.

Simple addressing

Syntax

Inputs:

PNIO: [<controllername>] I<format><address>
. <bit|stringlength> { , <quantity> }

Outputs:

PNIO: [<controllername>] Q<format><address>
. <bit|stringlength> { , <quantity> }

Explanations

PNIO

PROFINET IO protocol for access to the process variables.

<devicename> = CTRL<index>

Protocol-specific device name. The index of the device name is specified during the configuration of the PC station. Range of values 1 to 32

Example: CTRL1

I

Identifier for an input. Inputs are read only.

Q

Identifier for an output. Outputs can be read and written.

<format>

Format of the delivered data.

Specifying the format specifies the data type.

In principle, all listed OLE data types can be read via the Automation and Custom interface of OPC. However, certain development tools such as Visual Basic only offer a restricted number of data types.

The following table therefore lists the corresponding Visual Basic type in which the variable value can be represented.

Data types longer than one byte are interpreted in Motorola format.

Format identifier	OLE data type	Visual Basic V6.0 type automation data type	Description
X	VT_BOOL	Boolean	<p>In addition, the bit address in the relevant byte must also be specified.</p> <p>Writing individual bits is possible only with expanded addressing due to consistency considerations; the OPC client should, however, always write all bits of the logical address at the same time.</p> <p>An array length can also be specified in bits. It is always possible to read bit arrays, however, due to consistency considerations, it is only possible to write when the bit address has the value 0 and the bit length is a multiple of 8.</p> <p>To retain uniformity, however, reading bit arrays is also only permitted when the bit address has the value 0 and the bit length is a multiple of 8.</p>
B or BYTE	VT_UI1	Byte	Byte (unsigned)
W or WORD	VT_UI2	Long	Word (unsigned)
D or DWORD	VT_UI4	Currency	<p>Double word (unsigned)</p> <p>Mapped to the more extensive Visual Basic type 6.0 Currency 8 bytes, since no corresponding data type exists.</p>
LWORD	VT_UI8	VT_UI8	Automation type integer 8 bytes (unsigned), not supported by Visual Basic 6.0.
CHAR	VT_I1	Integer	Byte (signed) mapped to the more extensive Visual Basic type integer 2 bytes.
INT	VT_I2	Integer (%)	Word (signed)
DINT	VT_I4	Long (&)	Double word (signed)
LINT	VT_I8	VT_I8	Automation type integer 8 bytes (signed), not supported by Visual Basic 6.0.
REAL	VT_R4	Single	Floating point 4 bytes
LREAL	VT_R8	Double	Floating point 8 bytes
String	VT_BSTR	String	<p>String</p> <p>The length reserved for the string must also be specified. When writing, shorter strings can also be written whereby the transferred data length is always the reserved string length in bytes. The bytes not required are padded with the value 0x00.</p>

<bit|stringlength>

Specifies data type X: Bit number in the addressed byte.

The range is between 0 and 7.

Specifies the data type STRING: Reserved string length.

For the format identifiers X and STRING, .<bit|stringlength> must also be specified.

<quantity>

Quantity of the elements

Reading and writing bit arrays is possible only when

- the bit address has the value 0 and
- the length is a multiple of 8 and
- the exact module length is reached.

<address>=<logical address>

Simple logical addressing

Logical base address of the device as set in the project engineering.
number range of values 0 to 4 294 967 295.

Note

Please note that writing and reading PROFINET IO data is possible only with exactly the configured length. For example, an IO module with a length of 4 bytes cannot be partially written to with only 2 bytes using an OPC item *PNIO:[CTRL1]QWORD0*. The following are permitted, for example *PNIO:[CTRL1]QB0, 4* or *PNIO:[CTRL1]QWORD0*.

Expanded addressing

The expanded address includes the data area of the entire submodule. This information can be found in the configuration.

Syntax

Inputs:

```
PNIO:<controllername>I<expanded address>,
      <format><offset>{.<bit|stringlength>}[,<quantity>]
```

Outputs:

```
PNIO:<controllername>Q<expanded address>,
      <format><offset>{.<bit|stringlength>}[,<quantity>]
```

Additional explanations**<expandedaddress> =**

B or BYTE<logicaladdress>,<numberofsubareas>

The expanded address describes the entire address area of a submodule within which any subareas down to individual bits can be addressed relatively.

The expanded address is described by the number of bytes starting at the logical base address.

<offset>

Byte offset within the addressed submodule at which the element to be addressed is located.

Note

With expanded addressing, it is possible to read and write individual bits. It is also possible to read and write arrays of individual bits without any restriction relating to the start address and array length within the expanded address.

Examples of process variables for PROFINET IO

Here you will find examples illustrating the syntax of variable names for PROFINET IO variables.

- **Inputs**

PNIO:[CTRL3]IB10

Controller index 3, logical address 10, first input byte.

PNIO:[CTRL3]IB4,3

Controller index 3, logical address 4, array of the first 3 bytes.

PNIO:[CTRL 1]ID2

Controller index 1, logical address 2, first double word.

PNIO:[CTRL 1]IX10.0,64

Controller index 1, logical address 10.0, array of the first 64 output bits. Only multiples of 8 from the start of the first byte.

PNIO:[CTRL 1]IReal4

Controller index 1, logical address 4, first floating-point number in the input area.

Expanded addressing:

PNIO:[CTRL 1]IB2,1,X1.0,2

Controller index 1, expanded address 2, area quantity 1 byte, offset 1 byte, array of the first 2 bits starting at input bit 0.

PNIO:[CTRL 1]IB10,4,X1.3,7

Controller index 1, expanded address 10, area quantity 4 bytes, offset 1 byte, array of the first 7 bits starting at input bit 3.

- **Outputs**

PNIO:[CTRL3]QB7

Controller index 3, logical address 7, first output byte.

PNIO:[CTRL 1]QW0,8

Controller index 1, logical address 0, array of the first 8 words.

PNIO:[CTRL3]QX2.0,64

Controller index 3, logical address 2.0, array of the first 64 output bits, readable and writable. Only multiples of 8 beginning at bytes.

PNIO:[CTRL 1]QSTRING100.32,3

Controller index 1, logical address 100, array of the first 3 strings of 32 characters.

Expanded addressing:

PNIO:[CTRL 1]QB1,1,X0.0

Controller index 1, expanded address 1, area quantity 1 byte, offset 0 bytes first output bit 0.

PNIO:[CTRL 1]QB2,4,X1.0,2

Controller index 1, expanded address 2, area quantity 4 bytes, offset 1 byte, array of the first 2 bits starting at output bit 0.

2.12.8 PROFINET IO data status

With the items for the status (IOPS and IOCS), an OPC client can query or set the status bytes of a logical address.

The following schematic shows the data status and flow of values in the process image.

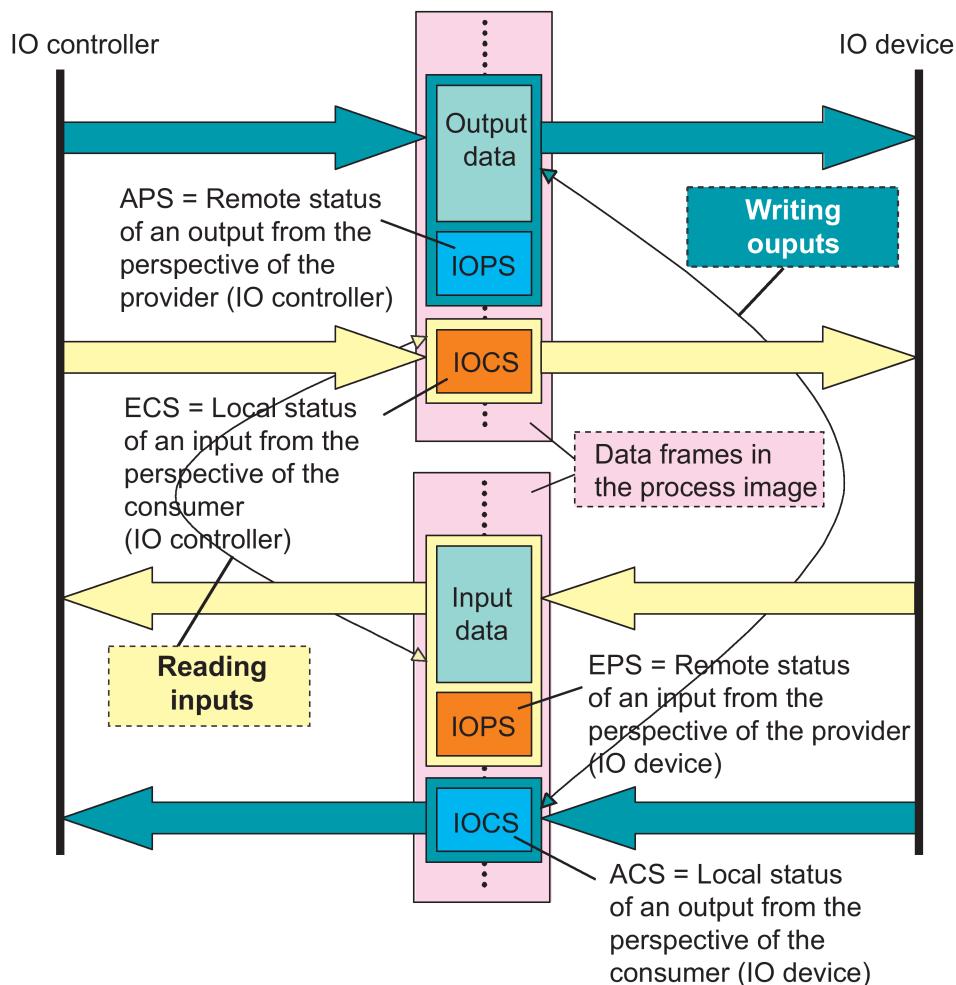


Figure 2-64 Overview of the data status in the process image in PROFINET IO communication between IO controller and an IO device

When the OPC client writes output data to the process image, it includes the remote status of the output from its perspective as provider with "QPS".

The device includes its own local status of the output from its perspective as consumer with "QCS".

When the OPC client reads input data from the process image, it includes its own local status of the input from its perspective as consumer with "ICS".

The device includes the remote status of the input from its perspective as provider with "IPS".

Note

Read access to cyclic output data

The reading, monitoring or activation of OPC items/nodes that map cyclic output data brings about a one-time or constant update of the underlying PROFINET IO protocol interface with the data contained in the OPC server cache including the provider status (PNIO_data_write()). Please remember that after the return of a PROFINET IO device or module, this output data is transferred to the device again after the point in time of the update.

Syntax

PNIO:[<controllername>]<I|Q><PS|CS><address>

Explanations

PNIO

PROFINET IO protocol for access to data records.

<controllername> = CTRL<index>

Protocol-specific controller name. The index of the controller name is specified during the configuration of the PC station. Range of values 1 to 32

I

Identifier for an input. Inputs are read only.

Q

Identifier for an output. Outputs can be read and written.

PS

Identifier for remote status (provider status).

CS

Identifier for local status (consumer status).

or

IPS Remote status input

QPS Remote status output

ICS Local status input

QCS Local status output

<address>=<logical address>

Simple logical addressing

Logical base address of the device as set in the configuration.

Numeric range of values 0 to 4 294 967 295.

Name	Explanation
IPS	<p>Remote status of an input address Data type: VT_I4, ReadOnly. The data provider of an input is the device from the perspective of the OPC server. The device also provides the status of the input data. The IPS is monitored by polling. The use of the item is not absolutely necessary for an OPC application because the content of the item is reflected in the OPC quality of the linked input items with the same address.</p>
ICS	<p>Local status of an input address Data type: VT_I4, ReadWrite. Default: <i>GOOD</i> = 0 The data consumer of an input is the OPC client from the perspective of the OPC server. The OPC client also informs the device of the status of the input data from its perspective. Normally, this status is always <i>GOOD</i>. When IO data is read, the last local status of an input address written by the OPC application is always included (written to the partner device). If the OPC application has not created the item or not written a value to it, the value <i>GOOD</i> is transferred when the corresponding input data is read. The status is read by the OPC client over the cache.</p>
QCS	<p>Local status of an output address Data type: VT_I4, ReadOnly. The data consumer of an output is the device from the perspective of the OPC server. The device also includes the status of the output data from its perspective. Reading the status is possible if the OPC client can first (or the same time) write a linked output item with the same address. The QCS is monitored by polling and the last output data written by the OPC client is always transferred to the function at the same time. If the OPC client was not yet able to write a value successfully, reading from the device results in an error or the OPC quality <i>bad</i>.</p>
QPS	<p>Remote status of an output address Data type: VT_I4, ReadWrite. Default: <i>GOOD</i> = 0 The data provider of an output is the OPC client from the perspective of the OPC server. The OPC client also informs the device of the status of the output data from its perspective. Normally, this status is always <i>GOOD</i>. When IO data is written, the last local status of the output data written by the OPC application and the output data of an output address are also included (written). If the OPC application has not created the IO item or not written an output value successfully, the value <i>GOOD</i> is transferred when the corresponding output data is written. When the OPC client writes a new status, the data value is used that was successfully written previously (or the same time) with the address of the linked output item. If the OPC client has not yet written a value successfully, writing the item results in the special return value (S_OP_COREPNOSTATUS). The written status is then written by the corresponding output item with the next successful write.</p>

Structure of the status value

The status value, type *VT_UI4* always has the following structure:

0 = GOOD
1 = BAD

2.12.9 PROFINET IO data records

Using the data record functions, device-specific information can be read out or parameters written.

The meaning of the data records is specified by the vendor. They could, for example, be used for the configuration data of a drive. When a data record variable is registered, the OPC server can only check that the syntax is correct and not whether the variable is valid on the partner device and the size of the data record is adequate based on the configuration of the device.

Syntax

```
PNIO:[<controllername>]<I|Q>DS<address>,DATA<datarecordindex>,
      {<datarecordlength>} {,<subelement>}
```

Explanations

PNIO

PROFINET IO protocol for access to data records.

<controllername> = CTRL<index>

Protocol-specific controller name. The index of the controller name is specified during the configuration of the PC station.

Range of values: 1 ... 32

I

Identifier for an input.

Q

Identifier for an output.

DS

Identifier for a data record.

<address>=<logical address>

Simple logical addressing

Logical base address of the device as set in the configuration.

Numeric range of values 0 to 4 294 967 295.

DATA

Identifier for data.

- **<datarecordindex>**

Data record index, device-specific parameter.

Range of values: 0 ... 65535).

Some of the data record indexes standardized in the PROFINET IO specifications are as follows:

0x8030 = 32816 Read input data object

0x8031 = 32817 Read output data object

0x8020 = 32800 Write substitute values for output data object

60k+10 = 0xF00A = 61450 Read diagnostics information IO device level

56k+10 = 0xE00A = 57354 Read diagnostics information AR level<

48k+10 = 0xC00A = 49162 Read diagnostics information slot level

32k+10 = 0x800A = 32788 Read diagnostics information subslot level

- **<datarecordlength>**

Data record length. When reading data records, the device can be informed of the exact length of the data record to be read in bytes. The OPC client must then specify the length. The item can also be written if a length is specified. The item cannot be written if no length is specified. When reading, the data record length is optional and can be omitted. In this case, the length of the array of the (return) values is variable. The user does not need to know how long the data record is (range of values: 0 ... 65535).

- **<subelement>**

To support the user when structuring data records, one or more subareas can be selected. Once again the valid data types for the IO process variables as described above are used.

Read: Since the structure and length of the data record read from the partner is not need to be fixed, it is not forbidden to define and request data outside the area. If the area can no longer be filled when receiving data, the quality of the variables changes accordingly.

Write: When writing to a subarea, the entire data record is always sent. If several subareas of the data record are written in an OPC write job, the data record is only sent after all its subareas have been updated.

In the data record, the data types are interpreted in Motorola format and converted to the corresponding Intel format.

`<subelement>=<format><offset>{.<bit/stringlength>}{,<quantity>}`

- **<offset>**

Byte offset of the subelement relative to the addressed data record.

The significance of the values `<format>`, `<bit/stringlength>` and `<quantity>` is as explained in the section "PROFINET IO process variables (Page 328)".

Note

It is also possible to read and write individual bits with data records.

It is also possible to read and write arrays of individual bits without any restriction relating to the start address and array length within a data record.

Note

By specifying the optional information type, address and quantity, you can define structured access to subareas of data records.

Variables for input or output data records with a different controller name or different address or different data record index or different length have independent memory areas.

The output data buffer is allocated and initialized with zero when an item is created for an independent memory area. A write to an ADS item is written to the internal write buffer and transferred.

The OPC read cache is a separate buffer that is updated only by device read jobs.

The data records are transferred acyclically. Simultaneous network jobs for the same data are possible. The complete send data record is always transferred. This also applies to subelement access or when several clients write to this item at the same time.

Writing simultaneously to the same send data record or subareas of the send data record by several clients can lead to inconsistencies. We therefore recommend that you always read or write the complete data record.

Value of the data record

Data type: *VT_ARRAY / VT_UI1*. Read-only; if the data record length is specified, it can also be written.

Contains the last data record read from the device or the last data record written to the device. The structure and length of the data record is flexible. The length of the ready data record is implicitly contained in the array length of the value.

Remember that depending on the addressed device and the data record index, it may not be possible to read back modified data records or to read them back only once. As a result, data records should only be polled (active item) if the data records support multiple reading.

If a data record with read restrictions or write restrictions is addressed, an error is returned on the OPC interface (communication error or access error).

Examples of PROFINET IO data records

PNIO:[CTRL 1]IDS16382,DATA61450,22

Controller index 1, diagnostics address 16382, data record index 61450, length 22 bytes, read diagnostics information IO device level.

PNIO:[CTRL 1]IDS32,DATA32788,100,W8

Controller index 1, logical address 32, data record index 32788, length 100 bytes. Read diagnostics information of the subslot. Word 8, if it exists, contains the error type of the first channel error.

PNIO:[CTRL 1]IDS10,DATA32768

Controller index 1, module address 10, data record index 32768, length not specified, item can only be read. Read diagnostics information of a slot.

PNIO:[CTRL 1]IDS16382,DATA61452

Controller index 1, diagnostics (base) address 16832, data record index 61452, length not specified, item can only be read. Read diagnostics information vendor-specific, block, slot, channel, and data.

PNIO:[CTRL 1]QDS10,DATA32768,22,B0,10

Controller index 1, logical address 10, data record index 32768, length 22 bytes specified, item can be written and read, subarea of the first 10 bytes. Diagnostics information of a slot.

PNIO:[CTRL 1]QDS10,DATA32768,22,X0.0,13

Controller index 1, logical address 10, data record index 32768, length 22 bytes specified, item can be written and read, subarea of the first 13 bits starting at bit 0. Diagnostics information of a slot.

PNIO:[Ctrl2]IDS11000,DATA2147483648,80,X0.0,1

This item can be inserted in an OPC group even if good quality cannot be achieved because this index cannot be configured and is outside the range of values.

2.12.10 Syntax of the systemspecific information variables

Syntax

`PNIO: [SYSTEM] &version()`

Explanation

Returns a version ID for the PROFINET IO OPC server, here the string, for example SIMATIC NET Core Server PNIO V 7.xxxx.yyyy.zzzz Copyright 2012

Data type: VT_BSTR

Access right: readonly

2.12.11 PROFINET IOspecific information variables

Syntax

```
PNIO:[<controllername>]&<informationparameter>
```

Explanations

PNIO

PROFINET IO protocol for access to the process variables.

<controllername>

Protocol-specific controller name. The index of the controller name is specified during the configuration of the PC station.

&<informationparameter>

Possible values are as follows:

- deviceactivate()
- deviceactivateval()
- devicedeactivate()
- devicedeactivateval()
- mode()
- modeval()

&deviceactivate()

Activation of a device

This mode can only be written. Setting the mode by writing one of the values shown below is only possible within the context of the PROFINET IO application environment.

The permitted mode changes must be kept to. These are as follows:

INACTIVE → ACTIVE

Input values as string:

Logical address of the device according to the configuration and the identifier for an input I or output Q: <Q|I><logicaldiagnosticsaddress>.

Example: I16373.

&deviceactivateval()

Activation of device with numeric values.

Data type VT_ARRAY | VT_I4, WriteOnly. Array length 2 elements (IO flag and logical address).

Input values as array of 2 numeric values:

The first element of the array is always a flag that specifies whether the address is an input or an output address.

0 = input address

1 = output address

The second element describes any logical address of the device.

Example: Value{0|16375} corresponds to I16375.

Note

Due to the activation of the device, the protocol may trigger the "Station return" interrupt.

&devicedeactivate()

Deactivation of a device.

This mode can only be written. Setting the mode by writing one of the values shown below is only possible within the context of the PROFINET IO application environment.

The permitted mode changes must be kept to.

These are as follows:

ACTIVE → *INACTIVE*

Input values as string:

Logical address of the device according to the configuration and the identifier for an input I or output Q: <Q||><logicaldiagnosticsaddress>.

Example: I16373.

The diagnostics address of a device can also be selected.

&devicedeactivateval()

Deactivation of device with numeric values.

Data type: VT_ARRAY | VT_I4, WriteOnly.

Array length: 2 elements (IO flag and logical address).

The first element of the array is always a flag that specifies whether the address is an input or an output address.

0 = input address

1 = output address

The second element describes any logical address of the device.

Example: Value{0|16373} corresponds to I16373.

Note

Due to the deactivation of the device, the protocol may trigger the "Station failure" interrupt.

&mode()

Query or set the mode of the controller.

When writing a new mode, all mode changes are permitted.

OFFLINE ↔ CLEAR ↔ OPERATE

or

OFFLINE ↔ OPERATE

This means, for example, that a direct change from OFFLINE to OPERATE and vice versa is possible.

The OFFLINE mode can be set, for example, to stop the controller. There is no automatic change to CLEAR but to OPERATE only during startup when the first client connects to the OPC server if "Set PNIO controller automatically to OPERATE" was configured with NCM.

When the OPC server is exited, the OFFLINE status is set automatically.

Data type VT_BSTR, writable, readable.

- OFFLINE
- CLEAR
- OPERATE

&modeval()

Query the mode of the controller or set as value.

Data type: VT_UI4, writable, readable

Possible Values	Meaning
0	OFFLINE
1	CLEAR
2	OPERATE

2.13 PROFINET IO communication with OPC UA over Industrial Ethernet

The system model of a PROFINET IO device assigned to the PROFINET IO controller is module-oriented. A PROFINET IO device can contain several modules, each module can contain several submodules. A submodule generally contains the physical terminals or driver blocks (channels) for the plant hardware to be connected, for example conveyor belts or sensors. Their input data or set values are addressed with the IO data. Reading and writing the IO data is the most important use case for PROFINET IO.

Both devices and modules and submodules can provide data records and generate alarms. The use of data records and alarms is device-dependent.

The PROFINET IO OPC UA server of SIMATIC NET has following characteristics:

- Process variables - Reading/writing/monitoring data
- Information variables - status queries
- Control variables - Enable/disable devices

2.13.1 SIMATIC NET OPC UA server for the PROFINET IO protocol

Introduction

The section below describes a configuration variant for the PROFINET IO protocol that OPC UA also supports. To do this, the PROFINET IO COM OPC Data Access server is set up as an outproc OPC server.

Configuration

The PROFINET IO UA server is activated by selecting "PROFINET IO" and "OPC UA" in the "Communication Settings" configuration program in the "OPC protocol selection" catalog:

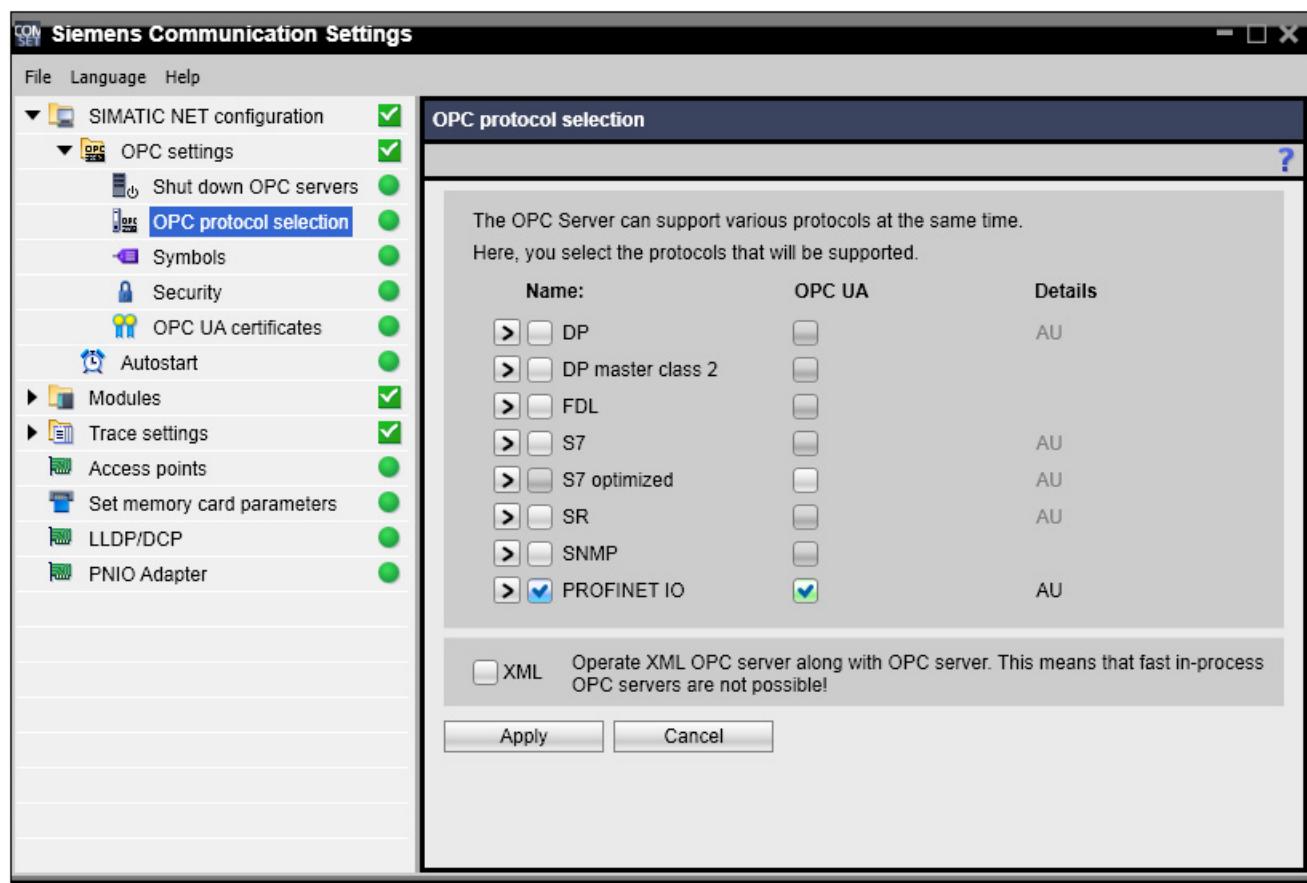


Figure 2-65 Window in the "Communication Settings" configuration program for selecting OPC UA for the PROFINET IO protocol

Advantages / disadvantages

When using the PROFINET IO OPC UA server, only the outproc mode of PROFINET IO is possible. The PROFINET IO OPC UA server process must be running to maintain readiness to receive. Even after all OPC UA clients have logged off, the PROFINET IO OPC UA server is not and must not be exited.

On the other hand, it does have the following advantages:

- COM/DCOM configuration is no longer necessary.
- High-speed, secure communication
- Only one server is required for events and data access.
- Several clients can use the server at the same time.

If you select the "OPC UA" and "PROFINET IO" check boxes, the PROFINET IO UA server is always started. With a suitable PROFINET IO configuration, this means that the PROFINET IO controller is in OFFLINE mode. Only when at least one OPC UA client has connected to the server, does the PROFINET IO controller change to OPERATE mode.

Note

The configured cycle time of the server is not automatically the time with which the underlying PROFINET IO protocol interface is called. Depending on the update time of an OPC client, the call rate can also be slower.

Note

If OPC UA is active for the PROFINET IO protocol, a configured PROFINET IO controller for the OPC server changes to the OPERATE mode as soon as the first OPC UA client has logged on with the OPC UA PROFINET IO server. If no OPC UA client has connected, e.g. after starting up the computer, a configured PROFINET IO controller remains in OFFLINE mode for the OPC server.

2.13.2 How is the PROFINET IO OPC UA server addressed?

Server URL

For the native binary TCP protocol, there are two ways for the OPC client to address the server:

- Direct addressing:
 - opc.tcp://<hostname>:55104
 - or
 - opc.tcp://<IP-Adresse>:55104
 - or
 - opc.tcp://localhost:55104

The PNIO OPC UA server has port 55104.

- The URL of the PROFINET IO OPC UA server can also be found using the OPC UA Discovery service.

The entry for locating the Discovery server is as follows:

- opc.tcp://<hostname>:4840
 - or
- opc.tcp://<IP-Adresse>:4840
 - or
- http://<hostname>:52601/UADiscovery/
 - or
- http://<IP-Adresse>:52601/UADiscovery/

The Discovery server has port 4840 (for TCP connections) and port 52601 (for HTTP connections).

IPv6 address

An IPv6 address can also be used as the IP address. The address must be in parentheses, for example [fe80:e499:b710:5975:73d8:14].

Endpoints and security modes

The SIMATIC NET PROFINET IO OPC UA server supports endpoints with the native binary TCP protocol and allows authentication using certificates and encrypted transfer.

The Discovery service on the addressed host signals the endpoints of the servers, in other words, their security requirements and protocol support.

The server URL "opc.tcp://<hostname>:55104" of the PROFINET IO OPC UA server provides the following endpoints:

- Endpoint in "SignAndEncrypt" security mode:

A signature and encryption are required to communicate with the server. Communication is protected by exchanging certificates and entering a password.

In addition to the security mode, the security policy Basic128Rsa15 is also displayed.

- Endpoint in "None" security mode:

In this mode, no security functions are required by the server (security policy "None").

For more detailed information on the security functions, refer to the section "Programming the OPC UA interface (Page 496)".

The security policies "Basic128Rsa18" and "None" are in the UA specification of the OPC Foundation at the following Internet address:

"<http://opcfoundation.org/UA>" > "Specifications" > "Part 7"

You will find more detailed information on the following Internet page:

OPC Foundation (<http://www.opcfoundation.org/profilereporting/index.htm>)
"Security Category" > "Facets" > "Security Policy"

The OPC UA Discovery of the OPC Scout V10

The OPC Scout V10 allows you to open the OPC UA Discovery dialog to enter UA endpoints in the navigation area of the OPC Scout V10.

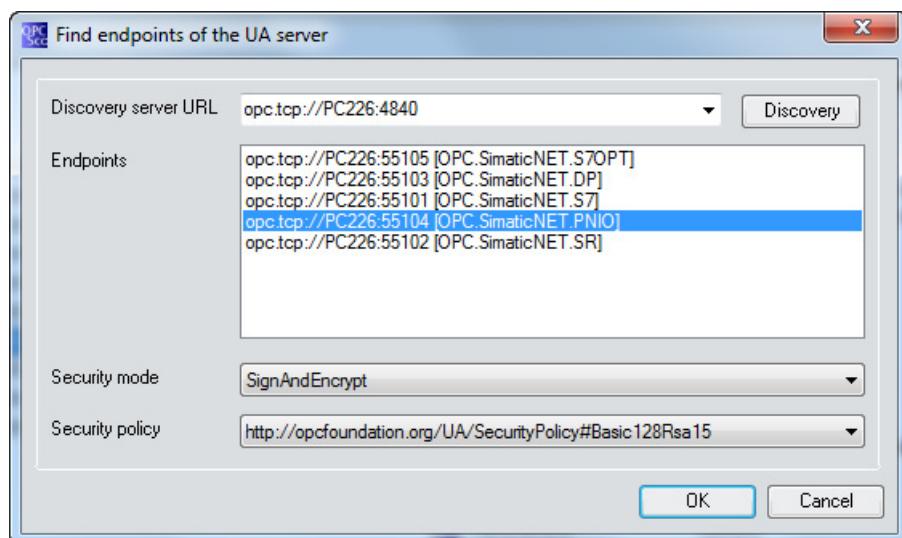


Figure 2-66 The "Find endpoints of the UA server" dialog of OPC Scout V10

The PROFINET IO OPC UA server can be found using the OPC UA Discovery service. For the entry, refer to "Server URL" above.

OPC Scout V10 contains a list of the OPC UA endpoints. These are persistent. The Discovery service on the addressed host then signals the registered OPC UA servers and their ports and security modes.

For more detailed information, refer to the online help of OPC Scout V10.

2.13.3 Which namespaces does PROFINET IO communication with OPC UA provide?

The PROFINET IO OPC UA server provides the following namespaces:

Table 2- 8 The namespaces of OPC UA

Namespace index	"Identifier" (namespace URI) / comment
0	"http://opcfoundation.org/UA/" specified by the OPC Foundation
1	"urn:Siemens.Automation.SimaticNET.PNIO:(GUID)" Unique identifier of the local high-speed PROFINET IO OPC UA server.
2	"PNIOTYPES:" Identifier for the namespace of the PROFINET IO types.
3	"PNIO:" Identifier of the local high-speed PROFINET IO OPC UA server with new simplified syntax (browsable and can be used with UA)
4	"PNIOCOM:" Identifier of the server with old syntax. (can be used with UA but cannot be browsed) In this namespace, the PROFINET IO OPC DA-compatible syntax can be used on the OPC UA server, however no identifiers can be used here that correspond to the alias items or symbols in the OPC DA server. Example: In the PNIOCOM namespace: There is the identifier PNIO:[cltr1]ab0 which is also an item on the OPC DA server.

Depending on the configuration, the assignment between the namespace index and namespace identifier can change on your OPC UA client for indexes 2 to 4. The identifier must therefore always be unique.

The namespace indexes 0 and 1 are reserved and their significance is specified by the OPC Foundation.

The assignment of the remaining namespace indexes to the identifiers (namespace URI) must be obtained via the data variable "NamespaceArray" at the beginning of an OPC UA session by the client specifying the identifier. The PNIOTYPES identifiers: PNIO: PNIOCOM: always exist with the PROFINET IO OPC UA server.

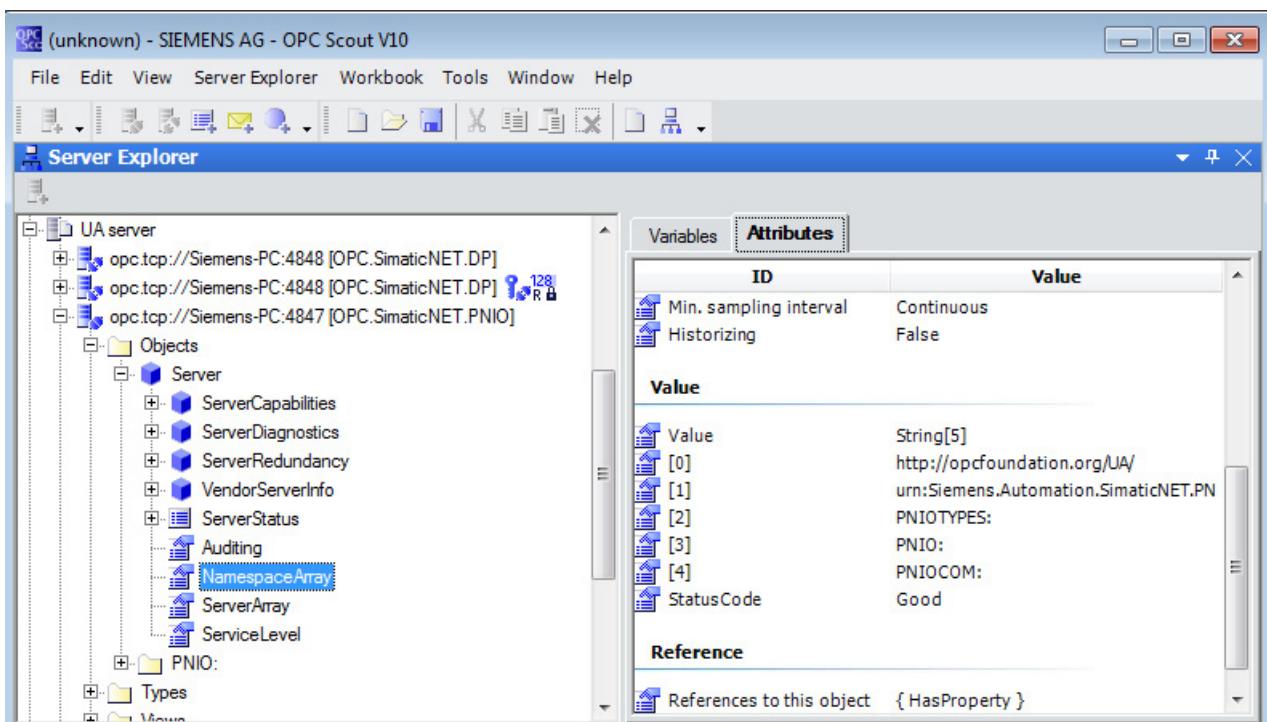


Figure 2-67 Display of the PROFINET IO namespaces using the browse function of the OPC Scout V10

How can the configured PROFINET IO namespace be browsed?

The configured PROFINET IO devices of the OPC server can be displayed with the browsing functions of OPC Data Access.

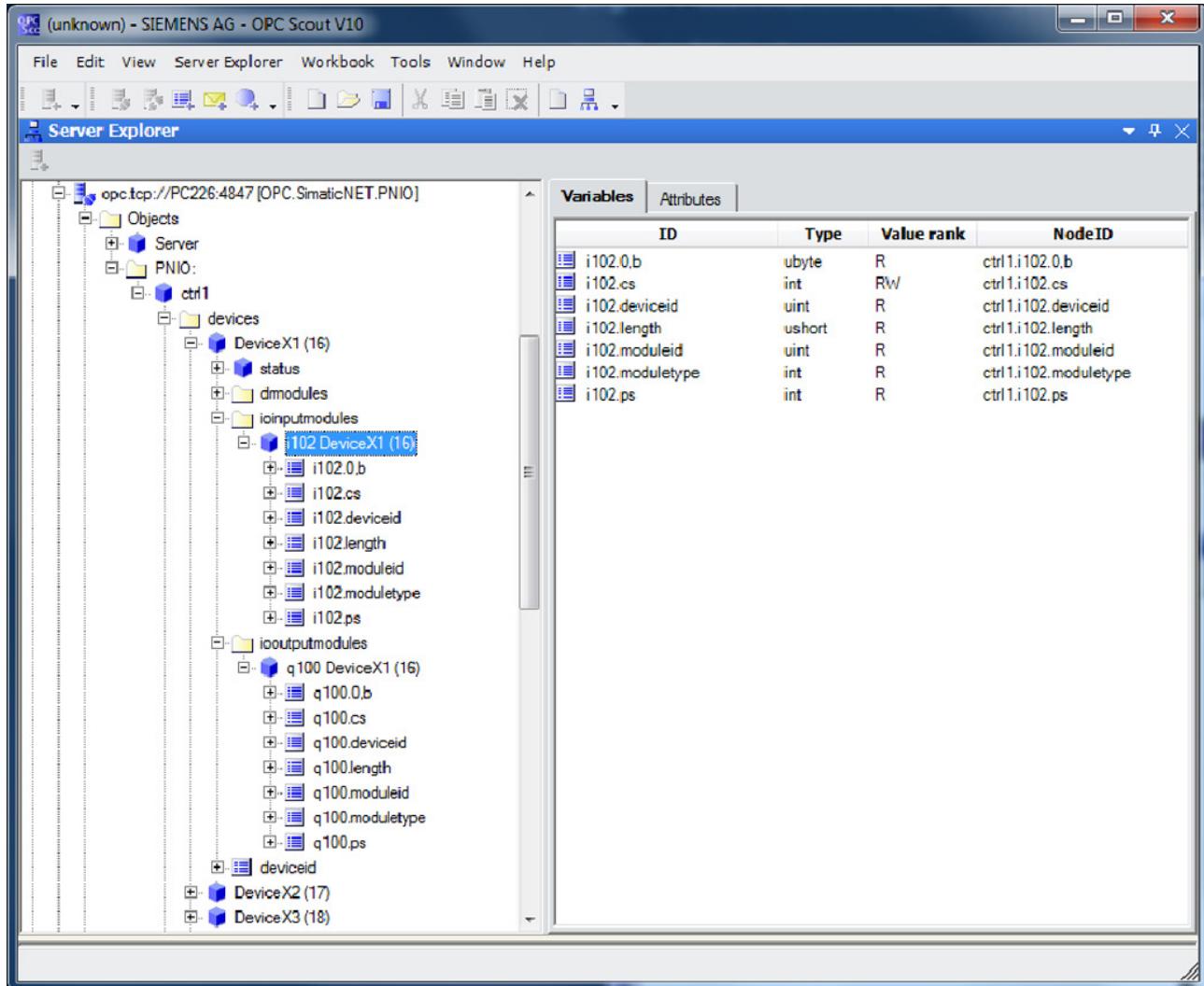


Figure 2-68 Browsing the configured namespace in OPC Scout

The existing inputs and outputs and their addresses are displayed. Since the configured data length is known, a predefined variable is available.

In the plant view, all configured devices and their names (with their submodules below them) are listed in the "devices" branch. All existing inputs and outputs and their addresses are displayed for each module. Predefined items already exist.

2.13.4 The Nodeld

Identification of a PROFINET IO process variable

The Nodeld identifies a PROFINET IO process variable with the aid of the following tuple:

- Namespace index
- Identifier (character string, numeric value)

Examples

- Nodeld:
 - Namespace URI:
PNIO:
(= namespace index 3) for Siemens.Automation.SimaticNET.PNIO
 - Identifier
ctrl1.q0.0,b
- Nodeld:
 - Namespace URI:
PNIocom:
(= namespace index 4)
 - Identifier
PNIO:[ctrl1]AB0

How does the new namespace adapted to OPC UA behave?

While the world of OPC Data Access items tends to be limited to reading and writing process variables, and is self-contained and exists fully isolated, for example from the alarm world, the OPC UA view concentrates on automation objects with various properties.

Accordingly, OPC UA no longer access items alone, but also objects and their subobjects. Data variables and methods are, for example, subobjects of a PROFINET IO device object. Properties define the objects in greater detail, all information is provided by attributes of the objects.

An OPC Data Access item for access to IO areas of a PROFINET IO device corresponds to an OPC UA data variable. An OPC Data Access item for activating/deactivating a device corresponds to an OPC UA method.

The qualified identifiers (Nodeld) have a greater significance in OPC UA than in OPC Data Access. Each individual access to an object, subobject, property and attribute uses its Nodeld.

The OPC namespace for PROFINET IO is represented as a tree structure and can be browsed for Nodelds. All the texts of the namespace are in English.

Syntax

The Nodelds of all OPC UA objects have the following structure:

<controllerobject>.<object>.<subobject>.<property>

An object can contain further subobjects.

A Nodeld that cannot be interpreted is rejected with an error. The letters "A-Z" are not case-sensitive for any objects.

Symbolic object representation

The OPC UA specification recommends a uniform symbolic representation for the hierarchical description of the address space. The symbols used in this document are as follows:

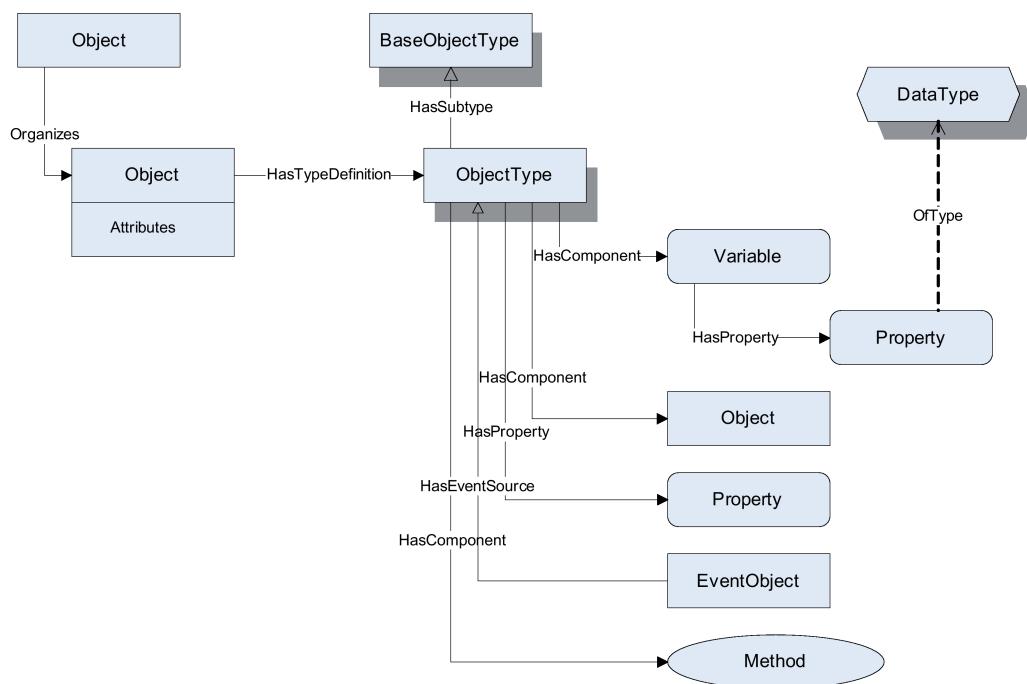


Figure 2-69 Symbols of the OPC UA address space

2.13.5 Board object and PROFINET IO controller

The board name of a PROFINET IO controller

All protocol-specific objects are always assigned to a board, in PROFINET IO this is the PROFINET IO controller. The board name is the PNIO controller configured in STEP 7 or the TIA Portal to identify the module.

Syntax

ctrl<slot>

Explanations

ctrl

Identifier for the PNIO controller

<slot>

Slot in the "Station Configuration Editor"

Example:

`ctrl112`

identifies the PNIO controller in slot 12 of the "Station Configuration Editor"

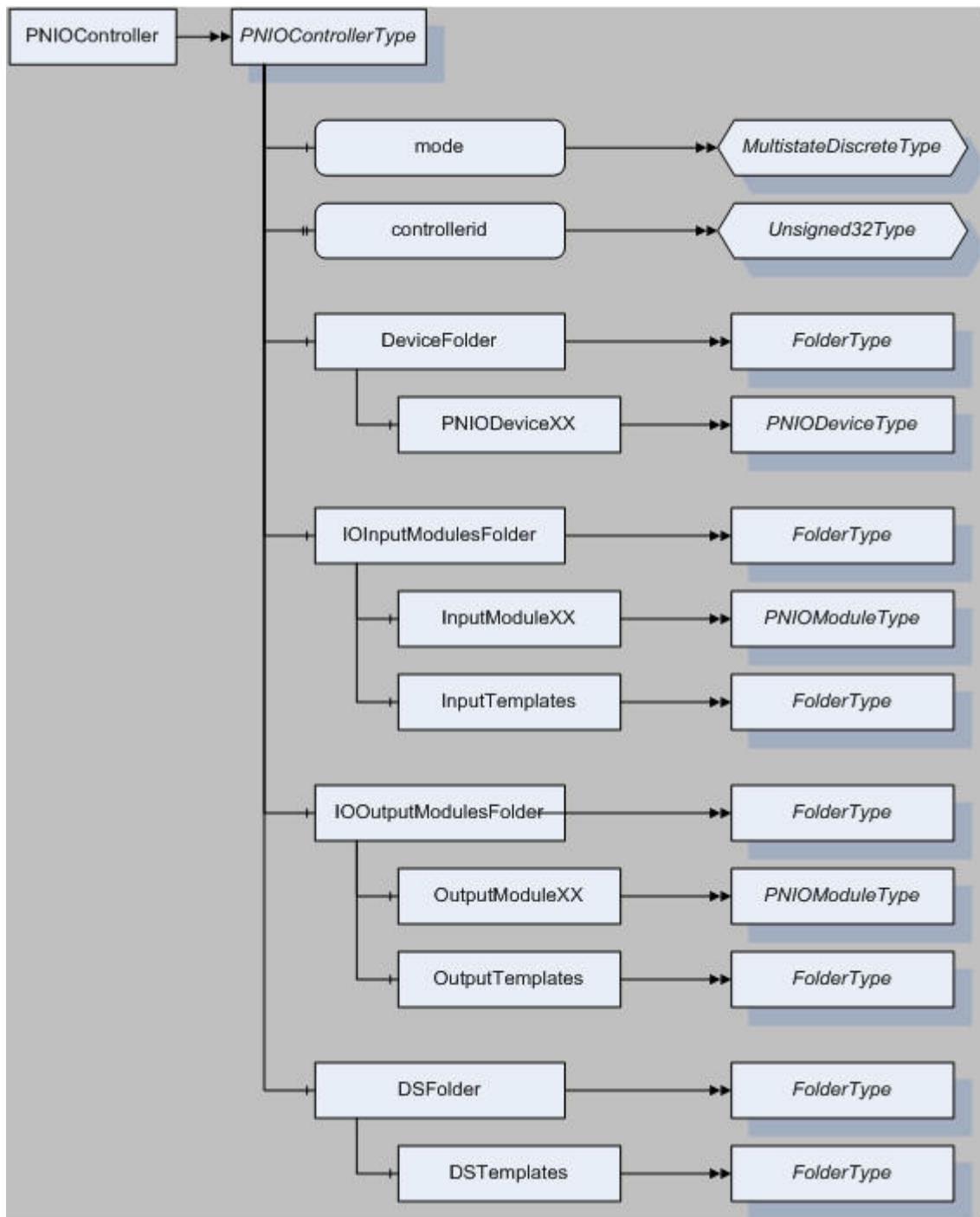


Figure 2-70 PROFINET IO controller class tree

2.13.6 Data variables of the PROFINET IO controller

Syntax for the data variables of the PROFINET IO controller

```
<controller>.<datavariable>
```

Explanations

<controller>

Protocol-specific connection name. The connection name is specified in the configuration. In the PNIO protocol, the connection name is the configured name of the communications module.

<datavariable>

Data variable	Description							
mode	Query or set the mode of the controller. Data variable of the UA type "MultistateDiscreteType", read and write. <table border="1" style="margin-left: 20px;"> <tr> <td>0</td><td>OFFLINE</td></tr> <tr> <td>1</td><td>CLEAR</td></tr> <tr> <td>2</td><td>OPERATE</td></tr> </table> When writing a new operating state, the permitted operating state changes must be kept in mind. These are as follows: OFFLINE ↔ CLEAR ↔ OPERATE If an error occurs, a communications error is returned. The CLEAR state is generally only implemented by DP slaves that are emulated as PROFINET IO devices via an IE-PB Link (link between Ethernet and PROFIBUS). Depending on the configuration value AutoOnline, the controller is automatically changed to the OPERATE mode by the OPC server the first time a client logs on. When the last client logs off, the controller is automatically changed to OFFLINE mode regardless of the configuration value AutoOnline. If a client crashes, this is detected by the UA protocol stack according to the keepalive mechanisms negotiated between client and server; after the selected wait time, the client is automatically logged off from the protocol stack. When the OPC server is shut down, the controller is closed.		0	OFFLINE	1	CLEAR	2	OPERATE
0	OFFLINE							
1	CLEAR							
2	OPERATE							
controllerid	Slot in the "Station Configuration Editor" Property of the type "UInt32", read-only							

Examples:

```
ctrl4.mode
ctrl4.controllerid
```

2.13.7 Device objects

Communication partners of the PROFINET IO controller are the PROFINET IO devices that have the input and output submodules.

Syntax

ctrl<slot>.dev<devicenumber>

Explanations

ctrl

Identifier for the PNIO controller

<slot>

Slot in the "Station Configuration Editor"

dev

Identifier for a PNIO device

<devicenumber>

Device number according to the STEP 7 or TIA Portal configuration

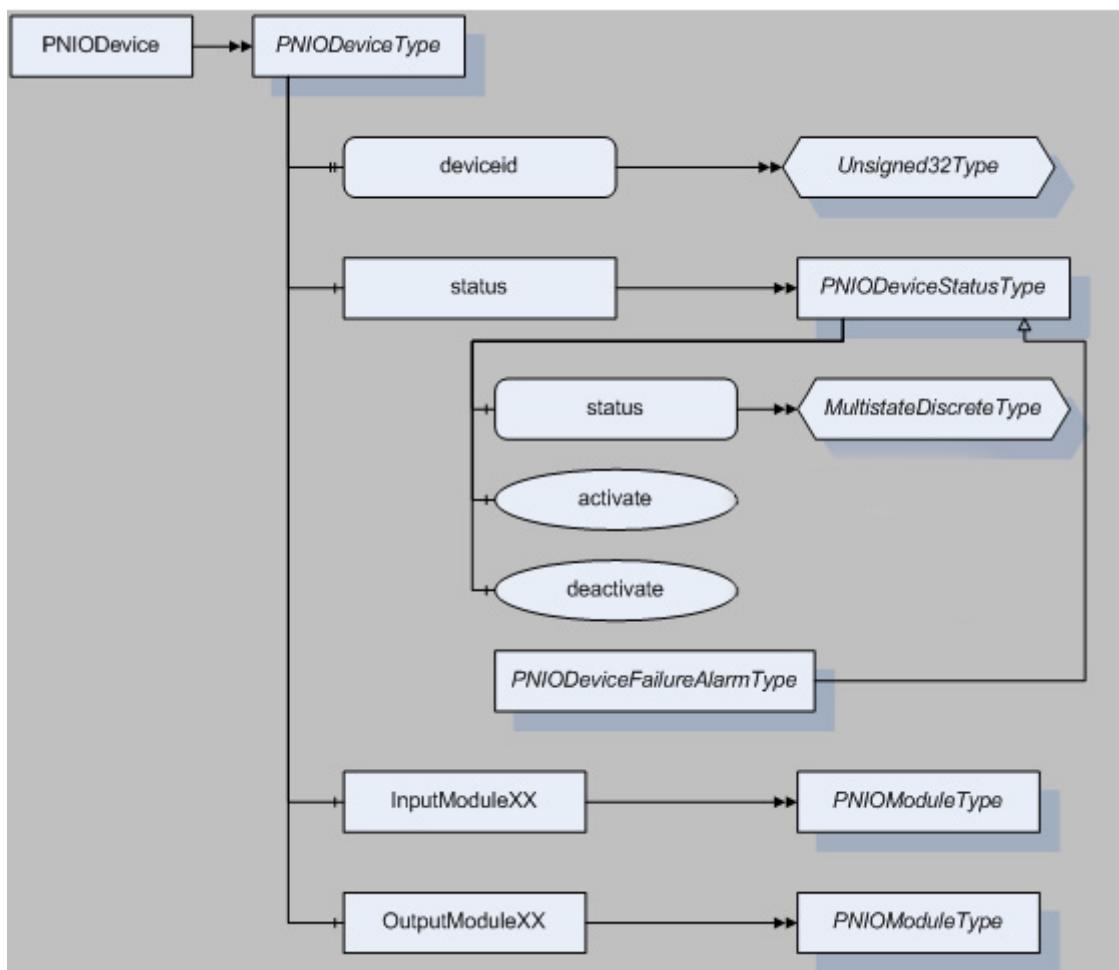


Figure 2-71 Device object

The device object of the PROFINET IO controller class tree is shown in Figure 2-48. Within the framework of addressing, the PROFINET IO devices have less significance and are used by the protocol stack only during connection establishment. On the OPC server-client

interface, the PROFINET IO devices are addressed solely via their modules. Even when a command (for example Device activate/deactivate) affects all modules of a PNIO device, the command only needs to address any one module of this PNIO device.

From the configuration, the OPC server obtains the I/O and diagnostics addresses assigned to this PNIO device. These are unique and used for the OPC UA Nodelds and communication.

All data variables of a device object, however, receive the device name configured in STEP 7 in the additional UA attribute "DisplayName" for the device part of the name.

The PROFINET IO device object is responsible for providing global status information about the PNIO device. The PNIO device status is managed in a lower-level object and also has two methods to activate and deactivate the PNIO device with all its modules.

2.13.8 Data variables and methods of the device object

Syntax of the data variables and methods of the device object

There are four possibilities:

- `ctrl<slot>.dev<devicenumber>.<datavariable>`
- `ctrl<slot>.dev<devicenumber>.<method>`
- `ctrl<slot>.dev<devicenumber>.status.<datavariable>`
- `ctrl<slot>.dev<devicenumber>.status.<method>`

Explanations

ctrl

Identifier for the PNIO controller

<slot>

Slot in the "Station Configuration Editor"

dev

Identifier for a PNIO device

<devicenumber>

PNIO device number according to the STEP 7 or TIA Portal configuration

status

Status object of the PNIO device

<datavariable> / <method>

Data variable / method	Description
deviceid	PNIO device number from STEP 7 or TIA Portal Property of the type "UInt32", read-only
status	Status of the PNIO device Data variable of the UA type "MultistateDiscreteType", read-only

Data variable / method	Description	
	0	FAILURE
	1	OPERATE
activate()	Activate PNIO device The "activate()" OPC UA method can only be written. It does not have any arguments. Successful completion of the method call indicates that the activation was successfully transported to the PNIO device. Whether or not the PNIO device is really active from the perspective of the PNIO controller, must be checked using the "status" data variable.	
deactivate()	Deactivate PNIO device The "deactivate()" OPC UA method can only be written. It does not have any arguments. Successful completion of the method call indicates that the deactivation was successfully transported to the PNIO device. Whether or not the PNIO device is really deactivated from the perspective of the PNIO controller, must be checked using the "status" data variable.	

Examples:

```

ctrl4.dev32.deviceid
ctrl4.dev32.status.status
ctrl4.dev32.status.activate()
ctrl4.dev32.status.deactivate()

```

2.13.9 PROFINET IO module objects**General**

The actual PNIO communication takes place via IO submodules. An IO submodule is always assigned to a PNIO device. The OPC server receives the type and length of an I/O submodule from the STEP 7 or TIA Portal configuration.

Syntax of the PNIO submodule objects

You have two options:

- `ctrl<slot>.i<address>`
- `ctrl<slot>.q<address>`

Explanations**ctrl**

Identifier for the PNIO controller.

<slot>

Slot in the "Station Configuration Editor".

q

Identifier for an output. Outputs can be read and written.

i

Identifier for an input. Inputs are read only.

<address>

Logical address of the IO submodule.

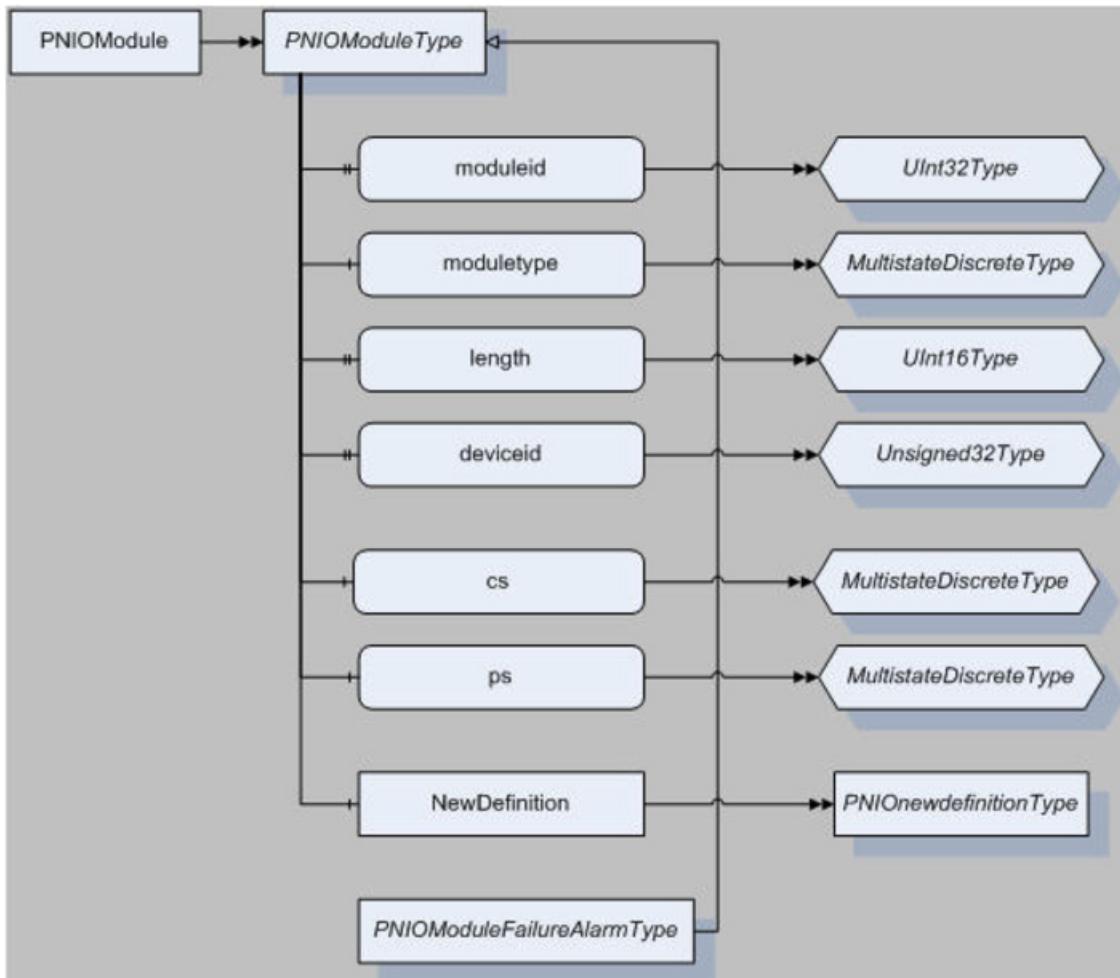


Figure 2-72 PROFINET IO module objects

2.13.9.1 Data variables of the PROFINET IO modules

Syntax for data variables of the PNIO module objects

You have two options:

- `ctrl<slot>.i<address>.<IOModulevariable>`
- `ctrl<slot>.q<address>.<IOModulevariable>`

Explanations

ctrl

Identifier for the PNIO controller.

<slot>

Slot in the "Station Configuration Editor".

q

Identifier for an IO output submodule. Outputs can be read and written.

i

Identifier for an IO input submodule. Inputs are read only.

<address>

Logical address of the IO submodule.

<IOModulevariable>

moduleid	Submodule address from STEP 7 or TIA Portal Property of the type "UInt32", read-only	
Module type	IO submodule type from STEP 7 or TIA Portal Data variable of the type "MultistateDiscreteType", read-only	
0	UNKNOWN	
1	I (Input) The value currently being delivered by the PROFINET IO UA server is I.	
2	Q (Output) The value currently being delivered by the PROFINET IO UA server is O.	
3	I (Diag) The value currently being delivered by the PROFINET IO UA server is I. No cyclic data, only data records.	
Variables for information purposes		
length	Submodule length in bytes from STEP 7 or TIA Portal Property of the type "UInt16", read-only Special module lengths: 0 No module from the I/O address range (modulename = 3)	
deviceid	PNIO device number belonging to the IO submodule from STEP 7 or TIA Portal Property of the type "UInt32", read-only	

Examples:

```
ctrl4.q27.moduleid
ctrl4.q28.modulename
ctrl4.q28.length
ctrl4.q28.deviceid
```

2.13.9.2 IO data variables of the PROFINET IO modules**General**

Reading and writing NodeIDs of cyclic IO data is the most common application. This allows simple access to the PNIO address space. The user must specify the PNIO controller name and the logical address of the IO submodule from the project engineering.

Syntax for data variables of the PNIO module objects

You have two options:

- `ctrl<slot>.i<address>.<offset>{,<PNIOtype>{,<quantity>}}`
- `ctrl<slot>.q<address>.<offset>{,<PNIOtype>{,<quantity>}}`

Explanations**ctrl**

Identifier for the PNIO controller.

<slot>

Slot in the "Station Configuration Editor".

q

Identifier for an IO output submodule. Outputs can be read and written.

i

Identifier for an IO input submodule. Inputs are read only.

<address>

Logical address of the IO submodule.

<offset>

Byte address in the data area of the IO submodule to be addressed.

<PNIObjectType>

Data type	OPC UA Datatype	Note
x<bitaddress>	Boolean	Bit (bool) In addition to the byte offset in the area, the <bitaddress> must also be specified in the byte. Range of values 0 to 7
b	Byte Byte string	Byte (unsigned) Used as the default if no <PNIObjectType> is specified.
w	UInt16	Word (unsigned)
dw	UInt32	Double word (unsigned)
lw	UInt64	Long word (unsigned)
c	SByte	Byte (signed)
i	Int16	Word (signed)
di	Int32	Double word (signed)
li	Int64	Long word (signed)
r	Float	Floating point (4 bytes)
lr	Double	Floating point (8 bytes)
s<stringlength>	String	The <stringlength> reserved for the string must also be specified. When writing, shorter strings can also be written whereby the transferred data length is always the reserved string length in bytes. The unnecessary bytes are filled with the value 0.

<quantity>

Number of elements. The data type of the variable is an array with elements of the specified format. Specifying a number of array elements causes an array of the corresponding type to be formed even when only a single array element is addressed.

Examples:

- | | |
|------------------------|---|
| ctrl4.i10.0,w | Input word starting at byte address 10 |
| ctrl4.q17.3 | Output byte starting at byte address 3 (BYTE default) |
| ctrl4.i10.0,x0 | Input bit starting at byte address 0, bit 0 |
| ctrl4.q17.1,x4,16 | Array of 16 output bits starting at byte address 1, bit 4 |
| ctrl4.i27355.100,s32,3 | Array with 3 strings each with 32 characters |

2.13.9.3 Data variables for IOPS and IOCS**General**

With the items for the status (IOPS and IOCS), an OPC client can query or set the status bytes of a logical address.

The following schematic shows the data status and flow of values in the process image.

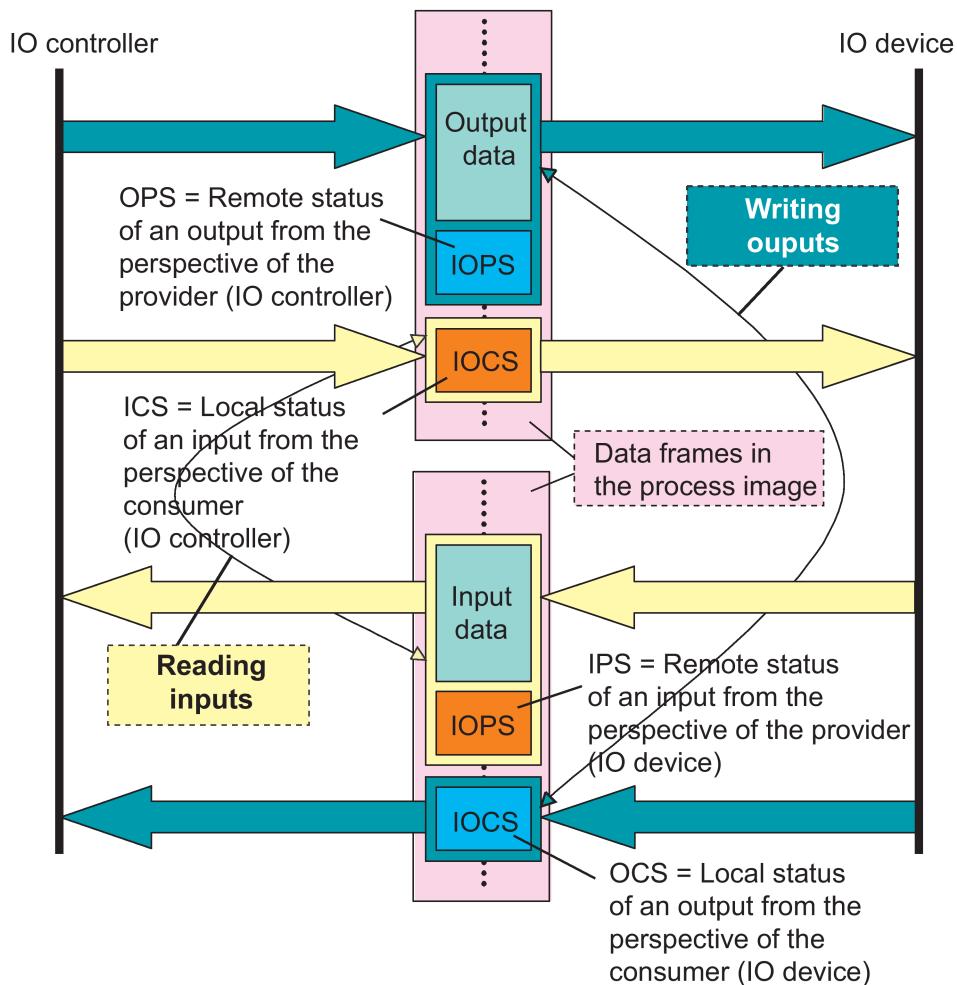


Figure 2-73 Overview of the data status in the process image in PROFINET IO communication between IO controller and an IO device.

When the OPC client writes output data to the process image, it includes the remote status of the output from its perspective as provider with "OPS".

The device includes its own local status of the output from its perspective as consumer with "OCS".

When the OPC client reads input data from the process image, it includes its own local status of the input from its perspective as consumer with "ICS".

The device includes the remote status of the input from its perspective as provider with "IPS".

Note

Read access to cyclic output data

The reading, monitoring or activation of OPC items/nodes that map cyclic output data brings about a one-time or constant update of the underlying PROFINET IO protocol interface with the data contained in the OPC server cache including the provider status (PNIO_data_write()). Please remember that after the return of a PROFINET IO device or module, this output data is transferred to the device again after the point in time of the update.

Syntax of the data variables for IOPS and IOCS

There are four options available:

IPS variable:

- `ctrl<slot>.i<address>.ps`

OPS variable

- `ctrl<slot>.q<address>.ps`

ICS variable:

- `ctrl<slot>.i<address>.cs`

OCS variable:

- `ctrl<slot>.q<address>.cs`

Explanations

ctrl

Identifier for the PNIO controller.

<slot>

Slot in the "Station Configuration Editor".

q

Identifier for an IO output submodule. Outputs can be read and written.

i

Identifier for an IO input submodule. Inputs are read only.

<address>

Logical address of the IO submodule.

ps

Provider status of an IO submodule.

Cs

Consumer status of an IO submodule.

Name	Explanation
IPSVariable	<p>Remote status of an input address</p> <p>Data variable of the UA type "MultistateDiscreteType", read-only</p> <p>The provider of an input is the device. The device also provides the status of the input data.</p> <p>The IPS is monitored by polling.</p> <p>The use of the item is not absolutely necessary for an OPC application because the content of the item is reflected in the OPC quality of the linked input items with the same address.</p>
ICSVariable	<p>Local status of an input address</p> <p>Data variable of the UA type "MultistateDiscreteType", writable and readable</p> <p>Default: <i>GOOD</i> = 0</p> <p>The consumer of an input is the OPC client. The OPC client also informs the device of the status of the input data from its perspective. Normally, this status is always <i>GOOD</i>.</p> <p>When IO data is read, the last local status of an input address written by the OPC application is always included (written to the partner device). If the OPC application has not created the item or not written a value to it, the value <i>GOOD</i> is transferred when the corresponding input data is read.</p> <p>The status is read by the OPC client over the cache.</p>
OCSVariable	<p>Local status of an output address</p> <p>Data variable of the UA type "MultistateDiscreteType", read-only</p> <p>The consumer of an output is the device. The device also includes the status of the output data from its perspective.</p> <p>Reading the status is possible if the OPC client can first (or the same time) write a linked output item with the same address.</p> <p>The OCS is monitored by polling and the last output data written by the OPC client is always transferred to the function at the same time. If the OPC client was not yet able to write a value successfully, reading from the device results in an error or the OPC quality <i>bad</i>.</p>
OPSVariable	<p>Remote status of an output address</p> <p>Data variable of the UA type "MultistateDiscreteType", writable and readable</p> <p>Default: <i>GOOD</i> = 0</p> <p>The provider of an output is the OPC client. The OPC client also informs the device of the status of the output data from its perspective. Normally, this status is always <i>GOOD</i>.</p> <p>When IO data is written, the last local status of the output data written by the OPC application and the output data of an output address are also included (written). If the OPC application has not created the IO item or not written an output value successfully, the value <i>GOOD</i> is transferred when the corresponding output data is written.</p> <p>When the OPC client writes a new status, the data value is used that was successfully written previously (or the same time) with the address of the linked output item.</p> <p>If the OPC client has not yet written a value successfully, writing the item results in the special return value OPCUA_GoodClamped. The written status is then written by the corresponding output item with the next successful write.</p>

Structure of the status value

The data variable of the UA type "MultistateDiscreteType" always has the following structure:

0 = GOOD

1 = BAD

In a normal situation, ctrl.xxx.cs and ctrl.xxx.ps do not need to be written to the UA interface, since these values are set by writing/reading the variables. The status is only set explicitly when the device needs to be informed of the lack of availability or processing possibility.

Examples:

ctrl1.i10.ps	Remote state of the logical input address 10
ctrl1.q20.cs	Local state of the logical output address 20

2.13.9.4 Data record data variables of the PROFINET IO modules

General

Using the data record data variables ("data record" = "dr"), device-specific information can be read out or written. The possible addresses are linked to configured module or diagnostics addresses.

Data records and length information

The syntax distinguishes data records with and without length information.

The length information is always required when writing data records where the variable does not address the entire data record as a byte string. The length of the data to be written is then known. This is the case when subelement addressing is used.

If you write without length information and without subelement addressing, the OPC UA server calculates the length of the data record based on the number of bytes to be written.

Without length information, reading and writing data records is restricted to 480 bytes.

Subelement addressing

The structuring of user data records is supported at the client end by the subelement addressing. Here, subareas of the data record are addressed.

Only readable data variables: Addressing a subelement is possible without length information of the data record. Since the device-specific length of the data record is not known in advance or can even change, subelements even outside the actual received length can be used. The OPC UA status code does, however, then show an error.

Readable and writable data variables: The writing of individual subelements or several subelements at the same time over the entire data record is consistent in the range specified with the length information. Any gaps are filled with the last written values (not read values) or with "0" values if this data record has never been written previously.

syntax of the data record data variables of the PROFINET IO submodules

There are four possibilities:

- `ctrl<slot>.i<address>.dr<number>{.<DRlength>,<offset>,<PNIOtype>{,<quantity>}}`
- `ctrl<slot>.q<address>.dr<number>{.<DRlength>,<offset>,<PNIOtype>{,<quantity>}}`
- `ctrl<slot>.i<address>.dr<number>.<offset>,<PNIOtype>{,<quantity>}`
- `ctrl<slot>.q<address>.dr<number>.<offset>,<PNIOtype>{,<quantity>}`

Explanations

ctrl

Identifier for the PNIO controller.

<slot>

Slot in the "Station Configuration Editor".

q

Identifier for an IO output submodule. Outputs can be read and written.

i

Identifier for an IO input submodule. Inputs are read only.

<address>

Logical address of the IO submodule.

dr

Identifier for data record data variables.

<number>

Data record number of the IO submodule.

<DRlength>

Data record length of the IO submodule.

<offset>

Byte address in the data record for the IO submodule to be addressed.

<PNIOtype>

Data type	OPC UA Datatype	Note
x<bitaddress>	Boolean	Bit (bool) In addition to the byte offset in the area, the <bitaddress> must also be specified in the byte. Range of values 0 to 7
b	Byte Byte string	Byte (unsigned) Used as the default if no <PNIOtype> is specified.
w	UInt16	Word (unsigned)
dw	UInt32	Double word (unsigned)
lw	UInt64	Long word (unsigned)
c	SByte	Byte (signed)

Data type	OPC UA Datatype	Note
i	Int16	Word (signed)
di	Int32	Double word (signed)
li	Int64	Long word (signed)
r	Float	Floating point (4 bytes)
lr	Double	Floating point (8 bytes)
s<stringlength>	String	The <stringlength> reserved for the string must also be specified. When writing, shorter strings can also be written whereby the transferred data length is always the reserved string length in bytes. The unnecessary bytes are filled with the value 0.

<quantity>

Number of elements. The data type of the variable is an array with elements of the specified format. Specifying a number of array elements causes an array of the corresponding type to be formed even when only a single array element is addressed.

Examples:

ctrl1.i10.dr61450

Data record diagnostics information of the device with a logical input address 10. Entire data record, Byte[], readable and writable

ctrl1.q99.dr32788,100.8,w,3

Data record with data record length, subelement addressing starting at byte 8, UInt16[3], readable and writable

ctrl1.q99.dr32788.8,dw

Data record without data record length, subelement addressing starting at byte 8, UInt32, read-only

2.13.10 PROFINET IO OPC UA templates

2.13.10.1 Template data variables

With the IO data variables and the data record data variables for the OPC UA PNIO protocol, you have flexible setting options with which you can obtain the process data of the plant in the data formats you require.

The wide variety of addressing options cannot, however, be put together in a fully browsable namespace. Even an IO submodule with the length of a single byte has approximately 40 different data format options - starting with Byte, SByte, arrays with one element of these, individual bits, arrays of bits with up to 8 array elements starting at different bit offsets.

The OPC UA server therefore supports the user with template data variables in the PNIO namespace. In a typical OPC UA client text input box, these templates can be converted to valid ItemIDs simply by changing a few characters.

Example:

ctrl4.i<x>.<o>,dw	Template for a UInt32 of an input module
-------------------	--

By replacing <x> with the module address and <o> the offset within the module, you obtain a valid Nodeld. → ctrl4.i8.15,dw

Further example:

ctrl4.i<x>.dr<dr>,<l>.<o>,c,<c>	Template for a data record variable, Char[]
→ ctrl4.i125.dr61450,100.0,c,100	

One great advantage of this concept is that it can be used by practically all OPC UA clients without any adaptation of the client being necessary. A further advantage is that the implementation involves little effort and that this concept can also be transferred to other protocol servers including COM of the SIMATIC NET OPC environment.

Note

The usability of OPC UA PNIO template data variables can be enabled and disabled in the "Communication Settings" configuration program in "OPC protocol selection".

Template data variables within the tree structure

The template data variables are sorted beside the corresponding folders in the namespace representation so that they can be used easily when required.

2.13.10.2 Directory of the template data variables

Special use of some of the attributes of the template data variables

In the template variables, the "Nodeld", "BrowseName" and "Description" attributes are used particularly intelligently.

Syntax of the template data variables

You have two options:

- ctrl<slot>.i<x>,<o>,<PNIOtypetemplate>,<c>
- ctrl<slot>.q<x>,<o>,<PNIOtypetemplate>,<c>

Explanations

ctrl

Identifier for the PNIO controller.

<slot>

Slot in the "Station Configuration Editor".

q

Identifier for an IO output submodule. Outputs can be read and written.

i

Identifier for an IO input submodule. Inputs are read only.

<x>

Placeholder for the number of the submodule containing the input or output area.

<o>

Placeholder for byte offset in the address space of the IO submodule.

<PNIOtypetemplate>

A PNIO template data type is converted to the corresponding OPC UA data type on the OPC UA server. The following table lists the type identifier and the corresponding OPC data type in which the variable value can be represented.

Data type	OPC UA data type	Note
x<bit>	Boolean	Placeholder for the bit offset (0 to 7)
b	Byte	Placeholder for byte (unsigned)
w	UInt16	Placeholder for word (unsigned)
dw	UInt32	Placeholder for double word (unsigned)
lw	UInt64	Placeholder for long word (unsigned)
c	SByte	Placeholder for byte (signed)
i	Int16	Placeholder for word (signed)
di	Int32	Placeholder for double word (signed)
li	Int64	Placeholder for long word (signed)
r	Float	Placeholder for floating point (4 bytes)
lr	Double	Placeholder for floating point (8 bytes)
s<sl>	String	Placeholder for the length of the string

<c>

Placeholder for the number of elements.

Examples:

NodeId	BrowseName	Description
ctrl.i<x>.<o>,x<bit>,<c>	template input bit	<x> Address of the submodule <o> Offset within the submodule <bit> Bit offset (0...7) <c> Size of the array
ctrl.i<x>.<o>,b<c>	template input byte	<x> Address of the submodule <o> Offset within the submodule <c> Size of the array
ctrl.i<x>.<o>,w,<c>	template input UInt16	<x> Address of the submodule <o> Offset within the submodule <c> Size of the array
ctrl.i<x>.<o>,s<sl>,<c>	template input string	<x> Address of the submodule <o> Offset within the submodule <sl> Length of the string <c> Size of the array

Nodeld	BrowseName	Description
ctrl.q<x>.<o>,x<bit>,<c>	template output bit	<x> Address of the submodule <o> Offset within the submodule <bit> Bit offset (0...7) <c> Size of the array
ctrl.i<x>.dr<dr>,<l>.<o>,b,<c>	template input dataset byte	<x> Address of the submodule <dr> Data record number <l> Length of the file <o> Offset within the submodule <c> Size of the array

2.14 Server diagnostics

What is server diagnostics?

OPC client and OPC server diagnostics information is available as of "SIMATIC NET PC Software V8.0" for OPC Data Access (COM, XML, UA). You can, for example, read out the number of connected clients and the number of active OPC groups. Version data of the OPC servers can also be read out.

Note

OPC server diagnostics (in the namespace under SERVER:) is available only with the OPC.SimaticNET server with several selected protocols (in "Communication Settings" ... Protocol selection"). With the individual protocol Data Access OPC servers such as OPC.SimaticNET.DP and OPC.SimaticNET.PD it is not. It is also not available with the OPC Alarms & Events servers.

The restriction also applies to OPC XML Data access.

2.14.1 Protocol ID

What is the protocol ID?

The protocol ID for the server diagnostics is called SERVER:

This ID is used for OPC DA and OPC XML DA.

For OPC UA DA, the server diagnostics is specified by the standard; you will find the Diagnose Nodes in namespace 0 under Server.

2.14.2 OPC DA server diagnostics items

Which OPC DA server diagnostics items exist?

The OPC servers provide a hierachic namespace in which the various diagnostics items are located. Based on the OPC UA standard, the diagnostics information is structured as follows:

Server:

Capabilities	MinSupportedUpdateRate
DiagnosticsSummary	CurrentSessionCount CumulatedSessionCount CurrentSubscriptionCount CumulatedSubscriptionCount
VendorInfo	ComponentVersion VendorInfoString

Diagnostics ItemID	Data type	Description
Server\Capabilities\MinSupportedUpdateRate	VT_UI4	Minimum update rate for an OPC group
Server\DiagnosicsSummary\CurrentSessionCount	VT_UI4	Current number of OPC client sessions (or number of OPC server instances)
Server\DiagnosicsSummary\CumulatedSessionCount	VT_UI4	Total number of OPC client sessions begun since the OPC server was started up. Some sessions may, however, have been closed again in the meantime.
Server\DiagnosicsSummary\CurrentSubscriptionCount	VT_UI4	Current number of OPC DA subscriptions, in other words, of OPC groups that are activated and can send OnDataChange callbacks to the OPC client.
Server\DiagnosicsSummary\CumulatedSubscriptionCount	VT_UI4	Total number of OPC DA subscriptions begun since the OPC server was started up. Some may, however, have been closed again in the meantime.
Server\VendorInfo\ComponentVersion	VT_BSTR	OPC server version string, for example "V03.08.00.00_01.14.00.01"
Server\VendorInfo\VendorInfoString	VT_BSTR	OPC server name, for example "SIMATIC NET OPC-Server DataAccess-V2.05/3.0 (C) SIEMENS AG 2010"

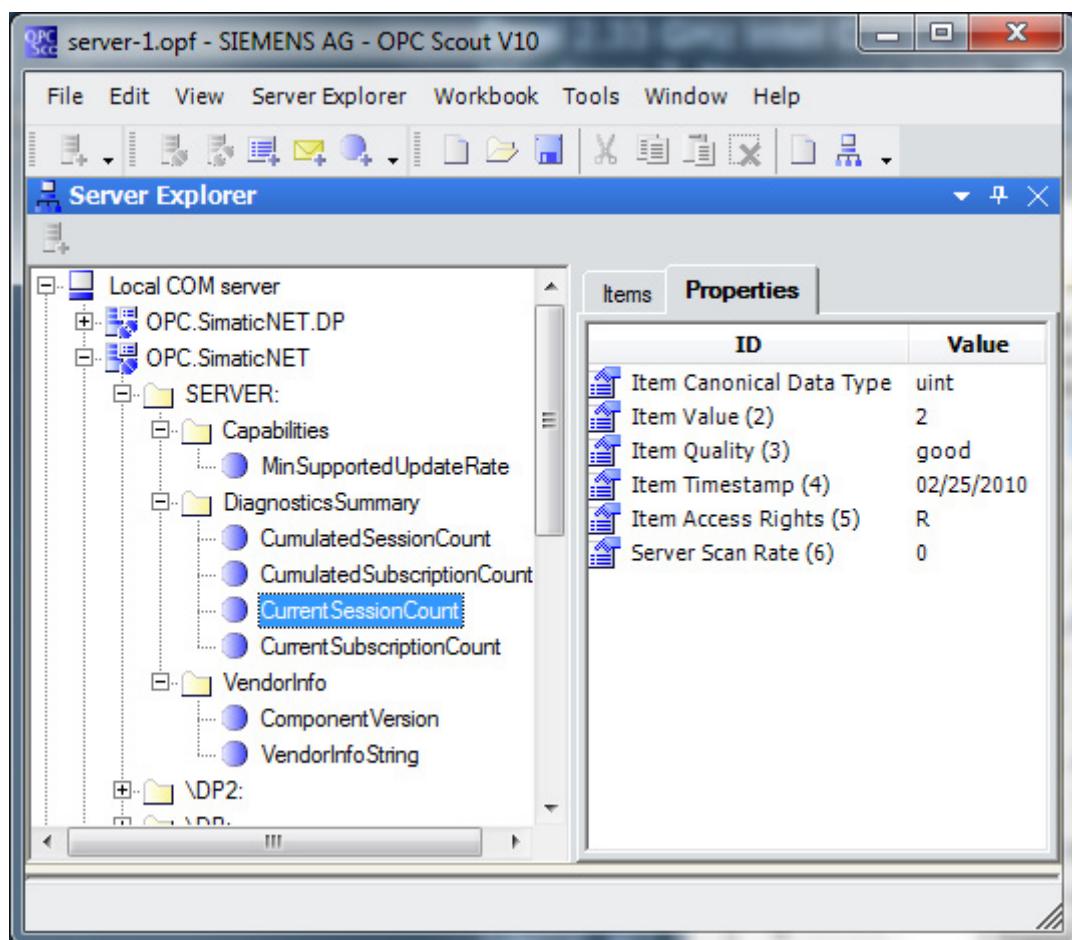


Figure 2-74 OPC server diagnostics items for COM/DCOM OPC.SimaticNET

What do the OPC XML DA server diagnostics items look like?

The diagnostics items are the same as in the OPC DA server, except for MinSupportedUpdateRate because there are no callbacks in OPC XML DA. Since no group registration is used either, the number of sessions CurrentSessionCount is implemented as follows.

In this context, the session means a communication relation between a client and a server that lasts for a longer period of time. Of the XML DA methods, only the subscription is comparable. All other methods (GetStatus, Browse, Read, Write) single accesses that cannot be assigned to a session. XML DA subscriptions are created by calling "Subscribe". The client obtains the data of these subscriptions using PolledRefresh. At the same time, it is guaranteed that all subscriptions of a list count as the same client and therefore to the same session. The SessionCount is calculated from the number of existing subscriptions of a client.

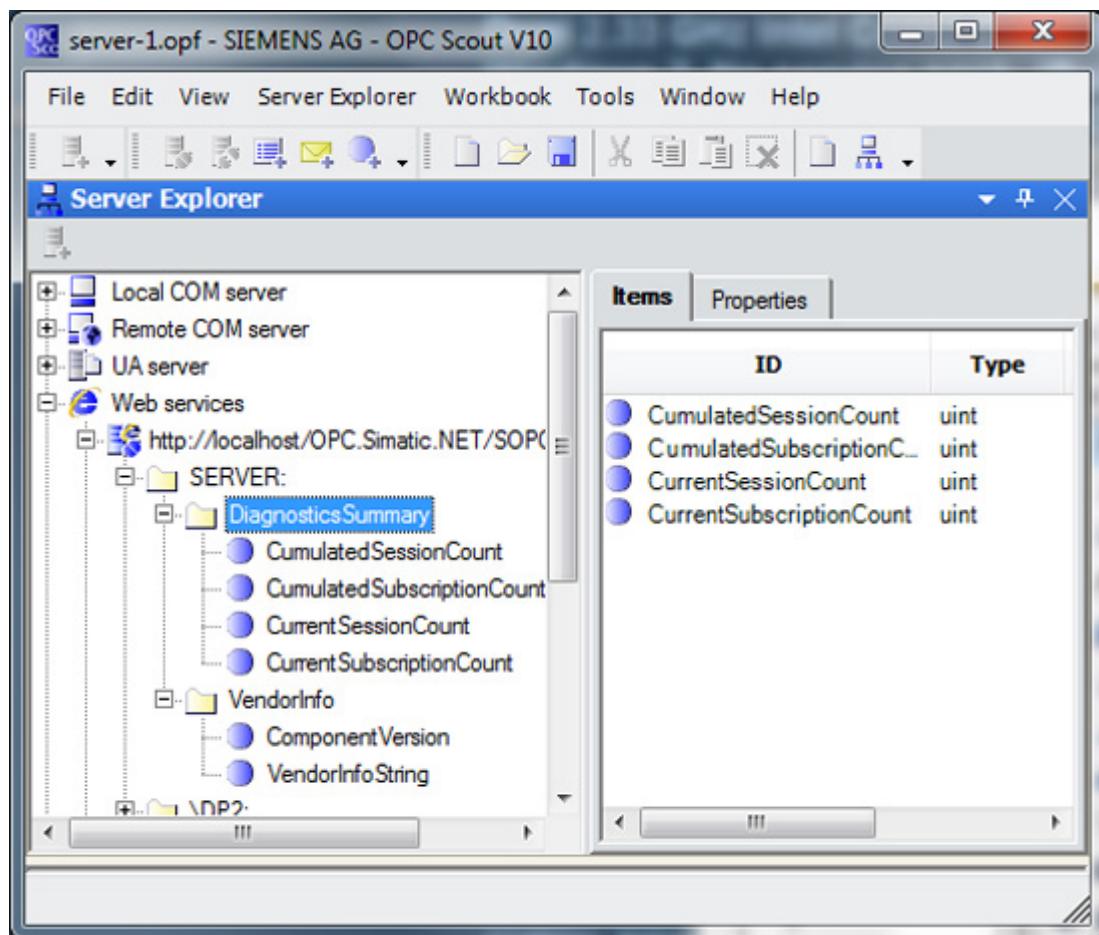


Figure 2-75 OPC server diagnostics items for XML DA
<http://localhost/OPC.Simatic.NET/SOPCWeb.asmx>

What does OPC UA server diagnostics provide?

The OPC UA server diagnostics serves as the model and provides the same as well as other diagnostics and information nodes. You will gain an impression from the following figure. You should also refer to the OPC UA specifications.

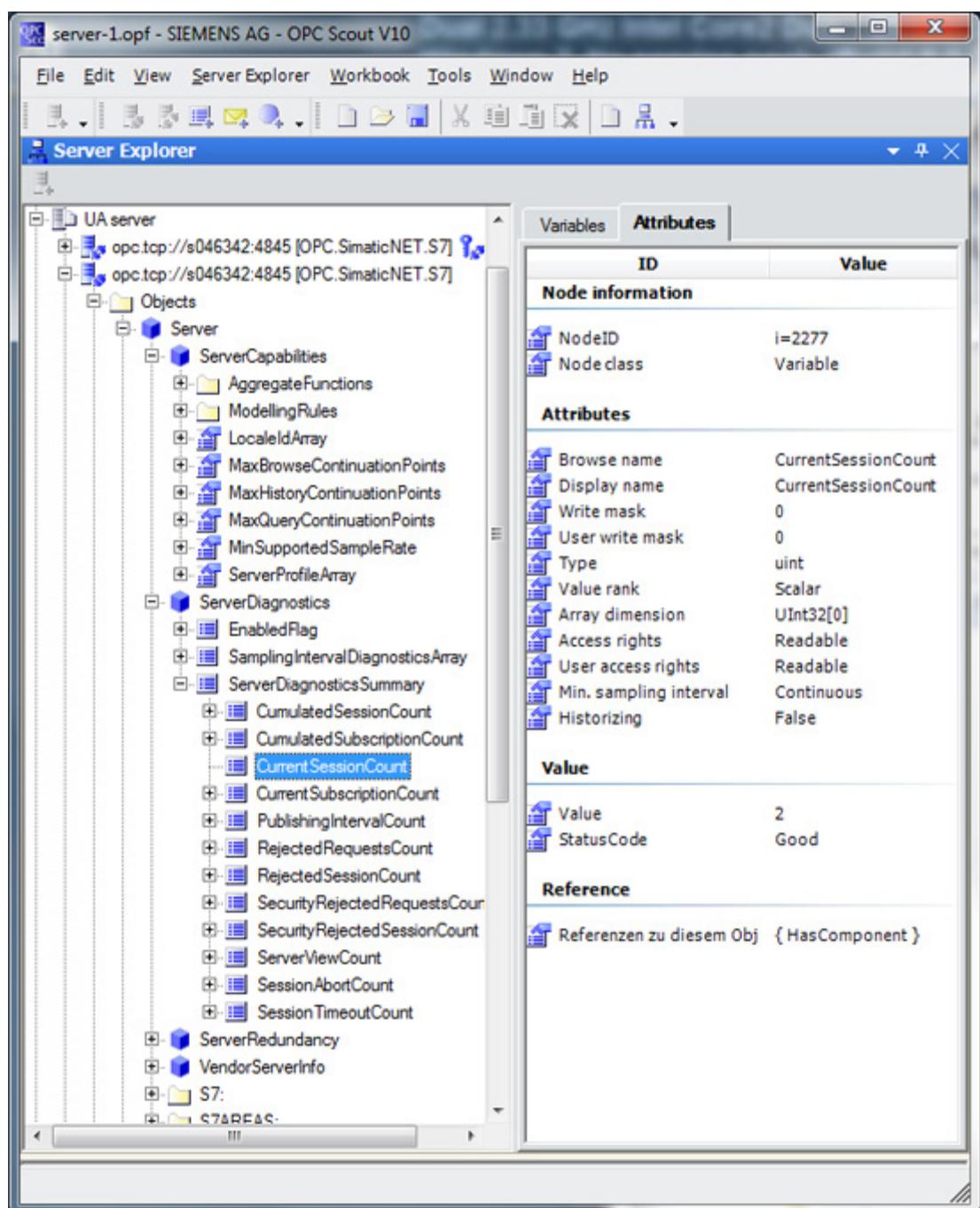


Figure 2-76 OPC UA server diagnostics nodes

2.15 Buffer-oriented services with the OPC interface

Introduction

This section describes the use of buffer-oriented services with OPC. You will learn how data buffers are mapped on OPC variables. The section also describes the special points to note when using TCP/IP native in conjunction with the buffer-oriented services.

2.15.1 Using buffer send/receive services

For the transfer of large data packets, S7 communication and S5compatible communication over Industrial Ethernet provide buffer send/receive services. Data packets are sent between communications partners. The transfer of the data puts load on the network only when a partner explicitly initiates a send job.

Using the OPC Server for SIMATIC NET, you can structure blocks of data. This allows individual parts of a data packet to be assigned to OPC items.

Note

The S7 buffer send/receive services are currently available only for devices of the S7400 and M7 series and PC stations. The S5 buffer send/receive services are available for practically all devices of the S5 and S7 series and PC stations. Note the information on the communication partner relating to future support of the bufferoriented services in the readme file in the main folder of the SIMATIC NET CD.

2.15.2 Properties of buffer-oriented communication

Supported services

In the buffer-oriented services, data buffers are transferred by the sender to a receiver using the communications system.

The following protocols support buffer-oriented services:

- S7 communication (BSEND/BRECEIVE)
- Open communication services (SEND/ RECEIVE) over Ethernet
- Open communication services (SEND/ RECEIVE) over PROFIBUS (SDA, SDN/Indication)

The distinguishing feature of the buffer-oriented services is that data is transferred only when the sender initiates the transfer. The receiver cannot initiate data transfer.

Activities of the sender and recipient when exchanging data buffers

Transmitter

- Puts together the send buffer and its content
- Sends the buffer on a connection to a communication partner
- Receives a confirmation of the result of the data transfer

Receiver

- Prepares a receive buffer for a connection
- Is informed when a partner sends a data buffer to it
- Sends an acknowledgment to the sender
- Evaluates the received data

2.15.3 Mapping data buffers on OPC variables

Differences between send and receive items

The OPC Data Access interface recognizes only process variables. To allow the advantages of buffer-oriented services to be used with OPC, the buffers must be mapped on OPC items:

Send item (S7 communication: BSEND, open communication: Send)

- An OPC item represents a send buffer or part of a send buffer.
- When the OPC item is written (synchronous / asynchronous), a write job is initiated on the network.
- If several items representing part of a buffer are written at once in a group operation, the entire send buffer made up of all parts is put together first and then sent.
- The read returns the last data sent from the send buffer. If nothing was sent, the item may possibly be readable, however is has the quality BAD.

Receive item (S7 communication: BRCV, open communication: RECEIVE)

- An OPC item represents a receive buffer.
- If the OPC item is read from the device (synchronous/asynchronous), the communication module is ready to receive. This state exists until a data packet is received or the timeout for the specific connection elapses. If no data packet is received before the timeout, the quality of the OPC item is BAD.
- If the OPC item is monitored (active receive item in an active group), a receive buffer is set up permanently on the communication module. When a data packet is received, the OnDataChange callback signals this to the application if the data differ from the previous data received.
- Receive items cannot be written.

2.15.4 Using buffer-oriented services

Handling send and receive items

Send items should only be written; reading or monitoring send items returns only the previously written data.

Receive items should be monitored; in other words, the receive item should exist as an active item in an active group. Regardless of the implementation of a callback function, the cache of the OPC server is then updated on the arrival of every data packet.

Reading from the cache and directly from the device

Read jobs both synchronous or asynchronous should read the cache. The cache contains the last block of data received if the receive item is monitored. Using the cache also ensures that several clients reading the item at the same time obtain the same value.

Reads to device (directly to the device) are not useful since there is only a receive buffer available during the timeout interval of the connection (normally several seconds). A sender must send within this time or the data buffer will not be accepted by the receiver.

Update rate

Make sure that the OPC Server fetches the receive buffer faster than the sender sends it. The rate at which the receive buffers are fetched is specified by the group-specific *Update Rate* OPC parameter.

Depending on the protocol, received blocks of data will otherwise be overwritten without the OPC client being informed (S7, SEND/RECEIVE with ISO and RFC 1006) or a backlog of packets will form and the receiver will only receive obsolete data (SEND/RECEIVE with TCP/IP native).

2.15.5 Points to note when using buffer-oriented services with TCP/IP native

Meaning of the socket interface

The SEND/RECEIVE protocol for open communication with TCP/IP native is based on the Windows Socket services. No explicit buffers are transferred, but rather a continuous data stream is transmitted on the Socket interface. The protocol uses an interim buffer both at the send and receive end and incoming and outgoing data is managed as in a FIFO buffer (First-In First-Out).

TCP/IP native and OPC variables

The protocol properties described above, also affect the application with OPC variables:

- Even if the partner does not accept any data, send jobs are acknowledged positively until the buffer at the sending end overflows. Prevent memory overflow in the user program.
- The interim buffer for receiving data is always available regardless of whether or not a variable is monitored or a read job is sent.
- If the data is not fetched quickly enough at the receiver end, there will be a *data backlog* in the receive buffer. As a result, the receiver does not obtain the data read last but data that is no longer up to date. Prevent data backlogs in the user program.

2.16 Restricting access rights of the OPC variables

Specifying access rights in STEP 7 or NCM

In STEP 7 or NCM, you can activate the access rights in the properties dialog of the OPC server. As default, the access rights of the variables are set to Read/Write (RW) by activating the relevant check box but this can be changed when necessary to Read (R), Write (W) or None.

In the Item-specific Access Rights dialog box, you can define access rights that differ from the settings for all items. The rights entered in this dialog box then overwrite the default access rights.

Assign OPC item-specific access rights

Here, enter the access rights for one or more OPC items. The default rights defined for the other OPC items are not affected. The syntax is defined as follows:

<OPCItem>=<rights>

<OPCItem>

Specifies one or more OPC items according to the syntax defined in the OPC documentation. The use of alias names is possible. The following placeholders can be used:

- | | |
|---|--------------------------|
| * | any number of characters |
| ? | exactly one character |

<rights>

<i>RW</i>	Read and write access
<i>R</i>	Read-only access
<i>W</i>	Write-only access
<i>NONE</i>	neither write nor read access

Rules for evaluation

The specific rights assigned here have priority over the default rights you assigned in the access protection area. The specific rights assigned here are evaluated in the order in which they are entered here. If there is a multiple assignment for an OPC item, the last assignment is always valid.

Example:

```
DP:[CP 5611]Slave040_QB*=RW  
DP:[CP 5611]Slave040_QB1=R  
DP:[CP 5611]Slave040_QB2=W  
DP:[CP 5611]Slave040_QB1=W  
DP:[CP 5611]Slave040_QB1*=R
```

As a result of the entry, the following access rights, among others, are effective:

```
DP:[CP 5611]Slave040_QB2=W  
DP:[CP 5611]Slave040_QB1=R
```

Note

Leading zeros in the syntax of the access rights assignment are always ignored. You should therefore not use the wildcards ? or * at positions where a leading zero could be located!

Example:

The definition *Slave0?9M06_QB1=R* must be replaced by the following two definitions:

```
Slave?9M06_QB1=R  
Slave9M06_QB1=R
```

Note

The restriction for access rights described here does not apply to OPC UA servers.

OPC Alarms & Events server for SIMATIC NET

3.1 Event server for S7 communication

3.1.1 Functions and alarm categories

Introduction

This section describes the events returned by the OPC server and which additional information is available in the form of attributes.

A&E server for SIMATIC NET

The OPC server for SIMATIC NET can be used as a "simple event server" or as a "conditional alarm and event server". A simple event server has no configuration information for alarm and event handling in an operator control and monitoring system.

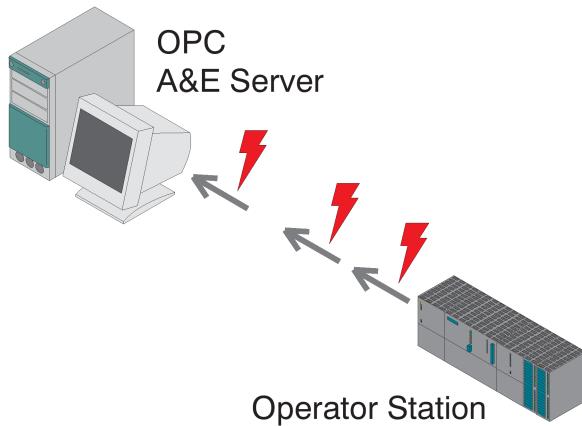


Figure 3-1 Principle of alarms and events communication between client and A&E server

'Condition-related events' can include subconditions. It is also possible to define several subconditions for an alarm. Such an event occurs when the subconditions are satisfied.

Condition-related events also allow evaluation of the conditions to be activated or deactivated by the server and the option of an acknowledgment.

Integrating the event server

A Simple Event Server passes on only simple messages to the underlying components. A higher-level alarm/event management server processes the messages of the lower level simple server taking into account configuration information.

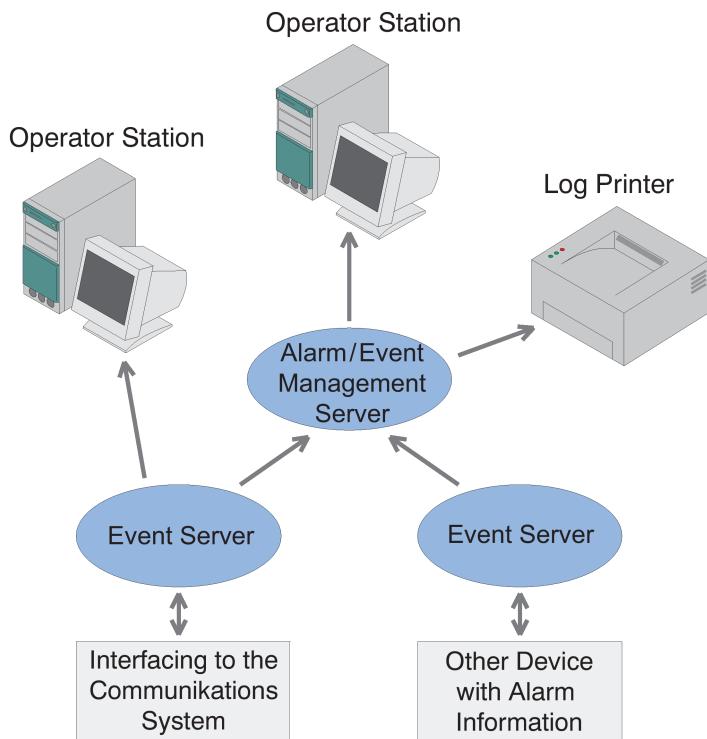


Figure 3-2 Integrating the event server

S7 Protocol

The S7 protocol provides the protocol mechanism "programmed alarms (ALARM)" for transferring events.

The message type is used by the Simple Event Server.

Alarm category

The following *Alarms & Events* categories are available for S7 communication:

Event category Value	Description
EVENTCATEGORY_S7SIMPLE 2	Simple events for the following S7 alarms: NOTIFY ALARM ALARM_8 ALARM_8P
EVENTCATEGORY_DIAGNOSIS 12	Contain the entries in the diagnostic buffer of the module (simple events).
EVENTCATEGORY_S7CONDITION 13	Condition events for the following S7 alarms: NOTIFY ALARM ALARM_8 ALARM_8P ALARM_S ALARM_SQ
EVENTCATEGORY_STATEPATH 14	Alarm for indicating an interrupted connection to a device (condition-related events).
EVENTCATEGORY_CAT_LEVEL 40	Level alarm Alarm signals violations of Min/Max values
EVENTCATEGORY_CAT_DEVIATION 41	Tolerance alarm Alarm signals deviations from tolerances
EVENTCATEGORY_CAT_ROC 42	Rate-of-change alarm Alarm signals frequency of change
EVENTCATEGORY_CAT_OFFNORMAL 43	Off normal alarm Alarm signals deviations from normal state Default for all S7 alarms SFB, SFC, SCAN
EVENTCATEGORY_CAT_TRIP 44	Trip alarm Alarm signals the tripping of a circuit breaker
EVENTCATEGORY_CAT_COS 45	Change-of-state alarm Alarm signals a change of state
EVENTCATEGORY_CAT_DEVICEFAILURE 46	Device fault Alarm signals a device fault
EVENTCATEGORY_CAT_SYSTEMFAILURE 47	System fault Alarm signals a system fault
EVENTCATEGORY_CAT_SYSTEMMESSAGE 60	System message (simple event)

Note

The event categories 2 to 14 are used for configurations of the OPC server < 8.0 and via the alarm text file. For OPC servers >= 8.0, categories 40 to 60 apply.

Table 3- 1 Alarms and scans are assigned to alarm category 43 (CAT_OFFNORMAL) or according to the set STEP 7 alarm class as shown in the following table:

STEP 7 alarm class	OPC alarm category
No alarm	Alarm is suppressed (can be set by user, for example if there are too many Alarm_8p alarm bits).
Unspecified or unknown alarm class	Alarm category 43 (EVENTCATEGORY_CAT_OFFNORMAL) (Default)
Alarm - high Alarm – low Warning – high Warning – low	Alarm category 40 (EVENTCATEGORY_CAT_LEVEL) Condition "HIHI" Condition "LOLO" Condition "HI" Condition "LO"
Tolerance – high Tolerance – low	Alarm category 41 (EVENTCATEGORY_CAT_DEVIATION) Condition "HI" Condition "LO"
AS process control alarm - fault AS process control alarm - error OS process control alarm - fault Preventive maintenance - general Process alarm - with acknowledgment Operating alarm - without acknowledgment Operator prompt - general Operator input message - general Status message - AS Status message – OS	Alarm category 43 (EVENTCATEGORY_CAT_OFFNORMAL) Condition "CFN"

3.1.2 Parameters for events

Parameters for all events

By calling the *OnEvent* callback function of the client, the client obtains a list of events.

Which parameters the OPC event server supplies depends on the type of event. The following parameters are relevant for all events:

Parameter	Meaning
szSource	As the source of the message, the OPC Event server specifies the connection information of the reporting S7 device. This is made up as follows: S7:<connectionname>
ftTime	The time at which the event occurred.
szMessage	The name of a message is made up of the message mechanism and the configured message number: ALARM<alarmnumber> Examples: ALARM55 (see note below)
dwEventType	The OPC Event server supports the following events: OPC_SIMPLE_EVENT This constant has the value 0x0001. OPC_CONDITION_EVENT This constant has the value 0x0004.
dwEventCategory	The following values are possible as the event category: For OPC servers < 8.0: EVENTCATEGORY_S7SIMPLE (2) EVENTCATEGORY_S7T0 (11) EVENTCATEGORY_DIAGNOSIS (12) EVENTCATEGORY_S7CONDITION (13) EVENTCATEGORY_STATEPATH (14) For OPC servers >= 8.0: EVENTCATEGORY_CAT_LEVEL (40) EVENTCATEGORY_CAT_DEVIATION (41) EVENTCATEGORY_CAT_ROC (42) EVENTCATEGORY_CAT_OFFNORMAL (43) EVENTCATEGORY_CAT_TRIP (44) EVENTCATEGORY_CAT_COS (45) EVENTCATEGORY_CAT_DEVICEFAILURE (46) EVENTCATEGORY_CAT_SYSTEMFAILURE (47) EVENTCATEGORY_CAT_SYSTEMMESSAGE (60)
dwSeverity	A predefined value is returned for the severity of the message. This value can be modified for individual message numbers of a communication partner in the configuration database of STEP 7.
dwNumEventAttrs	The number of attributes supplied with the message depends on the number of supplied associated values.
pEventAttributes	This structure contains the attributes supplied with the event. The attributes also include the associated values of the message supplied by the partner device.

Note**Alarm block message weighting before specified priority for alarm messages**

With the OPC Alarms & Events server "OPC.SimaticNetAlarms", S7 alarms with the alarm weighting ("Severity") can be received. These messages can be configured in STEP 7 / NetPro in Properties S7 Connection of the OPC server. Please note:

The programmed alarm weighting of a block ALARM, ALARM_8P or NOTIFY in an S7 program prevails over the configurable priority set for alarm messages.

Alarm priorities configured in NetPro for certain alarm numbers also prevail over the general priority setting for alarm messages and over programmed alarm weighting. ALARM_S and ALARM_SQ have no severity, so that the configured alarm weighting is always used.

Parameters for condition-related events

For *condition-related events*, the following additional parameters are transferred:

Parameter	Meaning
szConditionName	Name of the condition.
szSubconditionName	Name of the subcondition. If there is no subcondition, this parameter has the name of the condition.
wChangeMask	Indicates which state change caused the message to be sent. Possible values include for example OPC_CHANGE_QUALITY or OPC_CHANGE_SUBCONDITION.
wNewState	The status of the subcondition contained in the szSubconditionName parameter. The following values are possible: OPC_CONDITION_ACTIVE OPC_CONDITION_ENABLE OPC_CONDITION_ACKED
wQuality	Contains information on the quality of the value to which the message relates.
bAckRequired	Indicates whether the message requires an acknowledgment.
ftActiveTime	The time at which the subcondition was activated. If no subcondition exists, this parameter contains the time from which the condition was satisfied.
dwCookie	Cookie assigned by the server for the message.
szActorID	Contains application-specific information about the cause of the event.

3.1.3**Event attributes**

An S7 station sends an associated values with a message or an alarm. With EVENT_ATTR_S7_DATA n , EVENT_ATTR_S7_DATA n _DATATYPE and EVENT_ATTR_S7_DATA n _VALUE_LEN the value for n can be up to 10 associated values. The associated values are available in the event attributes via the OPC Alarms & Events interface.

Programmed messages (ALARM) provide the following attributes:

- EVENT_ATTR_S7_MSGTEXT
- EVENT_ATTR_S7_ALARMSTATE
- EVENT_ATTR_S7_TYPE
- EVENT_ATTR_S7_SEVERITY
- EVENT_ATTR_S7_CATEGORYID
- EVENT_ATTR_S7_CATEGORYDESC
- EVENT_ATTR_S7_COMMENT
- EVENT_ATTR_S7_ACTIVETIME
- EVENT_ATTR_S7_PCTIME
- EVENT_ATTR_S7_S7TIME
- EVENT_ATTR_S7_STATE
- EVENT_ATTR_S7_ACK_STATE
- EVENT_ATTR_S7_EVENT_STATE
- EVENT_ATTR_S7_NO_DATA
- EVENT_ATTR_S7_DATA n
- EVENT_ATTR_S7_DATA n _DATATYPE
- EVENT_ATTR_S7_DATA n _VALUE_LEN
- EVENT_ATTR_S7_EVENT_EVENTID
- EVENT_ATTR_S7_EVENT_SUBEVENTID
- EVENT_ATTR_S7_CONNECTION
- EVENT_ATTR_S7.Areas
- EVENT_ATTR_S7_INFOTEXT
- EVENT_ATTR_S7_TEXT1
- EVENT_ATTR_S7_TEXT2
- EVENT_ATTR_S7_TEXT3
- EVENT_ATTR_S7_TEXT4
- EVENT_ATTR_S7_TEXT5
- EVENT_ATTR_S7_TEXT6
- EVENT_ATTR_S7_TEXT7
- EVENT_ATTR_S7.TEXT8
- EVENT_ATTR_S7.TEXT9

EVENT_ATTR_S7_MSGTEXT

Value:	-1
Data type:	VT_BSTR

Attribute value: Configured alarm text, language dependent. If not configured, identical to EventID

EVENT_ATTR_S7_ALARMSTATE

Value:	-2
Data type:	VT_UI4

Attribute value: Alarm states (EVENT_ATTR_S7_ALARMSTATE)

EVENT_ATTR_S7_ALARMSTATE	Description
CHANGE_ACTIVE ACTIVE ACKREQUIRED	The values of the individual #Defines (ACTIVE...) are specified in the opc_ae.h header files. ALARM/ALARM_8/ALARM_8P/ALARM_SQ/ALARM_DQ/SCAN entered state
CHANGE_ACTIVE ACTIVE ACKED	ALARM_S/ALARM_D/NOTIFY/NOTIFY_8P entered state (implicitly acknowledged) The CHANGE_ACKED flag must not be set.
ACTIVE ACKREQUIRED	ALARM/ALARM_8/ALARM_8P/ALARM_SQ/ALARM_DQ/SCAN active (only with refresh)
ACTIVE ACKED	ALARM_S/ALARM_D/NOTIFY/NOTIFY_8P active (implicitly acknowledged) (only with refresh)
ACTIVE CHANGE_ACKED ACKED	ALARM/ALARM_8/ALARM_8P/ALARM_SQ/ALARM_DQ/SCAN active and acknowledgment arrived
ACTIVE ACKED	ALARM/ALARM_8/ALARM_8P/ALARM_SQ/ALARM_DQ/SCAN active and acknowledged (only with refresh)
CHANGE_INACTIVE ACKREQUIRED	ALARM/ALARM_8/ALARM_8P/ALARM_SQ/ALARM_DQ/SCAN exited state and not yet acknowledged
CHANGE_INACTIVE ACKED	ALARM_S/ALARM_D/NOTIFY/NOTIFY_8P exited state (implicitly acknowledged)
CHANGE_INACTIVE ACKED	ALARM/ALARM_8/ALARM_8P/ALARM_SQ/ALARM_DQ/SCAN exited state and acknowledged
ACKREQUIRED	ALARM/ALARM_8/ALARM_8P/ALARM_SQ/ALARM_DQ/SCAN inactive and not yet acknowledged (only with refresh)

EVENT_ATTR_S7_TYPE

Value:	-5
Data type:	VT_UI4

Attribute value: EVENT_TYPE_CONDITION

EVENT_ATTR_S7_SEVERITY

Value:	-6
Data type:	VT_I4

Attribute value: Severity

The severity value indicates the weighting or priority of a subcondition. The value range is between 1 and 1000. The higher the value, the higher the weighting.

EVENT_ATTR_S7_CATEGORYID

Value:	-9
Data type:	VT_I4

Attribute value: Category identifier

Possible values are listed above in the "Alarms & Events Categories" table in the section "Functions and alarm categories (Page 383)".

EVENT_ATTR_S7_CATEGORYDESC

Value:	-10
Data type:	VT_BSTR

Attribute value: Category description

EVENT_ATTR_S7_COMMENT

Value:	-13
Data type:	VT_BSTR

Attribute value: Configured additional text 9, that is language dependent.

EVENT_ATTR_S7_ACTIVETIME

Value:	-15
Data type:	VT_DATE

Attribute value: Time of signal change from "alarm entered state" from time stamp.

EVENT_ATTR_S7_PCTIME

Value:	6000
Data type:	VT_DATE

Attribute value: Time at which the OPC event server received the alarm.

EVENT_ATTR_S7_S7TIME

Value:	6001
Data type:	VT_DATE

Attribute value: Time at which the message was generated on the partner device.

EVENT_ATTR_S7_STATE

Value:	6002
Data type:	VT_UI1 for condition events VT_UI2 for simple events

Attribute value: General status indicating whether or not a message exists.

ALARM messages

The bit	indicates:
0x00H (all bits 0)	The message exists
Bit 0 set	Initialization
Bit 1 set	Overflow signal
Bit 2 set	Overflow instance
Bits 3 to 5	0, reserved
Bit 6 set	Additional values cannot be entered (size)
Bit 7 set	Additional values not obtainable

EVENT_ATTR_S7_ACK_STATE

Value:	6003
Data type:	VT_UI2

Attribute value: Acknowledgment status of the alarm on the S7 programmable controller

It is not possible to acknowledge messages using the OPC Event server for SIMATIC NET. Messages can, however, be acknowledged by other user interface systems.

ALARM messages:

The bit	indicates:
0	Acknowledgment executed for "message 1 entered state"
1 ... 6	Acknowledgment executed for "messages 2 to 7 entered state" (only ALARM_8 / ALARM_8P)
7	Acknowledgment executed for "message 8 entered state" (only ALARM_8 / ALARM_8P)
8	Acknowledgment executed for "message 1 exited state"
9 ... 14	Acknowledgment executed for "messages 2 to 7 exited state" (only ALARM_8 / ALARM_8P)
15	Acknowledgment executed for "message 8 exited state" (only ALARM_8 / ALARM_8P)

It is possible to acknowledge only "message n entered state" using the OPC Event Server for SIMATIC NET. Messages can, however, be acknowledged by other user interface systems.

EVENT_ATTR_S7_EVENT_STATE

Value:	6004
Data type:	VT_UI1 for condition-related events VT_UI2 for simple events

Attribute value: Event state

The bit	indicates:
0	Current status "message 1" (1 = active)
1 ... 6	Current status "message 2 to 7" (1 = active) (ALARM_8 / ALARM_8P only)
7	Current status "message 8" (1 = active) (ALARM_8 / ALARM_8P only)

EVENT_ATTR_S7_NO_DATA

Value:	6005
Data type:	VT_UI1 for condition events VT_UI2 for simple events

Attribute value: Number of associated values. Range of values: 1...10

EVENT_ATTR_S7_DATAAn

Value:	where $n = 0$ where $n = 1$ where $n = 2$ where $n = 3$ where $n = 4$ where $n = 5$ where $n = 6$ where $n = 7$ where $n = 8$ where $n = 9$	6008 6012 6016 6020 6024 6028 6032 6036 6040 6044
Data type:	VT_ARRAY VT_UI1	

Attribute value: The relevant bytes of associated value no. "n" as array of bytes.

EVENT_ATTR_S7_DATAAn_DATATYPE

Value:	where $n = 0$ where $n = 1$ where $n = 2$ where $n = 3$ where $n = 4$ where $n = 5$ where $n = 6$ where $n = 7$ where $n = 8$ where $n = 9$	6006 6010 6014 6018 6022 6026 6030 6034 6038 6042
Data type:	VT_UI2: VT_UI1:	With simple events With condition-related events

Attribute value: Data type of associated value no. "n".

Parameter value	Description
S7_DATATYPE_ERROR	Error (0x0)
S7_DATATYPE_BOOLEAN	Boolean (0x03)
S7_DATATYPE_INTEGER	Integer (0x05)
S7_DATATYPE_UNSIGNED	Integer (0x06)
S7_DATATYPE_FLOAT	Float (0x07)
S7_DATATYPE_OCTET_STRING	String (0x09)
S7_DATATYPE_BITSTRING	Bit string (0x04) Note: Length information in bytes instead of bits
S7_DATATYPE_DATE	Date (0x30) Note: Days since 01.01.1990

Parameter value	Description
S7_DATATYPE_TIME_OF_DAY	Time (0x31) Note: ms since the start of the day
S7_DATATYPE_TIME	Time (0x32) Note: ms
S7_DATATYPE_S5TIME	Time (0x33) Note: BCD coded
S7_DATATYPE_DATE_AND_TIME	Date and time (0x34)

EVENT_ATTR_S7_DATA_n_VALUE_LEN

Value:	where $n = 0$ where $n = 1$ where $n = 2$ where $n = 3$ where $n = 4$ where $n = 5$ where $n = 6$ where $n = 7$ where $n = 8$ where $n = 9$	6007 6011 6015 6019 6023 6027 6031 6035 6039 6043
Data type:	VT_UI2: VT_UI2:	With simple events With condition-related events

Attribute value: Number of relevant bytes of associated value no. "n".

EVENT_ATTR_S7_EVENT_EVENTID

Value:	6046
Data type:	VT_UI4 (with condition-related events only)

Attribute value: Alarm number

EVENT_ATTR_S7_EVENT_SUBEVENTID

Value:	6047
Data type:	VT_UI1 (with condition-related events only)

Attribute value: SubEventID (range of value 1 to 8) with messages of the type ALARM_8 / ALARM_8P, otherwise 1.

EVENT_ATTR_S7_CONNECTION

Value:	6050
Data type:	VT_BSTR

Attribute value: S7 connection name

EVENT_ATTR_S7 AREAS

Value:	6051
Data type:	VT_ARRAY of VT_BSTR

Attribute value: Areas in which the diagnostics event is enumerated.

EVENT_ATTR_S7_INFOTEXT

Value:	6060
Data type:	VT_BSTR

Attribute value: Configured info text, that is language dependent.

EVENT_ATTR_S7_TEXT1

Value:	6061
Data type:	VT_BSTR

Attribute value: Configured additional text 1, that is language dependent.

EVENT_ATTR_S7_TEXT2

Value:	6062
Data type:	VT_BSTR

Attribute value: Configured additional text 2, that is language dependent.

EVENT_ATTR_S7_TEXT3

Value:	6063
Data type:	VT_BSTR

Attribute value: Configured additional text 3, that is language dependent.

EVENT_ATTR_S7_TEXT4

Value:	6064
Data type:	VT_BSTR

Attribute value: Configured additional text 4, that is language dependent.

EVENT_ATTR_S7_TEXT5

Value:	6065
Data type:	VT_BSTR

Attribute value: Configured additional text 5, that is language dependent.

EVENT_ATTR_S7_TEXT6

Value:	6066
Data type:	VT_BSTR

Attribute value: Configured additional text 6, that is language dependent.

EVENT_ATTR_S7_TEXT7

Value:	6067
Data type:	VT_BSTR

Attribute value: Configured additional text 7, that is language dependent.

EVENT_ATTR_S7_TEXT8

Value:	6068
Data type:	VT_BSTR

Attribute value: Configured additional text 8, that is language dependent.

EVENT_ATTR_S7_TEXT9

Value:	6069
Data type:	VT_BSTR

Attribute value: Configured additional text 9, that is language dependent.

Meaning of the various time stamps

Attributes	Arrival of new alarms from the S7 device:	Refresh
ParameterTimeStamp	<p>Depends on the setting of the time stamp origin configuration:</p> <p>1) Arrival of edge change: S7 time of the edge of the signal</p> <p>2) Arrival of acknowledgment: S7 time of the acknowledgment</p> <p>Or</p> <p>1) Arrival of edge change: S7 time of the edge of the signal +/- offset</p> <p>2) Arrival of acknowledgment: S7 time of the acknowledgment +/- offset</p> <p>Or</p> <p>1) Arrival of edge change: PC time of the notification</p> <p>2) Arrival of acknowledgment: PC time of the notification</p>	<p>Depends on the setting of the time stamp origin configuration:</p> <p>Last S7 time of the edge of the signal or acknowledgment</p> <p>Or</p> <p>Last S7 time of the edge of the signal or acknowledgment +/- offset</p> <p>Or</p> <p>PC time of the last notification of edge change or acknowledgment</p>
EVENT_ATTR_S7_S7TIME	<p>1) Arrival of edge change: S7 time of the edge of the signal</p> <p>2) Arrival of acknowledgment: S7 time of the acknowledgment</p>	Last S7 time of the edge of the signal or acknowledgment
EVENT_ATTR_S7_PCTIME	<p>1) Arrival of edge change: PC time of the notification</p> <p>2) Arrival of acknowledgment: PC time of the notification</p>	PC time of the last notification of edge change or acknowledgment

When a connection was first established or interrupted, the S7 alarm update does not provide an S7 time stamp with the signal types ALARM/ALARM_8/ALARM_8P/ALARM_SQ/ALARM_DQ/SCAN. The ATTR_S7_S7TIME attribute then contains the time stamp 01.01.1990 (oldest possible S7 time), the time stamp is otherwise formed based on the time stamp 01.01.1990 or based on the PC time depending on the setting in the configuration of the time stamp origin.

Note

Note that not all status changes of an alarm are forwarded by an event on the OPC interface. If there is a fast change in the alarm status, it is possible that some status changes are not reported by an event and only the last and now current status is sent. This depends on OPC parameters such as "dwBufferTime" as well as the configuration and the computer performance.

3.1.4 Attributes for entries in the diagnostic buffer of the module

EVENT_ATTR_DIAGNOSIS_MSGTEXT

Value:	-1
Data type:	VT_BSTR

Attribute value: Configured alarm text, language dependent. If not configured identical to EventID.
Ideally, however, a SIMOTION environment uses its own alarm text server.

EVENT_ATTR_DIAGNOSIS_TYPE

Value:	-5
Data type:	VT_I4

Attribute value: EVENT_TYPE_SIMPLE

EVENT_ATTR_DIAGNOSIS_SEVERITY

Value:	-6
Data type:	VT_I4

Attribute value: Severity

EVENT_ATTR_DIAGNOSIS_CATEGORYID

Value:	-9
Data type:	VT_I4

Attribute value: Category identifier

EVENT_ATTR_DIAGNOSIS_CATEGORYDESC

Value:	-10
Data type:	VT_I4

Attribute value: Category description

EVENT_ATTR_DIAGNOSIS_S7_PCTIME

Value:	6001
Data type:	VT_DATE

Attribute value: Time stamp of the SIMATIC NET S7 OPC server

EVENT_ATTR_DIAGNOSIS_S7_DIAGNOSISID

Value:	6048
Data type:	VT_I4

Attribute value: Diagnostic event number, analog EventID

EVENT_ATTR_DIAGNOSIS_S7_DIAGNOSISDATA

Value:	6049
Data type:	VT_ARRAY or VT_UI1

Attribute value: Information on the diagnostic event or its effects, data length depending on the event class.

Note

You will find a detailed description of SFC 52 (S7 diagnostics data) in the document "STEP 7 System and Standard Functions for S7-300/400".

EVENT_ATTR_DIAGNOSIS_S7_CONNECTION

Value:	6050
Data type:	VT_BSTR

Attribute value: S7 connection name

EVENT_ATTR_DIAGNOSIS_S7.Areas

Value:	6051
Data type:	VT_ARRAY VT_BSTR

Attribute value: Areas in which the diagnostics event is enumerated. Currently a maximum of one area is available. Default: empty

Or

If there is any configuration information about the alarm:

connections\<connectionname>

Or

Configured area

EVENT_ATTR_DIAGNOSIS_S7_INFOTEXT

Value:	6060
Data type:	VT_BSTR

Attribute value: Configured info text, language dependent. If not configured, empty.

EVENT_ATTR_DIAGNOSIS_S7_TEXT1

Value:	6061
Data type:	VT_BSTR

Attribute value: Configured additional text 1 (possibly identical to source), language dependent. If not configured, empty.

EVENT_ATTR_DIAGNOSIS_S7_TEXT2

Value:	6062
Data type:	VT_BSTR

Attribute value: Configured additional text 2 (possibly used as area), language dependent. If not configured, empty.

EVENT_ATTR_DIAGNOSIS_S7_TEXT3

Value:	6063
Data type:	VT_BSTR

Attribute value: Configured additional text 3, language dependent. If not configured, empty.

EVENT_ATTR_DIAGNOSIS_S7_TEXT4

Value:	6064
Data type:	VT_BSTR

Attribute value: Configured additional text 4, language dependent. If not configured, empty.

EVENT_ATTR_DIAGNOSIS_S7_TEXT5

Value:	6065
Data type:	VT_BSTR

Attribute value: Configured additional text 5, language dependent. If not configured, empty.

EVENT_ATTR_DIAGNOSIS_S7_TEXT6

Value:	6066
Data type:	VT_BSTR

Attribute value: Configured additional text 6, language dependent. If not configured, empty.

EVENT_ATTR_DIAGNOSIS_S7_TEXT7

Value:	6067
Data type:	VT_BSTR

Attribute value: Configured additional text 7, language dependent. If not configured, empty.

EVENT_ATTR_DIAGNOSIS_S7_TEXT8

Value:	6068
Data type:	VT_BSTR

Attribute value: Configured additional text 8, language dependent. If not configured, empty.

EVENT_ATTR_DIAGNOSIS_S7_TEXT9

Value:	6069
Data type:	VT_BSTR

Attribute value: Configured additional text 9, language dependent. If not configured, empty.

EVENT_ATTR_DIAGNOSIS_S7_S7TIME

Value:	6100
Data type:	VT_DATE

Attribute value: Time stamp

Meaning of the various time stamps

Attributes	Arrival of new diagnostics events from the S7 device:
ParameterTimeStamp	Depends on the setting of EventTimeBias: S7 time of the diagnostics event Or S7 time of the diagnostics event + offset Or PC time of the arrival of the diagnostics event.
EVENT_ATTR_DIAGNOSIS_S7TIME	S7 time of the diagnostics event
EVENT_ATTR_DIAGNOSIS_PCTIME	PC time of the arrival of the diagnostics event

See also

[Attributes for alarms indicating an interrupted connection \(Page 403\)](#)

3.1.5 Attributes for alarms indicating an interrupted connection

EVENT_ATTR_STATEPATH_MSGTEXT

Value:	-1
Data type:	VT_BSTR

Attribute value: Configured alarm text, language dependent. If not configured, identical to EventID.

EVENT_ATTR_STATEPATH_ALARMSTATE

Value:	-2
Data type:	VT_UI4

Attribute value: Alarm states (EVENT_ATTR_STATEPATH_ALARMSTATE)

EVENT_ATTR_STATEPATH_ALARMSTATE	Description
CHANGE_ACTIVE ACTIVE	The values of the individual #Defines (ACTIVE...) are specified in the opc_ae.h header files. STATEPATH entered state (not with "Refresh")
ACTIVE	STATEPATH active
CHANGE_INACTIVE INACTIVE	STATEPATH exited state (not with "Refresh")

EVENT_ATTR_STATEPATH_TYPE

Value:	-5
Data type:	VT_I4

Attribute value: EVENT_TYPE_CONDITION

EVENT_ATTR_STATEPATH_SEVERITY

Value:	-6
Data type:	VT_I4

Attribute value: Severity

EVENT_ATTR_STATEPATH_CATEGORYID

Value:	-9
Data type:	VT_I4

Attribute value: Category identifier

EVENT_ATTR_STATEPATH_CATEGORYDESC

Value:	-10
Data type:	VT_BSTR

Attribute value: Category description

EVENT_ATTR_STATEPATH_COMMENT

Value:	-13
Data type:	VT_BSTR

Attribute value: Comment

EVENT_ATTR_STATEPATH_ACTIVETIME

Value:	-15
Data type:	VT_DATE

Attribute value: Time of signal change to "alarm entered state" from time stamp.

EVENT_ATTR_STATEPATH_CONNECTION

Value:	6050
Data type:	VT_BSTR

Attribute value: S7 connection name

EVENT_ATTR_STATEPATH.Areas

Value:	6051
Data type:	VT_ARRAY of VT_BSTR

Attribute value: Areas in which the state path alarm is enumerated.

3.1.6 Configuring alarm texts, source and area

Which alarms are supported?

Der OPC Alarms & Events server supports the following alarms:

- Symbol-related alarms (SCANs)
These allow the monitoring of bits in the areas I, Q, M and DB of the CPU asynchronous to the PLC user program.
- Block-related alarms (alarm SFB, alarm SFC)
Acknowledgeable alarm SFBs are: ALARM (SFB 33), ALARM_8 (SFB 34) and ALARM_8P (SFB 35)
The following SFBs are not acknowledgeable: NOTIFY (SFB 36) and NOTIFY_8P (SFB 31)
Acknowledgeable alarm SFCs are: ALARM_SQ (SFC 17) and ALARM_DQ (SFC 107)
The following SFCs are not acknowledgeable: ALARM_S (SFC 18) and ALARM_D (SFC 108)
- Diagnostics alarms
System diagnostics (ID 0x1000-0x79FF, 0xC000-0xEFFF, 0xF900-0xF9FF)
user diagnostics (ID 0x8000-0xB9FF) with WR_USMSG (SFC 52)
- Statepath alarm
Is generated by the OPC Alarms & Events server when the connection status changes.

What can be specified with an alarm?

This section shows some of the settings that can be made in the configuration for alarm texts, sources and areas.

Configured alarm texts

- Entry for programmed block-related alarms (alarm SFBs SFCs):
The alarm text is entered in STEP 7 for the corresponding instance data block in ... "Special Object Properties" > "Message"
The alarm text is entered in the "Message text" column in the dialog. Associated value can be integrated in the text.

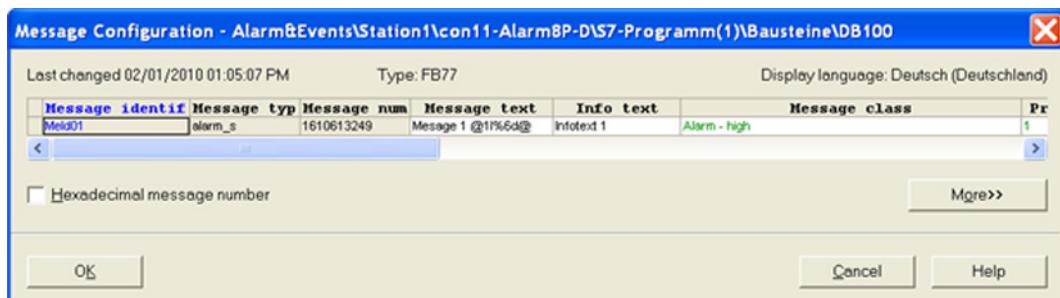


Figure 3-3 Configuration dialog of an alarm SFC with ALARM_SQ (SFC17) that was called in FB1, instance DB is DB1.

- Input for symbol-related alarms (SCANs):
The alarm text is entered in STEP 7 in the symbol editor shortcut menu "Special Object Properties" > "Message..." of an alarm symbol. For these symbols, the "Message" check

3.1 Event server for S7 communication

box is selected.

The alarm text is entered in the "Message text" column in the dialog.

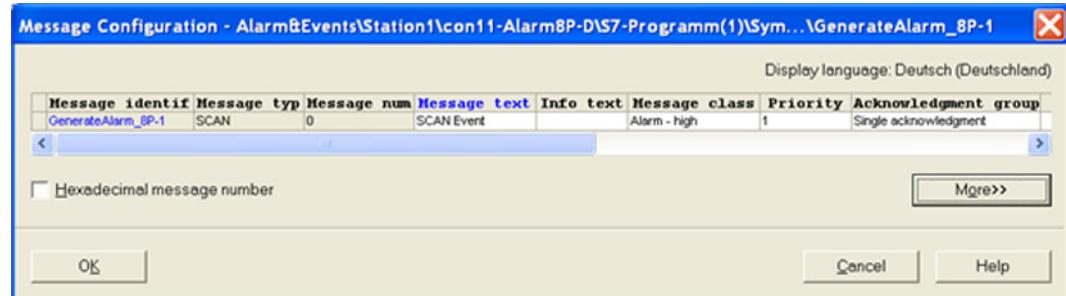


Figure 3-4 Configuration dialog of a symbol-related alarm for the memory bit 3.1

- Input for diagnostics alarms:

Alarm texts entering and exiting state for a user diagnostics alarm WR_UMSG (SFC52) are entered in the columns "Incoming message text" and "Outgoing message text". The input dialog is available in the SIMATIC Manager in the shortcut menu of S7 programs of SIMATIC S7 stations "Special Object Properties" > "Message..."

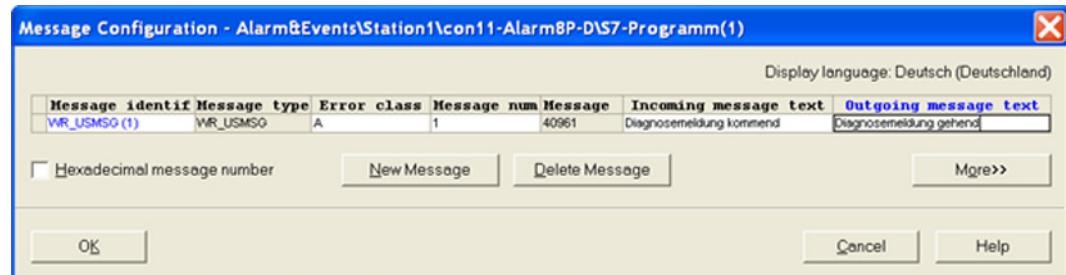


Figure 3-5 Example of configuring a user diagnostics alarm for WR_UMSG (SFC52)

- Configured source:

In the dialogs in which the alarm text is entered, you can also enter the alarm source. As source, the first additional text (source when attribute is set for data block S7_alarm_ui=1) is used, see Figure 3-6.

If this is not set, the path name to the block assigned to the alarm text in STEP 7 is used.

For example, SIMATIC400(1)/CPU416.3DP/S7_program(1)/DB10/DB100.

For the StatepathAlarm, the configured connection name is used as the source.

- Configured area:

As the area, the 2nd additional text (OS area when the attribute is set S7_Alarm_ui=1) is used, see Figure 3-6.

If this is not set, the path name from the STEP 7 project is also used here.

For the StatepathAlarm, the configured connection name is used as the area.

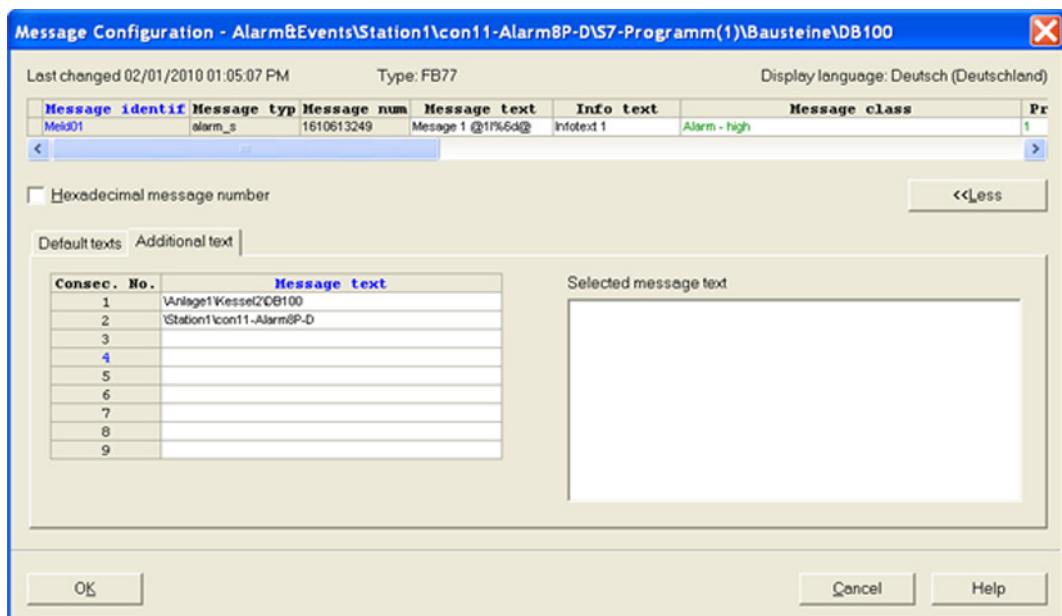


Figure 3-6 1. additional text (source) and 2nd additional text (area)

How are associated values embedded in alarms?

Alarm and info texts can contain formal parameters. Some of these parameters can be replaced by formatted associated values of the alarm.

You will find the structure and the meaning of the possible formal parameters described in STEP 7 in "Help" > "Inserting associated values".

The OPC server for S7 communication, however, also supports further format information, identified by *) in the list below. The formal parameters are only replaced by formatted associated values in message and info texts. With the OPC servers for S7 communication, the formal parameters are not replaced in the additional texts.

The formal parameters have the following structure: @<number><type><format>@

<number>	Stands for the number of the associated value to be used for the formal parameter.																												
<type>	<p>Describes the type of associated value. The following values are possible:</p> <p>Y - Byte W - Word D - DWord I - Integer D - Double Integer B - Bool C - Char R - Real <empty> - no type specified: Note: No type may be specified for the formats "%t", "%s" and "%Y". For the other formats select a type suitable for the associated value and format. There is no type conversion of the specified associated value.</p>																												
<format>	<p>Describes the formatting of the associated value in the text. The following formats are possible:</p> <table border="1"> <tr> <td>%[i]X</td> <td>Hexadecimal representation of associated value with uppercase letters.</td> </tr> <tr> <td>%[i]x</td> <td>Hexadecimal representation of associated value with lowercase letters. *)</td> </tr> <tr> <td>%[i]u</td> <td>Representation as decimal number without sign.</td> </tr> <tr> <td>%[i]d</td> <td>Representation as decimal number with sign.</td> </tr> <tr> <td>%[i]i</td> <td>Representation as decimal number with sign. *)</td> </tr> <tr> <td>%[i]b</td> <td>Representation as binary number.</td> </tr> <tr> <td>%[i][.y]f</td> <td>Representation as a fixed-point number with y decimal places.</td> </tr> <tr> <td>%[i][.y]e</td> <td>Representation as a floating-point number with y decimal places with exponential notation with lowercase letters. *)</td> </tr> <tr> <td>%[i][.y]E</td> <td>Representation as a floating-point number with y decimal places with exponential notation with uppercase letters. *)</td> </tr> <tr> <td>%[i][.y]g</td> <td>Representation as a fixed-point number or as a floating-point number with y decimal places with lowercase letters. *)</td> </tr> <tr> <td>%[i][.y]G</td> <td>Representation as a fixed-point number or as a floating-point number with y decimal places with uppercase letters. *)</td> </tr> <tr> <td>%[i]s</td> <td>The associated value value is interpreted as a 0 terminated string.</td> </tr> <tr> <td>%t#<file></td> <td>The associated value is interpreted as an index in the text library <file> and the text is inserted from the text library.</td> </tr> <tr> <td>%Y[locale] [#DT format]</td> <td> <p>The associated values are interpreted as DATE_AND_TIME and formatted in the DT format taking locale into account. *)</p> <p>"locale" stands for a locale name, for example, German, deu or us. If this setting is missing the current locale is used.</p> <p>.DT format corresponds to the format specification of the standard function strftime() with the extension that %s stands for the three-digit output of milliseconds. If DT format is not specified, %c is used corresponding to the date and time representation.</p> </td> </tr> </table>	%[i]X	Hexadecimal representation of associated value with uppercase letters.	%[i]x	Hexadecimal representation of associated value with lowercase letters. *)	%[i]u	Representation as decimal number without sign.	%[i]d	Representation as decimal number with sign.	%[i]i	Representation as decimal number with sign. *)	%[i]b	Representation as binary number.	%[i][.y]f	Representation as a fixed-point number with y decimal places.	%[i][.y]e	Representation as a floating-point number with y decimal places with exponential notation with lowercase letters. *)	%[i][.y]E	Representation as a floating-point number with y decimal places with exponential notation with uppercase letters. *)	%[i][.y]g	Representation as a fixed-point number or as a floating-point number with y decimal places with lowercase letters. *)	%[i][.y]G	Representation as a fixed-point number or as a floating-point number with y decimal places with uppercase letters. *)	%[i]s	The associated value value is interpreted as a 0 terminated string.	%t#<file>	The associated value is interpreted as an index in the text library <file> and the text is inserted from the text library.	%Y[locale] [#DT format]	<p>The associated values are interpreted as DATE_AND_TIME and formatted in the DT format taking locale into account. *)</p> <p>"locale" stands for a locale name, for example, German, deu or us. If this setting is missing the current locale is used.</p> <p>.DT format corresponds to the format specification of the standard function strftime() with the extension that %s stands for the three-digit output of milliseconds. If DT format is not specified, %c is used corresponding to the date and time representation.</p>
%[i]X	Hexadecimal representation of associated value with uppercase letters.																												
%[i]x	Hexadecimal representation of associated value with lowercase letters. *)																												
%[i]u	Representation as decimal number without sign.																												
%[i]d	Representation as decimal number with sign.																												
%[i]i	Representation as decimal number with sign. *)																												
%[i]b	Representation as binary number.																												
%[i][.y]f	Representation as a fixed-point number with y decimal places.																												
%[i][.y]e	Representation as a floating-point number with y decimal places with exponential notation with lowercase letters. *)																												
%[i][.y]E	Representation as a floating-point number with y decimal places with exponential notation with uppercase letters. *)																												
%[i][.y]g	Representation as a fixed-point number or as a floating-point number with y decimal places with lowercase letters. *)																												
%[i][.y]G	Representation as a fixed-point number or as a floating-point number with y decimal places with uppercase letters. *)																												
%[i]s	The associated value value is interpreted as a 0 terminated string.																												
%t#<file>	The associated value is interpreted as an index in the text library <file> and the text is inserted from the text library.																												
%Y[locale] [#DT format]	<p>The associated values are interpreted as DATE_AND_TIME and formatted in the DT format taking locale into account. *)</p> <p>"locale" stands for a locale name, for example, German, deu or us. If this setting is missing the current locale is used.</p> <p>.DT format corresponds to the format specification of the standard function strftime() with the extension that %s stands for the three-digit output of milliseconds. If DT format is not specified, %c is used corresponding to the date and time representation.</p>																												

Information in square brackets [] is optional. The value "i" stands for the number of characters to be used for the representation.

The following strings are replaced as follows outside formal parameters:

- \n Is replaced by the character <new line> (0x0a).
- \r Is replaced by the character <CR> (0x0d).
- \t Is replaced by the tab character (0x09).
- \\" Is replaced by double quotation marks (0x22).
- \@ Is replaced by the @ character (0x64). This means that the character does not introduce any formal parameters.

3.1.7 How can source information, sources and conditions be browsed and filtered in the namespace?

The "IOPC EventAreaBrowser" interface provides methods for browsing source area, sources and conditions. Using the filter functions, you can filter according to subareas.

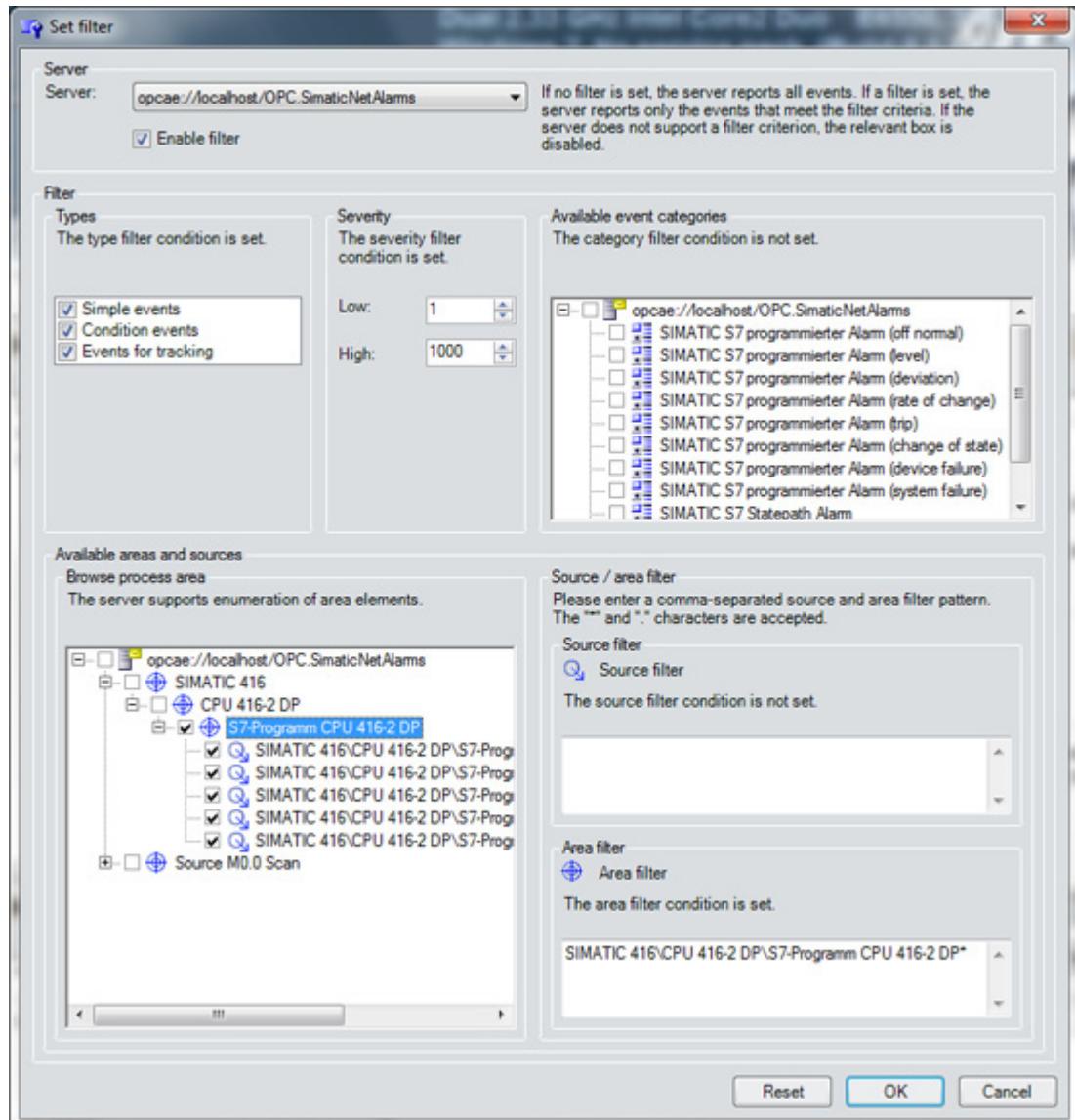


Figure 3-7 Filter dialog in the OPC Scout V10

To be able to include lower-level areas as well, a "*" character must be appended to the filter string.

For example, all alarms of a CPU 416 are obtained with the filter "SIMATIC 400(1)/CPU 416-3DP**"

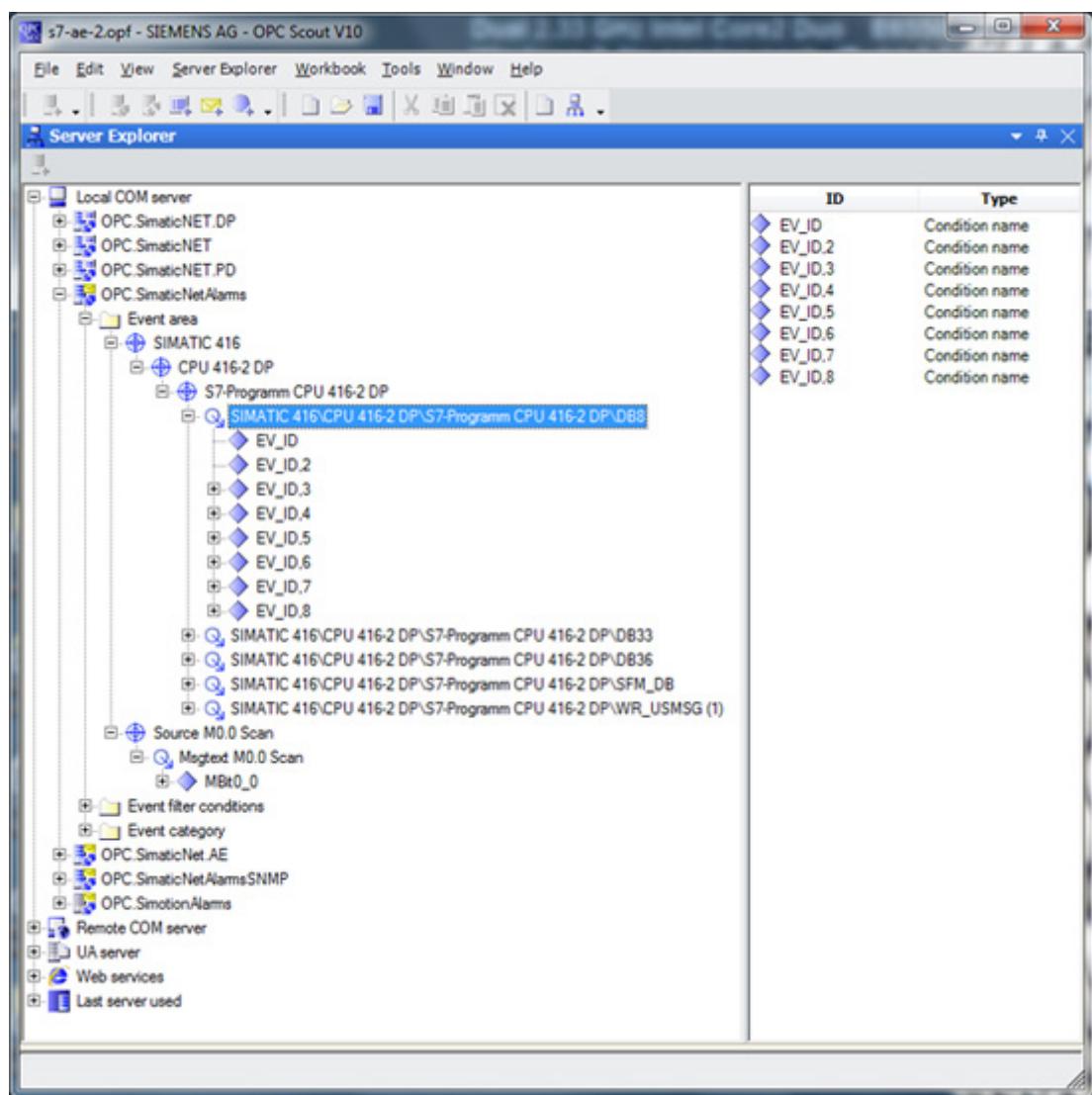


Figure 3-8 Source namespace of a CPU 416. The source can be configured specifically using the 1st additional text.

3.1.8 Mapping STEP 7 configuration values to OPC S7 Alarms & Events parameters

The following table shows the mapping of configuration data such as the "EventID" from STEP 7 to OPC A&E parameters such as "Condition":

Configured parameters	OPC value	Meaning
EventID ⇒ Condition ⇒ configured EventID is mapped to Condition EVENTCATEGORY_CAT_LEVEL EVENTCATEGORY_CAT_DEVIATION EVENTCATEGORY_CAT_ROC	"HIHI" "HI" (Default) "LO" "LOLO"	"HIHI" state "HI" state "LO" state "LOLO" state An EventID configured by the user may only adopt one of the four possible values.
EventID ⇒ Condition EVENTCATEGORY_CAT_OFFNORMAL	"CFN"	"CFN" state
EventID ⇒ Condition EVENTCATEGORY_CAT_TRIP	"TRIP"	"TRIP" state
EventID ⇒ Condition EVENTCATEGORY_CAT_COS	"COS"	"COS" state
EventID ⇒ Condition EVENTCATEGORY_CAT_DEVICEFAILURE EVENTCATEGORY_CAT_SYSTEMFAILURE	"FAILURE"	"FAILURE" state
Origin/source ⇒ Source	<connectionname>/ ALARM"<event ID> {,<subevent ID>} Or <connectionname>/ STATEPATH e.g. "s7_conn1/ALARM10 .5" e.g. "s7_conn1/STATEPA TH"	Unique combination of connection name and EventID or STATEPATH Alternative: Configured source, language dependent. Source must be unique!
Message weighting ⇒ Severity	1 ... 1000 See following table	Severity from the configuration or from the alarm data
Message class ⇒ Category	40 41 42 43 44 45 46 47	Category CAT_LEVEL Category CAT_DEVIATION Category CAT_ROC Category CAT_OFFNORMAL Category CAT_TRIP Category CAT_COS Category CAT_DEVICEFAILURE Category CAT_SYSTEMFAILURE
Configured source of the time stamp ⇒ TimeStamp	00.00.0000 00:00:00.0000	Time stamp (CPU, CPU + OFFSET, PCTime)

Priority: The priority is converted to an OPC alarm severity according to this table:

S7 priority	OPC alarm severity
0	1
1	63
2	125
3	188
4	250
5	313
6	375
7	438
8	500
9	563
10	625
11	688
12	750
13	813
14	875
15	938
16	1000

3.1.9 S7 demo alarms

The DEMO connection makes data block objects in the namespace available for data access. The DEMO connection is intended to familiarize you with the SIMATIC NET OPC systems and can be activated via the configuration.

S7 messages can also be simulated. These messages can be triggered by setting and resetting the bits of the first byte of the DEMO data block "db20".

COM DA S7

ItemID:

S7: [DEMO] DB20, X0.0

S7: [DEMO] DB20, X0.1

...

S7: [DEMO] DB20, X0.7

OPC UA S7

NameSpaceID

S7:

NodeID:

...

DEMO.db20.0, x7

Changing the setting of the bits from "False" to "True" triggers OPC alarms entering state, resetting the bits from "True" to "False" triggers OPC alarms exiting state.

The alarms are implicitly acknowledged when the corresponding bits in the third byte of the DEMO data block "db20" are set to "True":

S7: [DEMO] DB20, X2.0 ... or

S7: DEMO.db20.2, x0 ...

By resetting these bits to "False", acknowledgeable alarms can be set.

By triggering the alarms via Data Access the following are generated:

- for OPC Alarms & Events for S7 communication
conditional alarms of the category EVENTCATEGORY_S7OFFNORMAL = 43
- for OPC UA Alarms & Conditions for S7 communication
UA alarms of the EventType = Nodeld(ns=S7TYPES:, i=43)
-> referenced Nodeld of the S7OffNormalAlarmType

3.2 Simple event server for SNMP communication

SNMP traps

Traps are messages that can be sent to the OPC Server without it requesting them. There are seven generic traps available on many SNMP-compliant devices. There are also device-specific traps that are described in the MIB file.

Note

Device-specific MIBs must be integrated using the MIB Compiler (part of STEP 7).

Traps must be activated on the relevant devices and their target station (here: SNMP OPC server) configured.

Generic traps

warmStart

Reboot of the computer if it already had a connection to the network.

This is sent after a warm restart on the device.

coldStart

Startup of the computer if it did not yet have a connection to the network.

This is sent after a cold restart on the device.

linkDown

This is sent when a connection from the device is terminated.

linkUp

This is sent when a connection from the device is established.

authenticationFailure

This is sent when there was unauthorized access to the device.

egpNeighborLoss

The EGP neighbor (EGP = Exterior Gateway Protocol) of the device is not working. The Exterior Gateway Protocol is used to exchange routing information between two neighboring gateway hosts.

enterpriseSpecific

This is sent when a device-specific trap is sent.

Forwarding traps

Traps are forwarded as socalled "simple events" to the OPC A&E server. The frequency of the occurrence and the description of a trap are also available as an OPC DA item.

Requirements

- The IP address of the PC with the OPC SNMP server must be entered as a trap recipient on the SNMPcompliant device and the traps must be activated. The conversion is made with device-specific configuration tools (see manual "Commissioning PC Stations"; Web based Management of OSM/ESM/SCALANCE).
- The Windows SNMP service must be installed.

OPC alarm server name: OPC.SimaticNetAlarmsSNMP

Alarms & events categories

The following Alarms & Events categories are available and are described in greater detail below:

Category Number	Name	Description
20	EVENTCATEGORY_SNMP_GENERICTRAP	SNMP trap event (generic trap)
21	EVENTCATEGORY_SNMP_SPECIFICTRAP	SNMP trap event (specific trap)

Generic traps are specified by standards and have the same significance for all devices. Specific traps are devicespecific. The SNMP OPC Server is informed whether or not a trap is generic or specific by the device profile file.

These categories "generic" or "specific" are to be understood in the context of the term "Category". Alarms and events of every category can occur on any partner device (source). Different alarms and the events of one and the same category from the same partner device are identified by their different EventIDs.

Traps are triggered by the communications partner and forwarded by the SNMP protocol to the SIMATIC NET SNMP OPC Server. With high communications loads, traps can be

discarded by the SNMP protocol so that the SIMATIC NET SNMP OPC Server does not necessarily receive all traps. This is a general restriction of the SNMP protocol.

Traps do not have a status and are not acknowledged and consequently the SIMATIC NET SNMP OPC Server does not manage a state machine for the traps.

By calling the *OnEvent* callback function of the client, the client obtains a list of events. The SNMP OPC event server returns the following serverspecific parameters:

Parameter	Meaning
dwEventCategory	The following value is possible as the event category: EVENTCATEGORY_SNMP_GENERICTRAP = 20 EVENTCATEGORY_SNMP_SPECIFICTRAP = 21
dwEventType	The SNMP OPC Event server supports only the following type: OPC_SIMPLE_EVENT This constant has the value 0x0001.
dwNumEventAttrs	The number of attributes supplied with the message depends on the number of supplied associated values.
dwSeverity	A predefined value is returned for the severity of the error. This value can be modified for individual message numbers of a communication partner in the configuration database.
pEventAttributes	This structure contains the attributes supplied with the event. The attributes also include the associated values of the message supplied by the partner device.
szMessage	Trap event name from the device profile file: <nametext> e.g. TrapPowerDown
szSource	As the source of the message, the SNMP OPC event server specifies the device information of the reporting SNMP device. This corresponds to the device name of an SNMP variable for OPC Data Access: SNMP:\<device> e.g. SNMP:\Switch33

Along with this event, additional information in the form of attributes can also be sent. SNMP traps provide the following attributes:

Attribute ID Value	Attributes	Meaning
EVENT_ATTR_SNMP_PCTIME 6100	VT_DATE	Time at which the OPC event server received the alarm.
EVENT_ATTR_SNMP_TIMESTAMP 6101	AsnTimeticks (VT_UI4)	Time at which the message was generated on the partner device.
EVENT_ATTR_SNMP_EVENTID 6102	LONG (VT_I4)	Unique trap number within the device

Attribute ID Value	Attributes	Meaning
EVENT_ATTR_SNMP_ENTERPRISE 6103	AsnObjectIdentifier (VT_BSTR)	Devicespecific trap
EVENT_ATTR_SNMP_AGENTADDRESS 6104	AsnNetworkAddress (VT_BSTR)	IP address of agent
EVENT_ATTR_SNMP_SOURCEADDRESS 6105	AsnNetworkAddress (VT_BSTR)	IP address of trap source
EVENT_ATTR_SNMP_VARBINDINGS 6106	SnmpVarBindList (VT_ARRAY VT_VARIANT)	Variable bindings of the sent trap

3.3 Alarms & Events server for S7 and SNMP communication

3.3.1 The A&E servers from SIMATIC NET

What A&E servers are there for SIMATIC NET?

Similar to OPC Data Access, where the "OPC.SimaticNET" server brings together all communications protocols, the server "OPC.SimaticNET.AE" can be used for OPC Alarms & Events and this brings together the SNMP and S7 communications protocols. This allows S7 alarms and SNMP traps to be received within one server.

The individual A&E servers such as "OPC.SimaticNetAlarms" and "OPC.SimaticNetAlarmsSNMP" can be used parallel to each other.

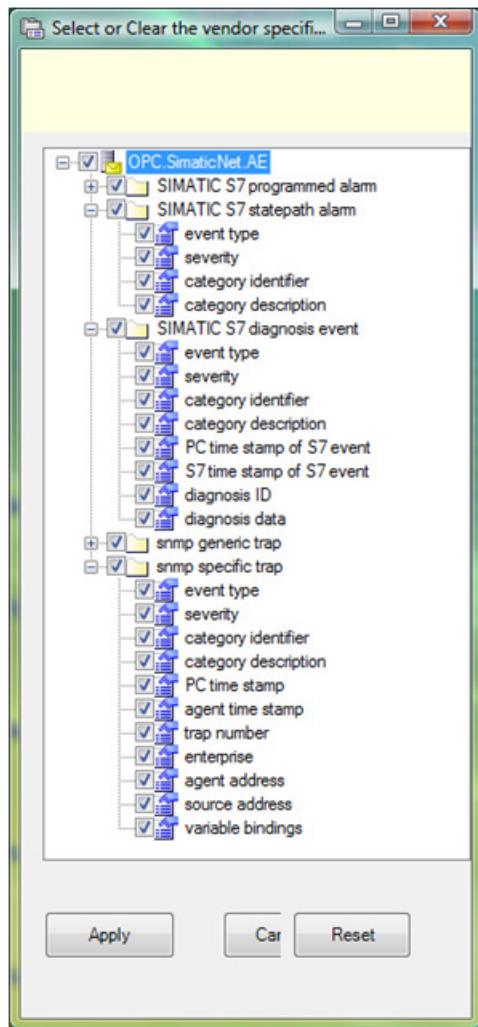


Figure 3-9 The Alarms & Events server "OPC.SimaticNET.AE"; view in OPC Scout V10

3.3.2 What are the names (ProgID) of the servers?

ProgID

The ProgIDs of the SIMATIC NET OPC Alarms & Events server are as follows:

- OPC.SimaticNET.AE
- OPC.SimaticNETAlarms
- OPC.SimaticNETAlarmsSNMP

3.3.3 How can the protocol of the alarm source be detected?

How can I find alarm sources?

The "IOPCEventAreaBrowser" interface is available to browse the possible alarm sources. This supports methods for filtering and querying OPC area and source parameters.

The following fixed area and source prefixes are returned:

- \S7::
- \SNMP::

Corresponding to the areas "\S7::" and "\SNMP::" it is possible to filter.

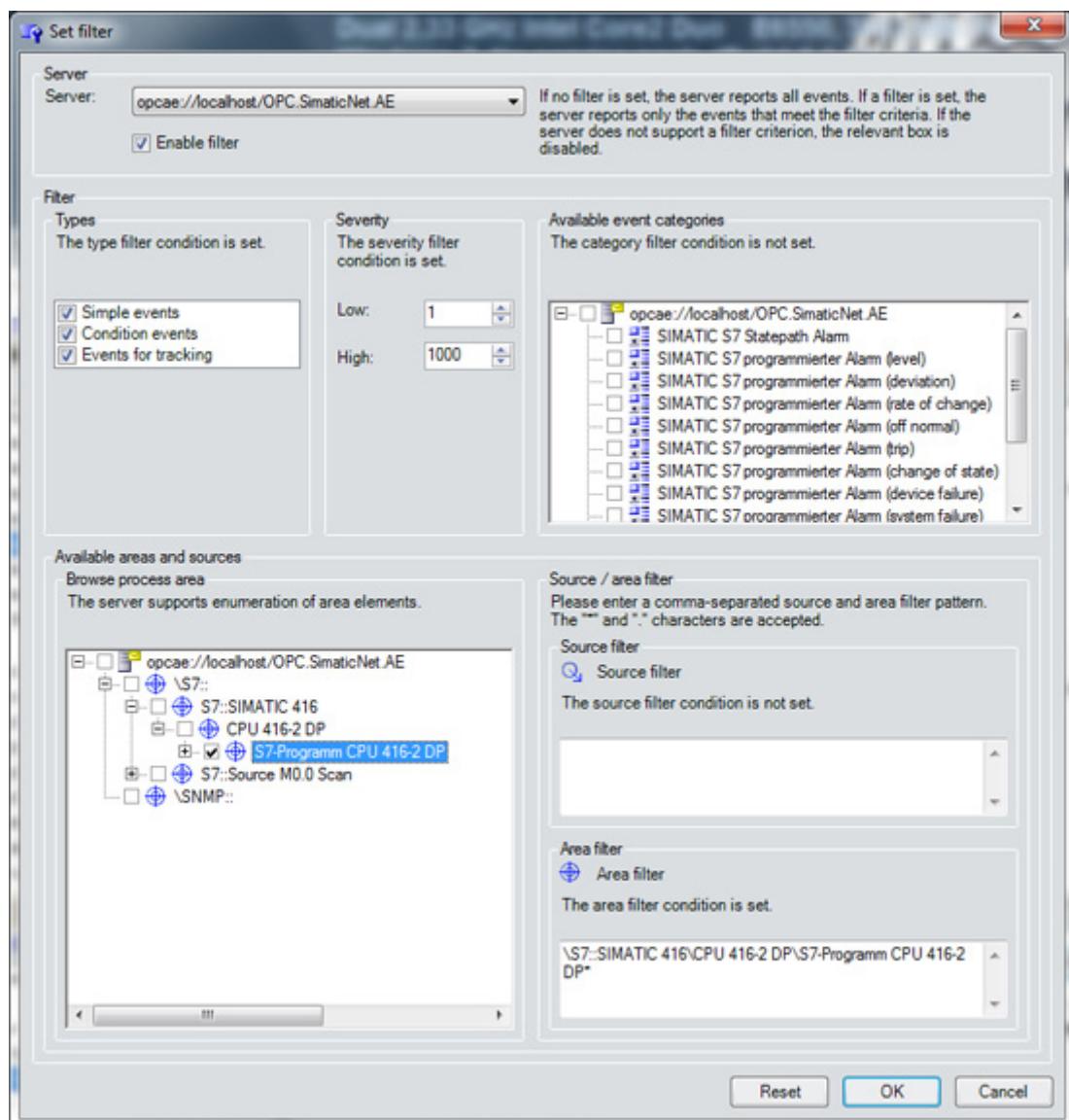


Figure 3-10 Filtering the alarm sources in OPC Scout V10

Special features of the "OPC.SimaticNET.AE" server

If the "OPC.SimaticNET.AE" server is used, this behaves differently with the source parameters of alarms:

- In addition to the protocol-specific source parameter (for example "\S7:\S7_connection_8"), there is also the area prefix "\S7::". This produces the following source parameter:
"\S7::\S7:\S7_connection_8"
- With a configurable source text "tank_pressure" of the S7 protocol, the text can also appear as follows:
"\S7::tank_pressure"

Otherwise the "OPC.SimaticNET.AE" server behaves like the individual servers "OPC.SimaticNetAlarms" and "OPC.SimaticNetAlarmsSNMP".

Using the OPC server

This section explains the various options open to you when you work with the OPC server.

4.1 Programming the automation interface

The Automation Interface is a supplement to the Custom Interface. With the Automation Interface, you have the convenience of modern development systems and script languages when programming OPC. There is both an Automation Interface for access to process variables (data access) and for processing events and alarms (Alarms & Events).

Application

You work with the Automation Interface when you want to create an application with a reasonable number of variables and medium data throughput based on an office application or with Visual Basic.

4.1.1 Programming the automation interface for data access

For data access, there is a simple class model, that groups the interfaces and their methods in classes.

The Automation Interface further refines the class model valid for the Custom Interface to provide the benefits and options available with structured development systems such as Visual Basic.

Note

The OPC Automation Interface for OPC Data Access has been released in a version from Siemens in which the errors have been corrected and is based on the version provided by the OPC Foundation.

4.1.1.1 What does the object model of OPC data access provide?

The classes of the Data Access class model contain the following objects:

- OPCServer
- OPCGroup
- OPCItem

Additional objects can be added for the automation interface. For administration of the "OPCGroup" and "OPCItem" objects there are separate collection objects "OPCGroups" and

"OPCItems". The collection objects provide functions for managing the objects assigned to them.

There is also an "OPCBrowser" collection object that contains the browsing functions.

Object model

The following graphic illustrates the objects and the relationships between the objects.

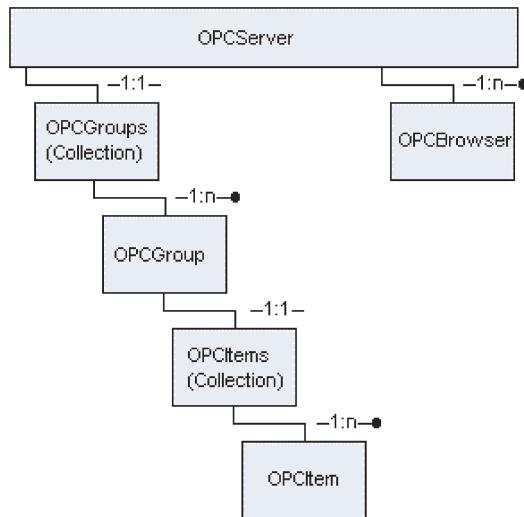


Figure 4-1 Object model of OPC data access

4.1.1.2 Points to remember when programming

- With the Automation Interface, optional parameters must be passed as variants. In Visual Basic, however, the optional parameters should be declared with the intended type of the optional variables. This ensures that the variant contains the correct data type.
- Asynchronous Functions**
The *IsSubscribed* group-specific property must be set to *True* to allow the variables to be monitored.
- The objects that will receive events must be declared with the supplement *withEvents* in Visual Basic.

Example:

Dim withEvents MyOPCGroup as OPCGroup

- According to the OPC Specification, arrays always begin with index 1. Define this in your program as follows:

```
Option Base 1 'ARRAYs are always Option Base 1
```

4.1.1.3 Objects of the automation interface for data access

Below, you will see a list of the properties, methods, and events for Data Access. Only the features specific to SIMATIC NET are described. For a detailed description of the properties, methods, and events, refer to the relevant OPC Specifications.

Objects

The following objects exist for the Automation Interface for Data Access:

OPCServer object

The OPCServer objects of the OPC server class are created by the client.

The properties of the OPCServer objects contain general information about the server. When an object is created, an OPCGroups collection object is also created as a property of the OPCServer object.

Properties of OPCServer

Below, you will find a list of the properties for the OPCServer object. Only the features specific to SIMATIC NET are described. You will find a description of the properties in the following OPC Specification:

Data Access Automation Interface

Version 2.02

February 4, 1999

Property	Meaning
Bandwidth	Returns the bandwidth of the server. Bandwidth is not supported by the OPC server for SIMATIC NET.
BuildNumber	Returns the build number of the server.
ClientName	Specifies the name of the client. ClientName is used primarily for test purposes.
CurrentTime	Returns the current time in UTC.
LastUpdateTime	Returns the time in UTC at which the server last sent data to the client.
LocaleID	Specifies the language for displayed texts. For SIMATIC NET, these are German and English.
MajorVersion	Returns the main version number of the server.
MinorVersion	Returns the secondary version number of the server.
OPCGroups	Specifies the collection of OPCGroup objects
PublicGroupNames	Returns the name of the public group of OPC server. Optional public groups are not supported by the OPC Server for SIMATIC NET.
ServerName	Returns the name of the connected OPC server.
ServerNode	Returns the name of the network node on which the OPC server is executed.
ServerState	Returns the status of the server.

Property	Meaning
StartTime	Returns the time at which the server started in UTC.
VendorInfo	Returns vendor information. The OPC server for SIMATIC NET returns "SIMATIC NET OPC server" as vendor information.

Methods of OPCServer

The methods for the OPCServer object are listed below. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Automation Interface
Version 2.02
February 4, 1999

Method	Meaning
Connect	Establishes a connection to the OPC server. The <i>ProgID</i> for OPC server is, for example, "OPC.SimaticNET".
CreateBrowser	Creates an OPCBrowser collection object.
Disconnect	Terminates a connection to the OPC server. The OPC Server for SIMATIC NET terminates all communication connections to an OPC client after the last link has been deleted.
GetErrorString	Gets an error message for an error code. The OPC Server for SIMATIC NET supports German and English error messages. Error messages from the Windows operating system are displayed in the language of the operating system installation.
GetItemProperties	Returns a list with the values of the requested properties.
GetOPCServers	Returns the names of the registered OPC servers. The name of the OPC server for SIMATIC NET is, for example, "OPC.SimaticNET".
LookupItemIDs	Returns a list with ItemIDs that correspond to the PropertyIDs. The OPC Server for SIMATIC NET does not return PropertyIDs that could be displayed as ItemIDs.
QueryAvailableLocaleIDs	Returns the available language codes. The OPC Server for SIMATIC NET supports German and English error messages. Error messages from the Windows operating system are displayed in the language of the operating system installation.
QueryAvailableProperties	Returns property codes and properties for an OPC item.

Events of OPCServer

Below, you will find a description of the event for the OPCServer object. Only the features specific to SIMATIC NET are described. You will find a description of the event in the following OPC Specification:

Data Access Automation Interface
Version 2.02
February 4, 1999

Event	Meaning
ServerShutdown	This is triggered when the OPC server is shut down. The OPC Server for SIMATIC NET triggers this event when the shutdown command is sent by the configuration program or when the PC station receives new configuration data.

Collection object OPCBrowser

Using the OPCBrowser collection object, you can browse through the namespace of the OPC server.

An object of the OPCBrowser class is created by the *CreateBrowser* method of the OPCServer object. Several OPCBrowser objects can be created for one server.

Properties of OPCBrowser

Below, you will find a list of the properties for the OPCBrowser collection object. Only the features specific to SIMATIC NET are described. You will find a description of the properties in the following OPC Specification:

Data Access Automation Interface
Version 2.02
February 4, 1999

Property	Meaning
AccessRights	Decides the access rights for the <i>ShowLeafs</i> method.
Count	Returns the number of entries.
CurrentPosition	Returns the current position in the namespace tree.
DataType	Decides the data type for the <i>ShowLeafs</i> method.
Filter	Decides the filter for the <i>ShowLeafs</i> and <i>ShowBranches</i> methods.
Organization	Returns the organizational structure of the namespace. The namespace of the OPC Server for SIMATIC NET is hierarchically structured.

Methods of OPCBrowser

Below, you will find a list of the methods for the OPCBrowser collection object. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Automation Interface

Version 2.02

February 4, 1999

Method	Meaning
GetAccessPaths	Gets the access path of an ItemID. For the OPC Server for SIMATIC NET, you should not use the AccessPath parameter.
GetItemID	When possible, this returns the ItemID for an element in the namespace of the OPC server.
Item	Specifies the name of an entry
MoveDown	Moves the current position in the namespace one level down
MoveTo	Moves the current position in the namespace to the specified position
MoveToRoot	Moves the current position in the namespace to the root
MoveUp	Moves the current position in the namespace one level up
ShowBranches	Specifies the name of the branches at the current browse position
ShowLeafs	Specifies the name of the leaves at the current browse position

Collection object OPCGroups

The OPCGroups collection object is a collection of OPCGroup objects for creating and managing OPC groups. The standard properties of OPCGroups specify standard values for all newly created OPC groups.

If the OPC server object executes a *Connect* call successfully, an OPCGroups collection object is created as a property of the OPC server object.

Note

The optional public groups are not supported by the OPC Server for SIMATIC NET.

Properties of OPCGroups

Below, you will find a list of the properties for the OPCGroups collection object. Only the features specific to SIMATIC NET are described. You will find a description of the properties in the following OPC Specification:

Data Access Automation Interface

Version 2.02

February 4, 1999

Property	Meaning
Count	Returns the number of groups.
DefaultGroupDeadband	Specifies the initial value for <i>Deadband</i> for newly generated OPCGroup objects.
DefaultGroupIsActive	Specifies the initial value for <i>ActiveState</i> for newly generated OPCGroup objects.

Property	Meaning
DefaultGroupLocaleID	Specifies the initial value for <i>LocaleID</i> for newly generated OPCGroup objects.
DefaultGroupTimeBias	Specifies the initial value for <i>TimeBias</i> for newly generated OPCGroup objects.
DefaultGroupUpdateRate	Specifies the initial value for <i>UpdateRate</i> for newly generated OPCGroup objects.
Parent	Returns the reference to the corresponding OPCServer object.

Methods of OPCGroups

Below, you will find a list of the methods for the OPCGroups collection object. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Automation Interface

Version 2.02

February 4, 1999

Method	Meaning
Add	Creates a new OPCGroup object and adds it to the collection.
GetOPCGroup	Specifies the reference to the name or server handle of an OPCGroup object.
Item	Returns the reference to the indexed object of the collection.
Remove	Deletes a group of the server.
RemoveAll	Deletes all groups and items of the server.

Events of OPCGroups

Below, you will find a description of the event for the OPCGroups collection object. Here, only the special features for SIMATIC NET are described. You will find a detailed description of the event in the following OPC Specification:

Data Access Automation Interface

Version 2.02

February 4, 1999

Event	Meaning
GlobalDataChange	Notifies value changes and the status of the active items of all active groups.

OPCGroup object

The OPC Group class manages the individual process variables, the OPC items. Using the OPCGroup object, a client can put OPC items into a group based on semantic considerations and process the group as a single unit.

Monitoring variables and read and write access are group-specific. As an example, monitoring of all the OPC items in a group can be activated by calling a single function.

You should therefore put all the process variables, for example, that are displayed together in one screen on an operator control and monitoring station into the same group and activate monitoring of the variables when the screen is opened.

Note

The optional public groups are not supported by the OPC Server for SIMATIC NET.

Properties of OPCGroup

Below, you will find a list of the properties for the OPCGroup object. Only the features specific to SIMATIC NET are described. You will find a description of the properties in the following OPC Specification:

Data Access Automation Interface
Version 2.02
February 4, 1999

Property	Meaning
ClientHandle	Specifies the handle for localizing data.
DeadBand	Specifies the bandwidth in which value changes do not lead to a notification.
IsActive	Specifies the active status of the group. Make sure you set <i>IsActive</i> to <i>True</i> if you want to monitor the variables of this group.
IsPublic	Indicates whether or not the group is a public group. The optional public groups are not supported by the OPC Server for SIMATIC NET.
IsSubscribed	Indicates whether the variables of a group will be monitored. Make sure you set <i>IsSubscribed</i> to <i>True</i> if you want to monitor the variables of this group.
LocaleID	Specifies the language for text string as returned by the server.
Name	Specifies the name of the group.
OPCItems	Specifies the collection object for managing the items of the group.
Parent	Returns the reference to the corresponding OPCServer object.
ServerHandle	Returns a unique handle for the group.
TimeBias	Specifies the time offset for converting the time stamp to local time.
UpdateRate	Specifies the interval at which a client is notified of changes to values or in the statuses of items.

Methods of OPCGroup

Below, you will find a list of the methods for the OPCGroup object. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Automation Interface
Version 2.02
February 4, 1999

Method	Meaning
AsyncCancel	Cancels an asynchronous job.
AsyncRead	Sends an asynchronous read command.
AsyncRefresh	Creates an event for every active OPC item with the current value from the cache.
AsyncWrite	Sends an asynchronous write command.
SyncRead	Starts synchronous reading of values, status information, and time stamp of one or more items of a group.
SyncWrite	Starts synchronous writing of values for one or more items of a group.

Events of OPCGroup

The OPC Automation Interface returns the changes to values of active items and the results of asynchronous operations using events.

Objects in Visual Basic that are intended to receive events must be declared with *withEvents*.

Below, you will find a list of the events for the OPCGroup object. Only the features specific to SIMATIC NET are described. You will find a description of the events in the following OPC Specification:

Data Access Automation Interface
Version 2.02
February 4, 1999

Event	Meaning
AsyncCancelComplete	This is triggered when a cancel job is completed.
AsyncReadComplete	This is triggered when a read job is completed.
AsyncWriteComplete	This is triggered when a write job is completed.
DataChange	This is triggered when one or more items have a changed value or a changed quality.

Collection object OPCItems

The OPCItems collection object is a collection of OPCItem objects for creating and managing OPC items. The properties of OPCItems specify standard values for all new OPC items that are created.

If an OPCGroup object is created, an OPCItems collection object is automatically created. OPCItems always exists as a property of a group and is used to monitor the OPC variables.

Properties of OPCItems

Below, you will find a list of the properties for the OPCItems object. Only the features specific to SIMATIC NET are described. You will find a description of the properties in the following OPC Specification:

Data Access Automation Interface

Version 2.02

February 4, 1999

Property	Meaning
Count	Returns the number of items in the group.
DefaultAccessPath	Specifies the initial value for AccessPath for newly added OPC items. For the OPC Server for SIMATIC NET <i>DefaultAccessPath</i> should be empty.
DefaultIsActive	Specifies the initial value for <i>ActiveState</i> for newly added OPC items.
DefaultRequestedDataType	Specifies the initial value for <i>RequestedDataType</i> for newly added OPC items.
Parent	Returns the reference to the corresponding OPCGroup object.

Methods of OPCItems

Below, you will find a list of the methods for the OPCItems object. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Automation Interface

Version 2.02

February 4, 1999

Method	Meaning
AddItem	Adds a new OPC item to the collection object.
AddItems	Adds several OPC items to the collection object.
GetOPCItem	Specifies the reference to the server handle created by <i>AddItem</i> .
Item	Specifies the reference to an item of the collection.
Remove	Deletes one or more items from a group.
SetActive	Sets the active status of one or more items of a group.
SetClientHandles	Changes the client handle for one or more items.
SetDataTypes	Sets the data type for one or more items.
Validate	Checks the validity of one or more OPC items.

OPCItem object

An object of the OPC Item class represents a process variable, for example an input module of a programmable controller. A process variable is a writable and/or readable data item of the process I/O such as the temperature of a tank. Each process variable is associated with a value, a quality and a time stamp.

Properties of OPCItem

Below, you will find a list of the properties for the OPCItem object. Only the features specific to SIMATIC NET are described. You will find a description of the properties in the following OPC Specification:

Data Access Automation Interface
Version 2.02
February 4, 1999

Property	Meaning
AccessPath	Returns the access path of the item. For the OPC Server for SIMATIC NET, Access Path should be empty.
AccessRights	Returns the access rights of the variables.
CanonicalDataType	Returns the original data type of the item.
ClientHandle	Specifies the handle for simpler assignment of process variables to internal data structures of the client.
EUInfo	Returns information about the units of the value (optional). The OPC Server for SIMATIC NET does not support units (Engineering Units).
EUType	Returns the unit of the returned value (optional). The OPC Server for SIMATIC NET does not support units (Engineering Units).
IsActive	Specifies whether notification events are generated for the item.
ItemID	Returns the unique name of the item.
Parent	Returns the reference to the parent OPCGroup object.
Quality	Returns the quality of the value read last.
RequestedDataType	Specifies the requested data type for the value of the item.
ServerHandle	Returns the server handle to identify an item.
TimeStamp	Returns the time at which the last value was acquired.
Value	Returns the last valid value of an item.

Methods of OPCItem

Below, you will find a list of the methods for the OPCItem object. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Automation Interface
Version 2.02
February 4, 1999

Method	Meaning
Read	Reads value, quality and/or time stamp of the variable synchronously.
Write	Sets the value of the variable synchronously.

4.1.2 Programming the automation interface for Alarms & Events

For Alarms & Events, there is a simple class model, that groups the interfaces and their methods in classes.

4.1 Programming the automation interface

The Automation Interface further refines the class model valid for the Custom Interface to provide the benefits and options available with structured development systems such as Visual Basic.

4.1.2.1 What does the object model of OPC Alarms & Events provide?

The classes of the class model of Alarms & Events contain the following objects:

- OPCEventServer
- OPCEventSubscriptions
- OPCEventSubscription
- OPCEventAreaBrowsers
- OPCEventAreaBrowser
- OPCEvents
- OPCEvent
- OPCEventCondition
- OPCEventSubConditions
- OPCEventSubCondition

Additional objects can be added for the Automation Interface.

Note

Since the Event Server of SIMATIC NET is a Simple Event Server, the following objects are not supported: OPCEventAreaBrowsers, OPCEventAreaBrowser, OPCEventCondition, OPCEventSubConditions, OPCEventSubCondition.

Object model

The following graphic illustrates the objects and the relationships between the objects.

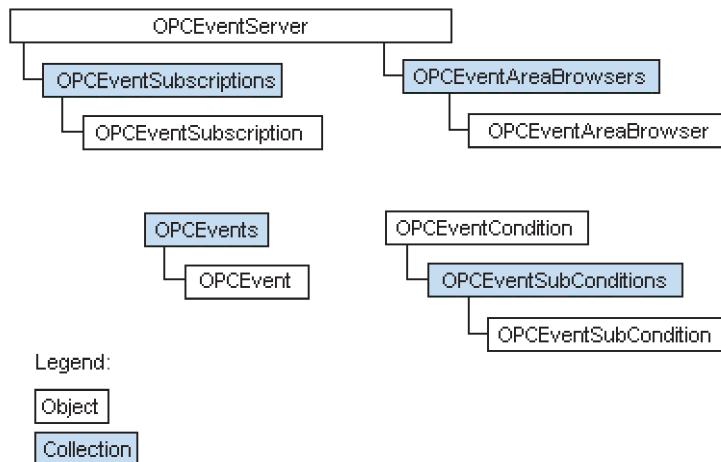


Figure 4-2 Object model of OPC Alarms & Events

4.1.2.2 Points to remember when programming

With the Automation Interface, optional parameters must be passed as variants. In Visual Basic, however, optional parameters should be declared with the target type. This ensures that the variant contains the correct data type.

- Asynchronous Functions
The *IsSubscribed* groupspecific property must be set to *True* to allow the variables to be monitored.
- The objects that will receive events must be declared with the supplement *withEvents* in Visual Basic.

Example:

`Dim withEvents MyOPCGroup as OPCGroup`

- According to the OPC Specification, arrays always begin with index 1. Define this in your program as follows:
Option Base 1 'ARRAYs are always Option Base 1

4.1.2.3 Objects of the automation interface for Alarms & Events

Below, you will see a list of the properties, methods, and events for Alarms & Events. Only the features specific to SIMATIC NET are described. For a detailed description of the properties, methods, and events, refer to the relevant OPC Specifications.

Objects

The following objects exist for the Automation Interface for Alarms & Events:

OPCEventServer object

OPCEventServer objects of the OPC Event Server class are created by the client. OPCEventServer must be created before the client can access other objects of Alarms & Events.

The properties of the OPCEventServer objects contain general information about the event server. When an object is created, an OPCEventServerSubscription collection object is also created.

OPCEventServer is connected to the event server by the *Connect* method.

Properties of OPCEventServer

Below, you will find a list of the properties for the OPCEventServer object. Only the features specific to SIMATIC NET are described. You will find a description of the properties in the following OPC Specification:

Alarm & Events Automation Interface Standard

Version 1.01

December 15, 1999

Property	Meaning
BuildNumber	Returns the build number of the server.
ClientName	Specifies the name of the client.
CurrentTime	Returns the current time in UTC.
FiltersByArea	Indicates whether or not the server can filter according to areas. The OPC Event Server does not support areas.
FiltersByCategory	Indicates whether or not the server can filter according to event categories.
FiltersByEventType	Indicates whether or not the server can filter according to event types.
FiltersBySeverity	Indicates whether or not the server can filter according to the severity of events.
FiltersBySource	Indicates whether or not the server can filter according to sources.
LastUpdateTime	Returns the time in UTC at which the server last sent data to the client.
LocaleID	Specifies the language for displayed texts.
MajorVersion	Returns the main version number of the server.
MinorVersion	Returns the secondary version number of the server.
OPCEventAreaBrowsers	Specifies the collection of <i>OPCAutoEventAreaBrowser</i> objects. OPC event server of SIMATIC NET does not support <i>OPCEventAreaBrowsers</i> .
OPCEventSubscriptions	Specifies the collection of <i>OPCEventSubscription</i> objects.
ServerName	Returns the name of the connected OPC server.
ServerNode	Returns the name of the network node on which the OPC server is executed.
ServerState	Returns the server status.
StartTime	Returns the time at which the OPC server was started.
VendorInfo	Returns vendor information.

Methods of OPCEventServer

Below, you will find a list of the methods for the OPCEventServer object. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Alarm & Events Automation Interface Standard
Version 1.01
December 15, 1999

Method	Meaning
AckCondition	Acknowledges one or more conditions on the event server.
Connect	Establishes the connection to the OPC server.
Disconnect	Terminates the connection to the OPC server.
EnableConditionsByArea	Sets all conditions for all areas to a defined status.
EnableConditionBySrc	Sets all conditions for all sources to a defined status.
GetConditionState	Gets current status information for the condition.
GetErrorString	Converts an error code to an error text.
GetOPCEventServers	Returns the names of all registered event servers.
QueryAvailableLocaleIDs	Returns the available language codes.
QueryConditionNames	Returns the names of the conditions valid for a specific event category.
QueryEventAttributes	Returns vendor-specific attributes
QueryEventCategories	Returns the categories supported by the event server.
QuerySourceConditions	Returns the names of conditions linked to a specific source.
QuerySubConditionNames	Returns the names of subconditions linked to a specific source.

Events of OPCEventServer

Below, you will find a list of the events for the OPCEventServer object. Only the features specific to SIMATIC NET are described. You will find a description of the event in the following OPC Specification:

Alarm & Events Automation Interface Standard
Version 1.01
December 15, 1999

Event	Meaning
EventServerShutDown	This is triggered when the OPC server is shut down. The OPC Server for SIMATIC NET triggers this event when the shutdown command is sent by the configuration program or when the PC station receives new configuration data.

Collection object OPCEventSubscriptions

The OPCEventSubscriptions collection object is a collection of OPCEventSubscription objects and the methods with which to create, remove, and manage them.

The properties of OPCEventSubscriptions specify standard values for all newly created OPCEventSubscription objects.

When the *Connect* call of the OPCEventServer object is successfully executed, an OPCEventSubscriptions collection object is created automatically. OPCEventSubscriptions always exists as a property of the OPCEventServer object and is used to manage event messages.

Properties of OPCEventSubscriptions

Below, you will find a list of the properties for the OPCEventSubscriptions object. Only the features specific to SIMATIC NET are described. You will find a description of the properties in the following OPC Specification:

Alarm & Events Automation Interface Standard
Version 1.01
December 15, 1999

Property	Meaning
Count	Returns the number of entries.
DefaultIsActive	Specifies the initial value for the active status for newly generated OPCEventSubscription objects.
DefaultbufferTime	Specifies the initial value for how often event notifications are sent for newly generated OPCEventSubscription objects.
DefaultMaxSize	Specifies the initial value for the highest number of events that can be sent with a single event notification for newly generated OPCEventSubscription objects.

Methods of OPCEventSubscriptions

Below, you will find a list of the methods for the OPCEventSubscriptions object. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Alarm & Events Automation Interface Standard
Version 1.01
December 15, 1999

Method	Meaning
Add	Creates a new OPCEventSubscription object and adds it to the collection.
Item	Specifies the reference to the indexed object of the collection.
Remove	Deletes an OPCEventSubscription object.
RemoveAll	Deletes all OPCEventSubscription objects.

OPCEventSubscription object

The OPCEventSubscription object represents a specific event subscription. A subscription is a season ticket to a set of events. The client starts a job for the regular sending of events to the event server.

Properties of OPCEventSubscription

Below, you will find a list of the properties for the OPCEventSubscription object. Only the features specific to SIMATIC NET are described. You will find a description of the properties in the following OPC Specification:

Alarm & Events Automation Interface Standard
Version 1.01
December 15, 1999

Property	Meaning
bufferTime	Decides how often event notifications are sent for the OPCEventSubscription object.
IsActive	Specifies the active status of the OPCEventSubscription object.
MaxSize	Specifies the highest number of events that can be sent with a single event notification.
Name	Specifies the name of the OPCEventSubscription object.

Methods of OPCEventSubscription

Below, you will find a list of the methods for the OPCEventSubscription object. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Alarm & Events Automation Interface Standard
Version 1.01
December 15, 1999

Method	Meaning
GetFilter	<p>Returns the current filter for the OPCEventSubscription object. The parameters of the S7 OPC Event Server for SIMATIC NET have the following significance:</p> <p><i>EventType</i> The event types OPC_SIMPLE_EVENT and OPC_CONDITION_EVENT are supported.</p> <p><i>EventCategory</i> The event categories are described in section "Event server for S7 communication (Page 383)"</p> <p><i>LowSeverity</i></p> <p><i>HighSeverity</i></p> <p><i>Areas</i> The OPC event server does not support areas.</p> <p><i>Sources</i> You can enter a connection name as the source.</p>
GetReturnedAttributes	Returns attributes for every event category that the server sends with the event notifications.
Refresh	Refreshes all conditions.
RefreshCancel	Cancels execution of the <i>Refresh</i> method. The S7 OPC Event Server for SIMATIC NET does not support <i>RefreshCancel</i> .

Method	Meaning
SelectReturnedAttributes	For each event category, this specifies the attributes that are returned with the event notifications by the <i>OnEvent</i> method.
SetFilter	Sets all filters so that the filter properties match those of the created events. The parameters of the S7 OPC Event Server for SIMATIC NET have the following significance: <i>EventType</i> The event types OPC_SIMPLE_EVENT and OPC_CONDITION_EVENT are supported. <i>EventCategory</i> S7_PROCESS_ALARM <i>LowSeverity</i> <i>HighSeverity</i> <i>Areas</i> S7 OPC event server does not support areas. <i>Sources</i> You can enter a connection name as the source.

Events of OPCEventSubscription

Below, you will find a list of the events for the OPCEventSubscription object. Only the features specific to SIMATIC NET are described. You will find a description of the event in the following OPC Specification:

Alarm & Events Automation Interface Standard
Version 1.01
December 15, 1999

Event	Meaning
ConditionEvent	This occurs when condition events are sent by the server.
RefreshCancel	This occurs when the <i>Refresh</i> method is interrupted. OPC event server of SIMATIC NET does not support <i>RefreshCancel</i> .
RefreshComplete	This occurs when the <i>Refresh</i> method is completed. OPC event server of SIMATIC NET does not support <i>RefreshComplete</i> .
RefreshConditionEvent	This occurs when events relating to the refreshing of conditions are sent by the server.
SimpleEvent	This occurs when a group of simple events are sent by the server.
TrackingEvent	This occurs when tracking events are sent by the server. OPC event server of SIMATIC NET does not support <i>TrackingEvent</i> .

Collection Object OPCAutoEventAreaBrowsers

The OPCAutoEventAreaBrowsers collection object is a collection of OPCAutoEventAreaBrowser objects and the methods with which to create, remove, and manage them.

Note

The OPC Event Server does not support OPCAutoEventAreaBrowsers.

OPCAutoEventAreaBrowser object

With the OPCAutoEventAreaBrowser object, a client can browse through the areas and sources of the server. The areas are grouped together to form OPCEventAreas and the sources form OPCEventSources.

The areas and sources can be used to filter events.

Namespace of the server

Servers have a flat or hierarchical namespace. If the namespace is flat, OPCEventAreas and OPCEventSources have the same areas and sources on the server.

If the namespace is hierarchical, the areas can be considered as branches in a tree and the sources as leaves. The OPCAutoEventAreaBrowser object moves like a pointer along the areas. OPCEventAreas and OPCEventSources contain only the areas and sources in which the object exists.

Note

The OPC Event Server does not support OPCAutoEventAreaBrowser.

Collection object OPCEvents

The OPCEvents collection object is the transfer parameter of the methods for handling events. It contains a collection of OPCEvent objects and the methods with which to create, remove, and manage them.

OPCEvents is generated from the event notifications sent by the server and therefore delivers the events that have occurred.

Properties of OPCEvents

Below, you will find a list of the properties for the OPCEvents object. Only the features specific to SIMATIC NET are described. You will find a description of the properties in the following OPC Specification:

Alarm & Events Automation Interface Standard

Version 1.01

December 15, 1999

Property	Meaning
Count	Returns the number of entries.
LastRefresh	Specifies that the OPCEvents collection object is the last in a series of refresh notifications.
Refresh	Specifies that the OPCEvents collection object belongs to a refresh notification.

Methods of OPCEvents

Below, you will find a list of the methods for the OPCEvents object. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Alarm & Events Automation Interface Standard

Version 1.01

December 15, 1999

Method	Meaning
Add	Creates a new OPCEvent object and adds it to the collection.
Item	Specifies the name of an entry.

OPCEvent object

The OPCEvent object contains the notification for a specific event.

Properties of OPCEvent

Below, you will find a list of the properties for the OPCEvent object. Only the features specific to SIMATIC NET are described. You will find a description of the properties in the following OPC Specification:

Alarm & Events Automation Interface Standard

Version 1.01

December 15, 1999

Property	Meaning
AckRequired	Indicates that the condition requires an acknowledgment.
ActiveTime	Returns the time at which the condition was activated.
ActorID	Returns the tracking and condition events that require acknowledgment.
ChangeAckState	Returns the event notification for the change to the <i>Acknowledge</i> property of a condition.
ChangeActiveState	Returns the event notification for the change to the <i>ActiveState</i> property of a condition.
ChangeAttribute	Returns the event notification for the change to the <i>Attribute</i> property of a condition.

Property	Meaning
ChangeEnableState	Returns the event notification for the change to the <i>Enable</i> property of a condition.
ChangeMessage	Returns the event notification for the change to the <i>Message</i> property of a condition. OPC event server of SIMATIC NET does not support <i>ChangeMessage</i> .
ChangeQuality	Returns the event notification for the change to the <i>Quality</i> property of a condition. OPC event server of SIMATIC NET does not support <i>ChangeQuality</i> .
ChangeSeverity	Returns the event notification for the change to the <i>Severity</i> property of a condition. OPC event server of SIMATIC NET does not support <i>ChangeSeverity</i> .
ChangeSubCondition	Returns the event notification for the change to the <i>SubCondition</i> property of a condition.
ConditionAcknowledged	Indicates that the new status of the condition is <i>Acknowledged</i> .
ConditionActive	Indicates that the new status of the condition is <i>Active</i> .
ConditionName	Returns the name of the condition that refers to the event notification.
Cookie	Returns a cookie defined by the server that is linked to the event notification. OPC event server of SIMATIC NET does not support <i>Cookie</i> .
EventCategory	Returns the code for the event category of the event.
Message	Returns a text describing the notification.
OPCEventAttributes	Returns the vendor-specific event attributes returned by the event notification.
Quality	Returns the quality associated with the status of the condition. OPC event server of SIMATIC NET does not support <i>Quality</i> .
Severity	Returns the severity of the event.
Source	Returns the source of the event notification.
SubConditionName	Returns the name of the current subcondition. OPC event server of SIMATIC NET does not support <i>SubCondition-Name</i> .
Time	Returns the time in UTC at which an event occurred.

OPCEventCondition object

The OPCEventCondition object describes the current status of a condition.

Collection object OPCEventSubConditions

The OPCEventSubConditions collection object is a collection of OPCEventSubCondition objects. The collection represents the various statuses of a condition on an OPC Event Server.

OPCEventSubCondition object

The OPCEventSubCondition object represents a specific subcondition of OPC Event Server. OPCEventSubCondition contains the attributes of the subcondition.

4.2 Programming the custom interface

The Custom Interface is designed so that it functions ideally to meet the given requirements. For access using script languages, it is not suitable. For this purpose, there is the Automation Interface.

There is both a Custom Interface for access to process variables (data access) and for processing events and alarms (Alarms & Events).

Application

You program the Custom Interface if you want to create an application with a large number of variables and high data throughput with C++.

CLSID

Each COM class can be uniquely identified by an identification code. This is 128 bits long and is called the CLSID. Using the CLSID, the operating system can find the "*.dll" or "*.exe" files in which the COM class is implemented. If the client wants to use an object of a class, it references it using the CLSID.

ProgID

To simplify identification of OPC servers, a readable ProgID is assigned to the CLSIDs. The CLSID and ProgID are specified by the vendor of the OPC server.

For the OPC server of SIMATIC NET, the following points apply to all protocols:

Interface	ProgID
Data access	OPC.SimaticNET (standard OPC server for multiple protocol operation) OPC.SimaticNET (high-performance OPC server for S7, SR, DP single protocol mode. Symbols and DX are also allowed) OPC.SimaticNET.DP (OPC server for highly efficient DP access)
Alarms & Events for S7	OPC.SimaticNetAlarms (A&E server for S7) OPC.SimaticNetAlarms (high-performance A&E server for S7 single protocol mode; symbols and DX also allowed) OPC.SimaticNetAlarmsSNMP (A&E server for SNMP)

Proceed as follows

You create a COM object and call its methods.

Note

A Windows object is an instance of a COM class. The COM class differs from the C++ class. A class in C++ is a type definition. A COM class is an object description and contains no types.

4.2.1 Creating COM objects and querying the status of the OPC server

You will find a detailed description of using the Custom Interface for OPC Data Access in the description of the examples. The description below simply outlines the basic sequence and makes no attempt to deal with error handling.

You create a COM object and query the status in five steps.

For examples of creating a COM object in Visual C++, refer to the section "Sample programs (Page 619)".

4.2.2 Objects of the custom interface for data access

This section lists the interfaces of the objects and their methods. Only the features specific to SIMATIC NET are described. For a detailed description of the interfaces, refer to the relevant OPC Specifications.

Return values of the methods of the interfaces

All methods return a result of the type *HResult*.

4.2.2.1 OPCServer object

An object of the OPC server class has various attributes that contain, for example, information on the status or the version of an OPCServer object. The class also has methods with which a client manages the objects of the OPC Group class.

A client application only addresses the OPC server object directly using COM mechanisms. All other objects are created by the relevant OPC methods.

Interfaces of OPCServer

The OPCServer object has the following interfaces:

- IOPCServer
- IOPCBrowse (new as of 3.00)
- IOPCItemIO (new as of 3.00)

- IOPCBrowseServerAddressSpace (2.0)
- IOPCCCommon
- IConnectionPointContainer
- IOPCItemProperties

The OPC DA specification with which the interface is supported is shown in parentheses.

The graphic below shows the interfaces of the OPCServer object.

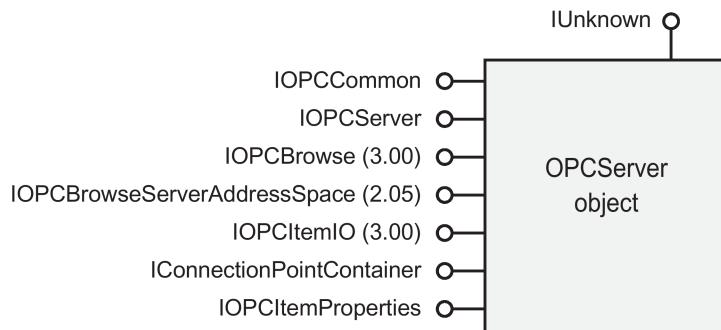


Figure 4-3 The OPCServer object

IOPCServer interface

The IOPCServer interface contains methods for managing the objects of the OPC Group class. Information can also be obtained about the current status of the server.

Below, you will find a list of the methods for the IOPCServer interface. Only the features specific to SIMATIC NET are described.

Method	Meaning
AddGroup	<p>Generates a group in the server object. The <i>LCID</i> parameter is not evaluated with the OPC Server for SIMATIC NET. The <i>pTimeBias</i> parameter specifies the time difference compared with the local time zone. The update rates used by the OPC Server for SIMATIC NET are multiples of the cycle time specified during configuration. The minimum update rate is the same as the cycle time. If <i>szName</i> is empty, a name is generated that starts with the tilde character (for example ~GROUP_1). User-defined names should therefore not begin with a tilde. The <i>pPercentDeadband</i> parameter causes a restricted change message to the OPC client if the configuration includes a high limit (EUHigh) and low limit (EULow) and the item value changes by (<i>pPercentDeadband</i>/100) * (EU High - EU Low). This applies only to analog data (EU-Type =1) and canonical types VT_I2, I4, R4, R8, BOOL, UI1.</p>
CreateGroupEnumerator	<p>Creates various enumerators for the group. The OPC Server for SIMATIC NET does not support public groups. The return values of the methods should therefore be identical with the input values ...PRIVATE and ...PUBLIC (for Public Groups) for the parameter <i>dwScope</i>.</p>
GetErrorString	<p>Gets an error message for an error code. The OPC Server for SIMATIC NET supports German and English error texts. Error messages from the Windows operating system are displayed in the language of the operating system installation.</p>
GetGroupByName	<p>Returns an additional interface pointer to the name of a private group; the reference counter is incremented.</p>
GetStatus	<p>Returns status information on the server. The return value is the name and the version of the OPC server.</p>
RemoveGroup	<p>Deletes a group in the server. The OPC server for SIMATIC NET does not support the use of <i>bForce</i>. You cannot delete groups if references to them still exist.</p>

You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
 Version 3.00
 March 4, 2003

IOPCBrowse interface

The essential advantage of this new browse interface is that ItemIDs and properties can be read out even when simply browsing the namespace. This allows the number of OPC server calls to be reduced significantly. The functionality is the same as that of OPC XML-DA.

Method	Meaning
Browse	This method browses a branch in the namespace and identifies no or several namespace elements. The namespace should have a hierarchical structure, a flat structure is shown to the OPC client without sub-elements. A hierarchical namespace has both branch and leaf subelements. The ItemIDs and properties can also be read out.
GetProperties	This method returns an array of OPCItemProperties for each ItemID.

You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard

Version 3.00

March 4, 2003

IOPCItemIO interface

With IOPCItemIO, it is possible to read/write items without groups. This is simpler and more efficient in particular for single calls (reading configuration variables).

You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard

Version 3.00

March 4, 2003

Method	Meaning
Read	Reads one or more values, qualities and timestamps synchronously. This function can be compared with the IOPCSyncIO::Read method.
WriteVQT	Writes one or more values synchronously. The writing of quality and timestamp is not supported; this is rejected with the message OPC_E_NOTSUPPORTED.

IOPCServerPublicGroups Interface

The OPC Server for SIMATIC NET does not support public groups and therefore does not support the optional IOPCServerPublicGroups interface.

IOPCBrowseServerAddressSpace interface

Using the OPC DA-2.0 interface IOPCBrowseServerAddressSpace, you can browse the namespace of the server. The namespace contains all the OPC items known to the server.

Below, you will find a list of the methods for the IOPCBrowseServerAddressSpace interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 2.05
December 17, 2001

Method	Meaning
BrowseAccessPaths	Gets the access path of an ItemID. BrowseAccessPaths is not required with the OPC Server for SIMATIC NET.
BrowseOPCItemIDs	Returns a string of the type <i>IEnumString</i> , whose content is specified by the parameters of the call. If the input parameter <i>dwBrowseFilterType</i> is set to OPC_BRANCH, the parameters <i>vtDataTypeFilter</i> and <i>dwAccessRightsFilter</i> have no effect. The rules for creating a filter are as follows: *: any character string including empty strings +: any character string, but there must be at least one character ?: exactly one character [] (square brackets): exactly one character from the specified set If you use a filter character, this must be preceded by a backslash (\).
ChangeBrowsePosition	Changes to the higher level or to a branch in the namespace.
GetItemID	Gets a complete ItemID in the hierarchical namespace. The OPC Server for SIMATIC NET supports GetItemID only for single leaves.
QueryOrganization	Returns the structure of the namespace. The namespace of the OPC Server for SIMATIC NET is hierarchically structured.

IOPCItemProperties interface

The IOPCItemProperties interface contains methods for querying server--specific information on an item.

Below, you will find a list of the methods for the IOPCItemProperties interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 2.05
December 17, 2001

Method	Meaning
QueryAvailableProperties	Returns a list of available properties for an item.
GetItemProperties	Returns the values of the properties of an item transferred in a list of PropertyIDs.
LookupItemIDs	Returns a list of ItemIDs for a list of PropertyIDs.

IconnectionPointContainer interface

The IConnectionPointContainer interface is a standard COM interface for reporting asynchronous events over connection points. For more detailed information on the use of connection points, refer to the literature on COM.

IOPCCCommon interface

The IOPCCCommon interface contains methods to inform the server of the language settings and the name of the client.

Below, you will find a list of the methods for the IOPCCCommon interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard

Version 2.05

December 17, 2001

Method	Meaning
SetLocaleID	Sets the language code of the server. The OPC Server for SIMATIC NET supports German and English.
GetLocaleID	Gets the language code of the server. The OPC Server for SIMATIC NET supports German and English.
QueryAvailableLocaleIDs	Returns all available language codes of the server. The OPC Server for SIMATIC NET supports German and English.
GetErrorString	Returns an error text for an error code.
SetClientName	Transfers a text describing the client to the server.

IPersistFile interface

The optional *IPersistFile* interface is not supported by the OPC Server for SIMATIC NET.

4.2.2.2 OPCGroup object

An object of the OPC Group class manages the individual process variables, the OPC items. Using the OPCGroup objects, a client can form meaningful units of OPC items and execute operations with them.

Interfaces of OPCGroup

The OPCGroup object has the following interfaces:

- IOPCGroupStateMgt
- IOPCGroupStateMgt2 (new as of 3.00)
- IOPCASyncIO2
- IOPCAAsyncIO3 (new as of 3.00)
- IOPCItemMgt
- IOPCItemDeadbandMgt (new as of 3.00)
- IOPCItemSamplingMgt (new 3.00, optional)
- IConnectionPointContainer
- IOPCSyncIO
- IOPCSyncIO2 (new as of 3.00)

The OPC DA specification with which the interface is supported is shown in parentheses.

The graphic below shows the interfaces of OPCGroup.

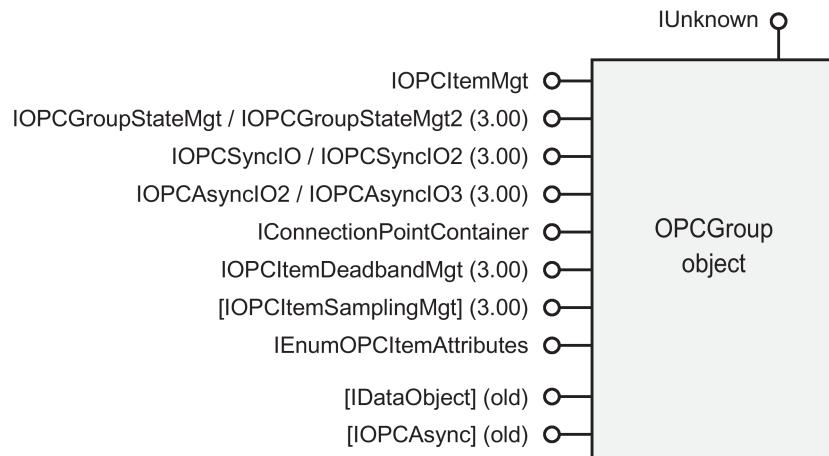


Figure 4-4 The OPCGroup object - [optional interfaces in parentheses]

IOPCGroupStateMgt interface

The IOPCGroupStateMgt interface provides methods for managing groups. You can edit group-specific parameters and copy groups.

Below, you will find a list of the methods for the IOPCGroupStateMgt interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 2.05
December 17, 2001

Method	Meaning
CloneGroup	Creates a copy of a group. The group attributes are copied with the exception of the following: The active status is set to false. A new server handle is assigned <i>szName</i> can be empty. In this case, a unique name is generated.
GetState	Gets the status of the group. The parameters <i>pTimeBias</i> and <i>pLCID</i> have no significance for the OPC Server for SIMATIC NET.
SetName	Changes the name of a group
SetState	Changes the properties of the group. The parameters <i>pTimeBias</i> and <i>pLCID</i> have no significance for the OPC Server for SIMATIC NET. The update rates used by the OPC Server for SIMATIC NET are multiples of the cycle time specified during configuration. The minimum update rate is the same as the cycle time.

IOPCGroupStateMgt2 interface

With this expansion of IOPCGroupStateMgt as of OPC DA 3.00, a monitoring time can be set up for each OPC group. The OPC server then sends a keepalive cyclically. This can be data or an empty callback (keepalive) if the active items have not changed.

Method	Meaning
SetKeepAlive	With this method, you can set the sign-of-life monitoring of the OPC server in the form of a cyclic callback. No GetStatus() monitoring is necessary on the OPC client.
GetKeepAlive	Reads out the current value

You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 3.00
March 4, 2003

IOPCPublicGroupStateMgt interface

The OPC Server for SIMATIC NET does not support public groups. For this reason, the optional IOPCPublicGroupStateMgt interface has no significance.

IOPCAsyncIO2 interface

The IOPCAsyncIO2 interface contains methods for asynchronous reading and writing of items.

Asynchronous means that the client starts a read or write operation and then continues to execute. The interface uses connection points. This simplifies processing of the transferred data.

With each value read, OPC provides a timestamp. Since the SIMATIC systems do not manage a timestamp, the time at which the value was received on the server is used as the timestamp.

Below, you will find a list of the methods for the IOPCAsyncIO2 interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 2.05
December 17, 2001

Method	Meaning
Read	Sends an asynchronous read command. The call is monitored for timeout on the server. If the timeout set in the configuration elapses, there is a notification with the status E_ABORT.
Write	Sends an asynchronous write command. The call is monitored for timeout on the server. If the timeout set in the configuration elapses, there is a notification with the status E_ABORT.
Cancel2	Cancels a pending job
Refresh2	Requests the current value from the cache for each active item

Method	Meaning
SetEnable	Allows deactivation of notification with <i>OnDataChange</i>
GetEnable	Returns the current value for notification with <i>OnDataChange</i>

IOPCAsyncIO3 interface

The IOPCAsyncIO3 interface extends IOPCAsyncIO2 as of OPC DA 3.00 by including the option of reading or writing Quality and Timestamp asynchronously based on MaxAge. The OPC server, however, rejects the writing of Quality and Timestamp with OPC_E_NOTSUPPORTED.

Note

Note that the two interfaces *IOPCAsyncIO2* and *IOPCAsyncIO3* use the same *Connection Point*, the same cookies and callback functions. You should therefore set up the callback only once if you are using both interfaces.

Method	Meaning
ReadMaxAge	Reads one or more values, qualities and timestamps asynchronously based on the MaxAge parameter
WriteVQT	Writes one or more values asynchronously. The writing of Quality and Timestamp is not supported; this is rejected with the message OPC_E_NOTSUPPORTED.
RefreshMaxAge	Forces the callback function IOPCDataCallback::OnDataChange for all active items taking into account MaxAge

You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 3.00
March 4, 2003

IOPCAsyncIO interface

This interface contains methods for asynchronous reading and writing of items. Asynchronous means that the client starts a read or write operation and then continues to execute.

Asynchronous operations return a transaction ID. When the server has completed the read or write operation, the client is notified by a message to its IAdviseSink interface (for more information on the IAdviseSink interface of the client, refer to the description of the *IDataObject* interface).

Note

In version 2, the *IOPCAsyncIO* interface was replaced by *IOPCAsyncIO2*.

IOPCAsyncIO2 and the newer *IOPCAsyncIO3* use *Connection Points* and are easier to handle. In future projects, use *IOPCAsyncIO2* or *IOPCAsyncIO3*.

With each value read, OPC provides a timestamp. Since the SIMATIC systems do not manage a timestamp, the time at which the value was received on the server is used as the timestamp.

Below, you will find a list of the methods for the IOPCAsyncIO interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard

Version 2.05

December 17, 2001

Method	Meaning
Cancel	Cancels a pending job.
Read	Sends an asynchronous read command. The call is monitored for timeout on the server. The corresponding parameter is <i>Read/Write Timeout</i> . If the timeout monitoring aborts, there is a callback with the status <i>E_ABORT</i> .
Refresh	Requests a current value for every active OPC item.
Write	Sends an asynchronous write command. The call is monitored for timeout on the server. The corresponding parameter is <i>Read/Write Timeout</i> . If the timeout monitoring aborts, there is a callback with <i>hrStatus=E_ABORT</i> .

IOPCItemMgt interface

The IOPCItemMgt interface contains methods for managing several items in a group.

Below, you will find a list of the methods for the IOPCItemMgt interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard

Version 2.05

December 17, 2001

Method	Meaning
AddItems	Adds one or more items to a group.
CreateEnumerator	Creates an enumerator. This is the <i>EnumOPCItemAttributes</i> object.
RemoveItems	Deletes one or more items from a group.
SetActiveState	Sets the active status of one or more items of the group.
SetClientHandles	Sets the client handle of one or more items of the group.
SetDataTypes	Sets the required data type of one or more items of the group.
ValidateItems	Checks the validity of an OPC item.

IOPCItemDeadbandMgt interface

With this new interface for OPC DA 3.00, it is possible to change the PercentDeadband parameter for specific items within a group. The PercentDeadband group parameter is used for all items with the set value of the group or the items are now evaluated with the item PercentDeadband values if this is set with this method (default = 0).

Method	Meaning
SetItemDeadband	Overwrites the group value for specific items
GetItemDeadband	Reads the current item-specific value
ClearItemDeadband	Resets the item-specific value to the group value

You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 3.00
March 4, 2003

IOPCItemSamplingMgt interface

This optional interface for OPC DA 3.00 is supported.

With this interface, the OPC client can set sampling rates (RevisedSamplingRate). These can be higher or lower than the group update rate (RevisedUpdateRate). If they are the same, there is no change compared with the previous reaction. This allows the sampling rate of individual items in an OPC group to be adapted more precisely to the actual rate of change of the items.

The group update rate to the client is not changed by samples.

If the sampling rate is lower than the group update rate, the OPC server can buffer several values. At the request of the OPC client, several item changes can be buffered on the server. This mechanism is known as item buffering in OPC Data Access 3.00. The default setting for buffering is 0 = "OFF".

Method	Meaning
SetItemSamplingRate	Overwrites the group update rate for specific items
GetItemSamplingRate	Reads the current item-specific value
ClearItemSamplingRate	Resets the item-specific value to the group value
SetItemBufferEnable	Enables or disables item buffering for items with a faster rate of change (sampling rate) than the group update rate.
GetItemBufferEnable	Reads the status for specific items

You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 3.00
March 4, 2003

IOPCSyncIO interface

The IOPCSyncIO interface contains methods for synchronous reading and writing. Synchronous means that the client waits until a read or write operation is completed before it continues. The client can use the result of the read or write operation for further processing.

Since the OPC Server for SIMATIC NET starts a separate thread for each client, other clients are not blocked while the client waits.

Below, you will find a list of the methods for the IOPCSyncIO interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 2.05
December 17, 2001

Method	Meaning
Read	Reads the values, a status information and the timestamp of one or more items of a group. The call is monitored for timeout on the server. The corresponding timeout is set during configuration for each specific protocol.
Write	Writes the value for one or more items of a group. The call is monitored for timeout on the server. The corresponding timeout is set during configuration for each specific protocol.

IOPCSyncIO2 interface

The IOPCSyncIO2 interface extends IOPCSyncIO as of OPC DA 3.00 by including the option of reading or writing Quality and Timestamp synchronously based on MaxAge. The OPC server, however, rejects the writing of Quality and Timestamp with OPC_E_NOTSUPPORTED.

Method	Meaning
ReadMaxAge	Reads one or more values, qualities and timestamps synchronously based on the MaxAge parameter
WriteVQT	Writes one or more values synchronously. The writing of quality and timestamp is not supported; this is rejected with the message OPC_E_NOTSUPPORTED.

You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 3.00
March 4, 2003

IDataObject interface

The IDataObject interface is a standard COM interface for data transfer. It contains methods for establishing a notification connection between client and a server group.

When the server sends the client a notification, it accesses the client over the IAdviseSink client interface by calling the *OnDataChange* method of IAdviseSink.

The following graphic illustrates the interaction of the interfaces `IAdviseSink` on the client and `IDataObject` on the server.

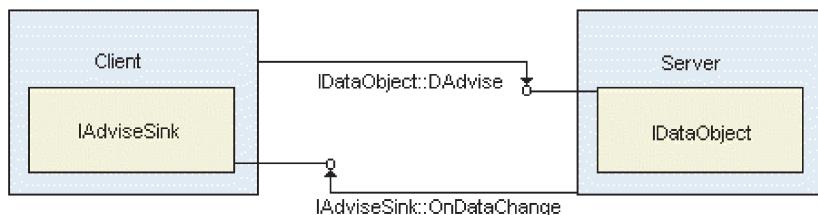


Figure 4-5 Interaction of `IAdviseSink` and `IDataObject`

Note

`IDataObject` was used in version 1 of the Data Access interface for asynchronous communication. As of version 2, *connection points* are used and are both simpler and more flexible.

In future projects, use version 2.

Below, you will find a list of the methods for the `IDataObject` interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 2.05
December 17, 2001

Method	Meaning
DAdvise	Establishes a connection between server and client
DUnadvise	Terminates a connection between server and client

IEnumOPCItemAttributes interface

The `IEnumOPCItemAttributes` interface is based on the standard `IEnum` interface. It returns the items of a group of. The interface is provided by the *CreateEnumerator* method of the `IOPCItemMgt` interface. It is not obtainable with *QueryInterface*.

Below, you will find a list of the methods for the `IEnumOPCItemAttributes` interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Data Access Custom Interface Standard
Version 2.05
December 17, 2001

Method	Meaning
Clone	Creates an identical copy of the <code>IEnumOPCItemAttributes</code> object
Next	Fetches the next OPC item of the group

Method	Meaning
Reset	Resets the list to the first item of the group
Skip	Skips a number of items in the list

4.2.3 Objects of the custom interface for Alarms & Events

The interfaces and their methods are listed below. Only the features specific to SIMATIC NET are described. For a detailed description of the interfaces, refer to the relevant OPC Specifications.

Return values of the methods of the interfaces

All methods return a result of the type *HResult*.

4.2.3.1 OPCEventServer object

An object of the OPC Event Server class is created by the client. The client then uses this object when it requires the services of Alarms & Events.

The following tasks can be performed using the OPCEventServer object:

- Create message object
- Execute queries
- Activate events
- Set the language for display texts
- Register to receive server-specific events

Interfaces of OPCEventServer

The graphic below shows the interfaces of OPCEventServer.

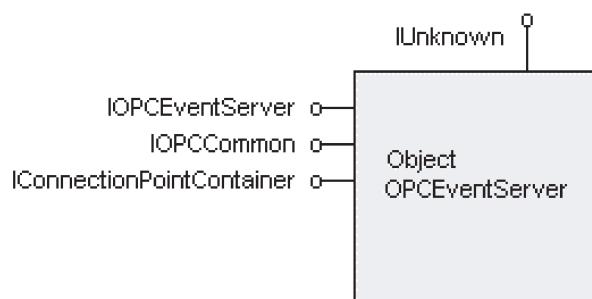


Figure 4-6 OPCEventServer object

IOPCCCommon interface

The IOPCCCommon interface contains methods to inform the server of the language settings and the name of the client.

Below, you will find a list of the methods for the IOPCCCommon interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Alarms & Events Custom Interface Standard
Version 1.10
October 2, 2002

Method	Meaning
SetLocaleID	Sets the language code of the server. The OPC Server for SIMATIC NET supports German and English.
GetLocaleID	Gets the language codes of the server. The OPC Server for SIMATIC NET supports German and English.
QueryAvailableLocaleIDs	Returns all available language codes of the server. The OPC Server for SIMATIC NET supports German and English.
GetErrorString	Gets an error message for an error code
SetClientName	Transfers a text describing the client to the server

IOPCEventServer

The IOPCEventServer interface is the central interface for Alarms & Events. It has the following tasks:

- Creating subscription objects
- Creating area browsers
- Browsing event categories
- Managing conditions

Note

SIMATIC NET does not support the optional areas and therefore no area browser is created.

Below, you will find a list of the methods for the IOPCEventServer interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Alarms & Events Custom Interface Standard
Version 1.10
October 2, 2002

Method	Meaning
GetStatus	Gets current status information about the OPC server.
CreateEventSubscription	Creates a message object to notify a client. The message object is called subscription and is like a ticket to a set of events. The interface required to access the message object is returned.

Method	Meaning
QueryAvailableFilters	Returns information about the filter options supported by the event server. OPC Event Server for SIMATIC NET does not support the following filters: OPC_FILTER_BY_EVENTS: 0x01 OPC_FILTER_BY_CATEGORY: 0x02 OPC_FILTER_BY_SEVERITY: 0x04 OPC_FILTER_BY_SOURCE: 0x16
QueryEventCategories	Returns the event categories provided by the event server. The OPC event server for SIMATIC NET returns event categories only when the parameter <i>dwEventType</i> has the value OPC_SIMPLE_EVENT or OPC_CONDITION_EVENT.
QueryConditionNames	Returns the conditions provided by the event server for a specific event category.
QuerySubConditionNames	Returns the subconditions provided by the event server for a specific event category.
QuerySourceConditions	Returns the conditions provided by the OPC Event Server for a specific source.
QueryEventAttributes	Returns the attributes provided by the event server for a specific event category. The OPC Event Server for SIMATIC NET provides special attributes. The attributes cannot be used as ItemIDs with Data Access.
TranslateToItemIDs	Gets the OPCItems corresponding to an event attribute to use with an associated OPC Data Access Server. The OPC Event Server for SIMATIC NET does not support <i>TranslateToItemIDs</i> .
GetConditionState	Returns information about the state of a condition of a source.
EnableConditionByArea	Activates all conditions for all sources within the specified area.
EnableConditionBySource	Activates all conditions for all specified sources.
DisableConditionByArea	Deactivates all conditions for all sources within the specified area. OPC Event Server for SIMATIC NET does not support conditions. E_NOTIMPL is returned.
DisableConditionBySource	Deactivates all conditions for all specified sources.
AckCondition	Transfers an event acknowledgment to the client. Only conditional events can be acknowledged.
CreateAreaBrowser	Creates an OPCEventAreaBrowser object to browse the process space. OPC Event Server for SIMATIC NET does not support areas. E_NOTIMPL is returned.

IConnectionPointContainer

The IConnectionPointContainer interface is a standard COM interface for reporting asynchronous events over connection points. For more detailed information on the use of connection points, refer to the literature on COM.

4.2.3.2 OPCEventSubscription object

An object of the OPC Event Subscription class sends event messages to the client that uses the IConnectionPointContainer interface of this object.

A client can use several OPCEventSubscription objects. It can define different filters for the various objects.

An OPCEventSubscription corresponds to a ticket for defined events.

Interfaces of OPCEventSubscription

The graphic below shows the interfaces of OPCEventSubscription.

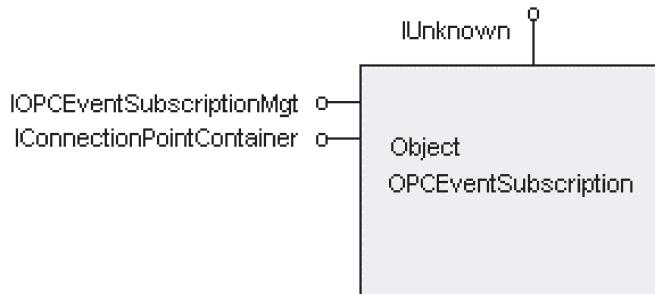


Figure 4-7 OPCEventSubscription object

IOPCEventSubscriptionMgt interface

The IOPCEventSubscriptionMgt interface is the central interface for managing the information on a specific event subscription. Over the interface, for example, the events relevant to the client can be selected.

Below, you will find a list of the methods for the IOPCEventSubscriptionMgt interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Alarms & Events Custom Interface Standard

Version 1.10

October 2, 2002

Method	Meaning
SetFilter	<p>Sets the filter for selecting certain events for this event subscription. The filter parameters have the following significance for the OPC Event Server for SIMATIC NET:</p> <p><i>Event Type</i> The OPC event server supports the event types OPC_SIMPLE_EVENT and OPC_CONDITION_EVENT.</p> <p><i>Event Categories</i> The event categories are described in section "Event server for S7 communication (Page 383)"</p> <p><i>Severity</i> The severity can be configured in STEP 7 or SIMATIC NCM PC/S7. The default value, for example for S7_PROCESS_ALARM is 500.</p> <p><i>Areas</i> The OPC event server for SIMATIC NET does not support areas.</p> <p><i>Source</i> You can enter a connection name.</p>
GetFilter	Returns the currently used filters of the event subscription. See <i>SetFilter</i> .
SelectReturnedAttributes	Specifies the attributes returned with an event message for an event category
GetReturnedAttributes	Returns the list of attributes returned with an event message for an event category.
Refresh	Sends all active and all inactive, unacknowledged condition messages that match the current filter setting to the client.
CancelRefresh	Cancels execution of a current refresh. Since no event message is sent with <i>Refresh</i> , <i>CancelRefresh</i> has no effect.
GetState	Returns the current state of the event subscription.
SetState	Sets various properties of an event subscription.

IConnectionPointContainer interface

The IConnectionPointContainer interface is a standard COM interface for reporting asynchronous events over connection points. For more detailed information on the use of connection points, refer to the literature on COM.

See also

Objects of the automation interface for data access (Page 423)

4.2.3.3 OPCEventAreaBrowser object

The OPC Event Server does not support areas and as a result the OPCEventAreaBrowser object cannot be used.

4.2.4 Interfaces of the client for alarms and events

Informing of events

The client is informed of events over connection points. To allow this, the client must provide a COM object with the IUnknown interface and the callspecific IOPCEventSink interface to receive the calls. When it registers at a connection point, the client passes a pointer to the IUnknown interface to the server.

Interfaces of the client object

The object that the client provides to receive messages must have the structure shown below:

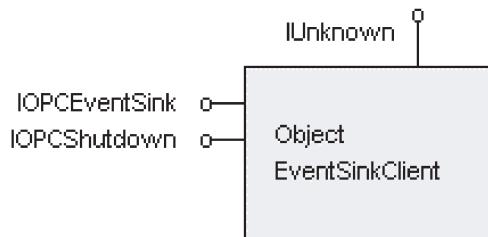


Figure 4-8 Interfaces of the client

4.2.4.1 IOPCEventSink interface

The IOPCEventSink interface is the central interface of the client for receiving messages. It contains one method that is called by the server to transfer events.

Below, you will find a list of the methods for the IOPCEventSink interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Alarms & Events Custom Interface Standard

Version 1.10

October 2, 2002

Method	Meaning
OnEvent	<p>Transfers one or more event messages to the client. The <i>*pEvents</i> structure contains one or more events. The OPC Event Server enters specific values in the following structure elements:</p> <ul style="list-style-type: none"> <i>SzSource</i> connection information. The connection information cannot be converted to an ItemID using TranslateToItemID and then be used by Data Access. <i>ftTime</i> Time at which the event happened on the partner device. The time at which the message is received on the OPC Event Server is entered in the EVENT_ATTR_S7_PCTIME attribute. <i>szMessage</i> <i>ALARM message number</i> <i>dwEventType</i> OPC_SIMPLE_EVENT <i>dwEventCategory</i> S7_PROCESS_ALARM <i>pEventAttributes</i> This structure contains the attributes delivered with the event. The attributes also include the associated values of the message supplied by the partner device. All other structure elements are irrelevant.

4.2.4.2 IOPCShutdown interface

Over this connection point-based call interface, the server can inform the clients before it is shut down or before it shuts itself down. This allows the clients to react and, if required, to shut down.

Below, you will find a list of the methods for the IOPCShutdown interface. Only the features specific to SIMATIC NET are described. You will find a description of the methods in the following OPC Specification:

Alarms & Events Custom Interface Standard

Version 1.10

October 2, 2002

Method	Meaning
ShutdownRequest	<p>The server notifies its clients that it is shutting down.</p> <p>You can set the text for the reason for the shutdown (<i>szReason</i>) in the "Communication Settings" configuration program.</p>

4.2.5 Error messages for OPC DA process variables

Error codes not dependent on the protocol

Error code	Meaning
OPC_E_INVALIDHANDLE (0xC0040001L)	The value of the handle is invalid.
IDS_DUPLICATE (0xC0040002L)	The same value was transferred several times.
IDS_UNKNOWNLCID (0xC0040003L)	The specified <i>locale ID</i> is not supported by server.
OPC_E_BADTYPE (0xC0040004L)	Conversion between the canonicalDatatype and the requestedDatatype is not supported by the server.
OPC_E_PUBLIC (0xC0040005L)	The required operation cannot be executed for a <i>public group</i> .
OPC_E_BADRIGHTS (0xC0040006L)	The required operation (read or write) is prevented by the access rights of the item.
OPC_E_UNKNOWNITEMID (0xC0040007L)	The name (item definition) does not exist in the namespace of the server.
OPC_E_INVALIDITEMID (0xC0040008L)	The name (item definition) has an illegal syntax.
OPC_E_INVALIDFILTER (0xC0040009L)	The character sequence of the filter is not permitted.
OPC_E_UNKNOWNPATH (0xC004000AL)	The connection name specified as the AccessPath is not permitted.
OPC_E_RANGE (0xC004000BL)	The value is outside the permitted range of values.
OPC_S_UNSUPPORTEDRATE (0x0004000DL)	The required update rate is not supported by the server. The nearest permitted value is used.
OPC_S_CLAMP (0x0004000EL)	The value transferred in the write was accepted but it was truncated when output.
OPC_S_INUSE (0x0004000FL)	The operation cannot be executed fully because there are still references to the object.
OPC_E_NOTFOUND (0xC0040011L)	The GetErrorString method returns a string in the form <i>dwError=%x</i> , where <i>%x</i> is a hexadecimal error identifying unknown to the OPC server.
0xC0048003L 0x00048003L	A timeout occurred, for example due to an interrupted connection.
0xC0048004L 0x00048004L	A service used internally has closed.
0xC0048005L 0x00048005L	The required operation (read or write) is prevented by the access rights of the item.
0xC0048006L 0x00048006L	An error has occurred in communication. Check the communication partner and the cable connection.
0xC0048007L 0x00048007L	The value is above the permitted range of values.
0xC0048008L 0x00048008L	The value is below the permitted range of values.

Error code	Meaning
0xC0048009L 0x00048009L	An error occurred during conversion.
0x85270181L 0x05270181L	The callback buffer has overflowed.
CONNECT_E_NOCONNECTION (0x80040200)	No connection. The message relates to an internal COM connection of the OPC server and not to an S7 or transport connection.
CONNECT_E_CANNOTCONNECT (0x80040202)	No connection can be established. Check the number of async connection points (only 1 async connection point is possible).

Error codes for the DP protocol

Error code	Meaning
0x85270101L 0x05270101L	The DP master is not operational.
0x85270102L 0x05270102L	The DP slave is not operational.
0x05270165L	The DP master is in the CLEAR or AUTOCLEAR mode.

Error codes for the S7 protocol

Error code	Meaning
0x85270201L 0x05270201L	Buffer send/receive services: The r_id is invalid.
0x85270202L 0x05270202L	Domain services: The access rights are not valid.
0x85270203L 0x05270203L	Domain services: A block error has occurred.
0x85270204L 0x05270204L	Domain services: A file error has occurred.
0x85270205L 0x05270205L	The consistency check for the SIMOTION project has failed.
0x85270206L 0x05270206L	There is no connection to the S7 communications partner, immediate error return was configured.

Error codes for the PROFINET protocol

Error code	Meaning
0x85270601L 0x05270601L	The requested value is not persistent.
0x85270602L 0x05270602L	The connection is not established.

Error code	Meaning
0x85270603L 0x05270603L	No connection
0x85270605L 0x05270605L	The value is uncertain.
0x85270606L 0x05270606L	Invalid description (too long, illegal characters, no separator, syntax error).
0x8527060aL 0x0527060aL	Invalid enumeration.
0x8527060bL 0x0527060bL	Invalid ID.
0x8527060cL 0x0527060cL	Invalid epsilon type or value.
0x8527060dL 0x0527060dL	The substitute value is invalid.
0x8527060eL 0x0527060eL	Illegal connection to itself.
0x8527060fL 0x0527060fL	Invalid cookie value.
0x85270610L 0x05270610L	Unsupported time.
0x85270611L 0x05270611L	Unsupported QoS type.
0x85270614L 0x05270614L	Destination already connected.
0x85270604L 0x05270604L	The value was only buffered.
0x85270607L 0x05270607L	Unknown object.
0x85270608L 0x05270608L	Unknown property.
0x85270609L 0x05270609L	The returned type does not match the expected type.
0x85270612L 0x05270612L	The QoS value (Quality of Service) is not supported.
0x85270613L 0x05270613L	The system is currently saving, a change to the configuration is not currently possible.
0x85270615L 0x05270615L	The action cannot currently be applied.
0x85270616L 0x05270616L	Access denied.
0x85270617L 0x05270617L	A hardware defect was detected.
0x85270A01L 0x05270A01L	The address information cannot be resolved.
0x85270A02L 0x05270A02L	The data length is invalid.

Error code	Meaning
0x05270A64L	The provider or consumer status was buffered but not yet transferred to the device.
0x05270A65L	The initial value from the controller was buffered but not yet transferred to the device (e.g. due to a problem on the device).

Error codes for the SEND/RECEIVE protocol

Error code	Meaning
0x85270301L	
0x05270301L	There is no connection to the SR communications partner, immediate error return was configured.

Windows error codes

Error code	Meaning
0x80070005	Access denied.
0x80070057	Bad parameter.

Note

Reply messages in which the first byte is zero (0x0....) are conditional success messages. The user can then assume that the variable value, the quality and the time stamp are defined although the quality is "bad" or "uncertain".

4.3 Programming the XML interface

Note

The OPC XML interface is available in the SIMATIC NET Software from Version 6.1 onwards.

Introduction

This section describes the methods of the XML DA interface. It introduces the tag structure of the methods graphically and describes the individual elements and attributes.

First the syntax of the method description is shown. This is followed by descriptions of the basic schemas. These are elements that occur more than once in the schema definition of OPC XMLDA. Following this, we will then look at the individual methods.

XML schema definition of the data types

In keeping with the XMLSchema2001 definition, the schema section of the WSDL of OPC XML-DA describes the complex data types that are used as the parameters of the individual methods. A precise description can be found in the OPC XMLDA Specification.

In the code of the Web service, a C# class is created for every complex data type from the XML schema. The methods of the Web service are described in an interface.

Notes on the use of the XML DA interface

Note

For product-specific error messages, with OPC XML DA, no error text but only the error number is returned. You can read the corresponding error text in the section "Error messages for OPC DA process variables (Page 463)".

Note

To operate OPC XML DA, enable remote OPC communication in "Security" in the "Communication Settings" configuration program.

Note

The OPC XML DA Web service uses other services for the use of symbols. These services require adequate rights for access to the folders in which the STI or ATI symbol files are stored and for the files themselves.

For this reason, please make sure that the following users have the "Full access" right for these files and folders:

Operating system: Windows 7 or Windows 8

User in the German operating system: User "NETZWERKDIENST"

User in the English operating system: User "NETWORK SERVICE"

Note

The OPC XML DA service assigns the following additional services for creating the trace files:

- in Windows 7 and Windows 8 the network services

Please assign read and write rights to these services for the folders in which the trace files are stored.

Note

The Internet Information Server can be configured so that it is automatically reset if certain events occur.

In Windows 7 and Windows 8, the automatic reset is set to 29 hours operating time as default. Please note:

- These settings can result in delayed replies.
 - Following each reset, active subscriptions are lost and must be sent again.
-

4.3.1 Description of the elements

Conventions used in the element descriptions

The descriptions of the OPCXML elements are preceded by a representation of the tag structure.

- **Elements** are indicated by **bold text** in the tag structure.
- *Attributes* are indicated by *italics* in the tag structure.

The nesting of the tags is reflected by the depth to which they are indented. The attributes of a tag are also indented beyond the corresponding tag.

Example

The tag structure might be as follows:

```
<tag1>
    <tag2 att1="yes" att2="no">content of tag2
    </tag2>
    <tag3>content of tag3
    </tag3>
</tag1>
```

In the description of **<tag1>**, you would then find the structure represented as shown below:

```
tag1
  tag2
    att1
    att2
  tag3
```

The elements **tag2** and **tag3** are indented further than the **tag1** element because they are subelements of **tag1**. Immediately following **tag2**, you can also see the attributes **att1** and **att2** of this element.

4.3.2 Basic schemas

Schema Definition for OPC XMLDA

There is a schema definition for all elements of OPCXMLDA. In these descriptions, we often refer to the definition of basic, commonly used elements. The section below describes some of these elements:

ItemProperty

An element used to define properties of an item.

ItemValue

An element with which the value of an item and any relevant additional information (for example, a time stamp) is specified.

OPCError

Contains the error code and description of an OPC error.

ReplyBase

An element with basic information on a server response (for example, time at which the response was sent).

RequestOptions

Contains information specifying a client request in greater detail.

4.3.2.1 ItemProperty

Tag structure

ItemProperty

Name
Description
ItemPath
ItemName
ResultID
Value

Elements and attributes

Name	Description
Name	Contains the name of the property.
Description	The description of the property.
ItemPath	The path information of an element. This attribute is not supported by SIMATIC NET.
ItemName	An element can be identified uniquely in the namespace of the server with the ItemName attribute.
ResultID	If an error or non-critical exception occurs, this attribute contains the name of the OPC error.
Value	The current value of the property.

4.3.2.2 ItemValue

Tag structure

```

ItemValue
  ValueTypeQualifier
  ItemPath
  ItemName
  ClientItemHandle
  Timestamp
  ResultID
  DiagnosticInfo
  Value
  Quality
    QualityField
    LimitField
    VendorField

```

Elements and attributes

Name	Description
<i>ValueTypeQualifier</i>	This attribute is used only with values of the type time, date, and duration to specify the data type. With all other data types, this attribute either does not exist or is ignored if it does exist.
<i>ItemPath</i>	The path information of an element. This attribute is not supported by SIMATIC NET.
<i>ItemName</i>	An element can be identified uniquely in the namespace of the server with the <i>ItemName</i> attribute.
<i>ClientItemHandle</i>	A character string specified by the client in the request that the server returns in its reply. It is used by the client in more complex systems to identify and assign the replies to the correct requests.
<i>Timestamp</i>	The time at which the values were last obtained from the server.
<i>ResultID</i>	If an error or non-critical exception occurs, this attribute contains the name of the OPC error.
DiagnosticInfo	Detailed server-specific diagnostics information.
Value	The value of the item. Since this attribute involves polymorphous information, the additional attribute <i>xsi:type</i> is necessary (for example, <i>: xsi:type=xsd:float</i>).
Quality	Information about the quality of the data. If the server does not return this attribute, the quality is <i>Good</i> . If the quality is <i>Bad</i> or <i>Uncertain</i> , the attribute is returned.
<i>QualityField</i>	If the quality has the value <i>Good</i> , this attribute is not returned. If the quality is <i>Bad</i> , and a previous value for the item is known, this attribute has the value <i>badLastKnownValue</i> .

Name	Description
LimitField	Contains a name with which the OPC limit bit field can be addressed. This attribute is always transferred except when the limit status is <i>none</i> .
VendorField	A numeric value that corresponds to the OPC vendor bit field. Whether or not this attribute is transferred depends on the vendor.

4.3.2.3 OPCError

Tag structure

```
OPCError
  ID
  Text
```

Elements and attributes

Name	Description
ID	Contains the name of the OPC error.
text	A description of the error in text form. The content of the string depends of the <i>LocaleID</i> attribute.

4.3.2.4 ReplyBase

Tag structure

```
ReplyBase
  RcvTime
  ReplyTime
  ClientRequestHandle
  RevisedLocaleID
  ServerState
```

Elements and attributes

Name	Description
RcvTime	Indicates the time at which the server received the request. RcvTime is a mandatory attribute.
ReplyTime	Indicates the time at which the server sent the reply; a mandatory attribute.
ClientRequestHandle	If this attribute was included in the client's request, it is returned by the server with the response.

Name	Description
RevisedLocaleID	If the client specifies a value for the <i>LocaleID</i> attribute that is supported by the server, the server returns the default value for the <i>LocaleID</i> attribute with the <i>RevisedLocaleID</i> attribute.
ServerState	Specifies the status of the server and is always returned; a mandatory attribute.

4.3.2.5 RequestOptions

Tag structure

RequestOptions

ReturnErrorText
ReturnDiagnosticInfo
ReturnItemTime
ReturnItemName
ReturnItemPath
RequestDeadline
ClientRequestHandle
LocaleID

Elements and attributes

Name	Description
ReturnErrorText	The default value for this attribute is TRUE. In this case, the server returns detailed error descriptions.
ReturnDiagnosticInfo	The server returns detailed diagnostics information if the value of this attribute is TRUE.
ReturnItemTime	Specifies whether or not a time stamp is returned for each item. The default is FALSE, no value is returned.
ReturnItemName	Specifies whether or not the <i>ItemName</i> attribute is returned for each item. The default is FALSE, no value is returned.
ReturnItemPath	Specifies whether or not the <i>ItemPath</i> attribute is returned for each item. If <i>ReturnItemPath</i> =true or false, the path is always returned (not zero). The default value is FALSE.
RequestDeadline	<p>Specifies the latest time that the client will wait for a response from the server. Data for items that are not available at this time is returned as an error. If the specified time is earlier than the current time on the server, the entire request fails. The client must supply the time information as UTC time.</p> <p>To use the value, <i>RequestDeadlineSpecified</i>=true must be set when the parameter is transferred.</p> <p>Exception OPC XML DA clients developed with the Microsoft Development Environment 2003 must transfer the local time and not the UTC time as <i>RequestOptions.RequestDeadline</i>. The Standard SOAP Writer of the Visual Studio 2003 always converts the transferred times into a UTC time in the format local time + time offset.</p>

Name	Description
ClientRequestHandle	If this attribute was included in the client's request, it is returned by the server with the response. In large and complex systems, this information helps the client to assign the responses to the requests. Specifying this attribute is optional.
LocaleID	An optional attribute with which the client specifies the language for certain returned data.

4.3.3 Read and write jobs

4.3.3.1 Read

Tag structure

```

Read
  Options
    ReturnErrorText
    ReturnDiagnosticInfo
    ReturnItemTime
    ReturnItemName
    ReturnItemPath
    RequestDeadline
    ClientRequestHandle
    LocaleID
  ItemList
    ItemPath
    ReqType
    MaxAge
    Items
      ItemPath
      ReqType
      ItemName
      ClientItemHandle
      MaxAge

```

Elements and attributes

Read

The <Read> element contains all the information for a read job.

Options

The <Options> element contains options that are available for the XMLDA requests. This element is of the type RequestOptions. You will find information on the attributes of this element in the relevant section of this documentation.

ItemList

The container element for the individual items. A request can contain several items.

The *ItemPath*, *ReqType* and *MaxAge* attributes can be used for the *ItemList* and *Items* elements. An attribute for an *item* overwrites the attribute of the same name for *ItemList*.

ItemPath

The path of an element. This attribute is not supported by SIMATIC NET.

MaxAge

A period of time of the type xsd:int that specifies the maximum age of the data. The data should not be older than the duration specified here.

ReqType

Specifies the data type for the value of the item requested by the client. All data types listed in the specification are supported.

Items

Element with which a single item is specified.

ItemName

The name of the item.

ClientItemHandle

A string assigned by the client. If this attribute is specified by the client, the server must return it.

Example

```
<soap:Body>
    <Read
        xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
        <Options
            ReturnErrorText="false"
            ReturnItemTime="true"
            ReturnItemName="true"
            LocaleID="en" />
        <ItemList>
            <Items ItemName="Simple Types/UInt" />
            <Items ItemName="Simple Types/Int" />
            <Items ItemName="Simple Types/Float" />
        </ItemList>
    </Read>
</soap:Body>
```

C# interface for the synchronous read web method

When using C# user programs, the following method is available for reading values. Only some of the parameters listed above need to be specified:

```
ReplyBase Read(RequestOptions Options,
                ReadRequestItemList ItemList,
                out ReplyItemList ItemValues,
                out OPCError[] Error);
```

4.3.3.2 ReadResponse

Tag structure

```
ReadResponse
  ReadResult
    RcvTime
    ReplyTime
    ClientRequestHandle
    RevisedLocaleID
    ServerState
  RItemList
    Reserved
    Items
      ValueTypeQualifier
      ItemPath
      ItemName
      ClientItemHandle
      Timestamp
      ResultID
      DiagnosticInfo
      Value
      Quality
        QualityField
        LimitField
        VendorField
    Errors
    ID
    Text
```

Elements and attributes

ReadResponse

Using the <ReadResponse> element, the server returns the results of a read job.

ReadResults

The <ReadResults> element contains basic information on the response of the server. This element is of the type ReplyBase. You will find information on the attributes of this element in the relevant section of this documentation.

RItemList

RItemList is the container element for the individual items of a response. A response can contain several items.

Reserved

This attribute prevents WSDL based programs from representing the return list as an array of items for code generation.

Items

Element with which a single item is specified.

This element is of the type ItemValue. You will find information on the attributes of this element in the relevant section of this documentation.

Errors

An array of errors that occurred with this response.

This element is of the type OPCError. You will find information on the attributes of this element in the relevant section of this documentation.

Example

```
<soap:Body>
    <ReadResponse
        xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
        <ReadResult
            RcvTime="2003-05-27T00:15:36.6400000-07:00"
            ReplyTime="2003-05-27T00:15:36.7500000-07:00"
            ServerState="running"
        />
        <RItemList>
            <Items
                <Item
                    ItemName="Simple Types/UInt"
                    Timestamp="2003-05-27T00:15:36.7343750-07:00">
                    <Value xsi:type="xsd:unsignedInt">4294967295</Value>
                </Item>
            <Items
                <Item
                    ItemName="Simple Types/Int"
                    Timestamp="2003-05-27T00:15:36.7343750-07:00">
                    <Value xsi:type="xsd:int">2147483647</Value>
                </Item>
            <Items
                <Item
                    ItemName="Simple Types/Float"
                    Timestamp="2003-05-27T00:15:36.7343750-07:00">
                    <Value xsi:type="xsd:float">3.402823E+38</Value>
                </Item>
            </Items>
        </RItemList>
    </ReadResponse>
</soap:Body>
```

4.3.3.3 Write

Tag structure

```
Write
  ReturnValuesOnReply
  Options
    ReturnErrorText
    ReturnDiagnosticInfo
    ReturnItemTime
    ReturnItemName
    ReturnItemPath
    RequestDeadline
    ClientRequestHandle
    LocaleID
  ItemList
    ItemPath
    Items
      ValueTypeQualifier
      ItemPath
      ItemName
      ClientItemHandle
      Timestamp
      ResultID
      DiagnosticInfo
      Value
      Quality
        QualityField
        LimitField
        VendorField
```

Elements and attributes

Write

The <Write> element contains all the information for a write job. The write job can be performed for one or more items.

ReturnValuesOnReply

With this attribute, the client can specify whether a value should be returned for each item. The value processed by the server for this write job is returned. This is the same value that the server would return if there was a read immediately following the write. If the write job fails, no values are returned. With write-only values, the server returns E_WRITEONLY.

Options

The <Options> element contains options that are available for the XML DA requests. This element is of the type RequestOptions. You will find information on the attributes of this element in the relevant section of this documentation.

ItemList

The container element for the individual items. A request can contain several items.

ItemName

An element can be identified uniquely in the namespace of the server with the ItemName attribute.

Items

Element with which a single item is specified. This element is of the type ItemValue. You will find information on the attributes of this element in the relevant section of this documentation.

Note

When writing and reading back *DateTime* items, errors up to 2 ms can occur caused by the .NET interop layer.

Example

```
<soap:Body>
    <Write
        xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
        <Options
            ReturnErrorText="false"
            ReturnItemName="true"
            LocaleID="en"
        />
        <ItemList>
            <Items ItemName="Simple Types/UInt">
                <Value xsi:type="xsd:unsignedInt">
                    >4294967295</Value>
            </Items>
            <Items ItemName="Simple Types/Int">
                <Value xsi:type="xsd:int">2147483647</Value>
            </Items>
            <Items ItemName="Simple Types/Float">
                <Value xsi:type="xsd:float">3.402823E+38</Value>
            </Items>
        </ItemList>
    </Write>
</soap:Body>
```

C# interface for the synchronous write web method

When using C# user programs, the following method is available for writing values. Only some of the parameters listed above need to be specified:

```
ReplyBase Write(RequestOptions Options,
                 WriteRequestItemList ItemList,
                 bool ReturnItemVal,
                 out ReplyItemList ItemValues,
                 out OPCError[] Error);
```

4.3.3.4 WriteResponse

Tag structure

```
WriteResponse
  WriteResult
    RcvTime
    ReplyTime
    ClientRequestHandle
    RevisedLocaleID
    ServerState
  RItemList
    Reserved
    Items
      ValueTypeQualifier
      ItemPath
      ItemName
      ClientItemHandle
      Timestamp
      ResultID
      DiagnosticInfo
      Value
      Quality
        QualityField
        LimitField
        VendorField
    Errors
    ID
    Text
```

Elements and attributes

WriteResponse

Using the <WriteResponse> element, the server returns the results of a write job.

WriteResult

The <WriteResult> element contains basic information on the response of the server. This element is of the type ReplyBase. You will find information on the attributes of this element in the relevant section of this documentation.

RItemList

The container element for the individual items. A response can contain several items.

Reserved

This attribute prevents WSDL based programs from representing the return list as an array of items for code generation.

Items

Element with which a single item is specified.

This element is of the type ItemValue. You will find information on the attributes of this element in the relevant section of this documentation.

Errors

An array of errors that occurred with this response.

This element is of the type OPCError. You will find information on the attributes of this element in the relevant section of this documentation.

Example

```
<soap:Body>
    <WriteResponse
        xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/">
        <WriteResult
            RcvTime="2003-05-27T05:19:26.3687500-07:00"
            ReplyTime="2003-05-27T05:19:26.4687500-07:00"
            ServerState="running"
        />
        <RItemList>
            <Items ItemName="Simple Types/UInt" />
            <Items ItemName="Simple Types/Int" />
            <Items ItemName="Simple Types/Float" />
        </RItemList>
    </WriteResponse>
</soap:Body>
```

4.3.4 Using subscriptions

Contact supported over several method calls

In contrast to most other Web services, the OPC XMLDA Web services support loose contact with a subscribed Web client over several method calls. This loose contact is known as a polled subscription below. The values of items can be read cyclically with a polled subscription.

Client logon and logoff

The client logs on the subscription with the OPC XML-DA Web service. This then notes which items are of interest to the client and queries their values. When a polled request arrives from the client, the current values are returned. The subscription is terminated by the client.

Identification using handles

To identify the subscription, a handle assigned by the server is transferred as a parameter with each future call. Is also possible to specify a list of handles so that several subscriptions are relevant per SubscriptionPolledRefresh. This increases the efficiency of polling.

The address of the subscription object is returned to the client as the subscription handle so that the subscription can be identified uniquely at the next SubscriptionPolledRefresh and SubscriptionCancel.

The following graphic illustrates the use of a subscription:

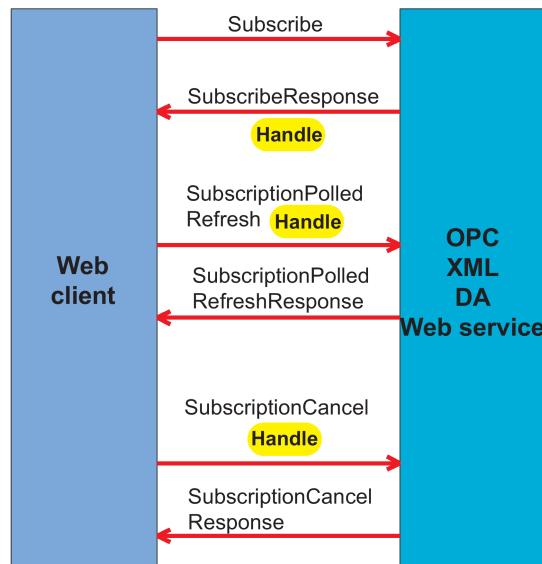


Figure 4-9 Use of a subscription

Period of validity of the monitored items

The **SubscriptionPolledResponse** method returns the values that were requested with the **SubscriptionPolledRefresh** method back to the client. **SubscriptionCancel** discards the subscriptions and their items. If no further **SubscriptionPolledRefresh** requests arrive from the client within a specified period (**SubscriptionPingRate**), the OPC DA Web service terminates the subscription automatically in the same way as with the **SubscriptionCancel** method.

4.3.4.1 Subscribe

Tag structure

```
Subscribe
  ReturnValuesOnReply
  SubscriptionPingRate
Options
  ReturnErrorText
  ReturnDiagnosticInfo
  ReturnItemTime
  ReturnItemName
  ReturnItemPath
  RequestDeadline
  ClientRequestHandle
  LocaleID
ItemList
  ItemPath
  ReqType
  Deadband
  RequestedSamplingRate
  EnableBuffering
Items
  ItemPath
  ReqType
  ItemName
  ClientItemHandle
  Deadband
  RequestedSamplingRate
  EnableBuffering
```

Elements and attributes

Subscribe

The <Subscribe> element contains all the information on a subscribe job.

ReturnValuesOnReply

If the value of this attribute is set to TRUE, the server returns values available for a `SubscribeResponse`. If the value is FALSE, the server returns no values with a `SubscribeResponse`.

SubscriptionPingRate

The value of this attribute specifies the interval at which the server checks the existence of the client. If the client has not communicated with the server during the specified period, the server can release all resources that were required for the client's subscription.

Options

The <Options> element contains options that are available for the XML DA requests. This element is of the type `RequestOptions`. You will find information on the attributes of this element in the relevant section of this documentation.

Item List

The container element for the individual items. A request can contain several items.

The *ItemPath*, *ReqTyp*, *Deadband*, *RequestedSamplingRate*, and *EnableBuffering* attributes can be used for the *ItemList* and *Items* elements. An attribute for an *item* overwrites the attribute of the same name for *ItemList*.

ItemPath

The path of an element. This attribute is not supported by SIMATIC NET.

ReqType

Specifies the data type for the value of the item requested by the client.

Deadband

This specifies a threshold value up to which value changes of an item do not trigger a SubscriptionPolledRefresh. This is specified as a percentage of the maximum value range of the item. The value for this parameter should therefore be between 0 and 100 percent and can only be used with items of the type integer and float.

RequestedSamplingRate

Time specified by the client in milliseconds after which the server checks the values for changes.

EnableBuffering

If the value of this attribute is set to TRUE, the server saves all value changes in a buffer according to the *RequestedSamplingRate* parameter. The server returns the saved data to the client at the next *PolledRequest*.

Item Name

The name of the item.

ClientItemHandle

A string assigned by the client. If this attribute is specified by the client, the server must return it.

4.3.4.2 SubscribeResponse

Tag structure

```

SubscribeResponse
  ServerSubHandle
  SubscribeResult
    RcvTime
    ReplyTime
    ClientRequestHandle
    RevisedLocaleID
    ServerState
  RItemList
    RevisedSamplingRate
  Items
    RevisedSamplingRate
  ItemValue
    ValueTypeQualifier
    ItemPath
    ItemName
    ClientItemHandle
    Timestamp
    ResultID
  DiagnosticInfo
  Value
  Quality
    QualityField
    LimitField
    VendorField
  Errors
    ID
    Text

```

Elements and attributes

SubscribeResponse

The <SubscribeResponse> element contains all the information on a SubscribeResponse.

ServerSubHandle

The value specified by the server for this value must be used with SubscriptionPolledRefresh requests and SubscriptionCancel requests. This attribute identifies the client that sends the request.

SubscribeResult

The <SubscribeResult> element contains basic information on the response of the server.

This element is of the type *ReplyBase*. You will find information on the attributes of this element in the relevant section of this documentation.

RItemList

Contains item elements. The available values for the items are only sent to the client when

the client has requested them with a corresponding value for the ReturnValuesOnReply attribute (see [Subscribe](#)).

RevisedSamplingRate

The fastest update cycle supported by the server. This value is returned by the server to the client.

Items

Element with which a single item is specified.

This element is of the type [ItemValue](#). You will find information on the attributes of this element in the relevant section of this documentation.

Errors

An array of errors that occurred with this response.

This element is of the type [OPCError](#). You will find information on the attributes of this element in the relevant section of this documentation.

4.3.4.3 SubscriptionPolledRefresh

Tag structure

SubscriptionPolledRefresh

HoldTime

WaitTime

ReturnAllItems

Options

ReturnErrorText

ReturnDiagnosticInfo

ReturnItemTime

ReturnItemName

ReturnItemPath

RequestDeadline

ClientItemHandle

LocaleID

ServerSubHandles

Elements and attributes

SubscriptionPolledRefresh

Using this element, the client requests the server to refresh the items that were defined in a prior [Subscription](#) request.

HoldTime

This attribute instructs the server not to send any refreshed values to the client until the internal time of the server is equal to or greater than the value of this attribute.

WaitTime

After reaching the time specified in the HoldTime attribute, the server waits for the time specified in WaitTime before returning the response. If the value of an item changes during the WaitTime, a response is sent immediately.

ReturnAllItems

If the value of this attribute is set to FALSE, the server returns only items that have changes since the previous SubscriptionPolledRefresh request and the current SubscriptionPolledRefresh request.

If this attribute is set to TRUE, the server returns all items that were defined in the corresponding Subscription request.

Options

The <Options> element contains options that are available for the XML DA requests. This element is of the type RequestOptions. You will find information on the attributes of this element in the relevant section of this documentation.

ServerSubHandles

This attribute is used by the server to identify the subscription to be polled.

4.3.4.4 SubscriptionPolledRefreshResponse

Tag structure**SubscriptionPolledRefreshResponse**

DataBufferOverflow

SubscriptionPolledRefreshResult

RcvTime

ReplyTime

ClientRequestHandle

RevisedLocaleID

ServerState

InvalidServerSubHandles**RItemList**

SubscriptionHandle

Items

ValueTypeQualifier

ItemPath

ItemName

ClientItemHandle

Timestamp

ResultID

DiagnosticInfo**Value****Quality**

QualityField

LimitField

VendorField

Errors

ID

Text

Elements and attributes

SubscriptionPolledRefreshResponse

This element contains all the information that the server sends to the client as the response to a SubscriptionPolledRefresh request.

DataBufferOverflow

If the value of this attribute is TRUE, there have been changes in the items that could not be saved due to lack of resources. The individual items indicate whether or not they are affected by the lack of resources.

SubscriptionPolledRefreshResult

This element contains basic information on the response of the server. It is of the ReplyBase type. You will find information on the attributes of this element in the relevant section of this documentation.

InvalidServerSubHandles

ServerSubHandles recognized as being invalid by the server.

RItemList

The <RItemList> element contains values for all items if the client transferred the ReturnAllItems attribute with the value TRUE with its SubscriptionPolledRefresh request. Otherwise the server returns only values that have changed.

SubscriptionHandle

Handle of the RItemList specified by the client in the subscription call.

Items

Element with which a single item is specified.

This element is of the type ItemValue. You will find information on the attributes of this element in the relevant section of this documentation.

Errors

An array of errors that occurred with this response.

This element is of the type OPCError. You will find information on the attributes of this element in the relevant section of this documentation.

4.3.4.5 **SubscriptionCancel**

Tag structure

SubscriptionCancel

ServerSubHandle

ClientRequestHandle

Elements and attributes

SubscriptionCancel

Using the <SubscriptionCancel> element, the client ends the subscription and the server can release the corresponding resources.

ServerSubHandle

The value assigned to this element by the server identifies the subscription that sends the request.

ClientRequestHandle

If this attribute was included in the client's request, it is returned by the server with the SubscriptionCancelResponse. In large and complex systems, this information helps the client to assign the responses to the requests. Specifying this attribute is optional.

4.3.4.6 SubscriptionCancelResponse

Tag structure

SubscriptionCancelResponse

ClientRequestHandle

Elements and attributes

SubscriptionCancelResponse

Using this element, the server confirms a SubscriptionCancel request of the client.

ClientRequestHandle

If the client used this attribute in its request, the server returns it in its response.

4.3.5 Further queries

4.3.5.1 Browse

Tag structure

Browse

LocaleID

ClientRequestHandle

ItemPath

ItemName

ContinuationPoint

MaxElementsReturned

BrowseFilter

ElementNameFilter

VendorFilter

ReturnAllProperties

ReturnPropertyValues

ReturnErrorText

PropertyNames

Elements and attributes

Browse

The <Browse> element contains all the information necessary for navigating through a hierarchical address space.

LocaleID

An optional attribute with which the client specifies the language for certain returned data.

ClientRequestHandle

If this attribute was included in the client's request, it is returned by the server with the response. In large and complex systems, this information helps the client to assign the responses to the requests. Specifying this attribute is optional.

ItemPath

The path to the item with which navigation through the address space starts. If there is a second browse request, this attribute must have the same value as in the previous request.

ItemName

The name of the item with which navigation through the address space starts. If there is a second browse request, this attribute must have the same value as in the previous request.

ContinuationPoint

If this is not an initial browse request, a point can be specified here from which the browse request is restarted.

MaxElementsReturned

Maximum number of return values.

If a negative value is transferred here, an exception occurred.

BrowseFilter

This attribute with the enumeration data type browseFilter (all, branch, item) specifies which subset of elements will be returned.

ElementNameFilter

A regular expression used to select elements.

VendorFilter

A vendor-specific expression used to select vendor-specific information. The effects on the ElementNameFilter element are not defined.

ReturnAllProperties

If the value TRUE is set for this attribute, the server returns all available properties for each returned element. The value of the PropertyNames attribute is then ignored in this case.

ReturnPropertyValues

If the value TRUE is set for this attribute, the server returns not only the names of the properties but also the values of the properties.

ReturnErrorText

If this attribute has the value TRUE, the server returns a detailed error description.

PropertyName

Names of properties returned with each element. If the value of the ReturnAllProperties attribute is set to TRUE, the server returns all properties regardless of PropertyNames.

4.3.5.2 BrowseResponse

Tag structure

```
BrowseResponse
  ContinuationPoint
  MoreElements
  BrowseResult
    RcvTime
    ReplyTime
    ClientRequestHandle
    RevisedLocaleID
    ServerState
  Elements
    Name
    Item Path
    ItemName
    IsItem
    HasChildren
  Properties
    Name
    Description
    ItemPath
    ItemName
    ResultID
    Value
  Errors
    ID
    Text
```

Elements and attributes

BrowseResponse

Contains all the information of a response to a Browse request.

ContinuationPoint

Here, the server can specify a point from which a following Browse request can be started.
This attribute is not supported by SIMATIC NET.

MoreElements

If the server does not support the ContinuationPoint attribute and the number of returned values exceeds the value of MaxElementsReturned, the server sets this attribute to TRUE.

BrowseResult

The <BrowseResponse> element contains basic information on the response of the server. This element is of the type ReplyBase. You will find information on the attributes of this element in the relevant section of this documentation.

Elements

Contains information on the parsed elements of the structure tree.

Name

Name of an element in the namespace displayed to the user (for example, in a structure view).

ItemPath

The path of an element. This attribute is not supported by SIMATIC NET.

ItemName

An element can be identified uniquely in the namespace of the server with the ItemName attribute.

IsItem

If the value TRUE is set for this attribute, this element is an item that can be used in read, write, and subscribe requests.

HasChildren

If the value TRUE is set for this attribute, the corresponding element has children.

Properties

The <Properties> element contains information on a property that can be made accessible using a Browse or GetProperties call.

This element is of the type ItemProperty. You will find information on the attributes of this element in the relevant section of this documentation.

Errors

An array of errors that occurred with this request.

This element is of the type OPCError. You will find information on the attributes of this element in the relevant section of this documentation.

4.3.5.3 **GetProperties**

Tag structure

GetProperties

LocaleID
ClientRequestHandle
ItemPath
ReturnAllProperties
ReturnPropertyValue
ReturnErrorText
ItemIDs
 ItemPath
 ItemName
PropertyName

Elements and attributes

GetProperties

Using the <GetProperties> element, you can query properties.

LocaleID

An optional attribute with which the client specifies the language for certain returned data.

ClientRequestHandle

If this attribute was included in the client's request, it is returned by the server with the response. In large and complex systems, this information helps the client to assign the responses to the requests. Specifying this attribute is optional.

ItemPath

The path of an element. This attribute is not supported by SIMATIC NET.

ReturnAllProperties

If the value TRUE is set for this attribute, the server returns all properties for each returned element. The value of the PropertyNames element is then ignored in this case.

ReturnPropertyValues

If the value TRUE is set for this attribute, the server returns not only the names of the properties but also the values of the properties.

ReturnErrorText

If this attribute has the value TRUE, the server returns a detailed error description.

ItemIDs

Contains a list of items whose properties will be queried.

ItemPath

The path of an element. This attribute is not supported by SIMATIC NET.

ItemName

An element can be identified uniquely in the namespace of the server with the ItemName attribute.

PropertyNames

Names of properties returned with each element. If the value of the ReturnAllProperties attribute is set to TRUE, the server returns all properties regardless of PropertyNames.

4.3.5.4 GetPropertiesResponse

Tag structure

```
GetPropertiesResponse
  GetPropertiesResult
    RcvTime
    ReplyTime
    ClientRequestHandle
    RevisedLocaleID
    ServerState
    PropertyLists
      ItemPath
      ItemName
      ResultID
    Properties
      Name
      Description
      ItemPath
      ItemName
      ResultID
      Value
    Errors
      ID
      Text
```

Elements and attributes

GetPropertiesResponse

The server responds to GetProperties requests with this element.

GetPropertiesResult

The <GetPropertiesResult> element contains basic information on the response of the server.

This element is of the type ReplyBase. You will find information on the attributes of this element in the relevant section of this documentation.

PropertyLists

An element of this type containing the requested properties of the item is returned for each item.

The *Item Path*, *ItemName*, and *ResultID* attributes can be used for the PropertyLists and Properties elements. If these attributes are specified in a Properties element, the values of the attributes with the same name in PropertyLists are overwritten for this element.

ItemPath

The path of an element. This attribute is not supported by SIMATIC NET.

ItemName

An element can be identified uniquely in the namespace of the server with the ItemName attribute.

ResultID

If an error or non-critical exception occurs, this attribute contains the name of the OPC error.

Properties

This element is of the type ItemProperty. You will find information on the attributes of this element in the relevant section of this documentation.

Errors

An array of errors that occurred with this response.

This element is of the type OPCError. You will find information on the attributes of this element in the relevant section of this documentation.

4.3.5.5 GetStatus

Tag structure

GetStatus

LocaleID

ClientRequestHandle

Elements and attributes

GetStatus

A GetStatus request checks the Web service and obtains vendor-specific information that is not accessible to other OPC methods.

LocaleID

An optional attribute with which the client specifies the language for certain returned data.

ClientRequestHandle

If this attribute was included in the client's request, it is returned by the server with the response. In large and complex systems, this information helps the client to assign the responses to the requests. Specifying this attribute is optional.

Example

```
<soap:Body>
    <GetStatus
        LocaleID="de-AT"
        xmlns="http://opcfoundation.org/webservices/XMLDA/1.0/"
    />
</soap:Body>
```

C# interface for the GetStatus web method

When using C# user programs, the following method is available for checking the Web service:

```
ServerStatus GetStatus(string LocaleID,
                        string ClientRequestHandle,
                        out ServerStatus);
```

4.3.5.6 **GetStatusResponse**

Tag structure

```
GetStatusResponse
  GetStatusResult
    RcvTime
    ReplyTime
    ClientRequestHandle
    RevisedLocaleID
    ServerState
  Status
    StartTime
    ProductVersion
    StatusInfo
    VendorInfo
    SupportedLocaleIDs
    SupportedInterfaceVersions
```

Elements and attributes

GetStatusResponse

Contains all the information of a response to a GetStatus request

GetStatusResult

The <GetStatusResult> element contains basic information on the response of the server. This element is of the type ReplyBase. You will find information on the attributes of this element in the relevant section of this documentation.

Status

The <Status> element contains information on the server status.

StartTime

Time at which the server was started in UTC format.

ProductVersion

Version number of the server made up of the major, minor, and build number.

StatusInfo

Contains additional information on the server status and may be language-dependent.

VendorInfo

Vendorspecific string with additional information on the server. The value of this attribute should include the company name and information on the supported devices.

SupportedLocaleIDs

The locales supported by the server. A server does not need to support all locales for all items.

SupportedInterfaceVersions

An array of strings containing the versions of the XMLDA specification supported by the server.

4.4 Programming the OPC UA interface

What will you find in this section?

This section describes the programming interface of the OPC UA server and the expanded configuration and certificate management.

Interfaces

There is both a C interface for simple, high-speed access from C programs as well as a .NET interface for convenient use in a .NET application. The corresponding libraries and assemblies are available after installing the SIMATIC NET PC software.

Availability of the OPC UA interface under SIMATIC NET

The OPC UA interface is available as of the SIMATIC NET CD with Edition 2008 (V7.1).

Locating an OPC UA server

An OPC UA server is located using the OPC UA Discovery services. Only the computer name or the IP address and the discovery port 4840 needs to be entered. The existing registered OPC UA servers then return their configured OPC UA endpoints.

Security and authentication

For secure communication (OPC UA CreateSecureChannel) between OPC UA client and server, X509 certificates must be available at both ends. These are exchanged when communication begins. The OPC UA server and the client decide at this point whether or not the supplied certificate will be trusted.

Communication

After accepting the certificates, an OPC UA session can be established and activated.

OPC UA nodes can be monitored (OPC UA MonitoredItems and Subscriptions) or simply read or written once. If nodes need to be read or written several times, to improve speed, it is strongly advised to register the nodes first (OPC UA RegisterNodes). When writing or reading, this is referenced.

Transparent redundancy

Higher availability of automation data without additional software for UA clients is achieved by the transparent redundant S7 UA server. For redundancy and load balancing, only the "Industrial Ethernet" standard is required.

Access to array elements in OPC UA using IndexRange

When accessing with IndexRange, there are restrictions depending on the target variables. Writing parts of bit arrays using IndexRange to S7 blocks is rejected for reasons of consistency. Only one-dimensional IndexRanges are supported. When reading with IndexRange, the entire variable is always read even if more is requested by the communications partner.

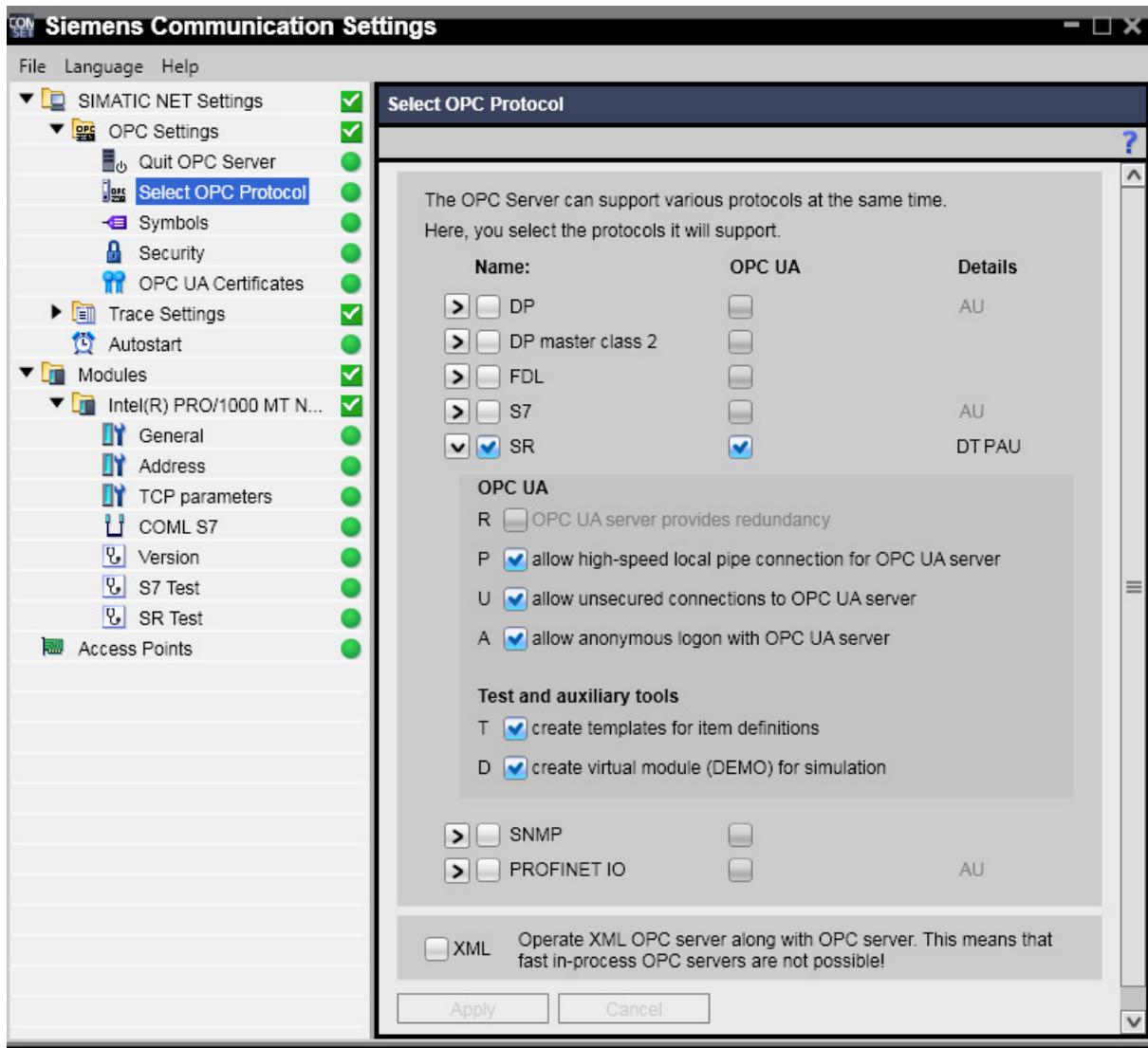
4.4.1 Configuring the OPC UA server

4.4.1.1 Authentication

User authentication

Following installation, authentication is deactivated for test purposes and simple commissioning. It is nevertheless advisable to activate user authentication.

The anonymous logon to the OPC UA server should be deactivated in the "Communication Settings" configuration program with "OPC settings" > "OPC protocol selection" > click the arrow symbol of the relevant protocol > "allow anonymous logon with OPC UA server". It can also be activated again here.



With this setting, existing Windows users and their passwords are queried during secure connection establishment. The user name and password are specified according to a client-specific procedure.

4.4.1.2 Endpoint security

Endpoint security of the OPC UA server

The endpoint security "None" (no security checks or encryption) "allow non-secure connections to OPC UA server" should be deactivated in the "Communication Settings" configuration program in "OPC protocol selection". Only secure endpoints are then available. It can also be activated again here.

4.4.2 Certificate management for the OPC UA server

Folders for certificate management

The certificate management for OPC UA server is based on OpenSSL (<http://www.openssl.org/>) and is located in the following folder:

- Windows 7/Windows Server 2008 R2 and Windows 8/Windows Server 2012
 - Hidden path in file system :
"<systemdrive>:\ProgramData\Siemens\OPC\PKI\CA"

OPC UA server certificates

The subfolder "\certs" in the folder named above is the location for the computer-specific OPC UA server certificates in X509 format with the extension "der"

- for S7 UA: OPC.SimaticNET.S7.der OPC.SimaticNET.S7.der including the public key,
- for S7OPT UA: OPC.SimaticNET.S7OPT.der including the public key,
- for PROFINET IO UA: OPC.SimaticNET.PNIO.der including the public key,
- for DP UA: OPC.SimaticNET.DP.der including the public key,
- for SEND/RECEIVE UA: OPC.SimaticNET.SR.der including the public key.

These certificates are transferred to the client at the start of communication. Depending on the OPC UA client, they can or must be imported into the certificate management of the OPC UA client.

Note

You should also note that only standard characters can be used in the computer name and therefore also in the certificate.

Standard names consist of the letters (A-Z, a-z), numbers (0-9) and (-). If you use other characters in computer names, no certificates can be generated.

Note

If you change the computer name after installing the "SIMATIC NET PC Software", the certificates installed for OPC UA are invalid and OPC UA will no longer work. Remedy: Create a new UA configuration with the "Communication Settings" program in "OPC UA certificates" > shortcut menu "Recreate OPC UA configuration".

UA client certificates for the OPC UA servers

The folder named above contains the location for computer-specific distinguishable OPC UA client certificates, once again in the "\certs" subfolder. For the locally installed OPC Scout V10, for example, it is the certificate "OPC.ScoutV10.der" including the public key.

This certificate is transferred to the OPC UA server at the start of communication. At this point it, must exist in the certificate management of the OPC UA server and it is compared with the certificate transferred by the OPC UA client at the start of communication. Without this certificate, secure communication is rejected.

UA revocation certificates for the OPC UA servers

A list of untrusted certificates can be created in the configurable "\crl" subfolder. Requests to establish secure connections with certificates from this list are rejected.

The "examplecrl.crl" list is already preinstalled and can be adapted.

Such lists can be created with the OpenSSL tool
(see <http://www.openssl.org/docs/apps/x509.html>).

SIMATIC NET S7 UA server names

The server name and the server URI are as follows:

```
<ServerUri>urn:Siemens.Automation.SimaticNET.S7:(%guid%)</ServerUri>
<ServerName>OPC.SimaticNET.S7</ServerName>
```

"%guid%" is a unique identifier for the server. It consists of 28 hexadecimal numbers and has the format ABCDEF01-2345-6789-0123456789AB.

Do not change this value.

SIMATIC NET S7OPT UA server names

The server name and the server URI are as follows:

```
<ServerUri>urn:Siemens.Automation.SimaticNET.S7OPT:(%guid%)</ServerUri>
<ServerName>OPC.SimaticNET.S7OPT</ServerName>
```

"%guid%" is a unique identifier for the server. It consists of 28 hexadecimal numbers and has the format ABCDEF01-2345-6789-0123456789AB.

Do not change this value.

SIMATIC NET PROFINET IO UA server names

The server name and the server URI are as follows:

```
<ServerUri>urn:Siemens.Automation.SimaticNET.PNIO:(%guid%)</ServerUri>
<ServerName>OPC.SimaticNET.PNIO</ServerName>
```

"%guid%" is a unique identifier for the server. It consists of 28 hexadecimal numbers and has the format ABCDEF01-2345-6789-0123456789AB.

Do not change this value.

SIMATIC NET DP-UA server names

The server name and the server URI are as follows:

```
<ServerUri>urn:Siemens.Automation.SimaticNET.DP: (%guid%)</ServerUri>
<ServerName>OPC.SimaticNET.DP</ServerName>
"%guid%" is a unique identifier for the server. It consists of 28 hexadecimal numbers and
has the format ABCDEF01-2345-6789-0123456789AB.
```

Do not change this value.

SIMATIC NET SEND/RECEIVE UA server names

The server name and the server URI are as follows:

```
<ServerUri>urn:Siemens.Automation.SimaticNET.SR: (%guid%)</ServerUri>
<ServerName>OPC.SimaticNET.SR</ServerName>
"%guid%" is a unique identifier for the server. It consists of 28 hexadecimal numbers and
has the format ABCDEF01-2345-6789-0123456789AB.
```

Do not change this value.

See also

Certificate management with a graphic user interface (Page 506)

4.4.3 Certificate management in the OPC UA client "OPC Scout V10"

Configuring certificate management for the OPC UA client "OPC Scout V10"

The supplied OPC UA client "OPC Scout V10" uses Windows certificate management.

How to manage certificates for a computer:

1. Log on as administrator with the system.
2. Click "Start" > "Run".
The input box opens.
3. Type in "mmc" and click "OK".
The console opens.
4. In the "File" menu, select "Add/ Remove Snap-in..." and then "Add".
The dialog box for selecting snap-ins opens.
5. Double-click on "Certificates" in the "Snap-in" list.
Select the "Computer account" option in the "Certificates snap-in" dialog and then click "Next".

6. Take one of the following actions in the "Select computer" dialog:
 - To manage certificates for the local computer, select the "Local computer" option and click "Finish".
 - To manage certificates for a remote computer, select the "Other computer" option, enter the computer name (or select it by clicking on "Browse") and then click on "Finish".
7. Click on "Close".
The "Certificates (computer name)" entry is displayed in the list of selected snap-ins for the new console.
8. Click "OK" if you want to add more snap-ins to the console.
9. To save this console, select the "File" menu and "Save" in the console and, if necessary, enter a name.
10. Close the console.

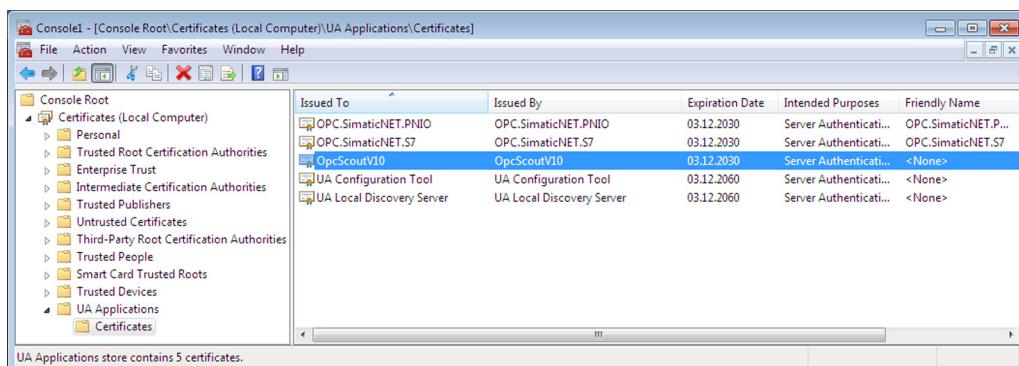


Figure 4-10 Windows of the console: Certificate management

OPC Scout V10 is listed and browsed for in the "Local computer" area in "UA Applications" under "OpcScoutV10".

How to export certificates from the certificate management of the client computer:

For the necessary secure communication with remote SIMATIC NET OPC UA servers, this certificate must first be exported from the certificate management of the client computer.

1. Open the console.
 "Action" > "All tasks" > "Export..." menu
2. Export the certificates without private key.

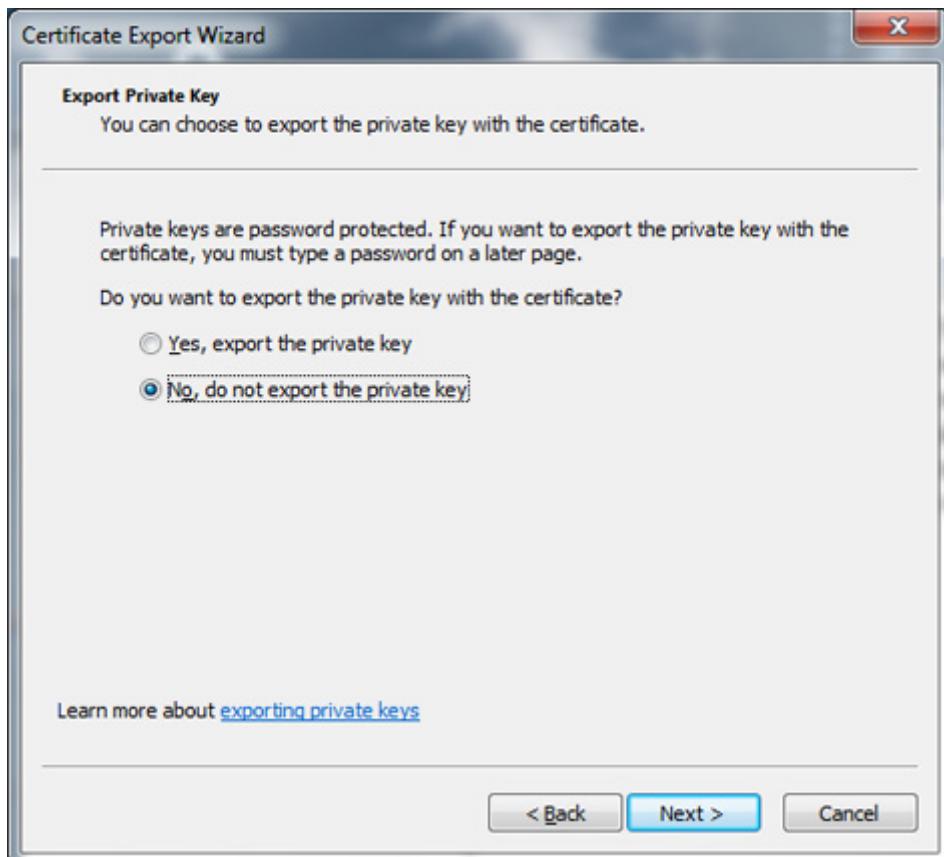


Figure 4-11 Exporting the certificate without private key

3. Export the certificates in the "DER-coded binary X.509" format.

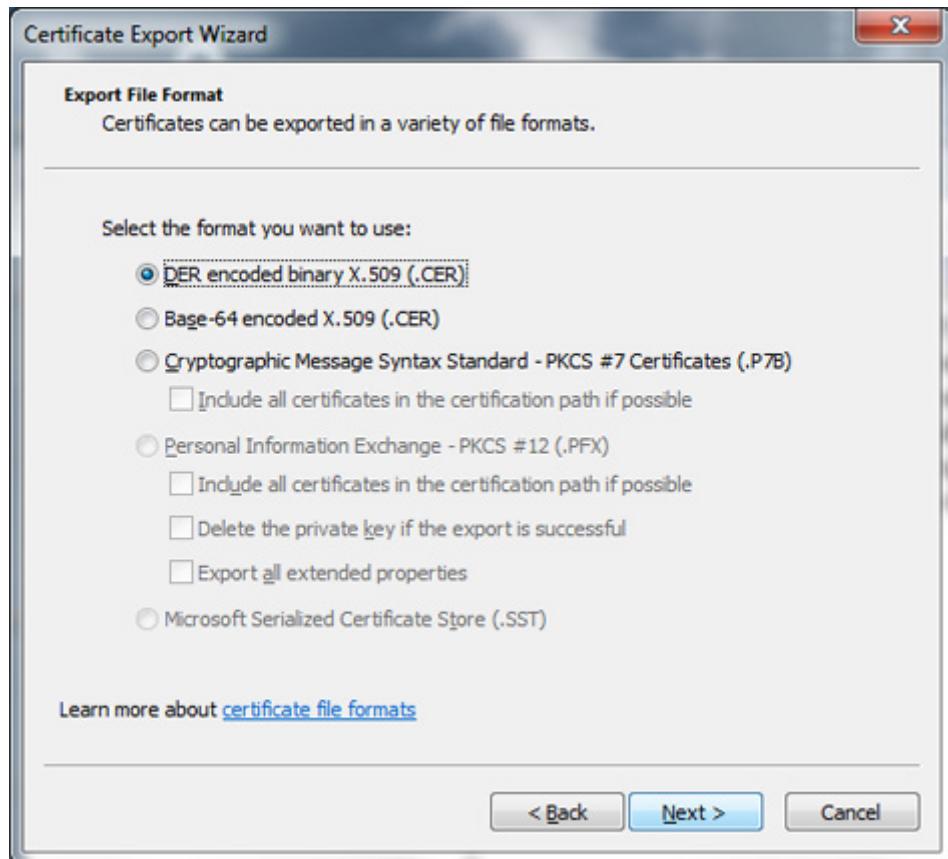


Figure 4-12 Exporting the certificate in the "DER" format

4. After selecting the required format, click "Next".

Remember that the file extension in the next dialog must also be "DER".

Change the file extension from "*.cer" to "*.der".

5. Click "Next" and "Finish" to complete the export.

The file is located in the folder with the same name in which the certificates of the server computer are also located.

Note

Certificate file in "*.der" format

If you created the certificate file DER-coded, but accidentally left the "cer" extension, you will need to change the file extension in the file folder of the client computer from "*.cer" to "*.der".

How to copy the certificates to a remote server:

For the necessary secure communication with remote SIMATIC NET OPC UA servers, this certificate must now be imported into the certificate management of the remote OPC UA server:

1. To do this, copy the certificate file of the client you have just exported to a suitable data medium.
2. Copy the certificate file of the client from this data medium to the relevant folder on the computer of the OPC UA server (see the section "Certificate management for the OPC UA server (Page 499)").

You can now establish a secure connection from the client computer to the OPC UA server.

Note

The computer name (host name) is coded in the certificate. Note that if the computer name is changed following installation, these certificates will become invalid. OPC UA communication is then no longer possible. The OPC UA configuration can, however, be recreated, refer to the section "Certificate management with a graphic user interface (Page 506)".

Note

You should also note that only standard characters can be used in the computer name and therefore also in the certificate.

Standard names consist of the letters (A-Z, a-z), numbers (0-9) and (-). If you use other characters in computer names, no certificates can be generated.

As of the SIMATIC NET PC software as of V8.0, the OPC UA server has certificate management also for incoming UA client certificates. These can be accepted or rejected in a graphic user interface in the "Communication Settings" configuration program. The client certificate of a remote server no longer needs to be transferred manually in advance by exporting and copying. This is only necessary if the client needs to be accepted during the first connection establishment.

4.4.4 Certificate management with a graphic user interface

As of the SIMATIC NET PC software V8.0, there is certificate management for the OPC UA servers with a graphic user interface in the "Communication Settings" configuration program.

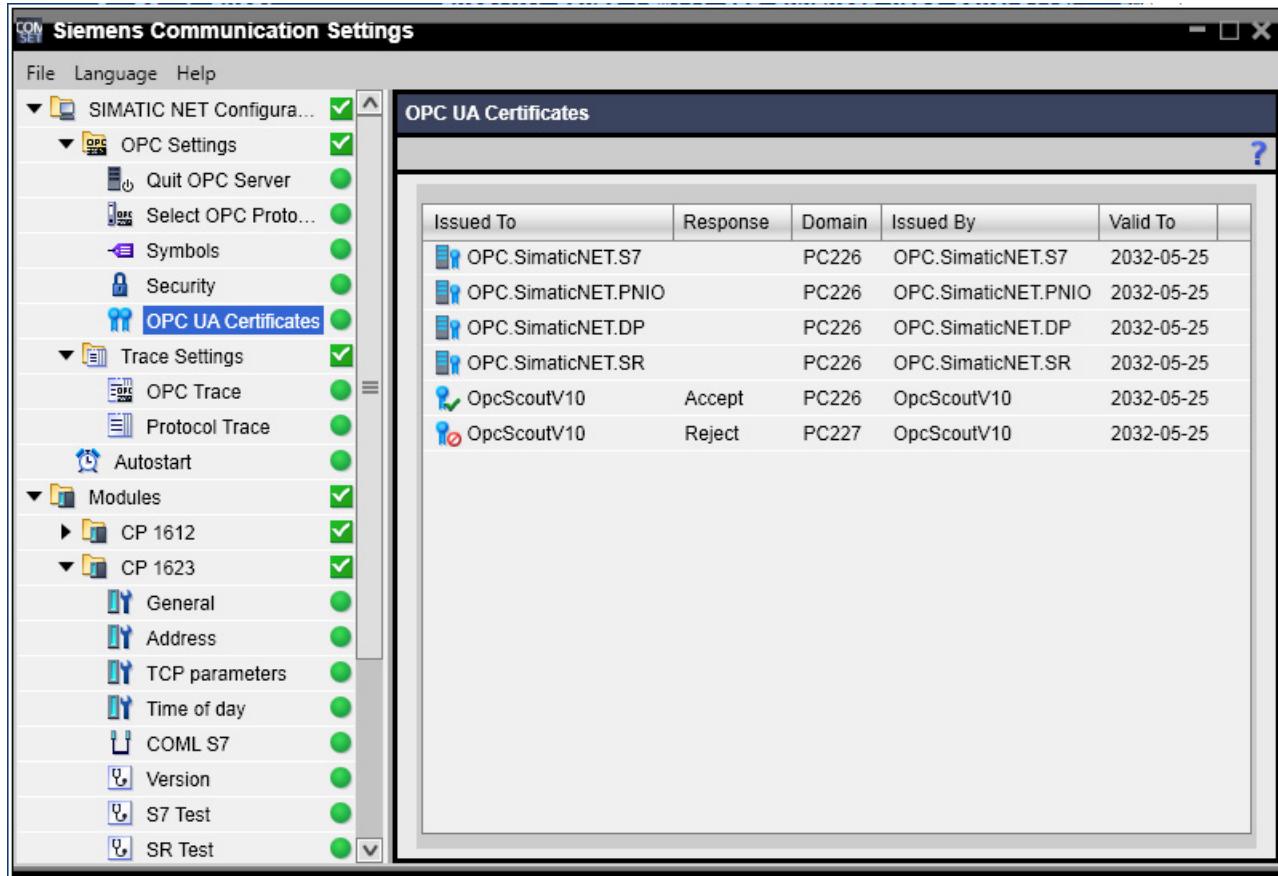


Figure 4-13 Certificate management in the "Communication Settings" configuration program

The certificates of the local OPC UA servers and the certificates with which OPC UA clients have identified themselves to the servers are displayed and can be managed here. The names of the computer with the software component for which the certificate was issued is also shown. The period of validity and the content of the certificate itself can also be displayed. Client certificates can be imported and server certificates exported as a DER-coded binary file (X.509).

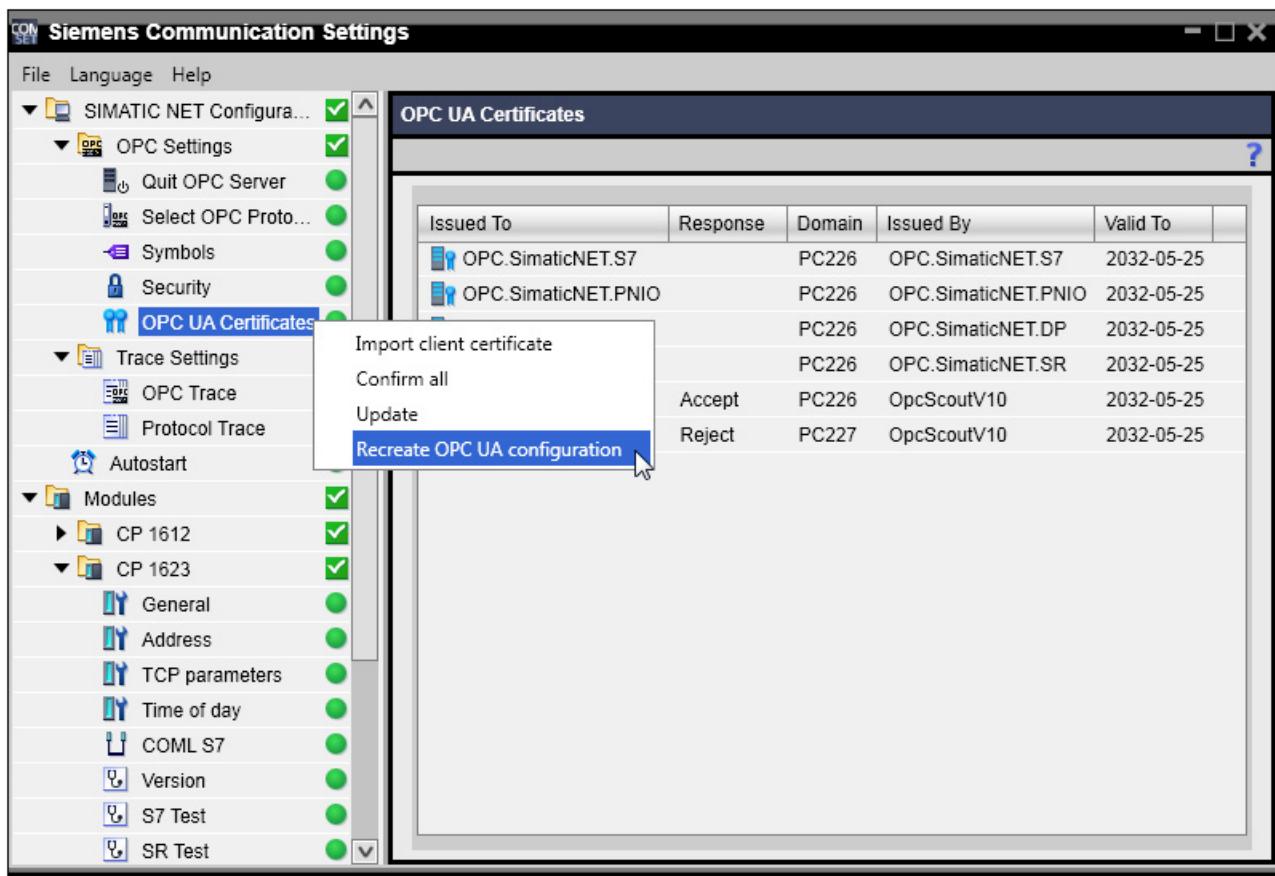


Figure 4-14 Configuring certificates in the "Communication Settings" configuration program

The local OPC UA configuration can, for example, be recreated if the computer name is changed. This involves the following:

- The configuration files of the local OPC UA servers for S7, S7 optimized, PROFINET IO, DP and SEND/RECEIVE are recreated (adaptation to the computer name).
- The OPC UA server certificates are deleted and recreated.
- The OPC UA client certificate for the local OPC Scout V10 is deleted and recreated.
- The certificate for the OPC UA discovery service is re-exported from the Windows certificate memory.
- The OPC server is exited and restarted.

Note

Any active SIMATIC NET communication should therefore be shut down before using this function.

4.4.5 OPC UA services

UA services

The OPC UA services are divided into logical function groups. The most important services and methods are described in this section. For more detailed information, refer to the OPC UA specifications at "<http://www.opcfoundation.org/UA>".

Discovery services

Using the discovery services, a UA client can obtain information on the configured endpoints and security requirements of a UA server:

- FindServers

This service returns the OPC UA servers known to the discovery server.

- GetEndpoints

This service returns the endpoints of a server and its security requirements to connect to a secure channel and a session.

Security services

The security services allow a secure communication channel to be established that ensures the integrity and confidentiality of exchanged messages between server and client.

- OpenSecureChannel

This service is used to open and to renew a secure channel. The certificates of client and server are exchanged. For more detailed information, refer to "<http://www.opcfoundation.org/UA>" under "Downloads" > "Specifications" > "Part 6".

- CloseSecureChannel

This service is used to close a secure channel.

Session services

The session services allow a client to authenticate the user wanting to use the session. Sessions can also be managed.

- CreateSession

This service is used by an OPC UA client to create a session. The server supplies two values that identify the session uniquely:

- The first value is the "SessionId" to identify the session uniquely in protocol files and in the namespace.
- The second value is the "AuthenticationToken" that assigns incoming queries to a session.

Prior to "CreateSession", "OpenSecureChannel" should be called to create a secure channel. The secure channel should be associated with the "AuthenticationToken" so that

only queries of this session are handled that can be associated with the session generated by the secure channel.

- **ActivateSession**

This service is used by the client to transfer certificates and the identity of the user of the session to the server. The certificates of the secure channel should also be the same as those used in the session. Client and server should check this and make sure it is the case.

- **CloseSession**

This service is used to close a session

- **Cancel**

This service is used to abort pending queries.

View services

The namespace can be browsed with these services. Nodes that will be written or read repeatedly can be registered.

- **Browse**

This service is used to read out the references of a node.

- **BrowseNext**

This service is used to request a next set of "Browse" or "BrowseNext" replies if a reply did not fit in a simple reply frame.

- **TranslateBrowsePathsNodeID**

The Browse paths of the NodeIDs are translated with this service. Each browse path begins with a start node and a relative path. The relative path includes a sequence of reference types and browse names.

- **RegisterNodes**

The "RegisterNodes" service should be used if nodes are accessed more than once or repeatedly (for example write or read values). This service allows the server to prepare for further accesses and to make subsequent access much more efficient. This achieves a noticeable improvement in performance on the OPC UA server. For more detailed information on the use of RegisterNodes, refer to the OPC UA specification at "<http://www.opcfoundation.org/UA>".

- **UnregisterNodes**

This service is used to unregister registered nodes.

Attribute (variable) services

The attribute services are used to read and write the attributes of nodes. Since the value of a variable is modeled as an attribute, these services are used to read and write values of variables.

- Read

With this service, one or more attributes can be read from one or more nodes. Array elements or ranges can also be read separately from arrays. Structured attribute elements can be indexed or read by area. The "maxAge" parameter instructs the server to read the data from a device or from a locally stored copy that is not older than maxAge.

- Write

This service is used to write one or more attributes of one or more nodes. Array elements or ranges can also be written separately in arrays. Structured attribute elements can be indexed or written by area.

Method services

This service is used to call methods. Input parameters can be transferred and output parameters received.

- Call

This service is used to call a list of methods.

Registration services

The registration services allow the client to generate, modify and delete subscriptions. Subscriptions send notifications - generated by MonitoredItems - to the client.

- CreateSubscription

This service is used to create a subscription. Subscriptions monitor a number of MonitoredItems for notifications and deliver these to the client as the response to a publish job.

- ModifySubscription

This service is used to modify a subscription.

- SetPublishingMode

This service activates the sending of notifications to one or more subscriptions.

- Publish

This service has two uses:

- Firstly, to confirm the receipt of notifications for one or more subscriptions.
- Secondly, it is used to request a notification or a "keep-alive" from the server.

- Republish

This service requests a repetition of the notification from the subscription.

- TransferSubscription

With this service, subscriptions and their MonitoredItems can be transferred from one session to another. This is possible both within a client and between clients.

- DeleteSubscription

This service is used to delete a subscription.

Monitoring services

The monitoring services are used along with the registration services to register for the monitoring of nodes in the namespace.

The monitoring services define services for generating, modifying and deleting item lists required to monitor attributes for data changes or objects for events.

- CreateMonitoredItems

Using this service, one or more MonitoredItems can be assigned to a subscription.

- ModifyMonitoredItems

Using this service, one or more MonitoredItems of a subscription can be modified.

- SetMonitoringMode

This service is required to set the monitoring mode of one or more MonitoredItems of a subscription.

- DeleteMonitoredItems

Using this service, one or more MonitoredItems of a subscription can be deleted.

4.4.6 Creating OPC UA clients

4.4.6.1 Interfaces under OPC UA

The programming interfaces in OPC UA

To create OPC UA clients, the programming interface should be selected depending on the application and required certificate management:

- The C interface for simple, high-speed access to OpenSSL certificate management
- The .NET interface for convenient use and access to the Windows certificate management

Note

You will find the relevant sample programs in the section "OPC UA interface in C (Page 720)".

4.4.6.2 The C interface with OPC UA

The C interface

The OPC UA C interface is suitable for OPC UA client applications that are, when necessary, easy to port to other systems.

You will find the necessary header files in the following folder after installation:

"<Installationprogrampath>\SIMATIC.NET\opc2\inc\"

The required import library "uastack.lib" and the loadable library "uastack.dll" are located in the following folders:

"<Installationprogrampath>\SIMATIC.NET\opc2\bins7\"

"<Installationprogrampath>\SIMATIC.NET\opc2\binpnio\"

"<Installationprogrampath>\SIMATIC.NET\opc2\bindp\"

"<Installationprogrampath>\SIMATIC.NET\opc2\binsr\"

"<Installationprogrampath>\SIMATIC.NET\opc2\bins7opt\"

The OpenSSL import library "libeay32.lib" and the loadable library "libeay32.dll" are also necessary for handling certificates. You will also find these in the directories named above.

This allows a secure OPC UA C client application to be created.

Note

For more information on the implementation, use the sample implementation from the OPC Foundation.

4.4.6.3 The .NET interface with OPC UA

The .NET interface

The OPC UA .NET interface is suitable for simple, convenient, programmable OPC UA client applications. OPC Scout V10, for example, uses the UA .NET interface.

You will find the necessary assembly files in the following folder after installation:

"<Installationprogrampath>\SIMATIC.NET\opc2\bin\"

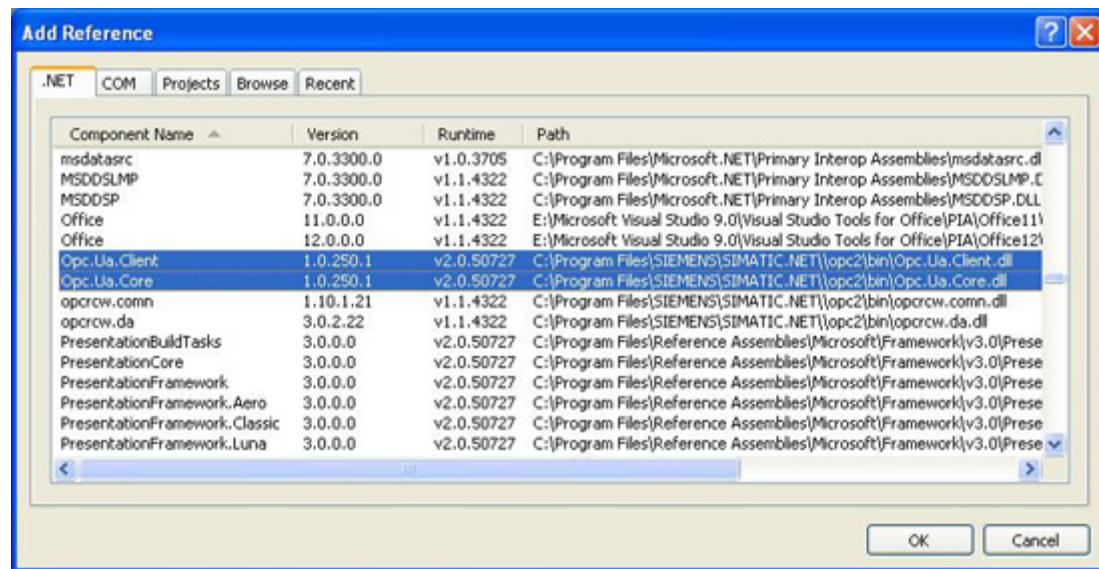


Figure 4-15 The assembly files of the .NET interface

The following components are required:

- The UA client component "Opc.Ua.Client.dll"
- The UA core component "Opc.Ua.Core.dll"

You can also use the corresponding OPC UA components in the OPC Foundation folder that is installed from the SIMATIC NET CD. You will find this in the following directory:

"<Installationprogrampath>\OPC Foundation\UA\V1.0\Bin"

4.4.7 Messages of the OPC UA server

System messages

The SIMATIC NET OPC UA servers output system messages that are output on the interface with an error code and in OPC Scout V10 with the symbolic ID.

The following table lists the most important system messages.

You will find more messages in the header files of OPC UA or in the specification of the OPC Foundation ("<http://opcfoundation.org/UA>") in Part 4 and Part 6.

Table 4- 1 System messages of the SIMATIC NET OPC UA server

Symbolic ID	Error code (hex)	Meaning
Good	0x00000000	Good - everything as expected
Good_Overload	0x002F0000	The current recording of data values has slowed down due to resource bottlenecks.
Good_Clamped	0x00300000	The value was accepted and written. The sensor (data destination) exceeded the permitted setting range.

Symbolic ID	Error code (hex)	Meaning
Uncertain	0x40000000	Uncertain - no further details
Bad	0x80000000	Invalid - no further details
Bad_NoCommunication	0x80310000	The communications endpoint is defined. Communication itself is not established and there is no last known value. The status / substatus relates to cache values before receiving the first value.
Bad_NodeIdInvalid	0x80330000	The syntax of the Nodeld is invalid.
Bad_NodeIdUnknown	0x80340000	The ID of the node references a node that does not exist in the server address space.
Bad_NodeClassInvalid	0x805F0000	The value of the node is invalid.
Bad_SourceNodeIdInvalid	0x80640000	The ID of the source node does not reference a valid node.
Bad_TargetNodeIdInvalid	0x80650000	The target node does not specify a valid node.
Bad_NoDeleteRights	0x80690000	The server does not allow the node to be deleted.
Bad_HistoryOperationInvalid	0x80710000	The "history details" parameter (details of the history) is invalid.
Bad_HistoryOperationUnsupported	0x80720000	The server does not support the requested procedure.
Bad_IndexRangeInvalid	0x80360000	The syntax of the area parameter of the index is invalid.
Bad_IndexRangeNoData	0x80370000	There is no data within the specified index.
Bad_DataEncodingInvalid	0x80380000	The data encoding is invalid.
Bad_DataEncodingUnsupported	0x80390000	The server does not support the requested data encoding for the node.
Bad_NotReadable	0x803A0000	The security level does not allow reading of or logging on at the node.
Bad_NotWritable	0x803B0000	The server does not allow writing to the node.
Bad_OutOfRange	0x803C0000	The value is outside the permitted range.
Bad_NotSupported	0x803D0000	The requested activity is not supported.
Bad_NotFound	0x803E0000	A requested element was not found or browsing was terminated without success.
Bad_ObjectDeleted	0x803F0000	The object cannot be used because it was deleted.
Bad_NotImplemented	0x80400000	The requested activity is not implemented.
Bad_MonitoringModelInvalid	0x80410000	The monitoring mode is invalid.
Bad_MonitoredItemIdInvalid	0x80420000	The ID of the monitoring item does not reference a valid monitored item.
Bad_MonitoredItemFilterInvalid	0x80430000	The parameter for the filter condition of the monitored item is invalid.
Bad_MonitoredItemFilterUnsupported	0x80440000	The filter condition of the monitored item is not supported by the server.
Bad_StructureMissing	0x80460000	A necessary structured parameter is missing or equals zero.
Bad_EventFilterInvalid	0x80470000	The event filter condition is invalid.
Bad_ContentFilterInvalid	0x80480000	The content filter condition is invalid.
Bad_FilterOperandInvalid	0x80490000	The operand used in the content filter condition is invalid.
Bad_ContinuationPointInvalid	0x804A0000	The return value of the continuation point remains valid.
Bad_NoContinuationPoints	0x804B0000	The requested job was not executed due to deleting all continuation points.

Symbolic ID	Error code (hex)	Meaning
Bad_ReferenceTypeIDInvalid	0x804C0000	The reference type does not reference a valid reference type node.
Bad_BrowseDirectionInvalid	0x804D0000	The browse direction is no longer valid.
Bad_NodeNotInView	0x804E0000	The node is not part of the view.

4.4.8 Migration from OPC Data Access / Alarms & Events to OPC UA

Which services differ between OPC Data Access / Alarms & Events and OPC Unified Architecture?

If you want to migrate your existing OPC DA / AE application to OPC UA, here, you will find an overview of the corresponding services in the two systems.

OPC Data Access / OPC Alarms & Events	OPC Unified Architecture
CoCreateInstanceEx()	OpenSecureChannel() CreateSession() ActivateSession()
ChangeBrowsePosition() BrowseOPCItemIDs() GetItemID() QueryAvailableProperties() GetItemProperties()	Browse() Read()
Access via the item name: Read() (DA 3.0) Write() (DA 3.0)	Access via the node ID: Read() Write()
Access to items of a group using a handle: AddItems() Read(...,handle,...) Write (... ,handle,...) RemoveItems()	Access to registered nodes using a handle: RegisterNodes() Read(...,handle,...) Write (... ,handle,...) UnregisterNodes()
AddGroup() SetState() RemoveGroup()	CreateSubscription() ModifySubscription() DeleteSubscription()
AddItems() RemoveItems()	CreateMonitoredItems() DeleteMonitoredItems()
DataChange()	Publish()

The SIMATIC NET OPC server continues to support the syntax of the previous specifications such as "OPC Data Access" or "OPC Alarms & Events". This simplifies migration to OPC UA.

4.5 Programming the OPC UA redundancy interface

Transparent redundant OPC UA servers

Redundant OPC UA servers are used in normal operation to distribute load. If one of the OPC UA servers fails, its communications functions are taken over by the redundant OPC UA server.

Transparent means that from the perspective of the OPC UA client, the OPC UA servers configured in a cluster appear to be a single OPC UA server that is proof against failure. The OPC UA client uses a single cluster IP address to address the OPC UA servers of a cluster. The data synchronization between the OPC UA servers ensures that the failover between two OPC UA servers is handled automatically and largely without any action being necessary on the part of the OPC UA client.

How the transparent redundant OPC UA server works

With the help of the OPC UA redundancy specification, you can use transparent server redundancy in the standard automation technology. A standard UA client requires no additional proprietary redundancy software.

When using transparent redundant servers, the tasks of the OPC UA clients are evenly distributed on the OPC UA servers assuming no server has failed. This ensures a balanced load. The uniform task distribution of the clients on the two OPC UA servers (see figure) allows better use of the resources.

The OPC UA client sessions and subscriptions are synchronized regularly between OPC UA servers. The servers are synchronized via industrial Ethernet at 10/100/1000 Mbps. This means that no extra cabling is necessary for the synchronization. If the connection between a client and an OPC UA server is interrupted, the 2nd OPC UA server takes over its tasks. The sessions and subscriptions are only briefly interrupted, allowing work to be resumed quickly.

You can configure the OPC UA redundancy server centrally with STEP 7. This allows you to create a configuration with only a few configuration specifications. Certificates are created and distributed when necessary.

The S7 protocol connection between the servers is not configured redundantly. In other words, if a server fails, data is read again by the 2nd server.

The figure below is a graphic representation of load balancing when two OPC UA servers are used.

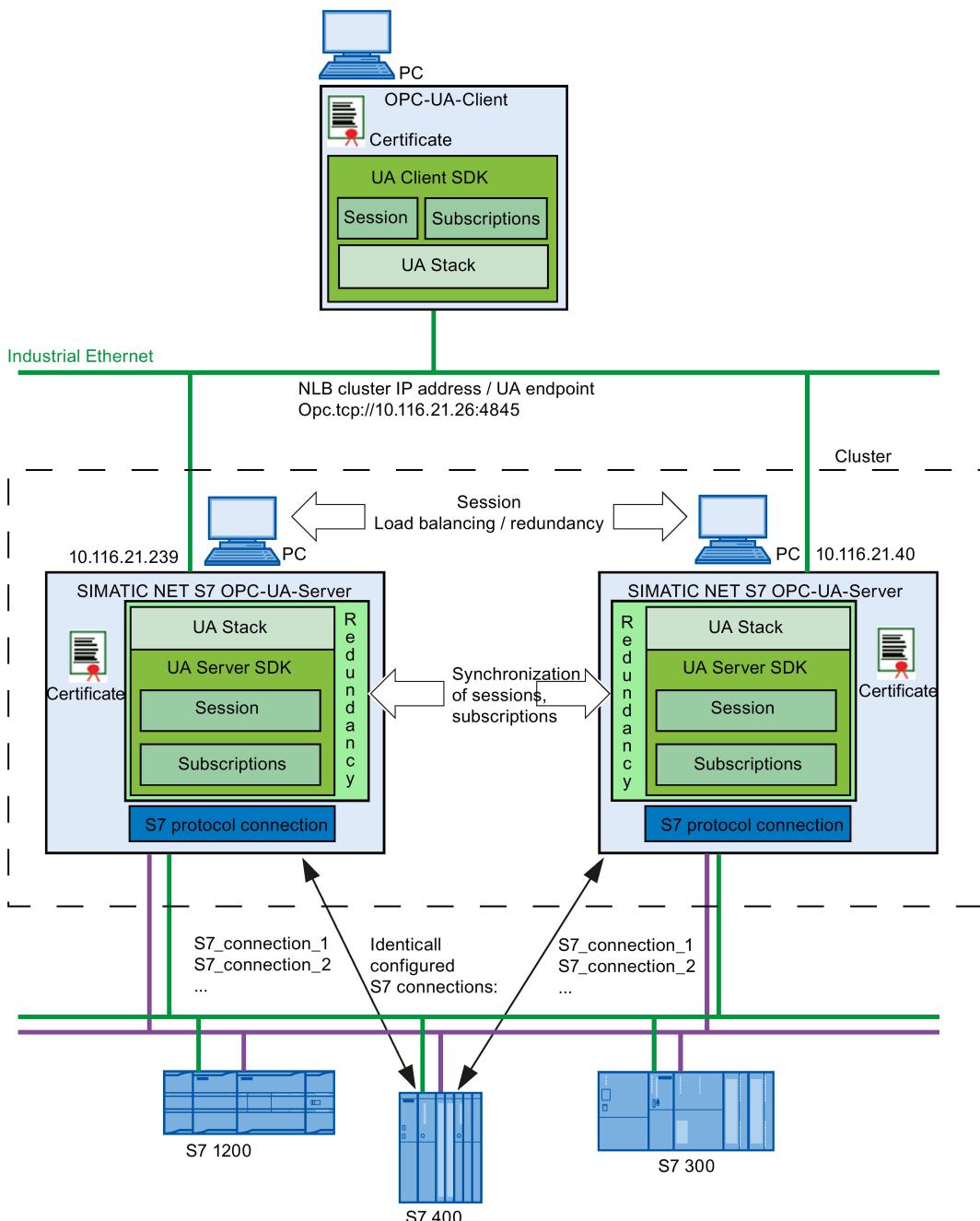


Figure 4-16 Schematic structure of a redundant network

The Windows feature "Network Load Balancing" (NLB) is required for transparent OPC UA redundancy. Since the NLB is available only with Windows server operating systems, a Windows server operating system is a minimum requirement. As of the SIMATIC NET PC software V8.1, "Windows Server 2008 R2 64-Bit" is required for this.

The "Network Load Balancing" feature (NLB)

The Windows "Network Load Balancing" (NLB) feature guarantees load distribution and redundancy of multiple server PCs.

With NLB, you can register various OPC UA server PCs in the network under one network address. To allow this, NLB provides a cluster IP address for a client behind which up to 32 physical server PCs can be grouped together. When a new TCP/IP connection is established, the connections are distributed evenly over the available servers. This ensures load distribution and high availability. No special configuration of the client is necessary for redundancy.

If the OPC UA server to which a connection exists fails, the client detects the interrupted connection depending on the configured connection monitoring time. In the reconnect scenario defined in the OPC UA specification, a new TCP/IP connection is automatically established to the redundant server with the help of NLB.

The synchronization of session and subscription information between the OPC UA servers of a redundancy set is only performed after the return message to the OPC UA client and also requires some time (see figure above). The synchronization is performed via Industrial Ethernet using an OPC UA connection with separate ports. This means that the client receives data again immediately after the TCP/IP reconnect without needing to create new subscriptions. The HMI operator does not receive any data updates during the failover. On most OPC UA clients, the quality of the data is also set briefly to "BAD" between the detection of the connection error and the reconnect. For the OPC UA client, the failure of an OPC UA server simply looks like a TCP/IP connection abort. If the connection between the OPC UA servers is disrupted, it is possible when there is a failover of the connection of the OPC UA client to the redundant OPC UA server that this does not have all the information about newly created sessions and subscriptions.

Condition

To be able to use this function, all servers must be identically configured. For more detailed information on identical configuration, refer to the section "Configuration of the cluster (Page 524)" in "Ensuring consistency of the OPC UA servers".

Notes on and restrictions for working with "Network Load Balancing" (NLB)

The NLB port rules decide the response of the servers during load balancing. In the default settings, the protocols are set to "Both" and filtering mode to "Multiple host" with affinity set to "Single". If you change these settings, the response of the OPC S7 redundancy server to load balancing can be changed. For example if protocols is configured to "TCP" and filtering mode at the same time to "Multiple host" with affinity "None". You should also check the repercussions on other server applications that use NLB.

4.5.1 Installing and configuring the "Network Load Balancing" feature

To allow redundant server operation, as of SIMATIC NET PC software V8.1, the "Windows Server 2008 R2 64-Bit" operating system with the "Network Load Balancing" Feature (NLB) is required.

Installing Network Load Balancing

To install the feature, follow these steps:

1. Start the installation.

Click "Start" > "Control Panel" > "Administrative Tools" > "Server Manager" > "Features", "Add Features"

Reaction: The "Add Features Wizard" dialog opens.

2. Select the "Network Load Balancing" feature and click "Install".
3. Follow the installation instructions.

Configuring Network Load Balancing

To configure the "Network Load Balancing" (NLB) feature, start the "Network Load Balancing Manager". Follow the steps outlined below:

1. Start the "Network Load Balancing Manager".

Click "Start" > "Control Panel" > "Administrative Tools" > "Server Manager" > "Network Load Balancing Manager".

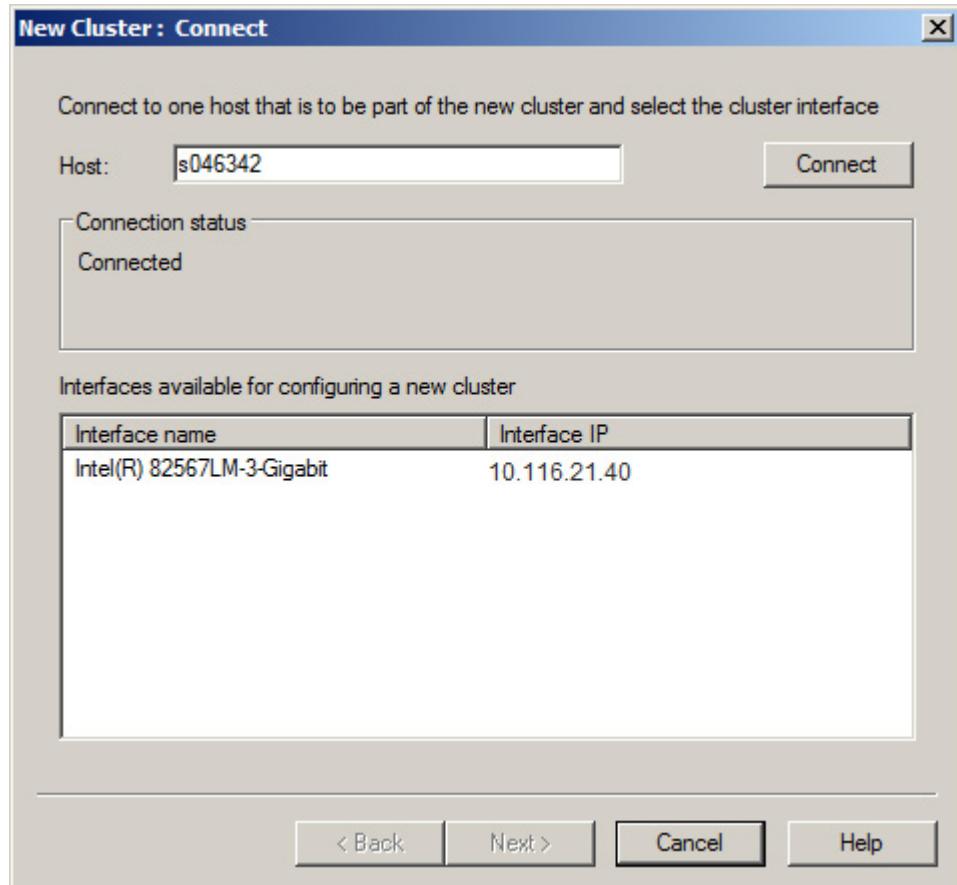
2. Create a new cluster.

Click "New" > "Cluster" in the menu

Reaction: The "New Cluster: Connect" dialog opens.

3. Connect the cluster first to the local host and the local Ethernet interfaces (see figure below).

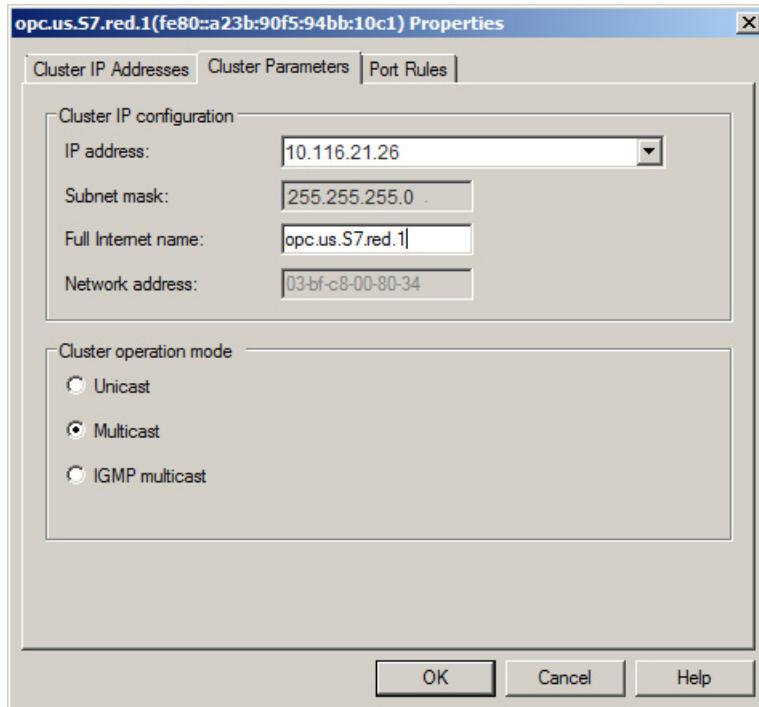
In the "Host:" input box enter the name of the host and click "Connect".



4. Open the "Properties" dialog.

Right-click on the interface name and select "Properties" from the shortcut menu.

Reaction: The "Properties of..." dialog opens.



5. Then configure the IP address and the Internet name of the cluster (see figure).

It is advisable to use the cluster operation mode "Multicast" or "IGMP multicast" to allow multiple use of the network adapter.

Note

Selecting the cluster operation mode

- With the parameters for the cluster operation mode, you specify whether or not a multicast MAC address (media access control) will be used for cluster operations. If multicast is activated, the cluster MAC address for the cluster adapter is converted to a multicast address by the network load balancing. This also ensures that the primary IP address of the cluster is resolved in this multicast address as part of the ARP (Address Resolution Protocol). The original integrated MAC address of the adapter that was deactivated in unicast mode can now be used.
- You need to activate the support for multicast before activating the support for the Internet Group Management Protocol (IGMP). IGMP support can also be activated on the network adapter.
- The IGMP multicast check box activates the IGMP support for restricting switch flooding. This restriction is achieved by restricting the data traffic to "network load balancing ports". Activate the IGMP support to ensure that the data traffic intended for an NLB cluster is directed only through the ports intended for the cluster hosts.

6. Confirm with "OK".

Adding further hosts to the cluster

Note

If you want to add further hosts (servers) to the cluster, you require administrator privileges on the remote host.

To obtain administrator privileges on the remote host, you need to log on with administrator privileges in the "Network Load Balancing Manager". Follow the steps outlined below:

1. Open "Credentials" in the "Network Load Balancing Manager".
Click "Options" > "Credentials".
Reaction: The dialog window "NLB Manager Default Credentials" is opened.
2. Enter a user name and the corresponding password.

With the servers in workgroup mode, use the default Windows user "Administrator" (builtin) with the corresponding password.



3. Confirm with "OK".

Note

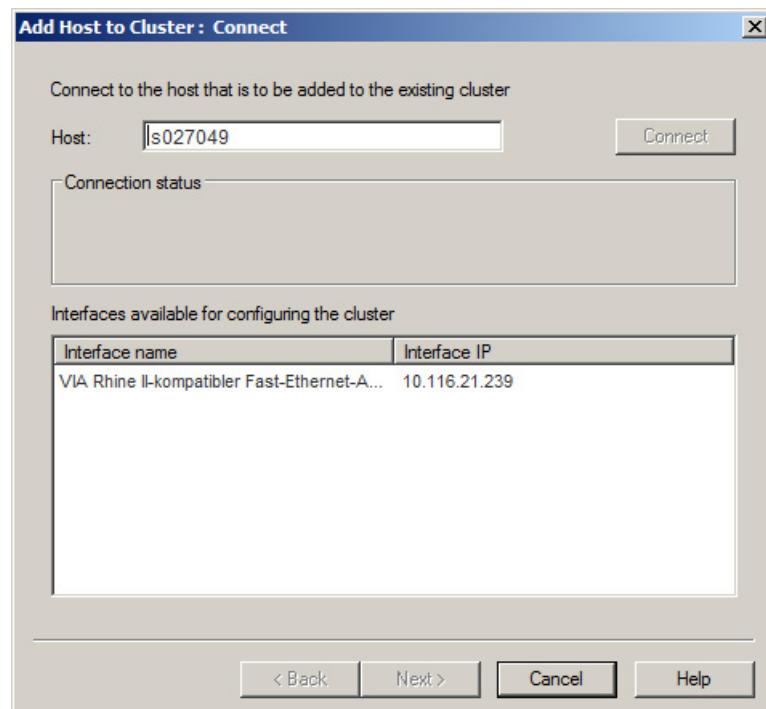
The domain mode of the server provides greater security since the user rights are managed by the Windows domain. This mode is recommended.

To add a further remote host to the cluster, follow these steps:

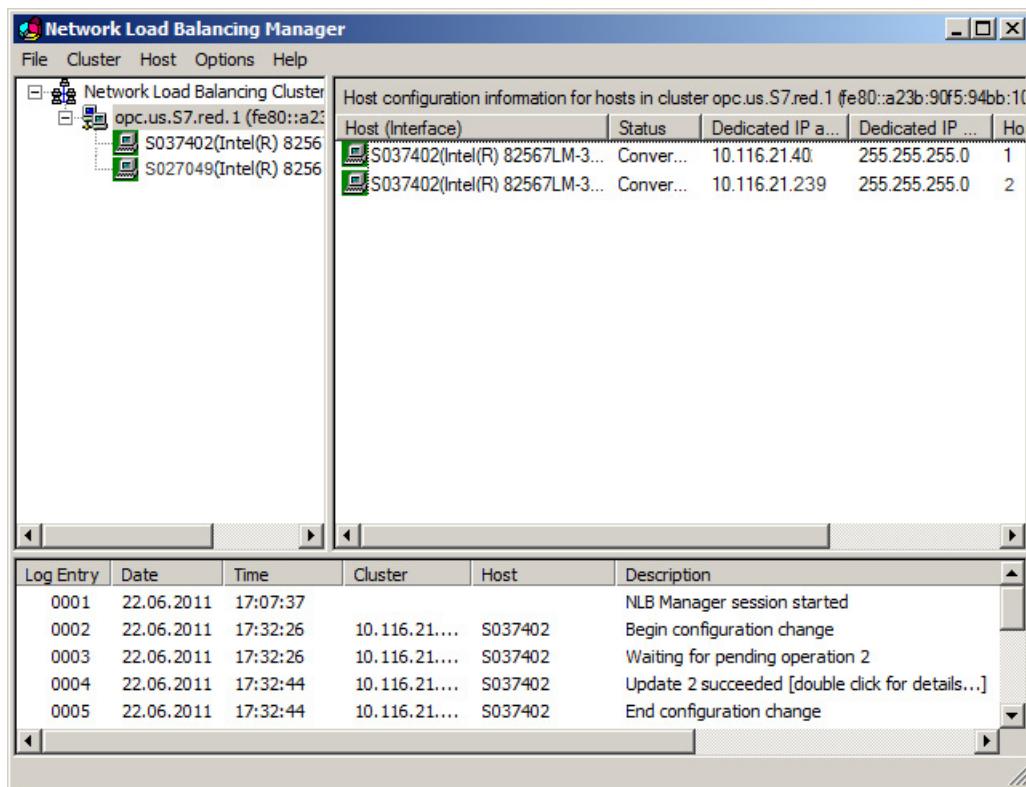
1. Click on the "Cluster" > "Add Host" menu command.

Reaction: The "Add Host to Cluster: Connect" dialog opens.

2. In the "Host:" input box enter the name of the host you want to add to the cluster and click "Connect".



Once the host connection is established successfully, a new cluster is created.



Note

Active and suitably configured "network load balancing" (NLB) is necessary for transparent OPC UA redundancy.

You can test whether or not the NLB is active using a PING in the operating system.

4.5.2 Configuration of the cluster

The initial configuration of the cluster is created in STEP 7. Among other things, the synchronization of sessions and subscriptions can be configured with STEP 7.

Using the configuration interface you can do the following:

- Configure the virtual network address of the cluster (for example redundant IP address and redundant port).
- Configure the port for server-to-server communication between the redundant OPC UA servers.
- Specify the IP addresses of the members of the other clusters.
- Select the local server from the list.

The certificate of redundant set can, if necessary, be created and displayed in the configuration.

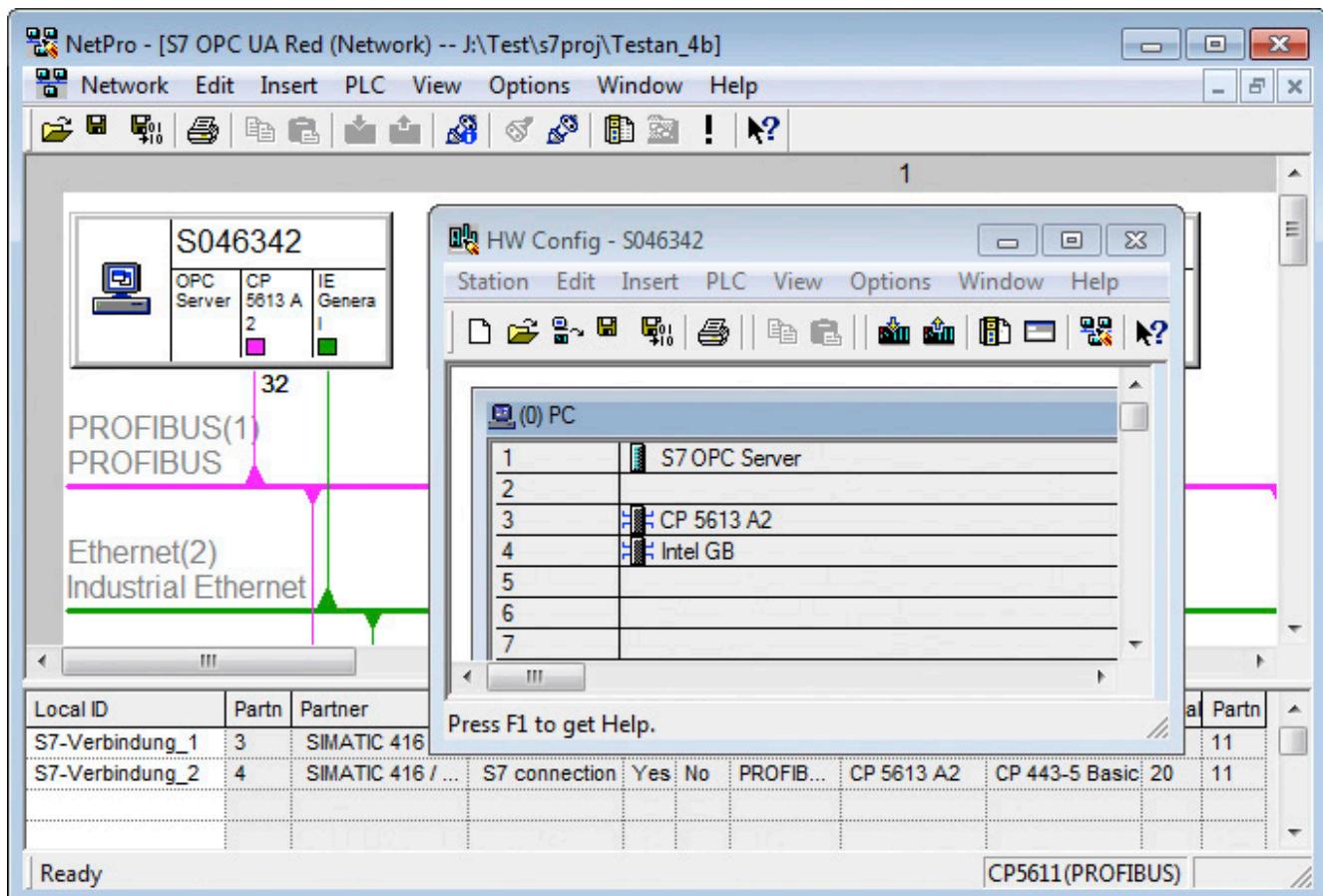
Note**Configuration interface**

- The list and the sequence must identical on all servers of the cluster.
 - Only IPv4 addresses can be used.
 - The configuration of "gateway" or "subnet mask" is not necessary.
 - MAC addresses do not need to be configured.
-

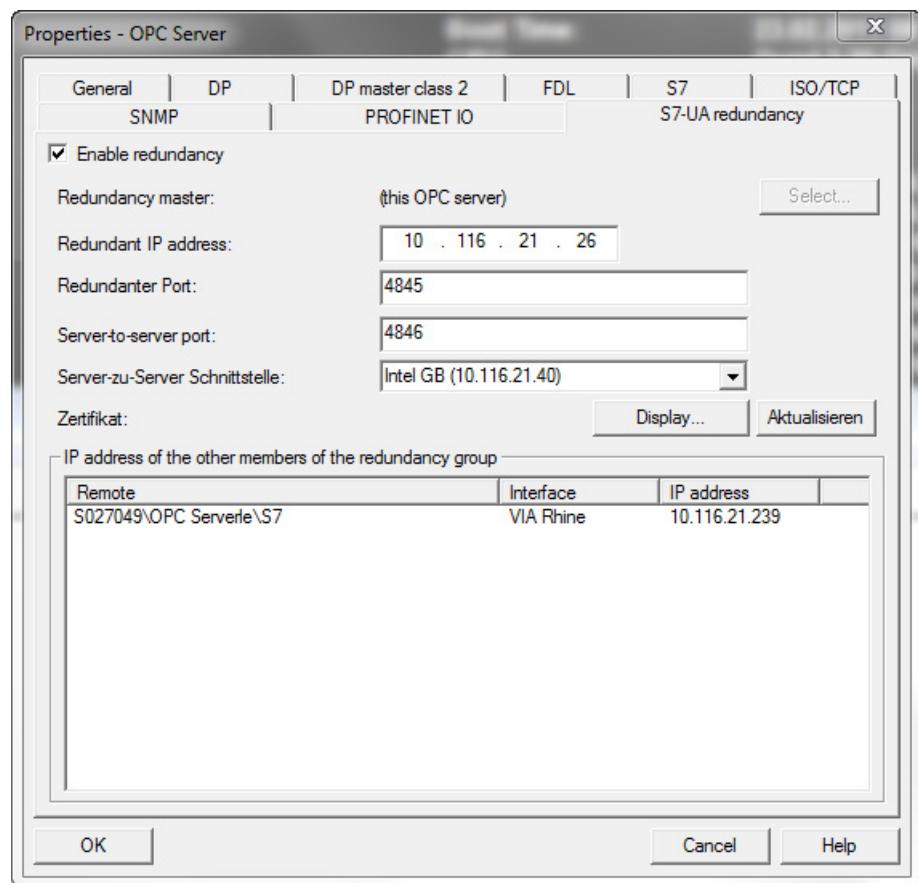
The "Network Load Balancing" (NLB) cannot be configured with STEP 7/NCM. Port blocking lists for the NLB are also not configured here. The configuration of the NLB is integrated in the operating system on the redundant servers and must be performed there.

Configuration example

The following example illustrates a STEP 7 project with a plant bus e.g. "PROFIBUS(1)" and an "Industrial Ethernet(2)" for the OPC client/server communication. The redundant servers each have two CPs, in the example "CP 5613 A2" and "IE General". As the destination station, an S7 station can be selected. These should be connected via the plant bus and suitable CPs.



The following figure shows an example with the necessary configuration options. This configuration is integrated in STEP 7. The parameters for S7 UA redundancy can be set in the Properties window of the OPC server (as of version 8.1).



Ensuring consistency of the OPC UA servers

In the consistency check, e.g. when you save and compile the project, the following factual information is checked:

- The server-to-server addresses within a cluster must be unique. A common IP address of a cluster may also only be used once within a project.
- Each member of a cluster must have the same S7 connections with the same connection names and the same communications partners.
- The hardware configuration of the PC stations in the clusters must be identical.

- The parameters of the S7 connections must be identical. This includes: connection establishment, alarms, optimizations, PDU size, number to parallel jobs, monitoring times.
- The S7 settings of the OPC UA servers must be identical. This includes: cycle time, access rights and S7 symbols.

Note

All servers can, however, be configured so that they can also reach other using an individual IP address in the same network.

Note

At the S7 protocol level, the S7 Redconnect product can also be used. It is, however, not a requirement for OPC redundancy.

Note

It is strongly advised that you configure active, permanently established S7 connections on the OPC UA servers to achieve the following:

- shorter failover times if there is a server failure.
 - the ability to check connection statuses of the passive redundant server (S7 connection diagnostics).
-

To have the same namespace on both servers, the two PC stations must be configured identically. To achieve this, we recommend that you first create a SIMATIC PC station and configure this in NetPro (for example to create the required S7 connections).

The created SIMATIC PC station, in the figure "S046342", can now be duplicated in the SIMATIC Manager. With the "Copy" and "Paste" menu commands, you can copy the PC station including S7 connections with the same name. Now adapt the station name, in the figure "S027049", the local addresses and the destination of the S7 connections. If further S7 connections are added, they must be configured identically on both stations.

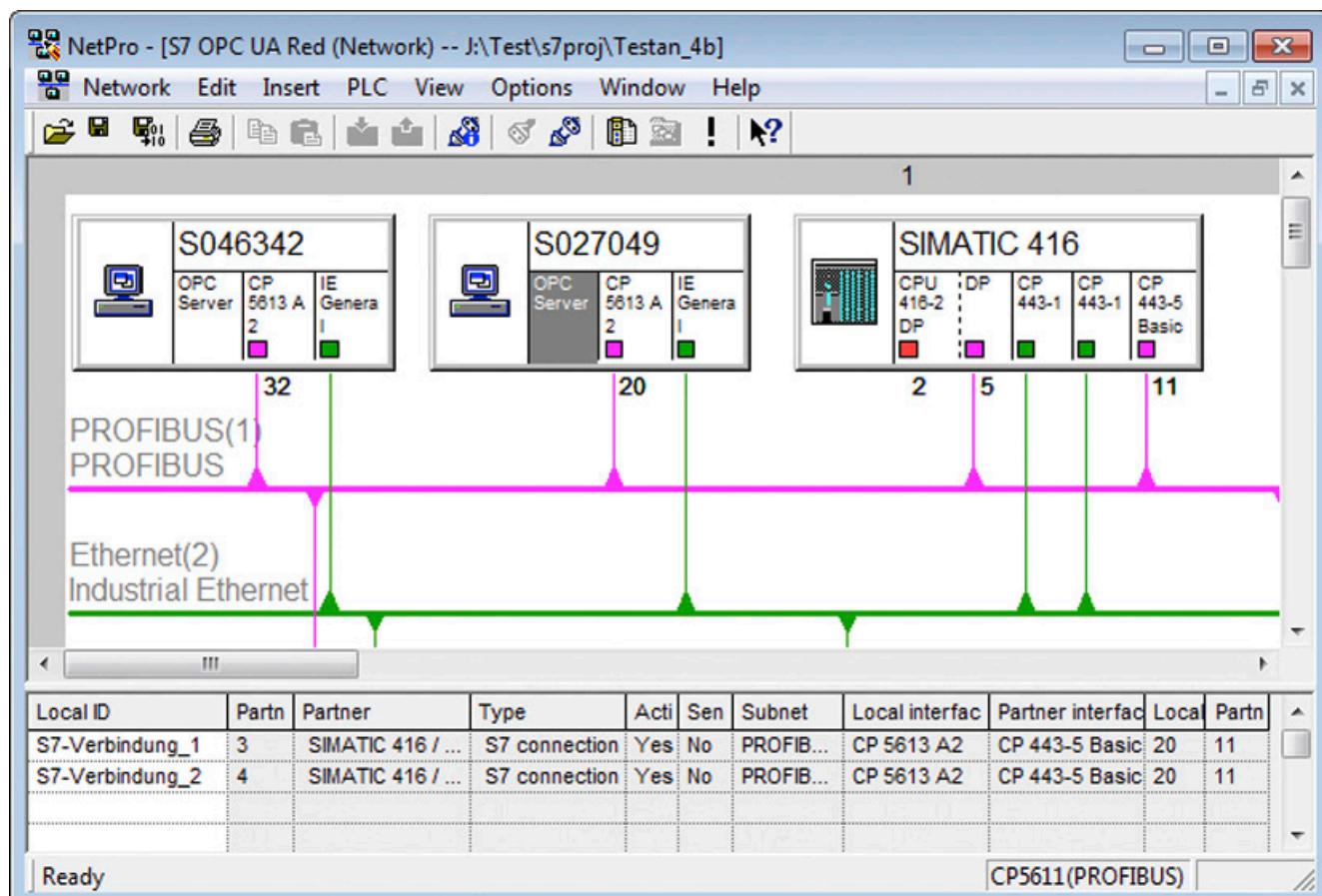


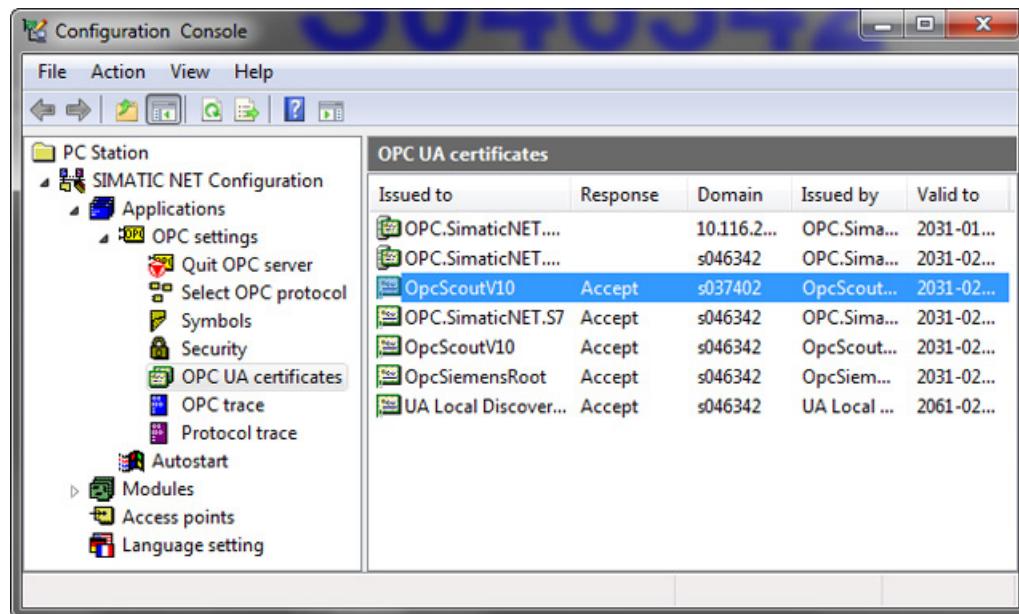
Figure 4-17 PC stations configured identically in NetPro with S7 connections

In the redundancy configuration, you also assign this station to the cluster of the first station. This is the simplest way to create a redundant server set with the same S7 connections to the same SIMATIC S7 station.

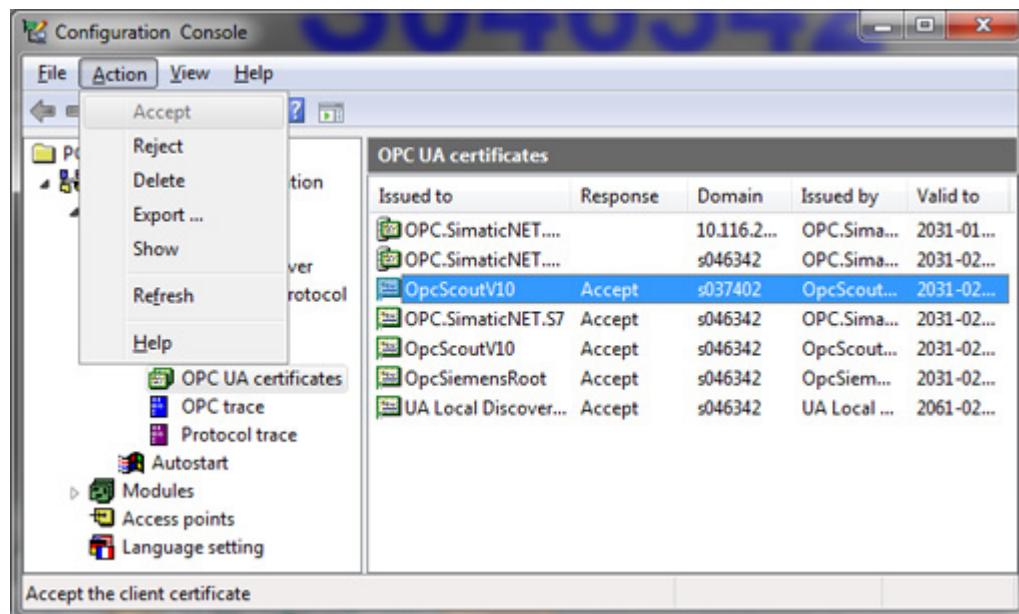
During configuration, remember that double the number of S7 connections is necessary for the SIMATIC S7 station.

Downloading configurations and updating UA certificates

Once the configuring is complete and the consistency check was successful, you will need to download the configurations for all nodes. The identical OPC UA certificate is also downloaded and activated on the two SIMATIC PC stations. You check this with the configuration program "Communication Settings" ("Applications" > "OPC Settings" > "OPC UA Certificates").



In the certificate management, you must also accept the certificates of the OPC UA clients. To do this, connect the OPC UA client with the redundant server using the redundant IP address (for example "10.116.21.26", compare the figure in the section "Configuring Network Load Balancing").



When the NLB is active, one of the two existing servers will be connected, accept the client certificate there on secure connections. Check the configured OPC UA namespace and usability of your items. To accept the client certificate on the second server as well, unplug the network cable from the first server. Then connect to the OPC UA client again using the redundant IP address. The NLB will now connect to the second server. Enter the computer name or the local IP address of the PC. The second redundant server should behave in exactly the same way as the first and supply the same values.

If the connection is not established, check your system structure and your configuration.

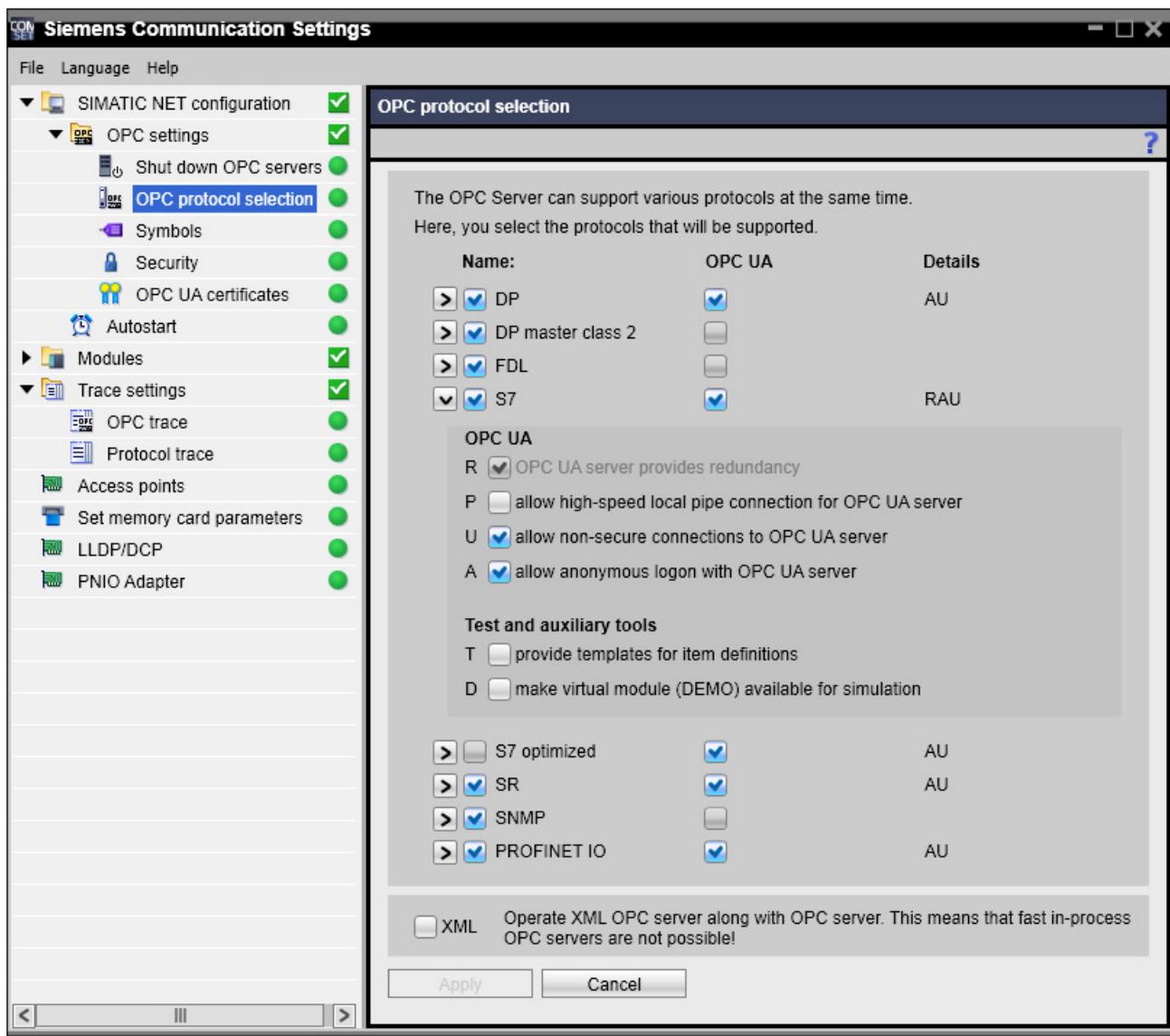


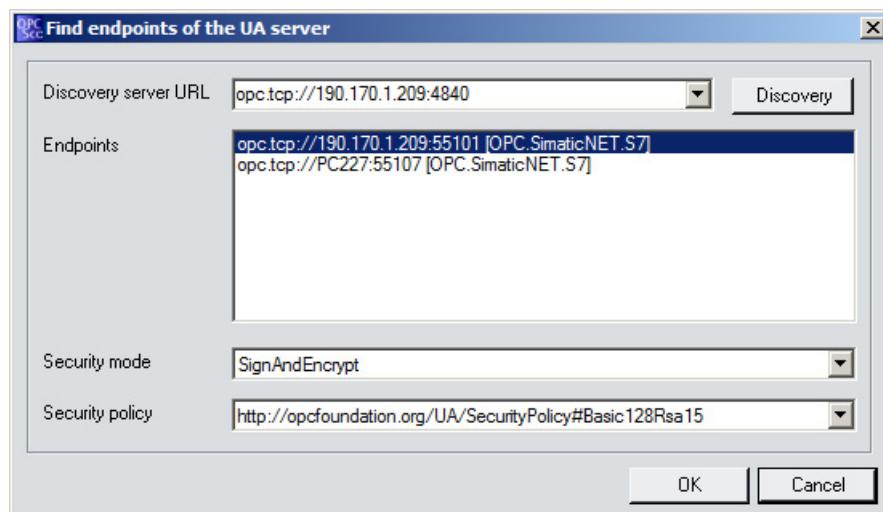
Figure 4-18 View of the protocol details when OPC UA server redundancy is functioning

Testing the redundancy failover

You can test the redundancy failover by unplugging the network cable of the active server. When the interruption on the network is detected on the client, there must be a reconnection to the existing other server. The failover response can be monitored using the diagnostics items. You will find more information on this topic in the section "Reading out the status of the cluster by the OPC UA client (Page 541)".

4.5.3 Locating the server endpoints with the help of the discovery server

Using the OPC UA discovery services, you can locate endpoints of redundant S7 OPC UA servers. By entering the common cluster IP address and the port ("4840"), the endpoint can be found using the OPC UA TCP protocol.



Since the TCP connection via NLB is transparent and inherently load balancing, it is not possible to predict which of the redundant servers will be connected. The list box displays all the endpoints on the server with the relevant security modes.

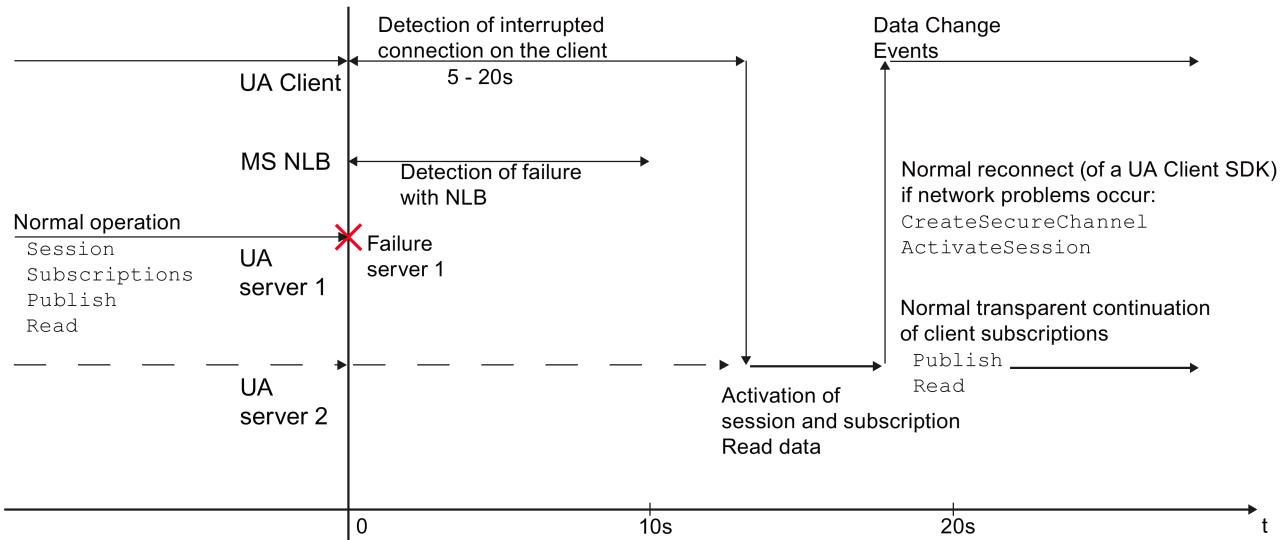
For a redundant, transparent OPC UA connection, select the endpoint with the configured common IP address and the configured server port. In the example "opc.tcp://190.170.1.209:55101".

For a deliberately non-redundant OPC UA connection, the endpoint of the specific S7 OPC UA server can also be selected. In the example "opc.tcp://PC227:55107". This connection can be used to exchange certificates with specific servers or to check the quality of the data of individual servers. Remember that there will be synchronization of the OPC UA sessions and subscriptions used with other redundant servers, however, not for this endpoint.

4.5.4 The OPC UA reconnect procedure

If a redundant OPC UA server fails, the failure must be detected by the Windows "Network Load Balancing" feature of the server operating system and by the OPC UA client. An interruption of the connection is detected based on the occurrence of network errors or a timeout.

The "Network Load Balancing" (NLB) requires approximately 10 seconds to detect the failure of OPC UA server 1. If there is a failure, the NLB directs new calls of the client to the available, redundant OPC UA server 2.



The detection time on the OPC UA client depends on the type of connection monitoring and the set monitoring cycles since these can have a direct influence on the detection time. OPC UA clients can monitor connections in two ways:

- Cyclic reading of the server status with a short timeout.

The detection time depends on the following components:

- The cycle time for the read job (for example 5 seconds).
- The timeout for the read job (for example 2 seconds).
- How often the read job is repeated in the case of an error until the connection is indicated as being disrupted (for example 1x).

- Monitoring the keepalive value of the subscription.

The detection time depends on the following components:

- The keepalive time (for example 5 seconds).
- The additional waiting time until the error message (for example 5 seconds).

After the connection error has been detected (network error, timeout), the normal OPC UA reconnect procedure of the client starts. This reconnect functionality is generally integrated in a client UA SDK and is transparent for the OPC UA client. During the reconnect procedure, the following occurs:

- A new "SecureChannel" is created with "CreateSecureChannel".

There is then a TCP/IP connect to the redundant IP address of the NLB. The NLB distributes the TCP connections evenly. Since the OPC UA server 1 has failed and cannot be reached, the TCP connection is established to OPC UA server 2.

- The existing session is activated by ActivateSession with the assignment of the SecureChannel.

The OPC UA client does not recognize that it is connected to a different server because the session and subscription IDs remain valid. The sessions and existing subscriptions are therefore activated again.

The OPC UA client can now continue its normal functionality transparently. For example monitoring with subscriptions that are available again or also simple reading and writing.

The OPC UA client sends a PublishRequests to the server to obtain new data and events. OPC UA server 2 is aware of the session and all the subscriptions and monitored items belonging to it are activated. The data is read using the SIMATIC NET communications protocol. A RefreshRequiredEvent is also triggered on the client if several event monitored items are created. As soon as data or events exist, they are sent to the client with a PublishResponse.

4.5.5 Redundancy support by OPC UA clients

Introduction

The properties required by an OPC UA client for the "optimum redundancy failover" are evaluated as conforming behavior in terms of the OPC UA standard.

The essential characteristics are as follows:

- Detection of a connection abort or connection monitoring
- Reconnect functions
- Refresh Required event

Detection of a connection abort

To detect a connection abort, the OPC UA client must be capable of connection monitoring.

The connection can, for example, be monitored as follows:

- Monitoring the status of the "Read" method
- Monitoring the keepalive value of the subscription

Once a connection abort has been detected, the OPC UA client should follow a reconnect logic.

Reconnect functions

With the reconnect, the OPC UA client uses the OPC UA option of continuing to use existing sessions and subscriptions.

To do this, the following sequence must be run through. The sequence is different from the initializing connect procedure.

1. CreateSecureChannel
2. ActivateSession
 - If there are errors in AcitvateSession
 - CreateSession
 - ActivateSession
 - create a new subscription
3. Wait for data changes / events or check whether the subscription still exists.
 - If there are subscription errors
 - create a new subscription

If the OPC UA client performs the reconnect procedure in the same way as the connect procedure, the only advantage is that the client can connect again to the same server endpoint.

RefreshRequiredEvent

The OPC UA client must react to the RefreshRequiredEvent of the server. To do this, the OPC UA client runs a suitable refresh. This sequence is described in the "OPC UA Part 9 - Alarms and Condition" specification.

Note

Check the OPC UA client you were using to make sure its behavior is in conformity with the standard. Clients are often based on SDKs that already include such mechanisms.

4.5.6

When does a server failover occur?

If a server fails, there is a failover if the OPC UA client detects a failure of the channel, session or subscription. For example when:

- Failure of the host PC
- OPC UA server crash
- Operating system crash

A failover does not occur when:

- S7 connection interruptions or other error states occur that prevent the OPC UA server from receiving S7 data from the controller.
- Error states occur on the OPC UA server, that do not cause the failure of channel, session or subscription or this failure is not detected by the OPC UA client.

4.5.7 Details of the server failover

Possible error messages to the OPC UA client

If write or read jobs are active when server 1 fails, errors are signaled for these jobs by the stack of the OPC UA client. These jobs must then be repeated by the client application.

Possible error messages to the user of the OPC UA client

Normally, OPC UA clients set the quality of the variables internally to "BAD" if the connection to the server aborts. Following the reconnect, the quality is reset again by the new data.

If there is a connection abort, the quality of the data, in other words "BAD", can be displayed to the user. This will only be for a few seconds or, depending on the client, it may not occur at all. The reason for this is that the quality of the data is only set to "BAD" after detecting the connection abort and the Reconnect is started automatically immediately after.

No synchronization of data and events

Data and events exchanged between the client and the PLC connected using the S7 communications protocol are not synchronized between the redundant OPC UA servers.

Each server in the redundant set processes this information only via its own S7 connection. Following a redundancy failover, the new server receives this information again from the PLC.

If a subscription is active following the failover, the following occurs:

- For the data type "Data Change Monitored Item", the current values are obtained using the SIMATIC NET communications protocol and returned as soon as they have been read.
- For the data type "Event Monitored Item", a RefreshRequiredEvent is sent to the client to trigger a refresh by the client. This sends the current status of the active alarms to the client with a new "EventId" known on the new server. A UA alarm client can assign the events to the known conditions based on the identical "ConditionId". It does, however, require an "EventId" known on the server to be able to take actions (example to acknowledge alarms). If there is a brief interruption and reconnection to the same server, no "RefreshRequiredEvent" is sent to the client.

Restrictions to the server failover

The following local data and statuses on the server are not synchronized:

- @Local server data that is written or read by the client or via remote access (S7 station to PC station, S7 connection) using PUT and GET function blocks.
- Local DEMO variables.
- Local buffers (including sections of them) of block services "BSEND/BRCV" since the block services are exchanged explicitly between the PLC and one of the redundant OPC servers.

- Local specifically configured connection parameters (baud rates, PDU size, timeouts) resulting from a differing configuration.
Differing configurations can result as follows:
 - Different projects or work statuses were downloaded to the two servers.
 - During runtime, different parameters were negotiated between the communications partners.
- Local specifically configured symbols (S7 symbol names with S7 connections, deadbands,...) resulting from differing configurations.
- Local alarm states, history and acknowledgements. This applies, for example, to locally generated Statepath alarms.
- Locally accepted or rejected OPC UA client certificates of the local OPC UA server in the redundant set.

Use the certificate management in the "Communication Settings" to accept the OPC UA client certificate on all OPC UA servers involved. This ensures that there is acceptance on the server if there is a failover. As of "SIMATIC NET PC software V8.2", OPC UA client certificates are automatically synchronized on redundant servers.

- Local failure of subcomponents and local error states of the SIMATIC NET communications protocol.

This data must be managed by the client and reinitialized or requested if a connection interruption is detected. This only works if the server namespaces were configured consistently and identically during the configuration of the cluster.

Points to note about S7 alarms

- Diagnostics alarms (simple events) are not forwarded between server and client if there is an interrupted connection. This also applies when there is a failover. The corresponding events are lost.
- S7 alarms. If an S7 alarm changes on the PLC, the change is sent separately to both OPC UA servers. The S7 alarm can arrive at different times at the OPC UA servers. The time difference can, for example, be caused by a lot of alarms or by the configuration being too large and therefore can be of the order of the failover time. If this is the case, the following can occur:
 - Prior to the server failure, the alarm status was changed from inactive to active on the OPC UA client. After the redundancy failover, a Refresh is triggered by the second OPC UA server. The alarm status is initially signaled with the "Inactive" alarm status and then with the "Active" alarm status. This also applies for the reverse situation.
 - Prior to the server failure, an alarm was acknowledged on the OPC UA client. After the redundancy failover, a Refresh is triggered by the second OPC UA server. The alarm is initially signaled with the status "Unacknowledged" and then with the "Acknowledged" status.

Special features when reading/monitoring BSEND/BRECEIVE variables

If the OPC UA server already has older values for a monitored BSEND/BRECEIVE item (variable) in the cache, when a subscription is activated, first the value from the cache including quality and time stamp is returned to the client before a more up-to-date value is obtained via the SIMATIC NET communications protocol and returned with a new quality and time stamp. The value from the cache may be older than a value that was returned by

another server of the redundant set to the client prior to the failover. The "Good" status is normally signaled. When reading/monitoring, the client must therefore note the corresponding time stamp.

4.5.8 Reaction of the SIMATIC NET OPC UA S7 A&C to a redundancy failover

Prior to the redundancy failover, OPC UA S7 server A sends events to the client if a change occurs in a condition. During the failover time, no events are sent to the client. After the failover, OPC UA S7 server B sends events to the client.

If following the failover, OPC UA S7 server B is active, OPC UA S7 server B sends a request to the client to perform a refresh for every event-monitored item using the RefreshRequired event (Nodeld i=2789). This refresh request ensures that the client is informed of the current states of the conditions.

The S7 interrupt blocks are mapped to OPC UA alarm conditions, or simply alarms, with corresponding alarm states.

Both OPC UA S7 servers (server A and server B) are connected to the S7 PLC so that both receive the same messages. Message means signaling the status change of an S7 interrupt block. Although both OPC UA S7 servers receive the messages, these two computers cannot be considered synchronized. The S7 PLC signals all connected OPC UA S7 servers and other operator control and monitoring stations one after the other in an unpredictable order and with an unpredictable execution time. If there are 2000 messages, it is possible that the same message is sent by the S7 PLC first to OPC UA S7 server A and then to OPC UA S7 server B 10 to 20 seconds later.

A change event for an OPC UA alarm has 4 relevant time stamps: ReceiveTimeClient (RTC time at which the event is received by the client), ReceiveTime (RT time at which the message is received by the server), Time (modified time stamp of the source - modified time of the triggering edge on the S7 interrupt block) and S7Time (time stamp of the source time of the triggering edge on the S7 interrupt block). The following information relates only to ReceiveTime (RT) and S7Time.

The ReceiveTime RT is always at least slightly different for the same message to both OPC UA S7 servers A and B due to the unpredictable execution times and the time being set slightly differently on computers A and B (which is normally the case).

The S7Time is set by the S7 PLC and is always identical on both OPC UA S7 servers for the same edge change.

By changing over an event subscription from OPC UA S7 server A to B, the following 4 situations can occur:

- A message is sent by the S7 PLC to both OPC UA S7 servers during the failover and does not trigger an event on the client during this time. The client is informed of the new alarm state following the failover. RT shows the time stamp of the OPC UA S7 server B.
- A message is sent by the S7 PLC to OPC UA S7 server A shortly before the failover and this triggers an event on the client with the server RT time stamp with the new alarm state. OPC UA S7 server B receives the same message from the S7 PLC at some other point in time (possibly even before A), however before completion of the failover so that OPC UA S7 server B does not trigger an event on the client. The refresh signals the

same alarm state with the identical S7Time and other identical properties, however with a different ReceiveTime RT. If the events are logged (relevant system events) for later examination, the alarm state that was received later on computer A or computer B must be discarded. Assuming time synchronization of the two computers, this is the computer with the more recent (higher) RT.

- A message is sent by the S7 PLC to OPC UA S7 server A shortly before the failover and this triggers an event on the client with the server RT time stamp with the new alarm state. OPC UA S7 server B receives the same message from the S7 PLC later following the failover, however before the refresh is begun. The refresh procedure is uninterruptible, in other words OPC UA S7 server B can send the corresponding event before the refresh or immediately afterwards. The same alarm state with the identical S7Time is signaled a total of three times in this case. The two events with the later (higher) RT originate from OPC UA S7 server B and need to be discarded for logging.
- The S7 PLC sends the message to OPC UA S7 server A shortly before the failover and still triggers an event with the new alarm state on the client with this server's RT time stamp, OPC UA S7 server B receives the same message from the S7 PLC later after the refresh has begun. The refresh therefore signals an out-of-date alarm state with an out-of-date S7Time. This is detected by the client and is therefore discarded. OPC UA S7 server B later triggers a further event for the identical alarm state as OPC UA S7 server A that can be discarded for the logging.

If the S7 connection supplying the messages to one of the computers is lost briefly, the OPC UA S7 server will establish this connection again. After successful establishment of the connection, the S7 PLCs are queried regarding the status of the S7 interrupt blocks. The information supplied by the S7 PLC is incomplete, in other words it does not include the additional values and S7Time. If alarms are incomplete, S7Time adopts the value 01.01.1990 00:00:01. Since this date can only occur with these incomplete alarms, it is advisable to exclude the incomplete alarms if there are several events for the same alarm state.

4.5.9

General recommendations and notes on redundant OPC UA S7 communication

The PC stations are redundant; the partner S7 station is generally single. This results in higher availability but requires more configuration effort. Note the following recommendations and points:

- Block-oriented services "BSEND/BRCV" and the S7 connection itself include address parameters for the specific PC station and the OPC UA server.

For the SIMATIC S7 station, configure a separate function block call "BSEND/BRCV" for each S7 connection. The use of permanent S7 connections is recommended. The PLC program must execute the function block calls for both S7 connections simultaneously. The PLC program is therefore more complicated.

Generally, the function block call to the passive redundant side is acknowledged negatively or needs to be aborted.

You should also note that the data on the PC stations only exists locally. There is no synchronization of the data sent between the redundant OPC UA servers on the PC stations when there is a failover.

Even if the function block call to one side was successful, data can be lost on the OPC UA client due to the redundancy failover.

- For each S7 connection of the SIMATIC S7 station, you need to program the S7 communications services "PUT/GET".

Each function block call sees the data areas of the corresponding OPC UA server. You should also note that the data on the PC stations only exists locally. There is no synchronization of the data sent between the redundant OPC UA servers on the PC stations.

- For the OPC UA client, the connection to an OPC UA server of the redundant set is transparent. The connection is distributed over the redundant servers by the NLB according to load balancing criteria. If a server fails, the reconnect connects to the redundant server. It is advisable to try out this failure test deliberately in advance by removing the network cable from the active OPC UA server. You should also test the quality of the S7 communication (variables) to both redundant servers.

To achieve a faster reconnection to the S7 variables of the redundant server if there is a server failure, we recommend that you configure active and permanently established S7 connections.

- OPC UA S7 redundant servers have no automatic function for synchronizing the time of day. To make sure that there are no unwanted time jumps when there is failover, synchronize the times of day of the various redundant servers manually. You require a configuration concept for this.
- If the ServerState is "Running" for all servers in the redundant server array, this does not necessarily mean that the entire redundancy set is operating free of errors. It simply means that the connected OPC UA server is in the "Running" status and that it can communicate with the partner servers. Another OPC server of the redundancy set could indicate the ServerState of its partner with "Unknown" because it cannot communicate with it.

The ServiceLevel of an OPC UA server with ServerState "Running" is 255 if it can reach its partner server, otherwise it is 127. The error-free status is indicated by the value "255" and "Running" for both servers in the redundant server array.

- When enumerating the endpoints of a redundant OPC UA server, the endpoints of its own address and those of the common address of the redundant pair are enumerated. Depending on the configuration of the network, the endpoints of the redundant partner can also be enumerated.
- When loading a redundancy configuration, an existing local endpoint is given port 14845 if its previous port is identical to the port of the configured redundant endpoint. When the redundancy configuration is deleted, endpoints with the port 14845 are again given the port they had prior to the redundancy configuration.
- Note that the results of the method calls for block services (e.g. password(), blockread()) are not available to the current server following the redundancy failover. The method calls must be reconfigured and called again for the current server.
- Special features when reading/monitoring status variables of the server:
Status variables are server-specific and can therefore have different values on each server at the same point in time. A typical example of this is the "CurrentServerId" that is, by definition, different for each server. Their time stamps do not change even during runtime of the server.

4.5.10 Reading out the status of the cluster by the OPC UA client

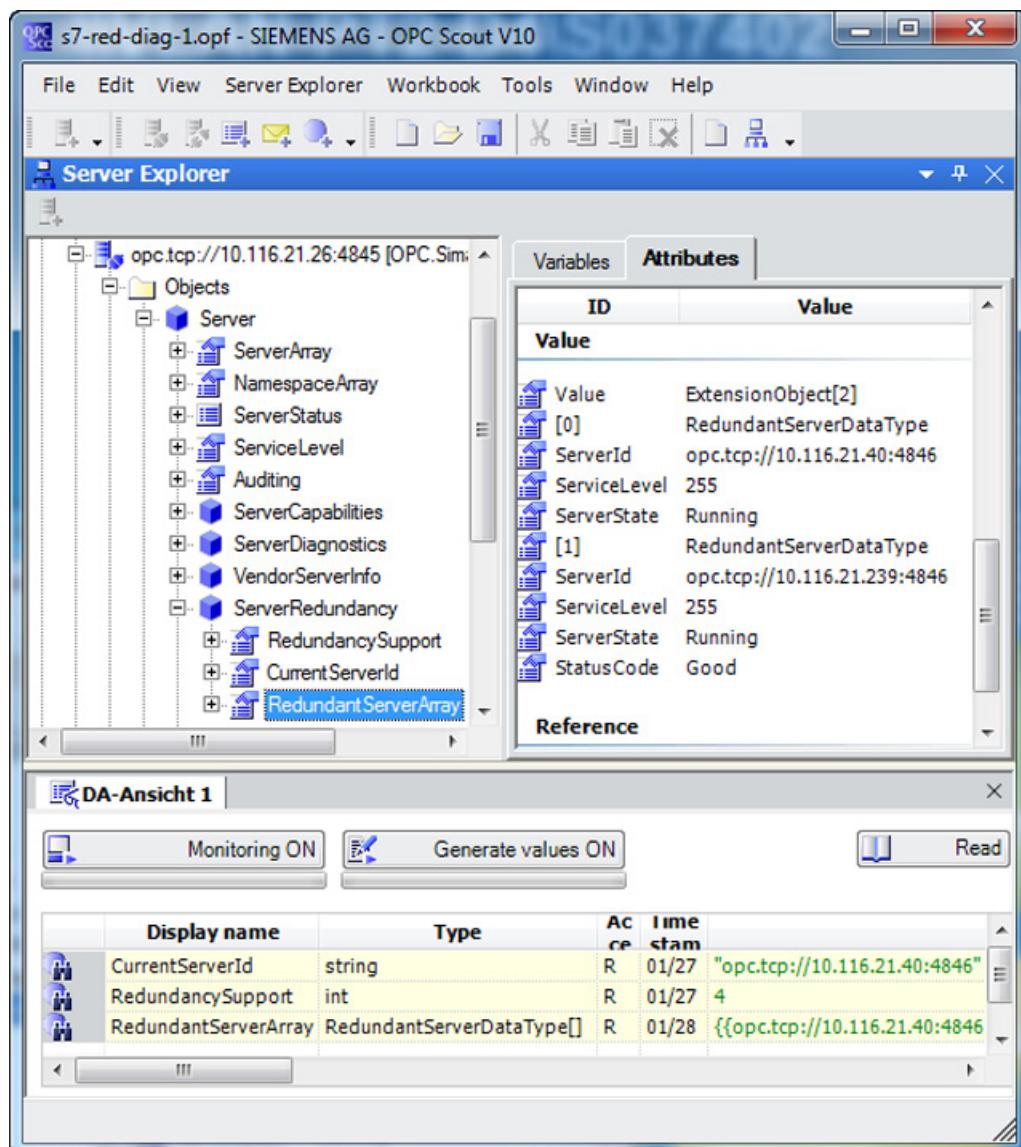
Using diagnostics variables, you can find out the type of redundancy support or the connection status between the OPC UA client and OPC UA server. The following diagnostics variables are made available by an OPC UA server of the cluster. The diagnostics variables can be displayed using the standardized server object in the namespace index "0" below the server node in the "Network Load Balancing Manager".

Under the server node, you will see the object "ServerRedundancy". This is an object of the type ServerRedundancyType with the property RedundancySupport "". With the help of this object, you can read out the type of redundancy support. The following values can be read out using the data type "enum_OpcUa_RedundancySupport":

- OpcUa_RedundancySupport_None = 0,
- OpcUa_RedundancySupport_Cold = 1,
- OpcUa_RedundancySupport_Warm = 2,
- OpcUa_RedundancySupport_Hot = 3,
- OpcUa_RedundancySupport_Transparent = 4

If transparent redundancy is supported, this object is of the type TransparentRedundancyType. This object type defines the additional elements CurrentServerId and "String". Although all servers have the same URI for the connection in transparent mode, the server of the cluster currently being used can be identified with the "CurrentServerId" identifier. Note that the value is only valid within a session.

If there is a server failure or failure of several sessions, different servers of the cluster can take over its tasks. The client can nevertheless identify the specific data source. The list of all servers in the cluster is contained in the "RedundantServerArray" of the type "RedundantServerDataType". This server list contains a variety of server-specific information or values (ServerId, ServiceLevel, ServerState). These values can change within a session.



Among other things, the "RedundantServerDataType" structure contains the following elements:

- ServerId

ID of a server. A string containing the unique identifier for the server in a cluster. This cannot be the server URI, because this has to be identical for all servers. The structure of the string is analogous to that of the server URI. As the section for the local PC, however, the unique IP address is used that is also used for server-to-server communication.

- ServiceLevel

A byte that displays how good the data transmission between server and client is. "0" (\triangleq "UNKNOWN") indicates the worst and "255" (\triangleq "RUNNING") the best possible status. This value shows the status of the availability of redundant servers.

- ServerState (enum OpcUa_ServerState)

can adopt the following values where the "UNKNOWN" status means that the server is not obtainable:

- RUNNING (OpcUa_ServerState_Running = 0)
- FAILED (OpcUa_ServerState_Failed = 1)
- NO_CONFIGURATION (OpcUa_ServerState_NoConfiguration =2)
- SUSPENDED (OpcUa_ServerState_Suspended = 3)
- SHUTDOWN (OpcUa_ServerState_Shutdown = 4)
- TEST (OpcUa_ServerState_Test = 5)
- COMMUNICATION_FAULT (OpcUa_ServerState_CommunicationFault =6)
- UNKNOWN (OpcUa_ServerState_Unknown =7)

Note

Note that you cannot switch over the active server using the client diagnostics variables for redundancy.

SIMATIC Computing

With the "SIMATIC Computing" software as part of the SIMATIC NET PC software, you can create simple applications with which you can access process data without a lot of programming effort. The process data can be read and written over any SIMATIC NET communication system.

In "SIMATIC Computing", there are various .NET assemblies available. You can configure and interconnect these controls to suit your task.

OPC data controls for .NET

For the .NET interfaces, there is a SIMATIC NET OPC data control available with simple graphic operator control. Simple applications can be created with the Microsoft Visual Studio. Here, process variables in the namespace of the SIMATIC NET OPC servers can be found and linked to user interface elements, for example a text box. These functionalities are available without writing any program code. You will find more information on this topic in the section "Auto-Hotspot".

For more extensive applications, there is also a programming interface available that nevertheless further simplifies the OPC interfaces.

Compatibility with OPC

The .NET Assemblies and ActiveX controls from SIMATIC NET use the OPC interface directly.

Applications created using the controls of SIMATIC NET can only be operated with OPC servers of the SIMATIC NET PC software.



CAUTION

When you interconnect a process variable via the controls, you establish access to process data. If you change the values of a process variable in a control, the values in the process can change as a direct result.

Changing process data can trigger unexpected reactions in the process that can lead to serious injury to persons or damage to equipment.

With this in mind, a degree of caution is necessary. You should, for example, restrict access rights for process variables. Install a physical emergency stop circuit for the machine or process.

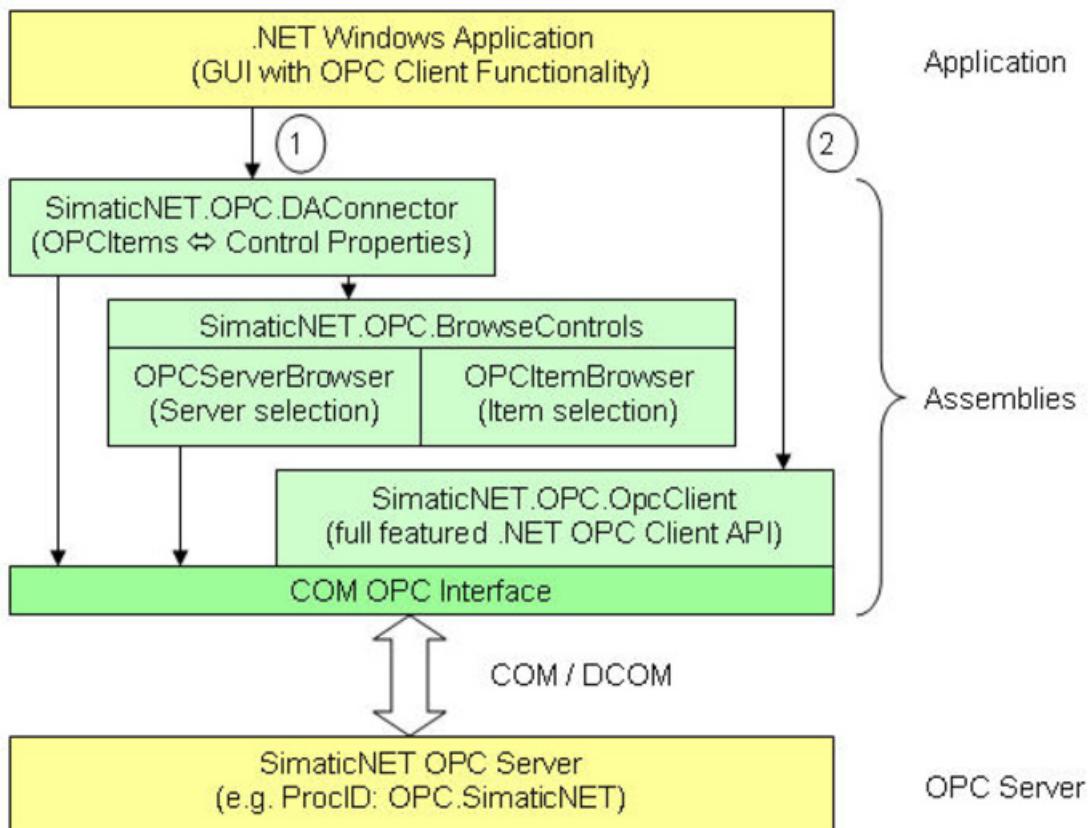
5.1 .NET OPC client control

5.1.1 Overview

Background

The Microsoft Visual Studio development environment supports the linking of additional controls that can be used in the .NET programming languages. With the SimaticNET .NET OPC client control, Siemens provides a simple option of interconnecting data points from an OPC server with standard controls from the MS Visual Studio, such as text boxes, labels or buttons. This allows simple OPC client applications to be created without needing to write a single line of code. This access path is labeled "1" in the overview.

Users who want to write more complex applications or want to process large amounts of data from OPC servers should use the SimaticNET .NET OPC Client API interface. This powerful data interface is based on a C++ library but simplifies access via .NET. This access path is labeled "2" in the overview.



Characteristics of .NET OPC client controls

The .NET Controls and the underlying C++ library have the following characteristics:

- The user does not require any detailed knowledge of the various OPC Data Access interfaces.
- The component completely hides the various basic technologies of OPC such as COM, DCOM, Web services, SOAP and XML.
- The component completely hides the handling of the connection to an OPC server with connection establishment and repeated connection establishment if an error occurs.
- The development of OPC client applications with C-Sharp .NET and Visual Basic .NET is possible with the SimaticNET .NET OPC client controls without additional programming.
- The conversion of the OPC data from various OPC Data Access interfaces to .NET data types.
- Fast and simple discovery of OPC COM servers locally and on the network.
- Graphically supported browsing of OPC items on the selected server.
- High-speed and optimized client-server communication with the implementation of core functionality in C++.
- Support of OPC UA and the certificate management for OPC UA.

Note

Using these controls, it is only possible to establish connections to OPC servers of the SIMATIC NET product family. Connections to other OPC servers or to OPC servers of other vendors are not possible.

5.1.2 Step 1 - Installation

SimaticNET .NET OPC client controls are installed using the setup of the "SIMATIC NET PC Software". The assemblies are installed in the "%ProgramFiles%/Siemens/Simatic.Net/OPC2/OCX.NET" path. The SimaticNET .NET OPC client control consists of the following files:

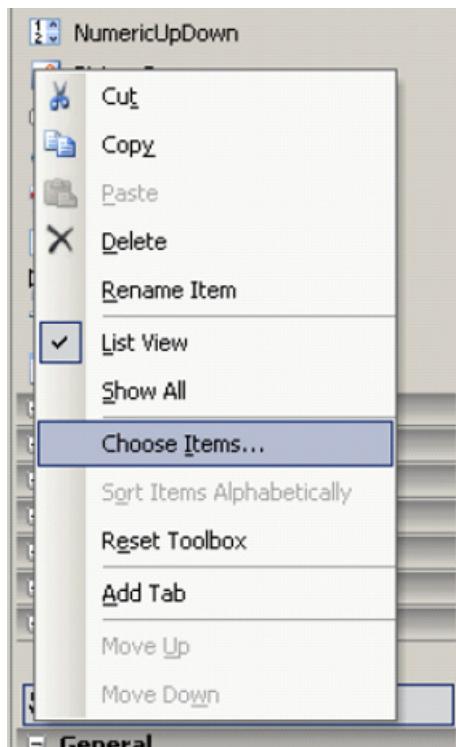
- "SimaticNET.OPC.DAConnector.dll"
- "SimaticNET.OPC.BrowseControls.dll"
- "SimaticNET.OPC.OpcClient.dll"

5.1.2.1 Referencing the controls

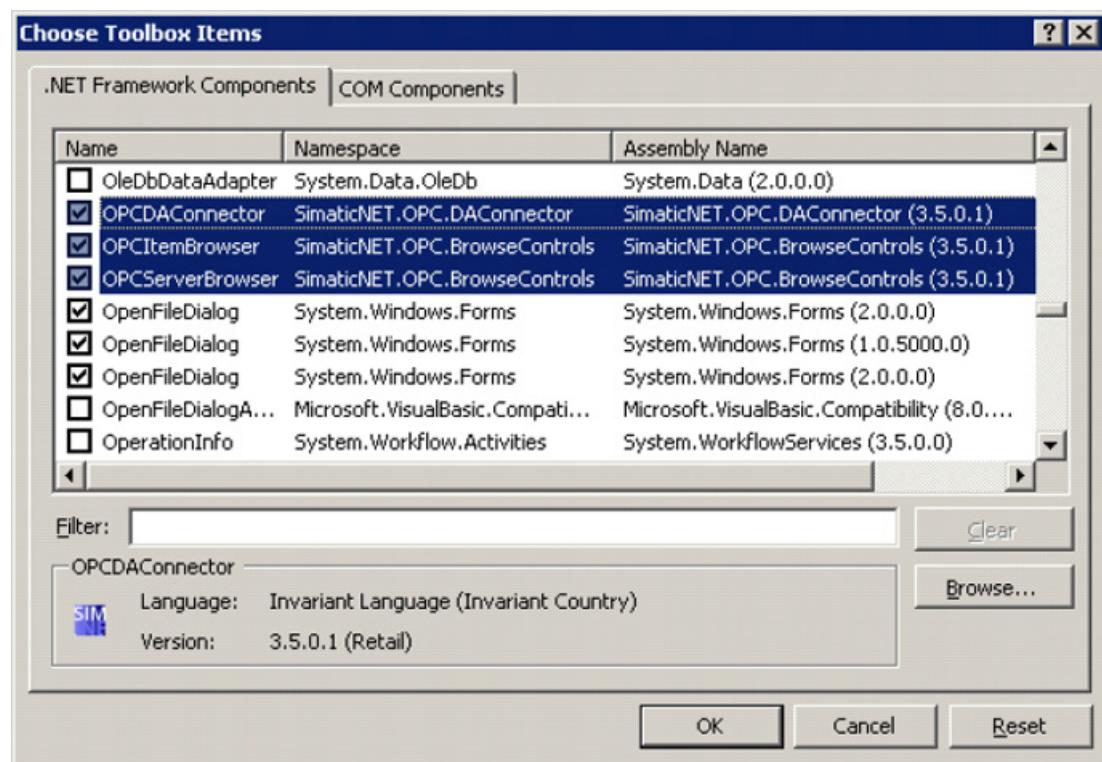
Before using the SimaticNET .NET OPC Client Controls the first time, the controls need to be added to the toolbox manually.

If you right-click in the Toolbox, you will see a submenu with which you can create a new tab with "Add tab" or add new components with "Select elements". In the next dialog box (Choose Toolbox Items), browse to the installation path named above. Add the following files:

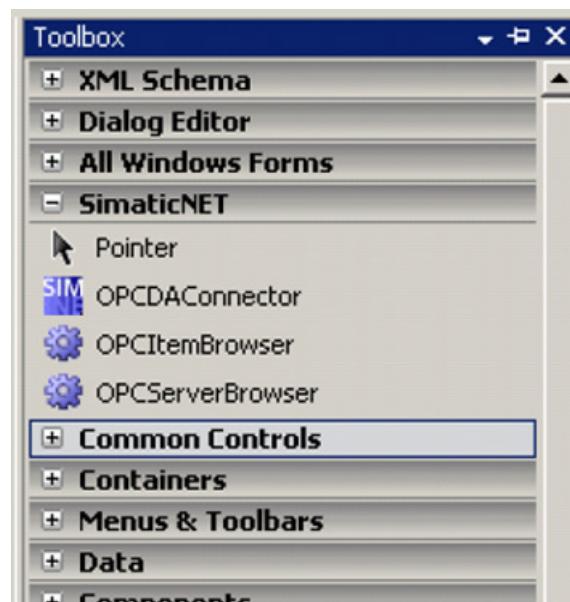
- "OPCDAConnector.dll"
- "OPCItemBrowser.dll"
- "OPCServerBrowser.dll"



⇒



A new tab appears in the Toolbox of the Microsoft Visual Studio that contains the individual components.



Note

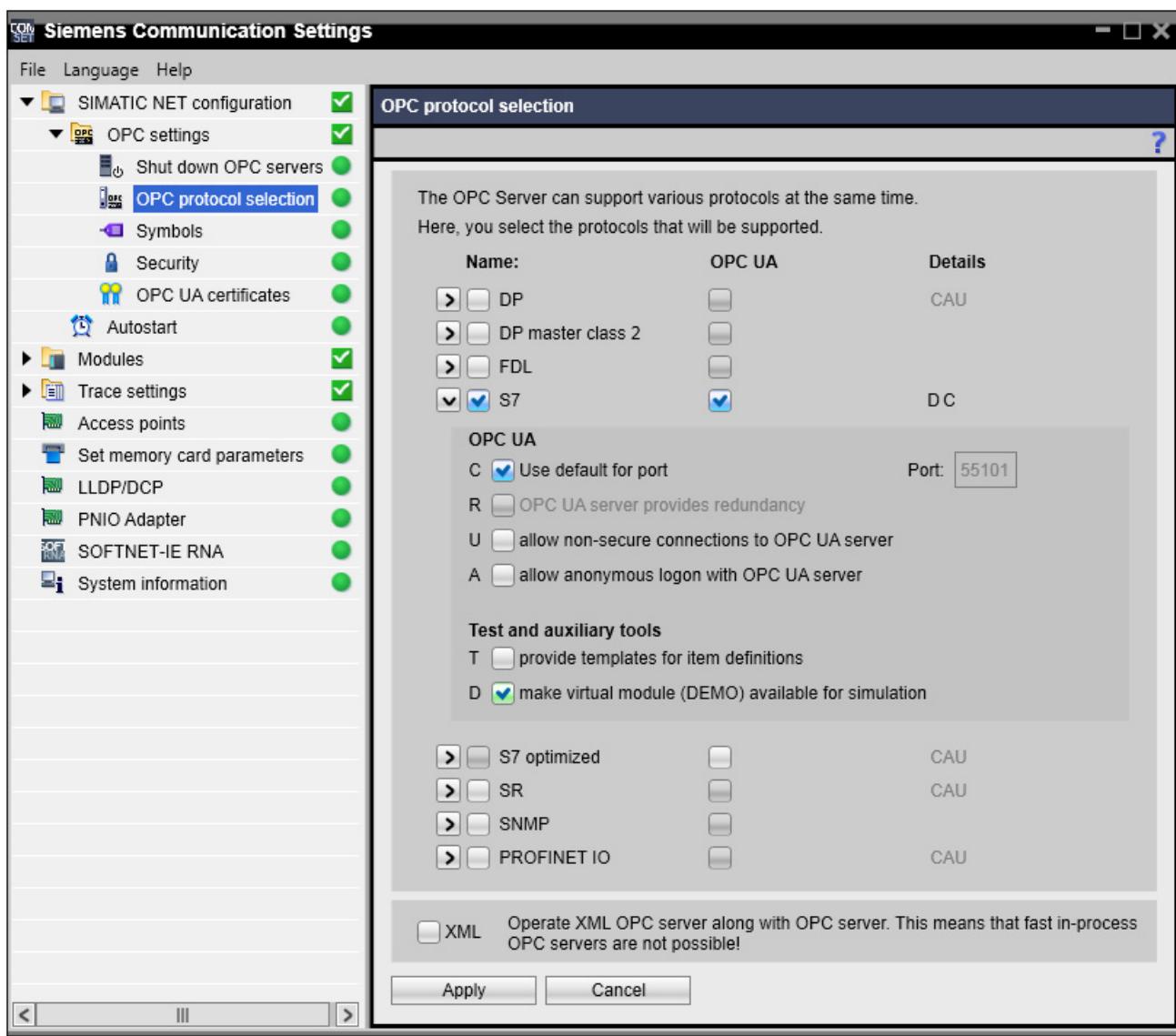
Assemblies that are referenced in MS Visual Studio must be located on a local hard disk and not on a network drive. This is a restriction of the Microsoft .NET development environment.

5.1.2.2 Setting a sample program

Before the current program can be run at all, the simulation connection of the OPC server must be enabled. The sample program uses demo variables of the S7 simulation connection.

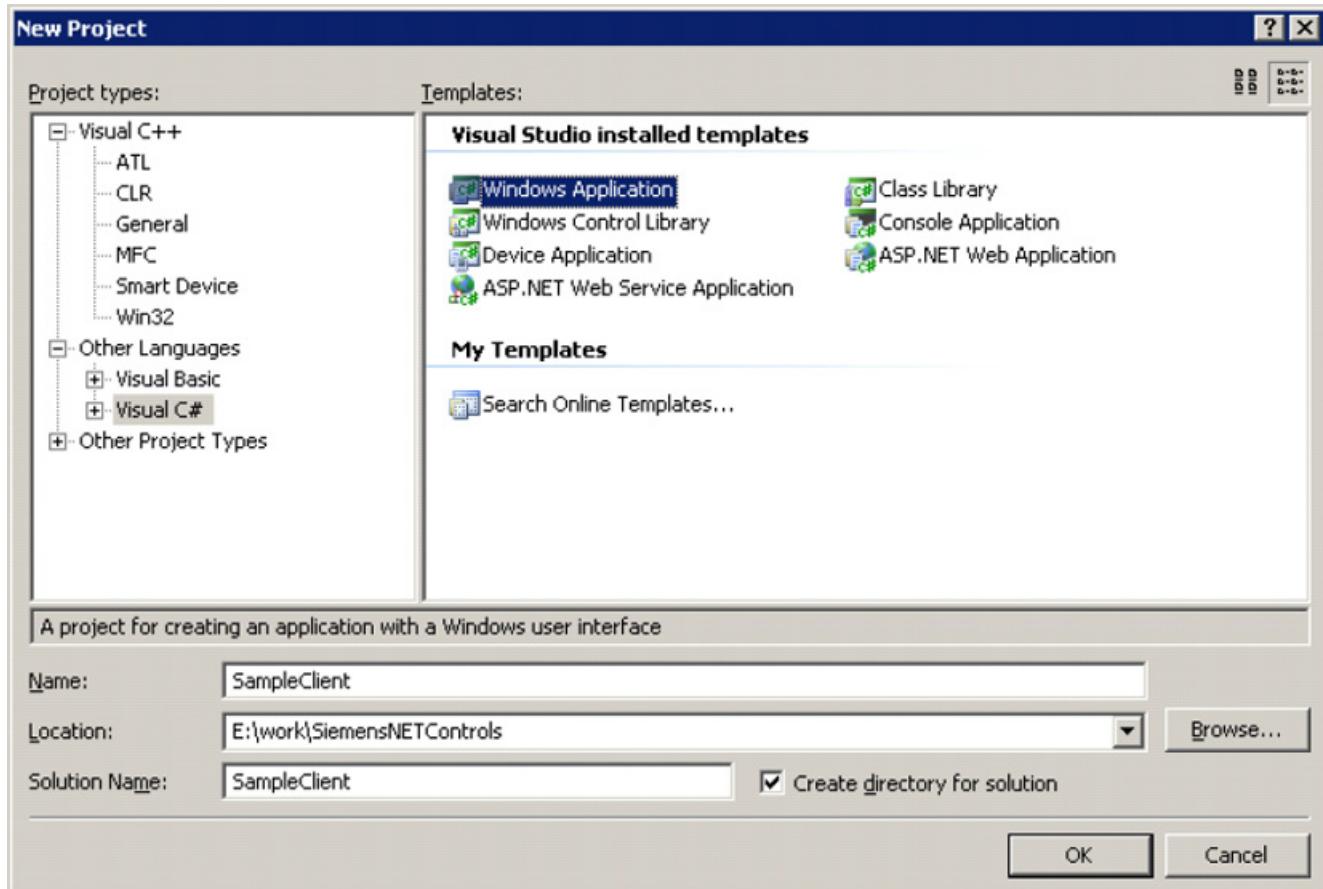
Follow the steps below to select a virtual module (DEMO) for the simulation:

1. Start the "Communication Settings" program from the Start menu:
"Start" > "SIMATIC" > "SIMATIC NET" > "Settings" > "Communication Settings"
2. In the left-hand navigation window, open the properties page for the OPC protocol selection.
3. Select the S7 connection and then click the arrow symbol beside it.
4. In the expanded parameter list, select the D check box "provide virtual module (DEMO) for the simulation".
5. Confirm your selection by clicking the "Apply" button.
6. Close the "Communication Settings" program.

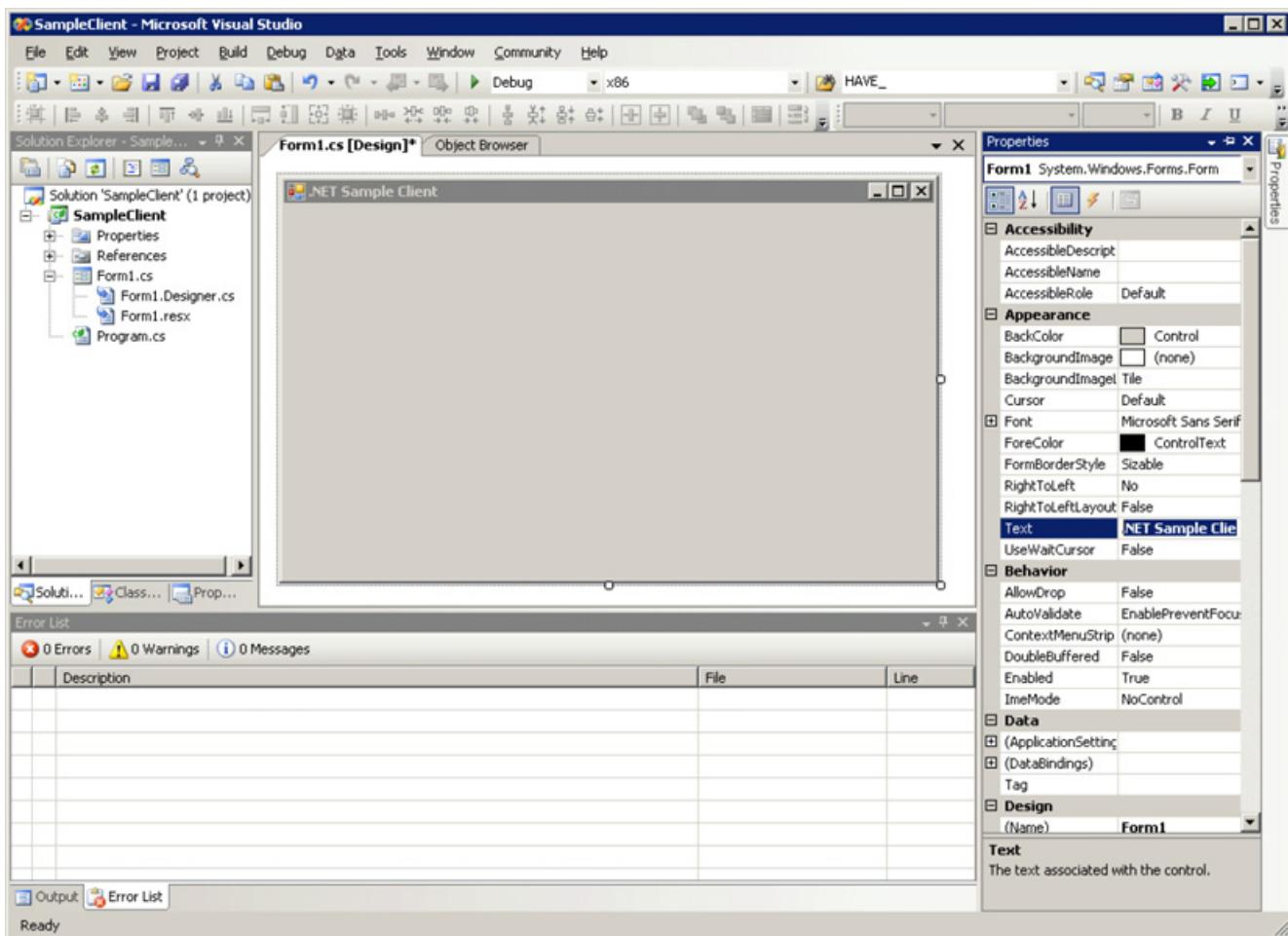


5.1.3 Step 2 - Controls in a WINDOWS FORM

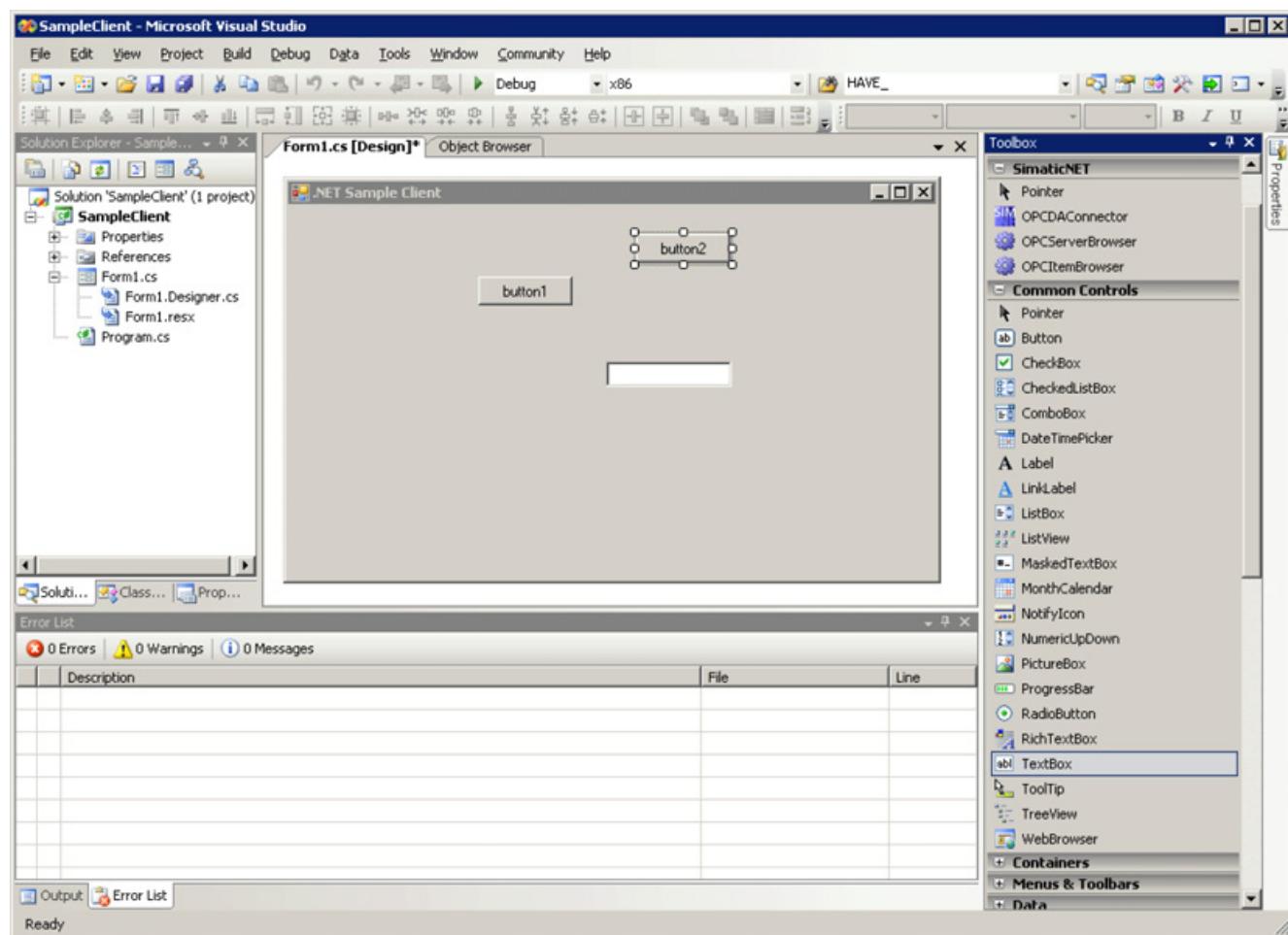
In Microsoft Visual Studio, create a new C# project from the "Windows Application" template type (VB.NET is also possible here).



You will see an empty application window (Form1) whose caption you can rename.

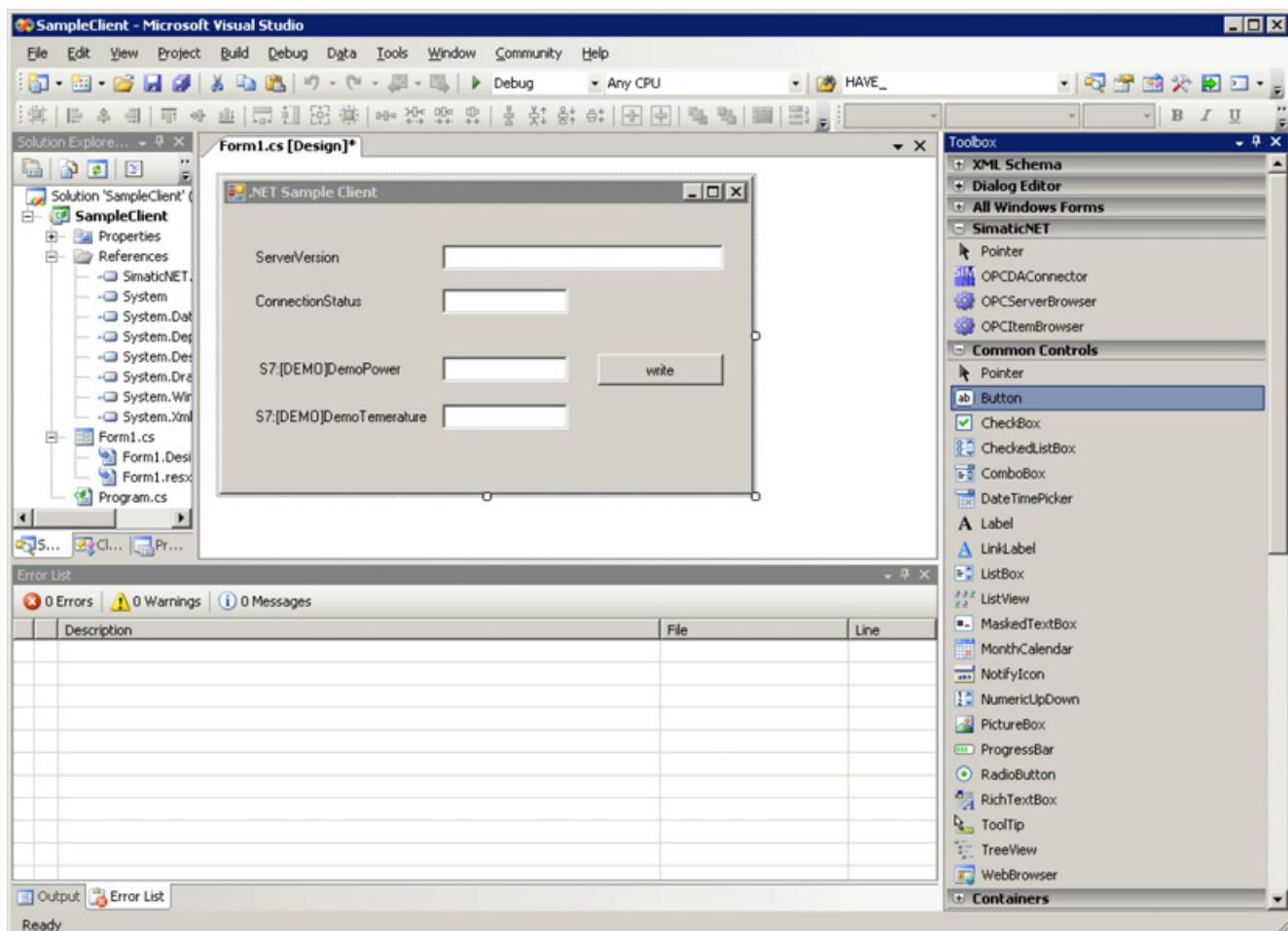


Open the Toolbox and drag the necessary components to the Forms Designer of your sample program.

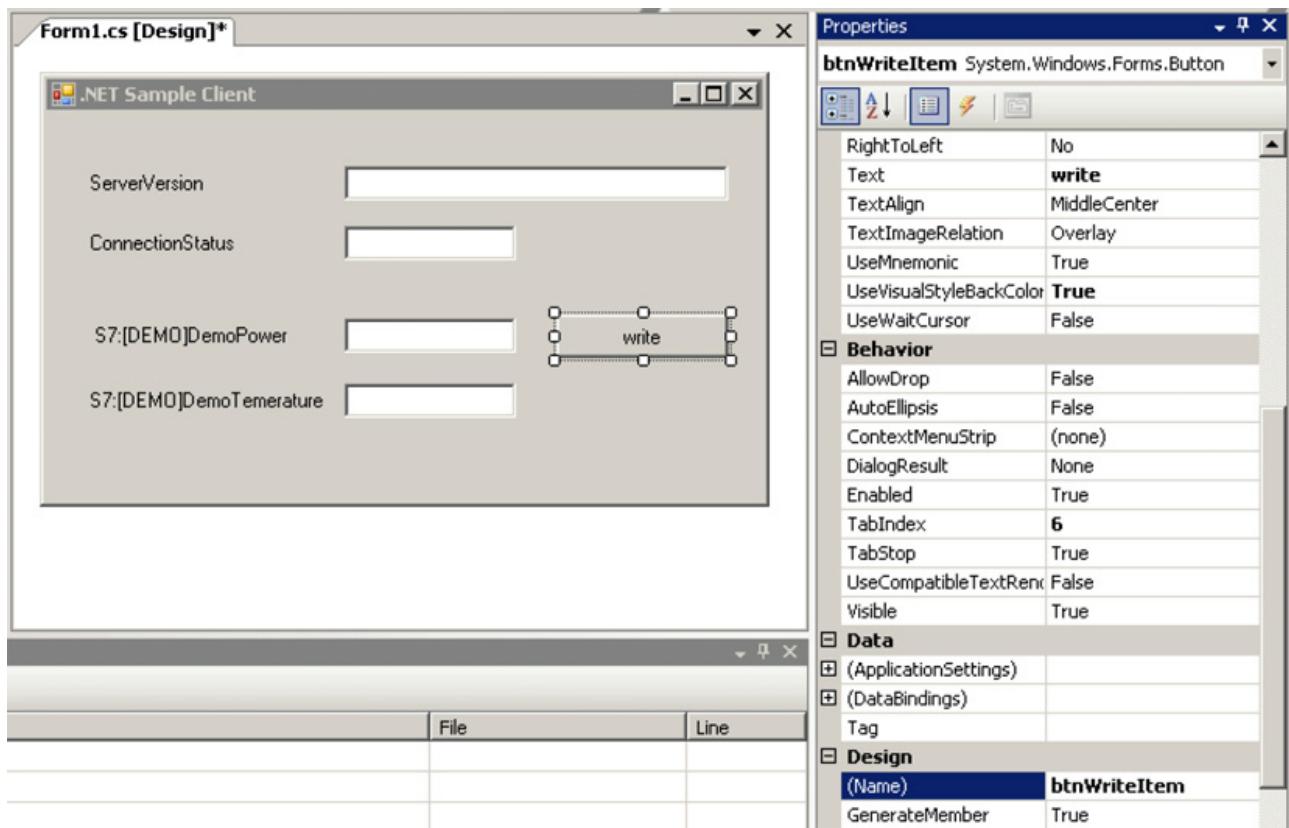


5.1.3.1 Adding Windows controls

Add four labels, four text boxes and a button to the empty form as shown in the figure. You will find these components in the "Common Controls" tab. Position the components as shown in the figure.



Where possible, use the names from the following table for this example because these will be referenced in the later explanation. Open the properties window and select the components whose properties you want to edit.

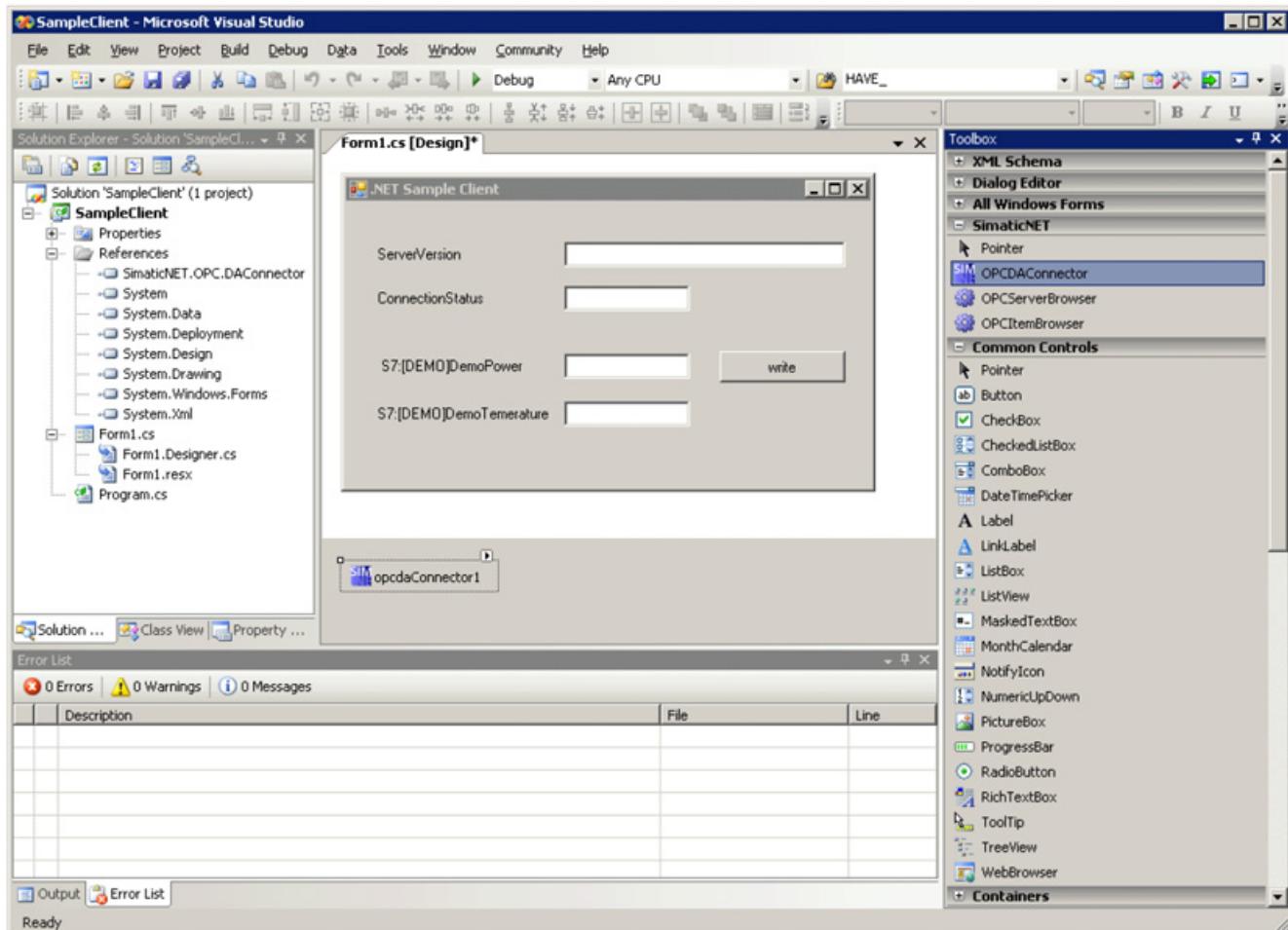


Repeat this for all the components you want to add.

Original name	New property (name)	New property (text)	New property (enabled)
Label1	lblServerVersion	ServerVersion	True
Label2	lblConnectionStatus	ConnectionStatus	True
Label3	lblItemID1	S7:[DEMO]DemoPower	True
Label4	lblItemID2	S7:[DEMO] DemoTemperature	True
TextBox1	txtServerVersion		False
TextBox2	txtConnectionStatus		False
TextBox3	txtDemoPower		True
TextBox4	txtDemoTemperature		True
Button1	btnWriteItem1	write	True

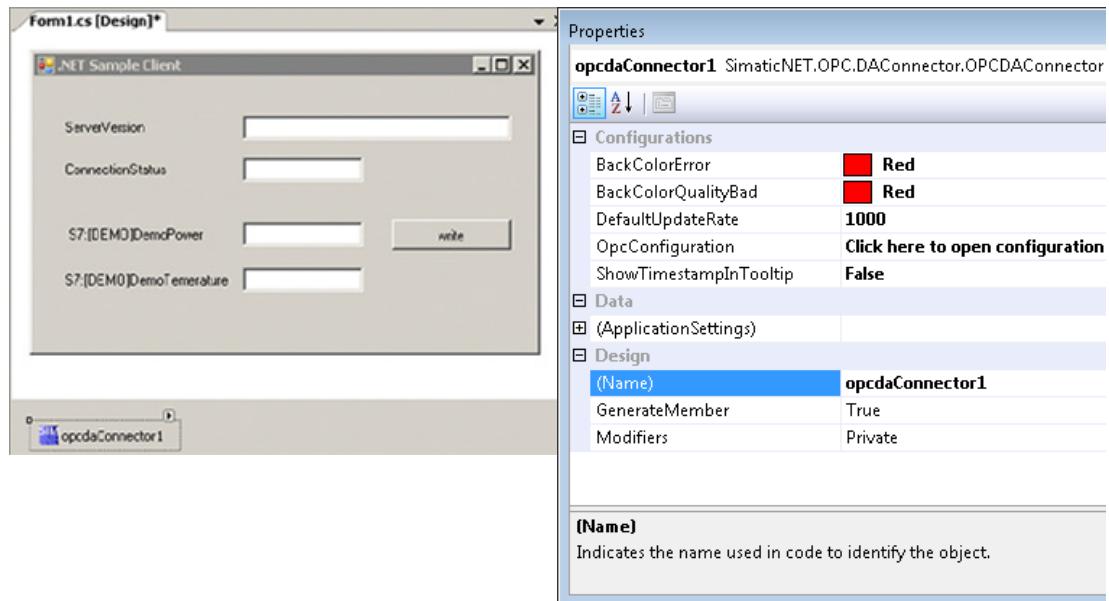
5.1.3.2 Adding the .NET OPC client control

Now take the "OPCDAConnector" control from the Toolbox and drag it to the application window. This step links the entire OPC client functionality into your application.



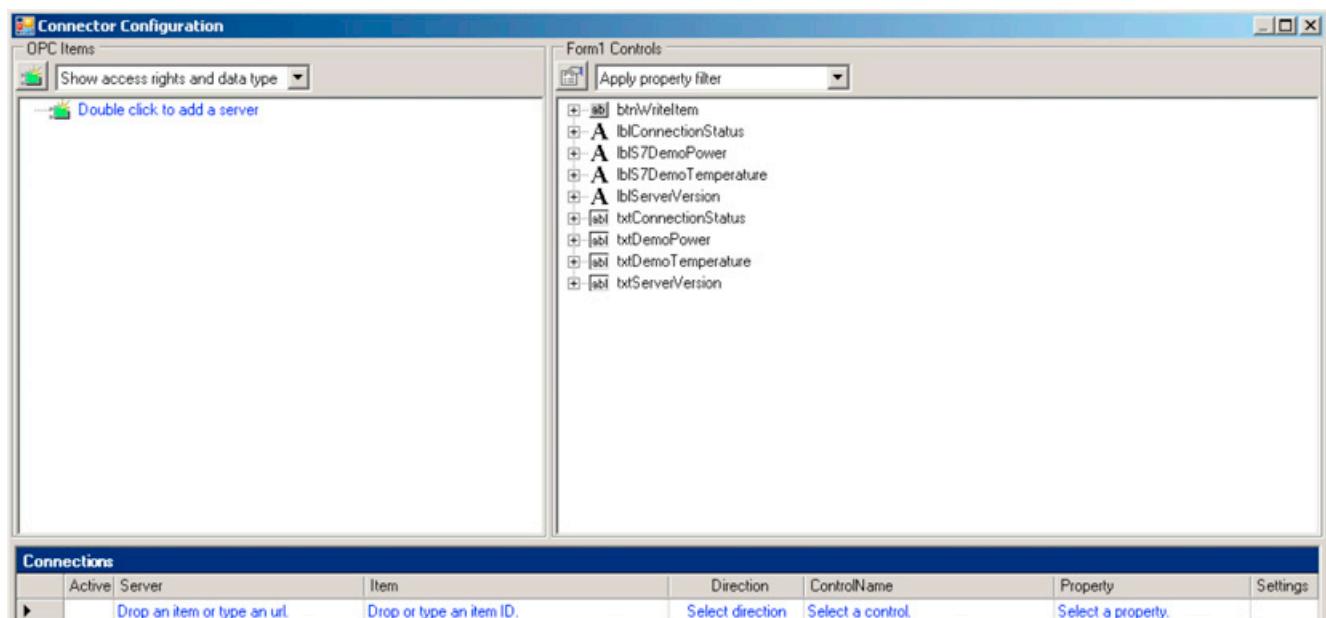
General settings for the OPC client control can be made in the properties window.

- With "Configuration", you open the configuration dialog for linking the OPC items.
- The update time (DefaultUpdateRate) is specified in milliseconds and is passed on to the OPC server to allow it to deliver data changes to the client application.
- The background color (BackColorError) is set for all controls that are linked to OPC items. If an OPC error occurs, the background color of the interconnected control changes to the color set here.
- The background color (BackColorQualityBad) is set for all controls that are linked to OPC items. If the quality of the returned OPC value changes to "Bad", the background color of the interconnected control will change to the color set here.
- The display of the OPC time stamp appears in the tooltip of the linked control and also shows the time (in UTC time) as well as the value.



5.1.4 Step 3 - Configuring data links

To open this dialog, click on the "Configure" box in the Properties window of the OPC client control or double-click the component in the Forms Designer.

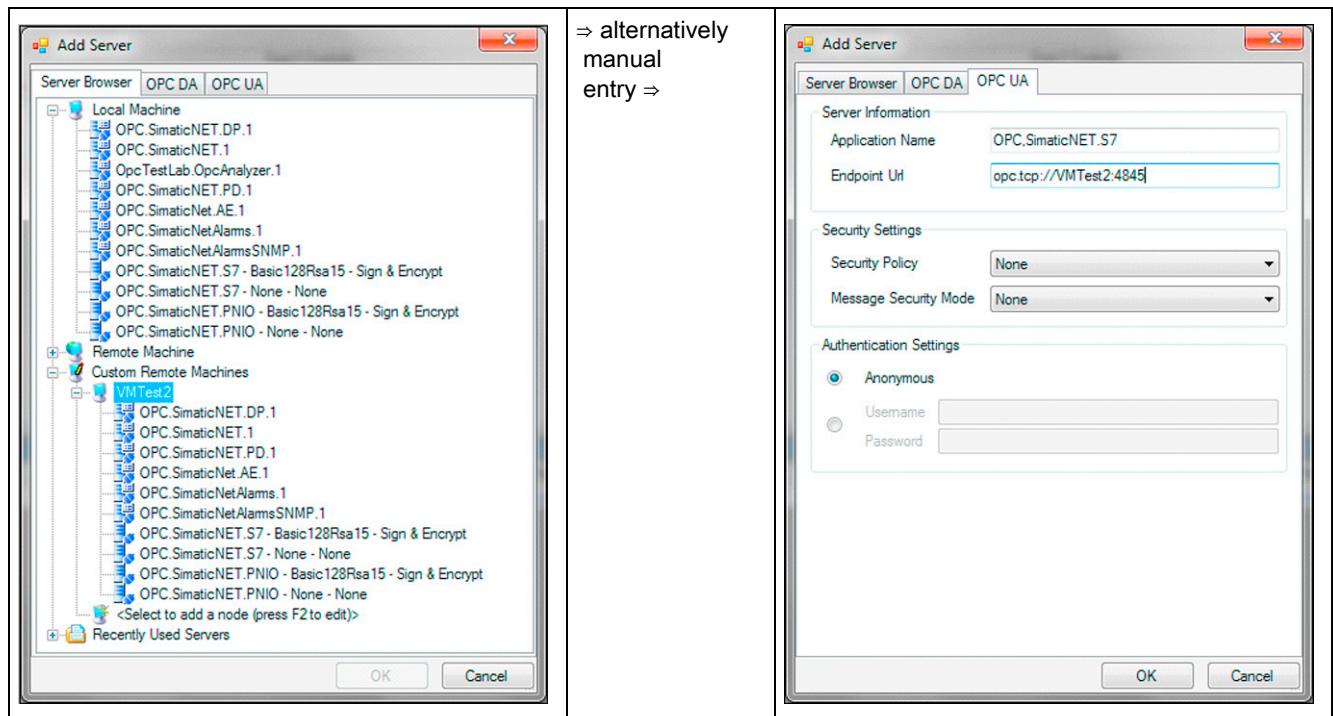


Essentially, the dialog is made up of three parts:

- Top left: The OPC browser window
Here you select the OPC server and then the OPC items to be interconnected
- Top right: The Windows component browser
Here you select the properties of the Windows controls to be interconnected
- Bottom: The list of all links
By dragging one OPC item and one control property to this list they are linked together

5.1.4.1 Browsing OPC items

In the top left window of the OPC client connector dialog, you will see OPC servers and their variables (OPC items). You first need to add an OPC server. You open the dialog for adding OPC servers (Add Server) by clicking the green icon or by double-clicking on the displayed text.



You can select a local OPC server and the later application will then attempt to start this server locally. If you select an OPC server in the network (Remote Machines), the computer name is stored in your application and will also be used later. Make sure that you have adequate DCOM rights to launch such remote OPC servers and to connect to them. In the second tab (OPC DA), you have the option of specifying an OPC server manually (ProgID and host name or IP address).

Note

Using these controls, it is only possible to establish connections to OPC servers of the SIMATIC NET product family. Connections to other OPC servers or to OPC servers of other vendors are not possible.

If you have added an OPC server, you can browse through its address space and select the OPC items you want to interconnect. Remember that the OPC items must have the required access rights before any interconnection is possible at all. You can change the view of the browser to display access rights and the data type with the combo box at the top left of the window. Select one of the two possible views:

- Show access rights and data type
Apart from the access right at item level, this also shows the data type of the OPC item
- Show only item names
Shows only the name of the OPC item

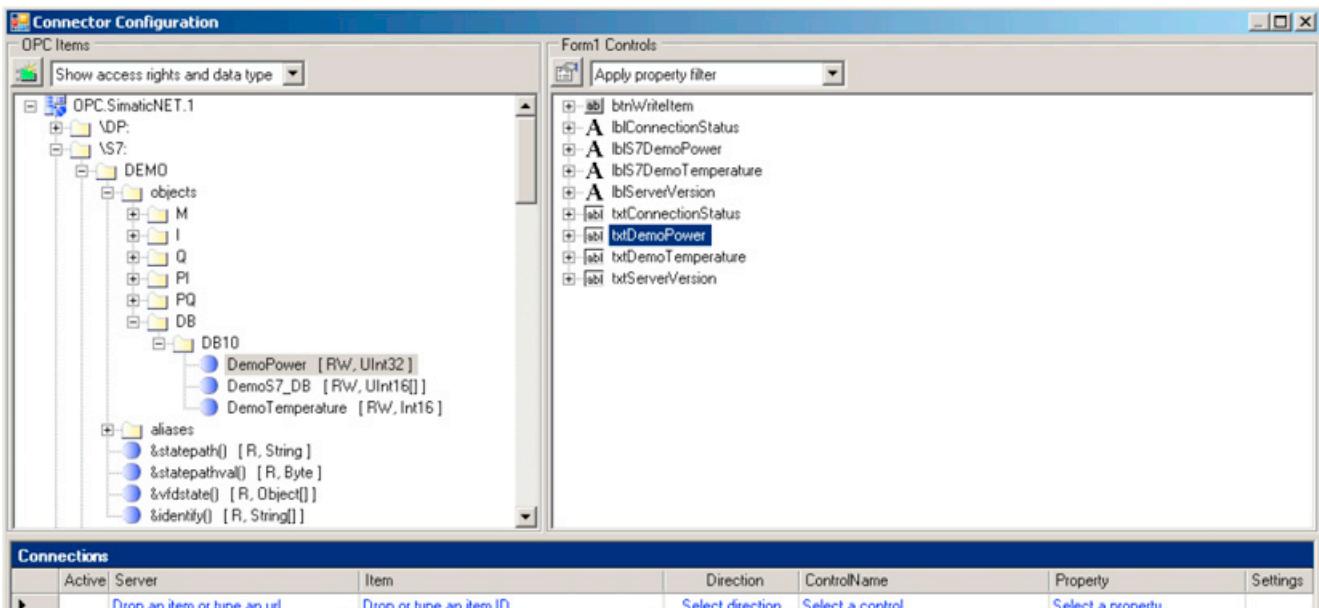
Since some of the Windows controls have large numbers of properties, you can set a filter for this view using the combo box at the top right in the window. Select one of the two possible views:

- **Apply property filter**

Here, the display is according to the filter settings. You can filter for the data type, the access rights and the level of the properties. The filter settings can be made in a dialog that you can open with the button to the left of the combo box (Apply Filter), see section "Browsing Windows control properties (Page 562)".

- **Show all properties**

Shows all properties of the Windows controls.



Now drag the OPC items "&version()", "&statepath()" and the variables of the demo connection "DemoPower" and "DemoTemperature" to the link list of the lower window.

Note

When connecting to a Unified Architecture server (UA server), depending on the selected endpoint it may be necessary to exchange certificates between the client and server before a connection is possible (refer to the section "Creating an OPC UA certificate for the NET OPC data control (Page 574)").

Access rights

To be able to read and write an OPC item, it must have the access rights RW (read write).

Data type

The data type of the OPC item must also be taken into account because it should match the data type of the property of the Windows control with which it will be interconnected. The OPC .NET client control requests the OPC items in the data type that matches the interconnected property, however not all conversions are possible and not all conversions are supported by the OPC server.

Status of the link

If a conversion is not possible or a loss of data could result, this is indicated by the relevant icons in the list of links (Connections). Position the mouse pointer over the icon to obtain more information.

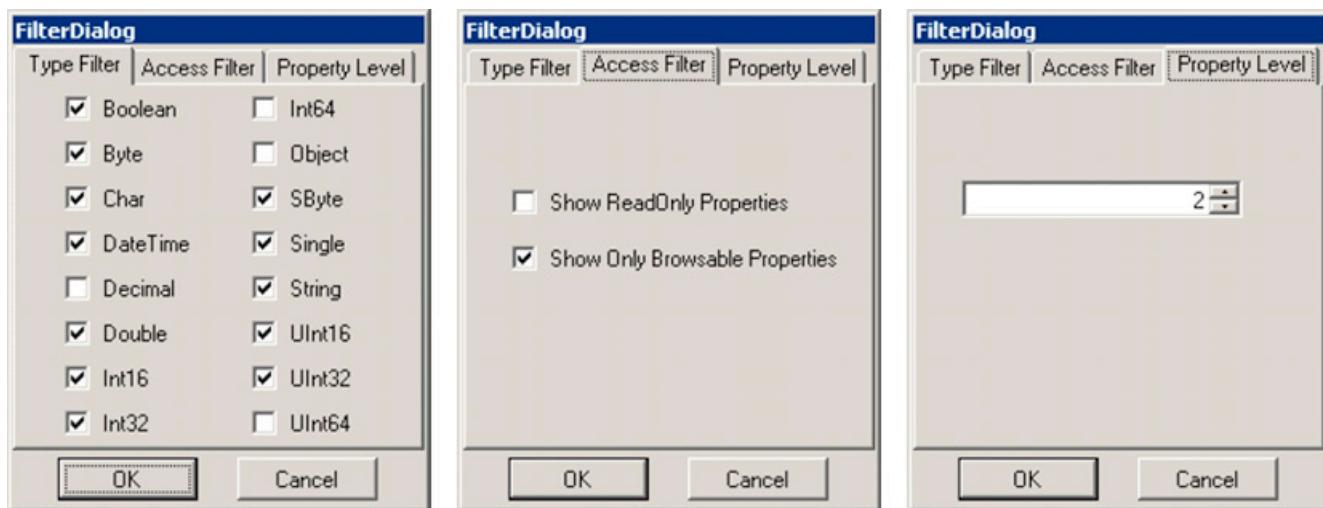
The following symbols are possible:

- Red cross
The interconnection is not possible or incomplete
- Orange check mark
The interconnection is possible but the data types only match with certain restrictions
- Green check mark
The interconnection will work

5.1.4.2

Browsing Windows control properties

In the upper right-hand window of the "Connector Configuration" dialog, all Windows components are displayed that are contained in the application window (Form1). The properties to be interconnected can be selected by opening the hierarchic view. To make the view clearer, you can use filters to hide properties that cannot be considered for interconnection with OPC items due to their data type.



Data type

The data type of the relevant property is displayed with an icon. The "Text" property of a text box has, for example, the "String" type. Since all OPC data types can be converted to a string, every value of an OPC item can be displayed in a text box.

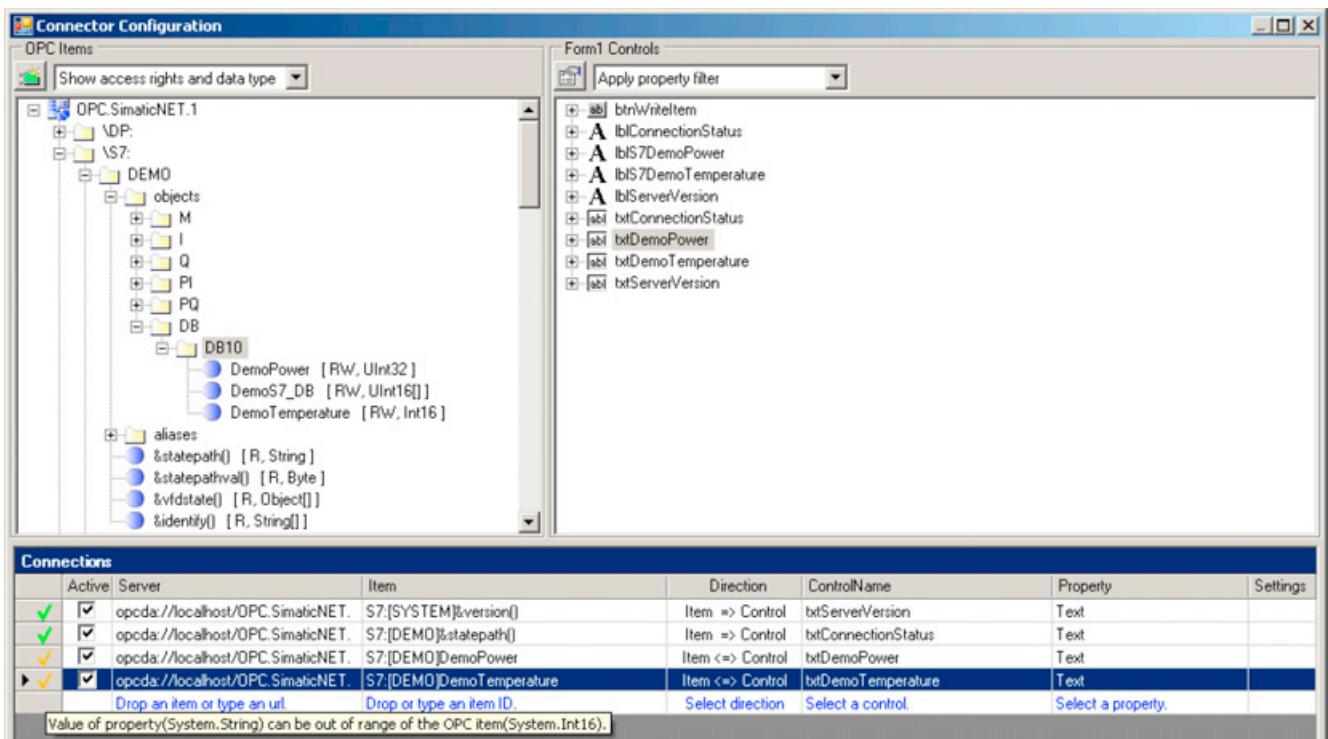
Access rights

Make sure that the selected property has the necessary access rights and that interconnection with an OPC item is possible. Some properties, for example, can only be read but not written.

Now drag the text properties of the relevant text box controls of the Windows form to the lower window (Connections). This creates a link to the OPC item in this row.

5.1.4.3 Creating a list of links

In the lower window of the "Connector Configuration" dialog, you will see the list of created links (connections). Using drag-and-drop, an OPC item is linked to a Windows control property and the data transfer direction is set. This specifies the direction in which the data is transferred.



The list of links automatically checks whether the created link will work and displays this based on various icons (green check mark, orange check mark, red cross). You can obtain further information by positioning the mouse pointer over the icon. Any conflicts that are displayed must be resolved to achieve a fully functional connection.

The following symbols are possible:

- Red cross
The interconnection is not possible or incomplete
- Orange check mark
The interconnection is possible but the data types only match with certain restrictions
- Green check mark
The interconnection will work

Subsequent editing and deactivation

You can edit links already entered in the list by clicking on the relevant box and editing the row. With the check mark (Active), a link can be temporarily deactivated.

Multiple interconnection

OPC items can be interconnected more than once with various Windows controls or with different properties of the same control.

Transfer direction

The transfer direction of the data is shown in the following table.

Direction	Function	Description
Item \Rightarrow Control	OPC item is read	Each time the value of the OPC item changes, this value is written to the linked property of a control.
Item \Leftarrow Control	OPC item is written	When a trigger event is fired, the value of the property is written to the linked OPC item
Item \Leftrightarrow Control	Bidirectional transfer	Each change to the value of the OPC item is written to the property and when a trigger is fired, the current value of the property is written to the OPC item. Writing the value change of the item can be prevented by setting the Option "Disable Datachange when Control has focus" option.

Set the transfer direction of the linked data connections correctly. The OPC items "&version" and "&statepath" are read-only and are linked by "Item \Rightarrow Control" to the text properties of the Windows controls "txtServerVersion" and "txtConnectionStatus". For the OPC items "DemoPower" and "DemoTemperature", select bidirectional transfer (Item \Leftrightarrow Control).

The bidirectional interconnection of the variables is displayed as "functional" but with an orange check mark. Although the value of the OPC item (Integer) can be written to the text property of the control (String) without any problems, in the opposite direction an error is possible because extremely high numbers or even letters could be entered in the text box that can no longer be written to the OPC item or that lead to errors.

5.1.4.4 Trigger for writing

To be able to write to OPC items, trigger events are necessary. These trigger must be configured and they then start the write action. By clicking in the relevant cell of the column at the extreme right (Settings) in the connection list of the "Connector Configuration" dialog, a button appears and a trigger event must be configured for every link that is intended to write an OPC item.

Connections							
	Active	Server	Item	Direction	ControlName	Property	Settings
✓	✓	opcda://localhost/OPC.Sim	S7:[SYSTEM]&version()	Item => Control	txtServerVersion	Text	
✓	✓	opcda://localhost/OPC.Sim	S7:[DEMO]&statepath()	Item => Control	txtConnectionStatus	Text	
▶ ✓	✓	opcda://localhost/OPC.Sim	S7:[DEMO]DemoPower	Item <=> Control	txtDemoPower	Text	
✓	✓	opcda://localhost/OPC.Sim	S7:[DEMO]DemoTemperature	Item <=> Control	txtDemoTemperatur	Text	
		Drop an item or type an url.	Drop or type an item ID.	Select direction	Select a control.	Select a property.	

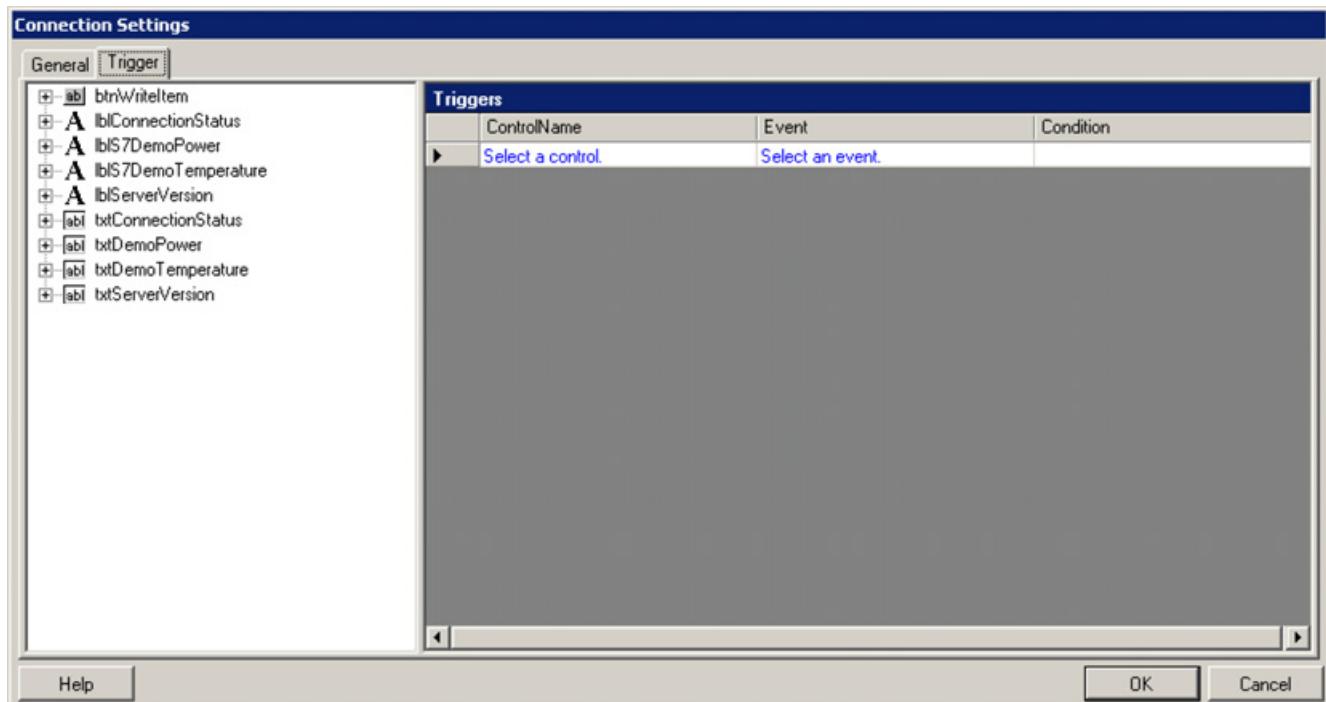
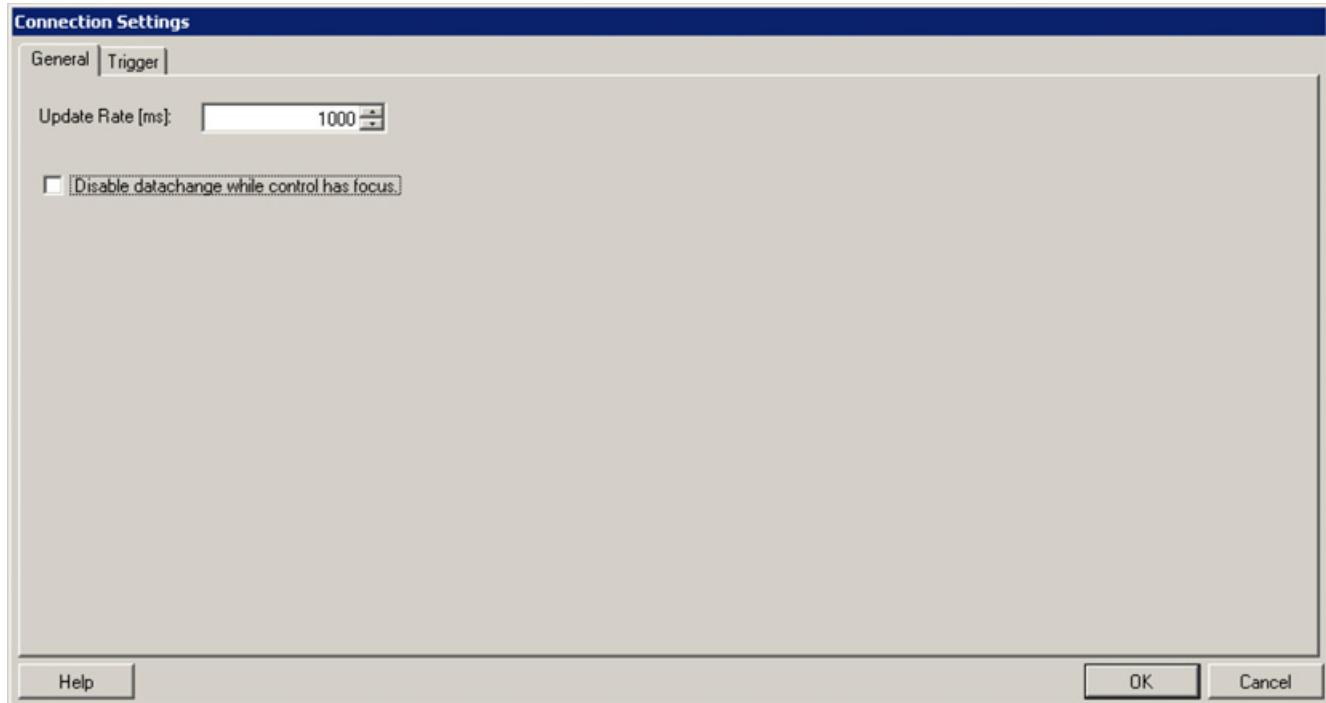
The dialog for configuring the trigger event of a link has two tabs for setting the general and the specific behavior.

General settings:

- **Updaterate**
The update rate in milliseconds at which the OPC item is registered with the server for monitoring
- **Disable DataChange when Control has focus**
Updating is temporarily disabled as long as the control has the focus; this setting should be used with bidirectional transfer

Trigger settings

- Selecting the event
- Setting special conditions in this event



Multiple trigger events

Multiple trigger events can also be configured. The write is then always performed for this interconnection when one of these trigger events is fired.

On the left-hand side of the trigger dialog, an event of the corresponding Windows control is selected and dragged to the list of triggers. With events that also return parameters, these can be queried as conditions.

Disable the updating of the boxes when the control has the focus (general setting of the trigger) for the text boxes "txtDemoPower" and "txtDemoTemperature".

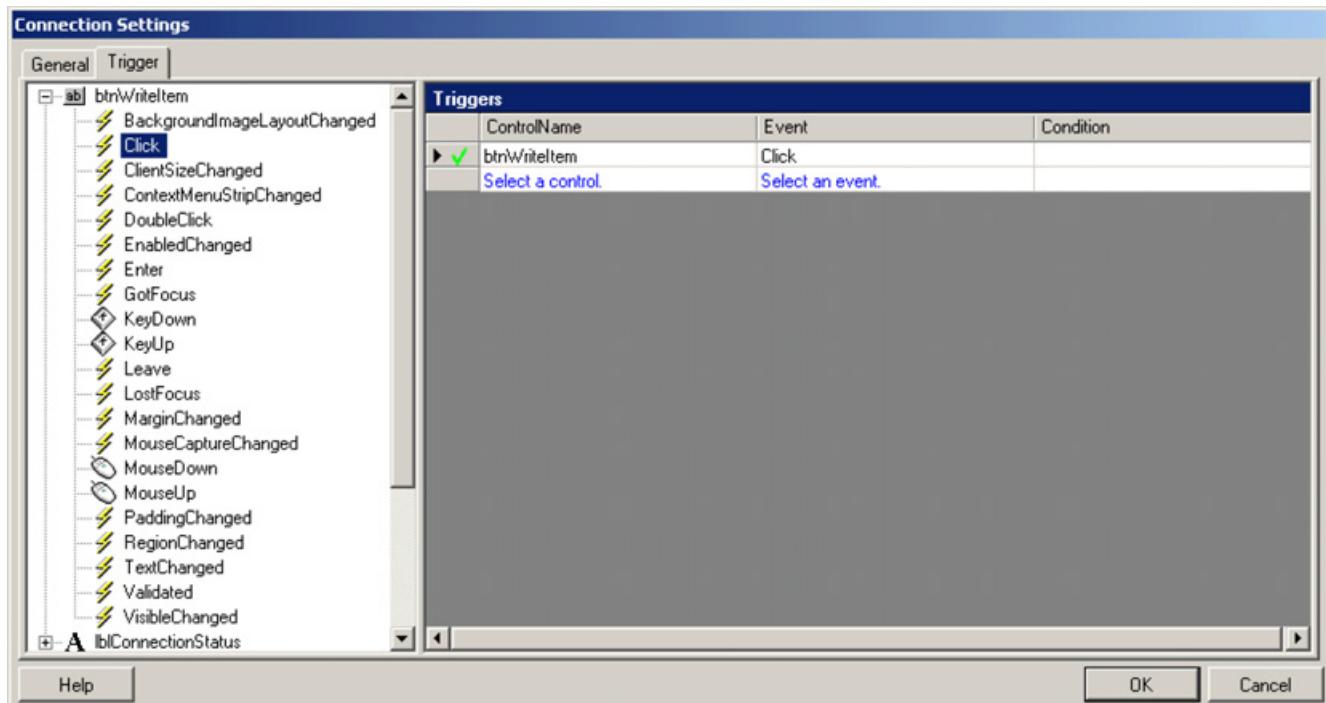
Configure the following special trigger events for the data links of the text boxes "txtDemoPower" or "txtDemoTemperature":

Control name	Event	Condition	Description
btnWriteItem1	Click	--	The current value of the Text property of the txtDemoPower text box is written to the OPC item S7:[DEMO]DemoPower when the btnWriteItem1 button is clicked.
txtDemoTemperature	KeyDown	Return	The current value of the Text property of the txtDemoTemperature text box is written to the OPC item S7:[DEMO]DemoTemperature when the Key Down event has the value "Return" (in other words, the return key was pressed).

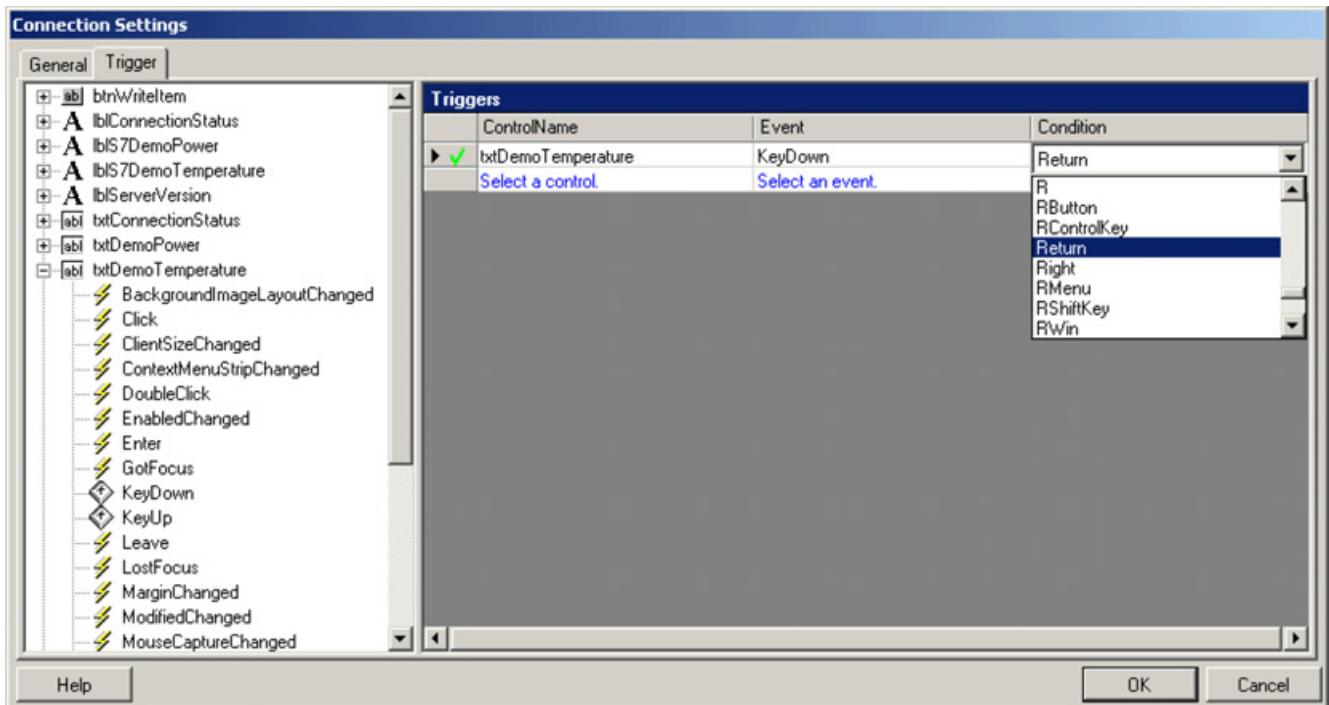
These two examples demonstrate the behavior of the data control when writing to OPC variables.

Example 1:

If the text box has the focus, in other words when the cursor is located in the text box, the text box is no longer updated with OPC values (disable DataChange when Control has Focus). This prevents the text box suddenly changing its value while the user is currently editing it. This allows the user to edit the box in peace and the click event is triggered only when the "btnWriteItem1" button is clicked. This then triggers the write job to the OPC server and the current value of the text box is written to the OPC item.

**Example 2:**

If you use the KeyDown event WITHOUT a condition, a total of three write jobs would be sent to the OPC server when writing the value "152" with each individual KeyDown event: Write(1), Write(15) and Write(152). If, however, you also configure the condition "Return", you can edit the text box without values being written to the OPC server. The value in the text box is only written to the OPC server when a KeyDown event of the Return key is detected.

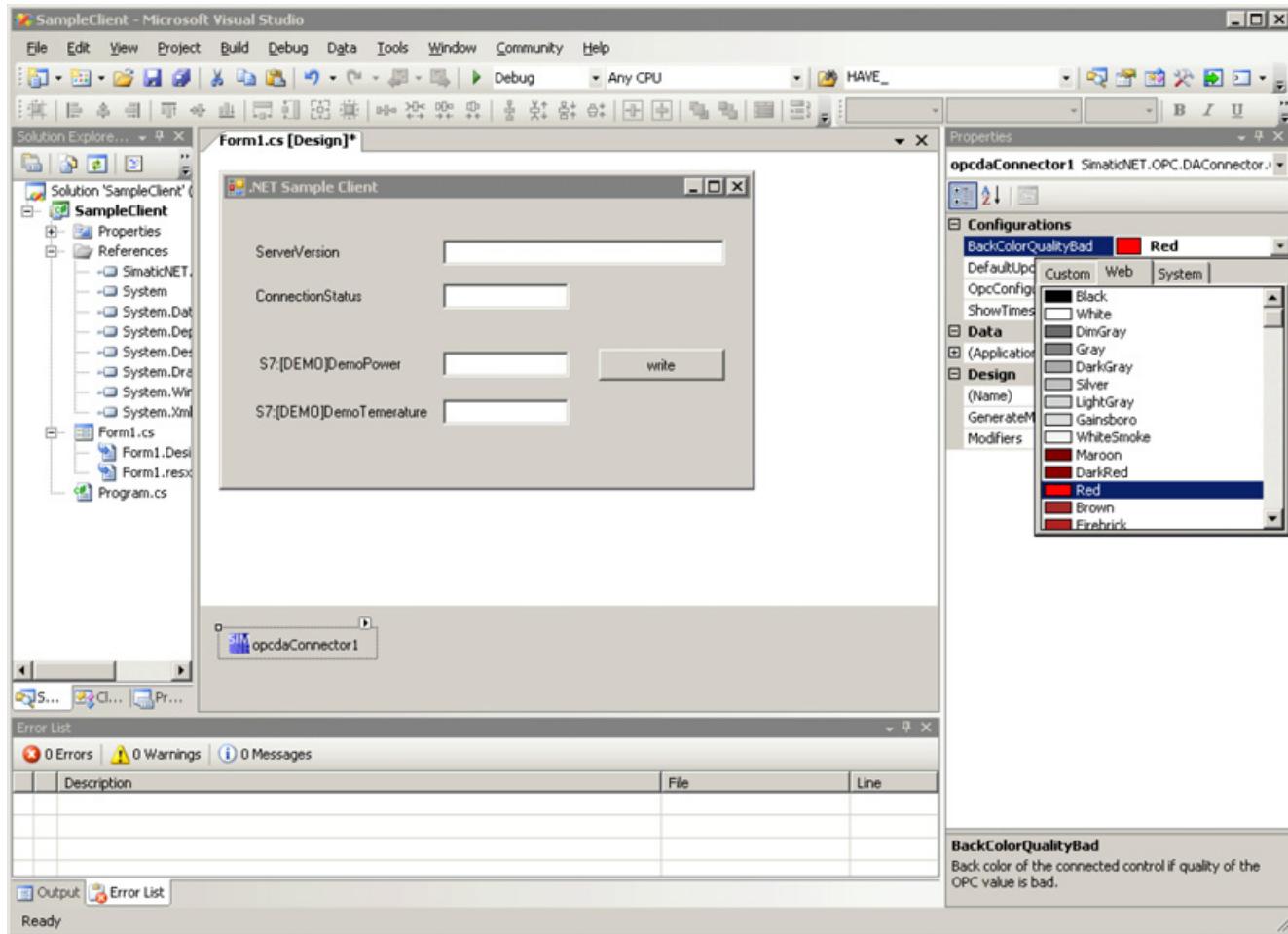


5.1.4.5 OPC quality and error

The linking of the OPC properties "Quality" and "Error" common to every OPC item is handled automatically by the DataControl. A separate interconnection of this OPC attribute is not possible with the SimaticNET .NET OPC Client Control.

Quality

The OPC quality of a read OPC item specifies whether the value that was read can be trusted. The quality can be "good", "uncertain" or "bad" and is mapped to the background color (BackColor property) of the interconnected Windows control and is also shown in the tooltip. Accordingly, the background color, for example of a text box, changes to red (default) if the quality of the returned OPC Value is not "good". This interconnection is made automatically, and in the "Connector Configuration" dialog, the OPC value is simply linked to the Windows Property control. The general setting of the color can be modified for the entire "OPCDAConnector" object.

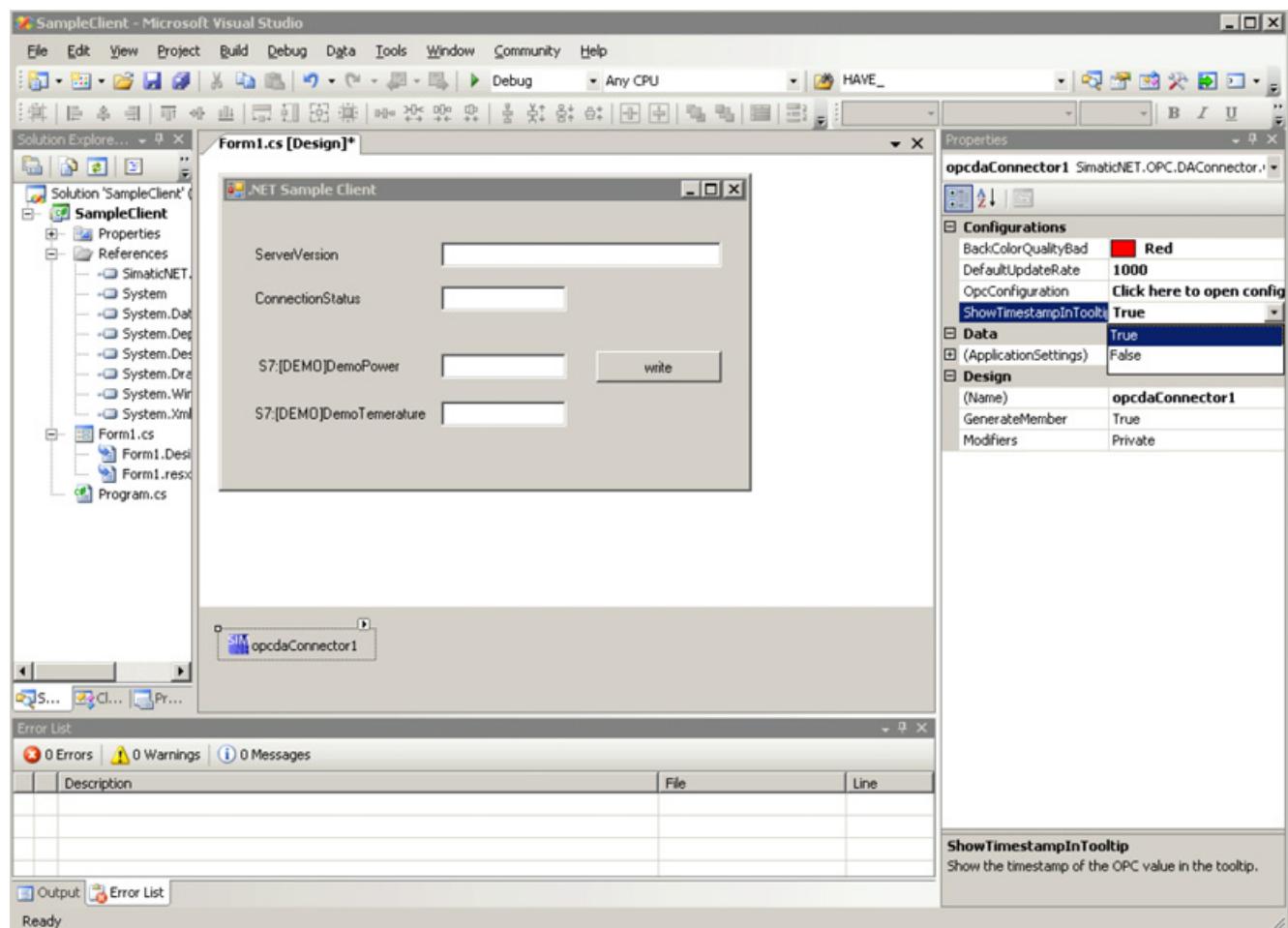


Error

If errors occur, for example when writing, an error code is returned. The relevant text describing the error is displayed in the tooltip of the interconnected Windows control. If you position the mouse pointer over this, you can display the last error text. This interconnection is also made automatically, and in the "Connector Configuration" dialog, the OPC value is simply linked to the Windows Property control, the interconnection to the corresponding tooltip is made in the background. These settings are fixed and cannot be modified. You can display an error in a separate color.

Timestamp

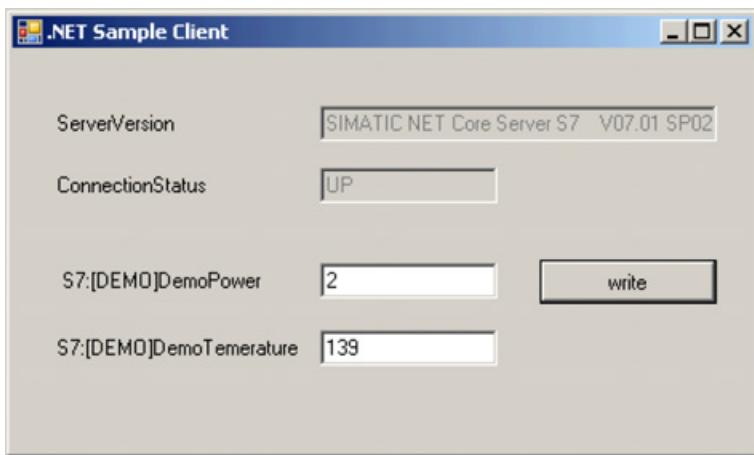
The OPC time stamp (UTC time) is also returned when an OPC item is read. This OPC attribute cannot be interconnected separately with the property of a Windows control. The interconnection with the tooltip of the corresponding Windows object is, however, made automatically if "ShowTimeStampInTooltip" is enabled.



5.1.5 Step 4 - Starting the sample program

The sample application is started by double clicking on the ".exe" file. This is in the "bin\Debug" or "bin\Release" subfolder below the Visual Studio sample project depending on the configuration used to compile.

To run the example, make sure that the S7 demo connection is activated (see section "Setting a sample program (Page 550)").



5.1.5.1 Debug and release

The sample application can be compiled as "Debug" or "Release". The assemblies of the .NET controls work for both variants.

5.1.5.2 Distributing the application

The newly created OPC client application is not normally run on the development computer but rather on a SIMATIC PC with the "SIMATIC NET PC Software" (local client) or on a different computer that has neither MS Visual Studio nor a SIMATIC NET installation (remote OPC client). In this case, you should make sure that all the necessary files are installed on the target system.

Which files need to be supplied with the application?

To be able to run the application on any computer, apart from the executable file ("*.exe") and the .NET assemblies, the OPC proxies/stubs and VC 2010 SP1 Runtime are also required. The .NET assemblies require .NET Framework 4.0 or higher. The OPC core components (proxy/stub) must be installed with each OPC product (client or server). These files already exist on any computer with a SIMATIC NET installation. As an alternative, merge modules or a redistributable setup can be downloaded from the OPC Foundation (www.opcfoundation.org)

Where are the DLLs located or where do they need to be installed?

When compiling the sample application, the Assemblies are automatically copied to the bin directory of the example project, these must be installed with the client application (in the same directory as the application).

- For the OPC core components, there is a merge module from the OPC Foundation. The files are installed to "%systemroot%\windows\system32\"
- For VC Runtime (2010 SP1), there is a merge module from Microsoft.

Which components need to be linked into a setup?

If you create a setup with the Visual Studio (deployment project), the Studio automatically recognizes the interdependencies and inserts the assemblies in the project. The core components of the OPC Foundation also need to be linked as well as the appropriate C Runtime 2010 SP1.

Note

As an alternative, the OPC Scout V10 client setup can also be installed. This already contains the OPC proxy/stub redistributables and .NET Framework.

5.1.6 Troubleshooting

Visual Studio LoaderLock Exception

If you start the sample application from the debugger of the Visual Studio, you may see a warning from the Studio (LoaderLock exception). The cause of this warning is the mixed use of managed and unmanaged code. This warning can be ignored because the DataControl starts the mixed "Assemblies" in the correct order so that a deadlock cannot occur. We nevertheless recommend that you disable the warning. To stop the debugger, follow these steps:

1. Select "Debug" > "Exceptions" in the menu of the Studio.
2. Open the "Managed Debug Assistance" entry
3. Disable the "LoaderLock" check box
4. Exit the dialog with "OK".

No connection can be established to the remote OPC server:

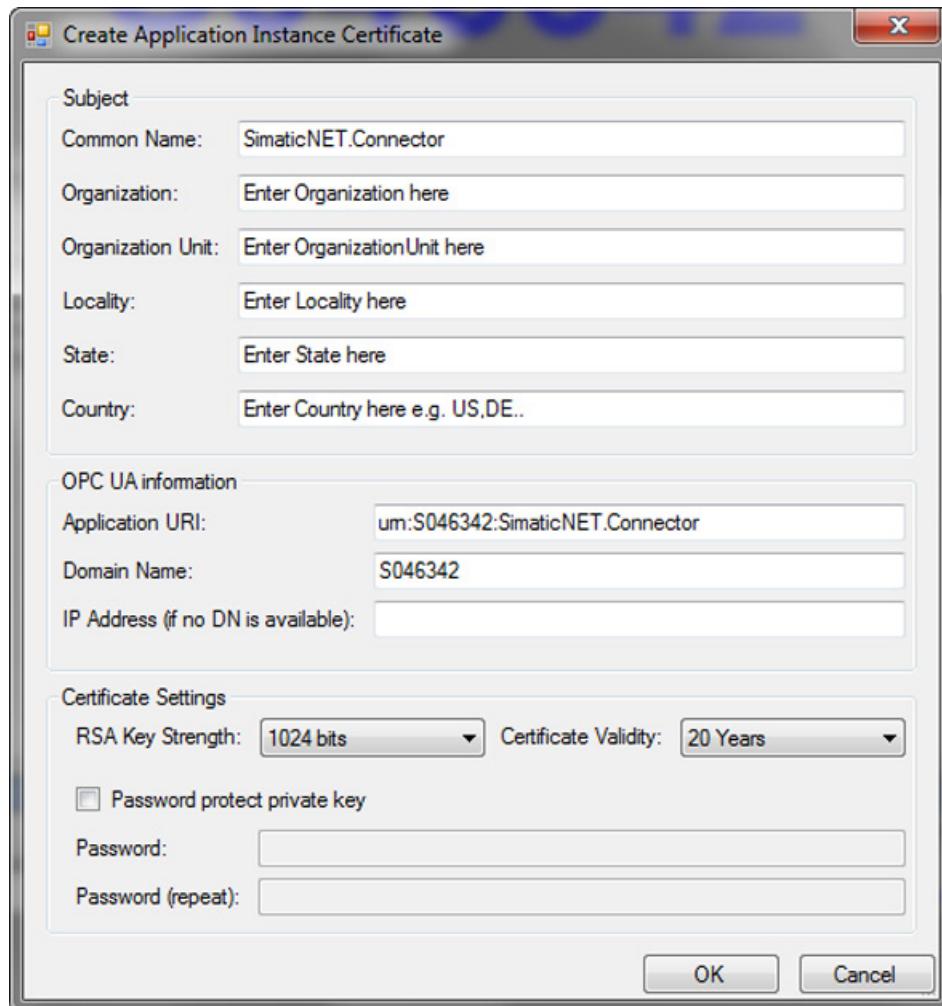
Check the DCOM settings and follow the instructions in the user documentation of the server. Remember that the application runs in the context of the Visual Studio (DevEnv.exe) if you start the application from the development environment. This also means that the DCOM rights for the context in which this process runs must be correctly set.

No data changes are reported by the OPC server

These security settings must be made by the application so that messages sent by the OPC server such as "Shutdown event" or "DataChange notification" can be received. To be able to do this, the client application needs to set the correct security level. This is done during the initialization phase of the application. The ColnitializeSecurity call with the appropriate setting (AuthenticationLevel = RPC_C_AUTHN_LEVEL_CONNECT and impersonation level = RPC_C_IMP_LEVEL_IDENTIFY) must be made before the application starts (for more information read the MSDN). If the COM security is not set by the application, the default settings are used. DCOM Security must then must be set accordingly ("Start" > "Run" > "dcomcnfg.exe").

5.1.7 Creating an OPC UA certificate for the NET OPC data control

In a dialog, ".NET OPC Data Control self-signed X.509 OPC UA Certificates" are generated in the Windows certificate store. It is created if no suitable certificate exists.



The certificate management in the Windows certificate store corresponds to the certificate management in the UA S7 client OPC Scout V10 (see section "Certificate management in the OPC UA client "OPC Scout V10" (Page 501)").

5.1.8 OPC UA certificates when connecting to an OPC UA server

The connection establishment to a Unified Architecture (UA) server using the .NET OPC Data control can normally make use of different endpoints. The SIMATIC NET OPC UA servers can be configured so that one endpoint allows unencrypted connections and one endpoint only accepts encrypted connections.

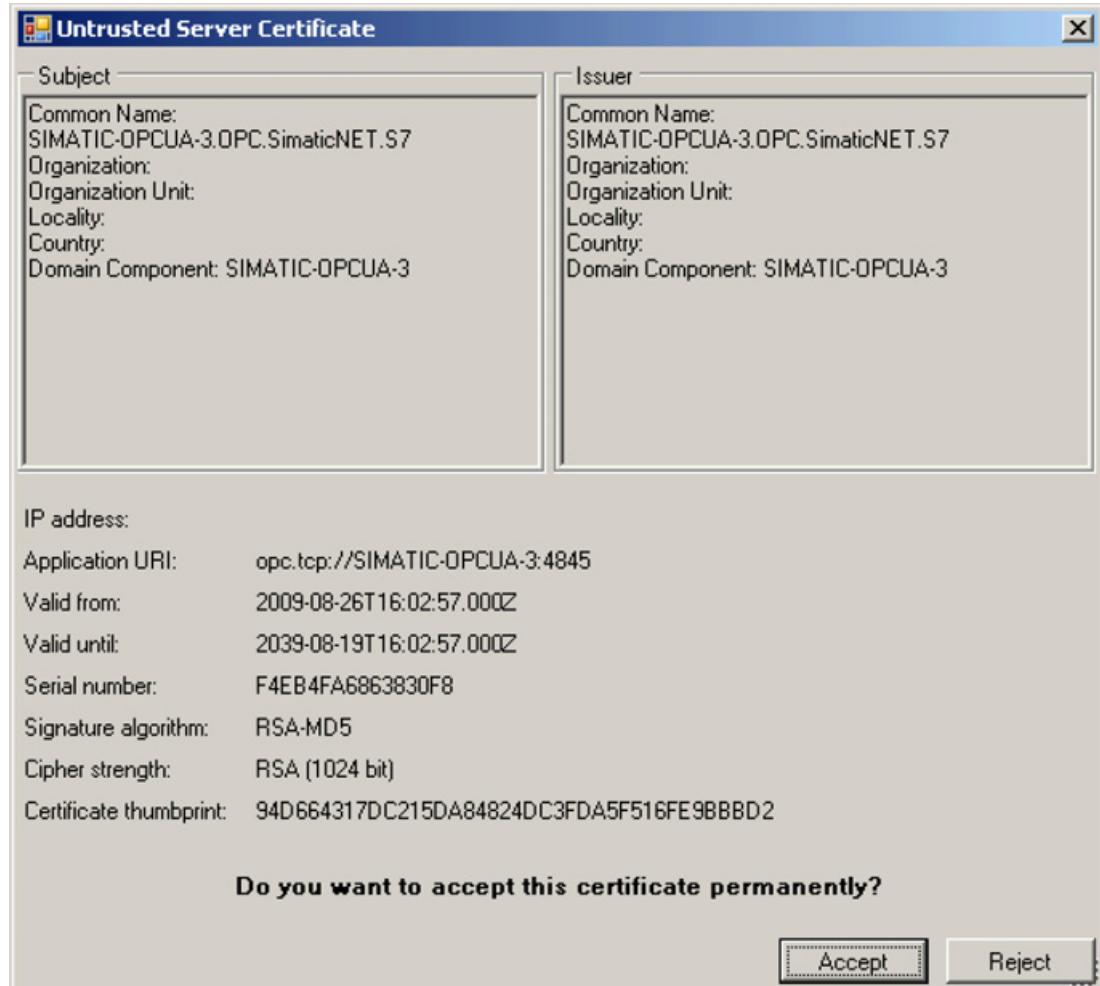
When connecting to an endpoint with security, in other words, the "SecurityPolicy" of the endpoint and the "MessageSecurityMode" are not "None", certificates must be exchanged between the client and server.

Connection establishment then requires the following steps:

- The client checks whether it is already has a certificate. If it does not, the user is prompted to create one.
- The client loads the certificate of the server.

If the server certificate is not yet known to or classed as trustworthy by the .NET OPC Data control UA client, the user will be asked whether or not to accept the certificate.

The certificates are located in "My" store and have a validity of 20 years.



When a client attempts to connect to a server and the client certificate is not known to or not classed as trustworthy by the server, the server rejects the connection.

You will find how to accept, reject, and manage SIMATIC NET OPC UA server certificates in the sections "Certificate management for the OPC UA server (Page 499)" and "Certificate management with a graphic user interface (Page 506)".

5.2 .NET OPC client API

Aims

The aim of the SIMATIC NET .NET OPC client API is to provide users of the languages C-Sharp and Visual Basic .NET with a simple and optimized class library that can be used intuitively and with which OPC client applications for access to OPC servers of the SIMATIC NET product series can be created quickly.

Characteristics of the .NET OPC client component

The .NET class library and the underlying C++ library have the following characteristics:

- Simple, intuitive .NET interface (API – Application Programming Interface).
- Reduction of the OPC Data Access interfaces to essential functions.
- The user does not require any detailed knowledge of the various OPC Data Access interfaces.
- The component completely hides the various basic technologies of OPC such as COM, DCOM, Web services, SOAP and XML.
- The component completely hides the handling of the connection to an OPC server with connection establishment and repeated connection establishment if an error occurs.
- Simple development of OPC client applications with C-Sharp .NET and Visual Basic .NET is possible with the SIMATIC NET .NET OPC client API.
- The conversion of the OPC data from various OPC Data Access interfaces to .NET data types.
- Fast and simple discovery of OPC COM servers locally and on the network.
- High-speed and optimized client-server communication with the implementation of core functionality in C++.
- Support of OPC UA and the certificate management for OPC UA.

Note

With this API, connections can only be established to OPC servers of the SIMATIC NET product family. Connections to other OPC servers or to OPC servers of other vendors are not possible.

Note

When connecting to an OPC UA server, depending on the selected endpoint it may be necessary to exchange certificates between the client and server before a connection is possible. Use the relevant classes and methods for this.

5.2.1 SIMATICNET.OPCDACLIENT namespace

The namespace SimaticNet.OpcDaClient of the .NET component provides the following functionality via the "DaServerMgt" object:

- Connection to the OPC server

With the "Connect" method, the connection can be established to the OPC server and can be terminated again with the "Disconnect" method. The connection is monitored by the .NET component. If connection errors occur, status changes are reported with the "ServerStateChanged event".

- Reading and writing OPC Data Access items

Using the "Read" and "Write" methods, the values of OPC items can be read and written synchronously and asynchronously.

- Signaling data changes

The .NET component provides a mechanism with which value changes can be signaled making cyclic reading unnecessary. Using the "Subscribe" method, items can be registered for monitoring. These can be canceled again with "SubscriptionCancel". Changed values are reported via "Event DataChanged".

- Obtaining information about the address space

The "Browse" method makes it possible to search through the address space of an OPC Data Access server for OPC items. Using the "GetProperties" method, properties of OPC items can be obtained.

5.2.1.1 Classes of the data model

The following section describes the classes provided by the .NET API. Apart from the "DaServerMgt" class, these only have properties and are therefore used for data encapsulation. "DaServerMgt" provides the API for access to an OPC Data Access server.

DaServerMgt class

The "DaServerMgt" class allows access to an OPC Data Access server. You will find a detailed description of the API and its methods in "The interface of the DaServerMgt object".

ServerState enumerator

The "ServerState" enumerator is transferred with "StateChange Event" and provides information on the current status of the server.

- CONNECTED
The connection to the OPC server is established.
- DISCONNECTED
There is no connection to the OPC server.
- ERRORSHUTDOWN
The OPC server has sent a Shutdown event. The connection is interrupted. A new connection establishment is attempted.

- **ERRORWATCHDOG**

The .NET API has detected an error on the connection to the OPC server. The connection is interrupted. A new connection establishment is attempted.

- **UNDEFINED**

The OPC server is in a status that does not correspond to any of those listed.

ItemIdentifier class

The "ItemIdentifier" class is required for the "Read", "Write", "GetProperties" and "Subscribe" methods. The class is used to identify an OPC item. Instances of the class are transferred with all methods as ref parameters (in/out). This is necessary because the .NET component stores information in the object the first time it is used and this is then used to optimize the OPC calls when used again. This reason, the objects should be re-used even in Read and Write calls and not reinitialized for each call.

The class also contains error information that needs to be checked following a call to be able to detect item errors.

Properties of the ItemIdentifier class:

- **string ItemName**

This property contains the item name (ItemID) of an OPC Data Access item.

- **string ItemPath**

In the case of an OPC XML DA server, the optional ItemPath can be specified here. In the case of a COM OPC Data Access server, this property is ignored.

- **object ClientHandle**

If an ItemIdentifier is generated for the "Subscribe" method, the transferred "Clienthandle" with the changed value of the item is transferred back to the client with "DataChanged". Since the "Clienthandle" is of the type object, any .NET type can be transferred here, for example a TextBox control or an object containing information about the continued processing of the data. In the "DataChange handler method", the "ClientHandle" allows the corresponding object in the application to be identified.

- **int ServerHandle**

Instances of the "ItemIdentifier" class are transferred with all methods as reference parameters (keyword ref). With the "Read" and "Write" method calls, the "int ServerHandle" property is set by the .NET API. If the same ItemIdentifier instances are transferred with "Read" or "Write" again, the .NET API can greatly optimize the calls to the OPC server.

- **System.Type DataType**

The data type in which the OPC server returns the value of the item with "Read" or "DataChange" can be specified here. If the property is not set, the native data type of the items on the OPC server is returned. In this case, the .NET API sets the data type the first time the ItemIdentifier instance is used.

- **ResultID ResultID**

If an item error occurs during an OPC call (for example unknown ItemName, attempt to write a read-only item etc.), the corresponding error code is stored in the object of the corresponding "ItemIdentifier". The "ResultID" class provides the error code(int), the name(string) and a localized description(string). This allows the program to react to errors.

ItemValue class

The "ItemValue" class is required for the "Read" and "Write" methods. When reading, "ItemValue" contains the value, quality and time stamp. When writing, "ItemValue" only contains the value of the OPC item.

When the "Read" method is called, the "ItemValue array" is an out parameter, which means that the object is generated completely by the .NET API.

When the "Write" method is called, the "ItemValue array" must be generated according to the "ItemIdentifier array" by the client and filled with the values to be written.

Properties of the "ItemValue" class:

object Value

The value that was read or that will be written. Since the "Value" property is of the "object" type, it can adopt or contain any data type. Normally, "Value" has the same type as was requested with the corresponding "ItemIdentifier".

QualityID Quality

The quality of the value. The "QualityID" class provides the quality code(int), the name(string) and a description(string).

System.DateTime TimeStamp

The time stamp of the value.

ItemValueCallback class

The "ItemValueCallback" class is derived from the "ItemValue" class and has the following additional properties over and above those described in "ItemValue":

- **object** ClientHandle
Here, the "ClientHandle" supplied with the "Subscribe" or "ReadAsync" method is returned. "ClientHandle" allows the client to uniquely identify the returned value.
- **ResultID** ResultID
Any errors relating to the item identified by "ClientHandle" are entered here. The "ResultID" class provides the error code(int), the name(string) and a localized description(string). This allows the program to react to errors.

ItemResultCallback class

The "ItemResultCallback" class is used in "WriteCompleted Callback" and has the following properties:

- **object** ClientHandle
Here, the "ClientHandle" supplied with the "WriteAsync" method is returned. "ClientHandle" allows the client to uniquely identify the returned value.
- **ResultID** ResultID
Any errors relating to the item identified by "ClientHandle" are entered here. The "ResultID" class provides the error code(int), the name(string) and a localized description(string). This allows the program to react to errors.

BrowseElement class

The "BrowseElement" class contains all data relating to an OPC data access item obtained with the "Browse" method.

Properties of the "BrowseElement" class:

string Name

The name of the returned element. Normally this name is used for the tree representation of the namespace of an OPC server.

string ItemName

The ItemName of the element.

string ItemPath

The ItemPath of the element.

bool HasChildren

Indicates whether or not the element has child elements in the namespace.

bool IsItem

Indicates whether or not the element is an OPC Data Access item.

ItemProperties ItemProperties

The properties of the element requested with the "Browse" method.

BrowseFilter enumerator

Transferring a "BrowseFilter" specifies the type of child elements of the specified element to be returned when the "Browse" method is called. Possible filters include:

- **ALL**
All elements are returned.
- **BRANCH**
Only elements of the type branch are returned.
- **ITEM**
Only elements of the type item are returned.

ItemProperties class

The "ItemProperties" class is generated only by the .NET API. It contains the properties of an OPC item.

Properties of the "ItemProperties" class:

- **ItemProperty[] RequestedItemProperties**

An array of objects of the "ItemProperty" class. This array contains all the properties of an OPC item requested with "GetProperties" or "Browse".

ItemProperty class

The "ItemProperty" class represents a property of an OPC item.

Properties of the "ItemProperty" class:

- **string ItemName**
If an OPC server supports the reading and writing of a property via an item, the "ItemName" of the property is returned here.
- **string ItemPath**
If an OPC server supports the reading and writing of a property via an item, the "ItemPath" of the property is returned here.
- **string Description**
The description of the property. This information can be used to represent a property in a graphic user interface.
- **object Value**
The value of the property.
- **ResultID ResultID**
If an error occurs obtaining the property, the corresponding error code is stored in this property.
- **System.Type DataType**
The data type of the property.
- **int PropertyID**
The ID of the property.

ResultID class

The "ResultID" class contains an error code, its string representation and a localized description of the error that has occurred. "ResultIDs" are used to display errors at the item level and are part of the OPCExceptions.

Properties of the "ResultID" class:

- **int Code**
The code transferred by the server when an action took place.
- **string Name**
The string representation of the code.
- **string Description**
The localized description of the error that occurred.
- **bool Succeeded**
With this property, it is possible to find out whether or not "ResultID" contains the code of a successful operation without needing to evaluate the code itself in detail.

An instance of the "ResultID" class can have the following values:

Value	Code	Name	Description
WIN_S_OK	0x00000000	S_OK	Operation succeeded.
WIN_S_FALSE	0x00000001	S_FALSE	The function was partially successful.

Value	Code	Name	Description
S_UNSUPPORTEDRATE	0x0004000D	OPC_S_UNSUPPORTEDRATE	The server does not support the requested data rate but will use the closest available rate.
S_INUSE	0x0004000F	OPC_S_INUSE	The operation cannot be performed because the object is being referenced.
S_DATAQUEUEOVERFLOW	0x00040404	OPC_S_DATAQUEUEOVERFLOW	Not every detected change has been returned since the server's buffer reached its limit and had to purge out the oldest data.
S_CLAMP	0x0004000E	OPC_S_CLAMP	A value passed to write was accepted but the output was clamped.
RPC_S_SERVER_UNAVAILABLE	0x800706BA	RPC_S_SERVER_UNAVAILABLE	The RPC server is unavailable.
RPC_S_CALL_FAILED	0x800706BE	RPC_S_CALL_FAILED	The remote procedure call failed.
E_UNKNOWNPATH	0xC004000A	OPC_E_UNKNOWNPATH	The item's access path is not known to the server.
E_UNKNOWNITEMID	0xC0040007	OPC_E_UNKNOWNITEMID	The item ID is not defined in the server address space or no longer exists in the server address space.
E_RATENOTSET	0xC0040405	OPC_E_RATENOTSET	There is no sampling rate set for the specified item.
E_RANGE	0xC004000B	OPC_E_RANGE	The value was out of range.
E_PUBLIC	0xC0040005	OPC_E_PUBLIC	The requested operation cannot be done on a public group.
E_NOTSUPPORTED	0xC0040406	OPC_E_NOTSUPPORTED	The server does not support writing of quality and/or timestamp.
E_NOTFOUND	0xC0040011	OPC_E_NOTFOUND	The requested object (e.g. a public group) was not found.
E_NOBUFFERING	0xC0040402	OPC_E_NOBUFFERING	The server does not support buffering of data items that are collected at a faster rate than the group update rate.
E_INVALIDITEMID	0xC0040008	OPC_E_INVALIDITEMID	The item ID does not conform to the server's syntax.
E_INVALIDHANDLE	0xC0040001	OPC_E_INVALIDHANDLE	The value of the handle is invalid.
E_INVALIDFILTER	0xC0040009	OPC_E_INVALIDFILTER	The filter string was not valid.

Value	Code	Name	Description
E_INVALIDCONTINUATIONPOINT	0xC0040403	OPC_E_INVALIDCONTINUATIONPOINT	The continuation point is not valid.
E_INVALIDCONFIGFILE	0xC0040010	OPC_E_INVALIDCONFIGFILE	The server's configuration file is an invalid format.
E_INVALIDARG	0x80070057	E_INVALIDARG	An argument to the function was invalid.
E_INVALID_PID	0xC0040203	OPC_E_INVALID_PID	The specified property ID is not valid for the item.
E_FAIL	0x80004005	E_FAIL	Unspecified Error.
E_DUPLICATENAME	0xC004000C	OPC_E_DUPLICATENAME	Duplicate name not allowed.
E_DEADBANDNOTSUPPORTED	0xC0040401	OPC_E_DEADBANDNOTSUPPORTED	The item does not support deadband.
E_DEADBANDNOTSET	0xC0040400	OPC_E_DEADBANDNOTSET	The item deadband has not been set for this item.
E_BADTYPE	0xC0040004	OPC_E_BADTYPE	The server cannot convert the data between the specified format and/or requested data type and the canonical data type.
E_BADRIGHTS	0xC0040006	OPC_E_BADRIGHTS	The item's access rights do not allow the operation.
DISP_E_TYPEMISMATCH	0x80020005	DISP_E_TYPEMISMATCH	Type mismatch.
CONNECT_E_NOCONNECTION	0x80040200	CONNECT_E_NOCONNECTION	The client has not registered a callback through IConnectionPoint::Advise.
CONNECT_E_ADVISELIMIT	0x80040201	CONNECT_E_ADVISELIMIT	Advise limit exceeded for this object.

QualityID class

The "QualityID" class contains all the information relating to a quality code transferred by the server.

Properties of the "QualityID" class:

- **int FullCode**
The code transferred by the server.
- **int Quality**
The code that describes the quality of the transferred value
- **int LimitBits**
The limit part of the code transferred by the server
- **int VendorBits**
The vendor part of the code transferred by the server
- **bool IsGood**
Using this property, it is possible to find out whether or not the quality of the read value is good.

- **string Name**
The string representation of the code.
- **string Description**
The localized description of the quality code.

ConnectInfo class

The "ConnectInfo" class contains initialization data that is evaluated by the server object when the "Connect" method is called.

Properties of the "ConnectInfo" class:

- **bool RetryAfterConnectionError**
If this flag is set, the OPC client API will attempt to re-establish a connection following a connection abort until establishment is successful. If the connection can be re-established, the group handles generated prior to the interruption remain valid. The "Eventhandler" methods remain registered for the events.
- **bool RetryInitialConnect**
If this flag is set, the OPC client API will continue to attempt to connect to the server if the first connection attempt was unsuccessful.
- **int KeepAliveTime**
The OPC client API checks the availability and the connection to the server during runtime. KeepAliveTime represents the time interval in milliseconds after which a validation takes place.
The initial value of "KeepAliveTime" is 10000 ms.
The interval for the reconnect to the OPC server begins with twice the "KeepAliveTime" and is increased up to 10 times the "KeepAliveTime" if the OPC server is unavailable for a longer time. The interval for a reconnect after an OPC server shutdown is one minute.
- **string LocalID**
LocalID can be used to send a country identifier ("us", "en", etc.) to the server. Setting a LocalID has the effect that localizable return values are transferred in a specific language. If the value does not exist in a localizable version, the default value is transferred. The following table shows several examples of LocalIDs:

Locale	LocalID
System default	Empty string
English	en
English - United States	en - us
English - United Kingdom	en - gb
German	de
German - Germany	de - de
German - Austria	de - at

Note

The changes to the connection status, should always be tracked using the "ServerStateChanged" event. Other methods such as "Browse", "Read" or "Subscribe" can only be executed successfully after the status of the server has changed to connected.

- String SecurityPolicyUri
The SecurityPolicy defines which type of security is used for transfer via the UA SecureChannel, for example, unencrypted or Basic128Rsa encrypted.
- byte MessageSecurityMode
The MessageSecurityMode defines which security is used at the message level, for example 1 = None, 2 = Encrypt, 3 = Encrypt&Sign.
- byte[] ServerCertificate
The X509 certificate of the OPC UA server to which the connection will be established.
The certificate can be loaded, for example, using "OpcServerEnum::getCertificateForEndpoint".
- byte[] ClientCertificate
the X509 certificate of the client.
- byte[] ClientPrivateKey
The private key for the X509 certificate of the client.
- string CertificateStoreName
The name of the "WindowsCertificateStore" in which the certificates for client and server are stored. Use "UA Applications".
- WinStoreLocation CertificateStoreLocation
The location in the "WindowsCertificateStore" e.g. LocalMachine

Note

The certificates and the settings for the "WindowsCertificateStore" are only needed if the connection to the OPC UA server is secure, in other words "SecurityPolicy" and "MessageSecurityMode" are not set to "None".

To use the location "LocalMachine", administrator privileges are required.

ReturnValue enumerator

Returning a "ReturnValue" shows whether the relevant function was successful or whether the quality of the values was not good when various methods are called.

- SUCCEEDED
The function was completed successfully.
- ITEMERROR
An error occurred with at least one item during the operation. Which item is involved and

what the exact error code is must be obtained with the help of the function parameters (for example ref ItemIdentifier[]).

- **QUALITYNOTGOOD**

The quality is not good for at least one item. Which item is involved and what the exact quality code is must be obtained with the help of the function parameters (for example ref ItemIdentifier[]).

- **ITEMERRORANDQUALITYBAD**

An error has occurred during the operation with at least one item and the quality is not good for at least one item (the same or a different item). Which items are involved must be obtained with the help of the function parameters (ref ItemIdentifier[]).

- **UNSUPPORTEDUPDATERATE**

The requested update rate is not supported by the server.

5.2.1.2 The interface of the DaServerMgt object

Generating the DaServerMgt object

As is typical for .NET types, an instance of the "DaServerMgt" class must first be generated:

```
DaServerMgt daServerMgt = new DaServerMgt();
```

Connect method

The "Connect" method establishes the connection to the server.

The "IsConnected" property indicates the connection status of the client to .NET API. The .NET API checks the connection to the OPC server when "connect" was called. If the connection to the server aborts, the .NET API attempts to re-establish the connection. The "ServerState" property shows the status of the OPC server. Possible values are DISCONNECTED, CONNECTED, ERRORSHUTDOWN, ERRORWATCHDOG. If the status is set to one of the two error states, the .NET API attempts to re-establish the connection. The client is informed of status changes by the "ServerStateChanged" event.

Parameter	Functionality
string url	The URL of the OPC server (see below: "Structure of the URL").
ConnectInfo	"ConnectInfo" contains several properties, that are used to initialize the .NET API. LocalID string: The country settings to be used for the language-specific return values. If an empty string is transferred, the default setting is used. "bool ReconnectAfterConnectionError" / "bool RetryInitialConnect ". A connection establishment to the OPC server can fail due to a network connection being temporarily unavailable. By setting the two parameters, the .NET API attempts to re-establish the connection if an error occurs. The "IsConnected" property is always set and the client can nevertheless create subscriptions despite the connection error. For the items in the subscriptions, however, a bad quality is returned in the "DataChanged" event until the connection to the server can be established. If connection establishment fails and this parameter is not set, the "Connect" method fails and an exception is reported. "int keepAliveTime" is a time interval in milliseconds at which the connection to the OPC server is checked.
int clientHandle	With this parameter, the application can assign an index for this server object. This "ClientHandle" is transferred by the .NET API when the "ServerStateChanged" event occurs to identify the object in the application.
out bool connectFailed	Indicates whether or not the connection establishment failed if the "retryConnect" parameter was set. If the connection establishment fails and the "retryConnect" parameter was not set, the "connect" method fails and an exception is thrown.

The required form of the URL can either be entered by the user or obtained automatically from the class "OpcServerEnum" in the "SimaticNet.OpcCmn" namespace.

```
string url = "opcda://localhost/OPC.SimaticNET/{b6eacb30-42d5-11d0-9517-0020afaa4b3c}  
";  
  
string localId = "de";  
  
int clientHandle = 0;  
  
int keepAliveTime = 5000; // 5 seconds
```

```
daServerMgt.Connect(url, localId, clientHandle, true, keepAliveTime, out
connectFailed);
```

Note

With this API, connections can only be established to OPC servers of the SIMATIC NET product family. Connections to other OPC servers or to OPC servers of other vendors are not possible.

Note

If this API is used in a multi-threaded application, make sure that the thread in which the Connect method is executed is executed in the same apartment state as the application itself.

Example:

If you develop an application with "ApartmentState STA" and Connect needs to be executed in a different thread from the base thread, set the "ApartmentState" of the new thread before the start as follows:

```
Thread MyThread = new Thread(MyThreadStart);
MyThread.SetApartmentState(ApartmentState.STA);
```

The procedure for "Apartment State MTA" is analogous.

Structure of URL

The URL that identifies the server uniquely is structured as follows:

- For OPC COM Data Access
[OpcSpecification]://[Hostname]/[ServerIdentifier]
- For OPC Unified Architecture
[Endpoint server URL]

URL part	Description
OpcSpecification	Specifies the OPC DA specification to be used <ul style="list-style-type: none"> • opcda for OPC Data Access 2.05A or 3.0 (COM) • http for OPC XML-DA 1.01
Hostname	Computer name or IP address of the computer on which the OPC server is operating. For the local computer, this is localhost. For OPC-XML-DA servers, a port can also be specified along with the IP address.

URL part	Description
ServerIdentifier	Identifies the OPC server on the specified computer. <ul style="list-style-type: none"> • OPC XML-DA - path to the Web service • OPC COM DA – [ProgID]/[optional ClassID]
Endpoint server URL	opc.tcp://[hostname]:[port] <ul style="list-style-type: none"> • opct.tcp is the OPC UA binary TCP protocol identifier • hostname is the computer name or IP address • port is the port number

Examples of a correct URL are shown below:

opcda://PC_001/OPC.SimaticNET/{b6eacb30-42d5-11d0-9517-0020afaa4b3c}
 opcda://localhost/OPC.SimaticNet.DP/{625c49a1-be1c-45d7-9a8a-14bedcf5ce6c}/
 http://192.168.0.120/Opc.Simatic.Net/WebService.asmx
 opc.tcp://PC1:55101 (for OPC.SimaticNET.S7)

Disconnect method

Calling this method terminates the connection to the OPC server. All existing subscriptions and resources are released.

IsConnected property

With this property, the current connect state to the client API can be queried. The status of the connection to the OPC server can be queried using the "ServerState" property.

```
if(daServerMgt.IsConnected == true)
{
    daServerMgt.Disconnect();
}
```

ServerState property

With this property, the current status of the connection to the OPC server can be queried.

Read method

Using the "Read" method, items of the OPC server can be read. If items are read cyclically, it is advisable to create a subscription and to receive the changed data with the "DataChanged" event.

Parameter	Functionality
int maxAge	The maximum age of the values to be read in milliseconds. If the values currently in the cache of the OPC server are older than specified by maxAge, the values are read from the device. If the value 0 is transferred, the values are always read from the device.
ref ItemIdentifier[] itemIdentifiers	The "itemIdentifiers" array specifies the OPC items to be read. Any item errors are returned in the "ResultID" object of the relevant "ItemIdentifiers". The parameter is an in/out parameter so that the .NET API can return error information and a "ServerHandle" in the object. If the same items are used more than once for "Read" and "Write" calls, the objects should only be created once and then used for all calls. This allows the .NET API to use the information saved in the object to optimize the OPC server access.
out ItemValue[] itemValues	The "itemValues" array contains the values, quality and timestamp for the read OPC items.

Return type	Functionality
ReturnCode	The return code indicates whether or not errors occurred relating to the transferred parameters and whether or not the quality of all read values is good. Based on the value of the enumerator, the "ItemIdentifier[]" should be checked.

The following code is an example of how two items are read with one "Read":

```

ReturnCode result;

int maxAge = 1000;

ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];

itemIdentifiers[0] = new ItemIdentifier();
itemIdentifiers[0].ItemName = "S7:[@LOCALSERVER]DB1,B0";

itemIdentifiers[1] = new ItemIdentifier();
itemIdentifiers[1].ItemName = "S7:[@LOCALSERVER]DB1,B1";

ItemValue[] itemValues = null;

result = daServerMgt.Read(maxAge,
    ref itemIdentifiers,
    out itemValues);

if (result != ReturnCode.SUCCEEDED)
{

```

```
// ToDo: implement error handling
}

else
{
    if (result != ReturnCode.ITEMERRORANDQUALITYBAD)
    {
        // all values are correct and can be used
        // ToDo: use the values in the application

        string strValue1;

        string strValue2;

        strValue1 = itemValues[0].Value.ToString();

        strValue2 = itemValues[1].Value.ToString();

    }
    else
    {
        // ToDo: check the ItemIdentifiers[]
    }
}
```

Note

All methods of the .NET API allow group calls, in other words, any number of items can be read with one "Read" call.

Where possible, group calls should be used because this strategy reduces the client-server communication to a minimum on the one hand and makes full use of the optimization mechanisms of the API on the other.

ReadAsync method

Using the "ReadAsync" method, items of the OPC server can be read asynchronously. The read values are delivered using the "ReadCompleted" event. If items are read cyclically, it is advisable to create a subscription and to receive the changed data with the "DataChanged" event.

Parameter	Functionality
int transactionHandle	A handle as identification for the "Read" transaction. The handle is returned to the client with the "ReadCompleted" event.
int maxAge	The maximum age of the values to be read in milliseconds. If the values currently in the cache of the OPC server are older than specified by "maxAge", the values are read from the device. If the value 0 is transferred, the values are always read from the device.
ref ItemIdentifier[] itemIdentifiers	The "itemIdentifiers" array specifies the OPC items to be read. Any item errors are returned in the "ResultID" object of the relevant "ItemIdentifiers". The parameter is an in/out parameter so that the .NET API can return error information and a "ServerHandle" in the object. If the same items are used more than once for "Read" and "Write" calls, the objects should only be created once and then used for all calls. This allows the .NET API to use the information saved in the object to optimize the OPC server access.

Return type	Functionality
ReturnCode	The return code indicates whether or not errors occurred relating to the transferred parameters and whether or not the quality of all read values is good. Based on the value of the enumerator, the "ItemIdentifier[]" should be checked.

Event ReadCompleted

Using the "ReadCompleted" event, the .NET API sends the result of a "ReadAsync" call to the application.

Parameter	Functionality
int transactionHandle	Handle transferred by the application with "ReadAsync".
bool allQualitiesGood	Indicates that there are no values with the quality bad in the "ItemValueCallback" array.
bool noErrors	Indicates that there are no errors in the "ItemValueCallback" array.
ItemValueCallback[] itemValues	Array with the value, quality and timestamp for every item in the "ReadAsync" call. The assignment is made using the "ClientHandle" in the "ValueCallback" item. If the value of "noErrors" is "false", the "ResultID" property of "ItemValueCallback" must also be checked.

To register for the "DataChanged" event of an OPC server, a "Handler" method must be registered for the event. The following code shows an example of how a "Handler" method is registered for the "DataChanged" event:

```

daServerMgt.ReadCompleted += new DaServerMgt.ReadCompletedEventHandler
(MyReadCompleted);

void MyReadCompleted(int transactionHandle,
bool allQualitiesGood,
bool noErrors,
ItemValueCallback[] itemValues)
{
    // ToDo: Error check
    // ToDo: Entry of the values in the application
}

```

Write method

Parameter	Functionality
ref ItemIdentifier[] itemIdentifiers	The "itemIdentifiers" array specifies the OPC items to be written. Any item errors are returned in the "ResultID" object of the relevant "ItemIdentifiers". The parameter is an in/out parameter so that the .NET API can return error information and a "ServerHandle" in the object. If the same items are used more than once for "Read" and "Write" calls, the objects should only be created once and then used for all calls. This allows the .NET API to use the information saved in the object to optimize the OPC server access.
ItemValue[] itemValues	The "itemValues" array contains the values to be written.

Return type	Functionality
ReturnCode	The return value indicates whether or not errors occurred relating to the transferred parameters. Based on the value of the enumerator, the "ItemIdentifier[]" should be checked.

The following code is an example showing how a single item has the integer value 11 or 22 written to it:

```

ItemIdentifier[] itemIdentifiers = new ItemIdentifier[2];
itemIdentifiers[0] = new ItemIdentifier();
itemIdentifiers[0].ItemName = "S7:@LOCALSERVER]DB1,B0";
itemIdentifiers[1] = new ItemIdentifier();
itemIdentifiers[1].ItemName = "S7:@LOCALSERVER]DB1,B1";
ItemValue[] itemValues = new ItemValue[2];
itemValues[0] = new ItemValue();

```

```

itemValues[0].Value = 11;
itemValues[0] = new ItemValue();
itemValues[0].Value = 22;
daServerMgt.Write(ref itemIdentifiers, itemValues);

```

Note

All methods of the .NET API allow group calls, in other words, any number of items can be written with one "Write" call.

Where possible, group calls should be used because this strategy reduces the client-server communication to a minimum on the one hand and makes full use of the optimization mechanisms of the API on the other.

WriteAsync method

Using this method, values are written asynchronously to the OPC server. The information as to whether or not the values were written successfully is returned by the "WriteCompleted" event.

Parameter	Functionality
int transactionHandle	A handle as identification for the "Write" transaction. The handle is returned to the client with the "WriteCompleted" event.
ref ItemIdentifier[] itemIdentifiers	The "itemIdentifiers" array specifies the OPC items to be written. Any item errors are returned in the ResultID object of the relevant "ItemIdentifiers". The parameter is an in/out parameter so that the .NET API can return error information and a "ServerHandle" in the object. If the same items are used more than once for "Read" and "Write" calls, the objects should only be created once and then used for all calls. This allows the .NET API to use the information saved in the object to optimize the OPC server access.
ItemValue[] itemValues	The "itemValues" array contains the values to be written.

Return type	Functionality
ReturnCode	The return value indicates whether or not errors occurred relating to the transferred parameters. Based on the value of the enumerator, the "ItemIdentifier[]" should be checked.

WriteCompleted event

Using the "WriteCompleted" event, the .NET API sends the result of a "WriteAsync" call to the application.

Parameter	Functionality
<code>int transactionHandle</code>	Handle transferred by the application with "WriteAsync".
<code>bool noErrors</code>	Indicates that there are no errors in the "ItemResultCallback" array.
<code>ItemResultCallback[] itemResults</code>	Array with "ClientHandle" and error code for each item in the "WriteAsync" call. The assignment is made using the "ClientHandle" in the "ItemResultCallback". If the value of "noErrors" is "false", the "ResultID" property of "ItemResultCallback" must also be checked.

To register for the "WriteCompleted" event of an OPC server, a "Handler" method must be registered for the event. The following code shows an example of how a "Handler" method is registered for the "WriteCompleted" event:

```
daServerMgt.WriteCompleted += new DaServerMgt.WriteCompletedEventHandler
    (MyWriteCompleted);

void MyWriteCompleted(int transactionHandle,
    bool noErrors,
    ItemResultCallback[] itemResults)
{
    // ToDo: Error check
}
```

Browse method

The Browse method can be used to browse the namespace of an OPC server. The namespace is normally displayed as a tree structure since this representation corresponds to the structure of the items and folders of a server.

Parameter	Functionality
<code>string itemName</code>	The "itemName" parameter specifies the element (directory) for which all child elements will be found. If you want to browse the namespace of the server at the root level, transfer an empty string.
<code>string itemPath</code>	With this parameter, you specify the "ItemPath" of the element for an OPC XML DA server.
<code>ref string continuationPoint</code>	If the number of elements returned by the client (maxElementsReturned parameter) or by the server is restricted to a certain number, the server returns a continuation point with the "continuationPoint" parameter for further "Browse" calls for the same element. If a "ContinuationPoint" is returned by the OPC server, the "Browse" call must be repeated with the same parameters and the returned "ContinuationPoint" to find the remaining child elements.

Parameter	Functionality
int maxElementsReturned	Here, you can specify the maximum number of elements to be returned when the function is called. If the transferred value is 0, all elements are returned.
BrowseFilter browseFilter	Here, a "BrowseFilter" must be defined that describes the type of elements to be returned. ("All", "Items" or "Branch")
int[] propertyIDs	Here, the IDs of the properties to be found when the "Browse" method is called can be transferred. The properties are returned in the relevant "BrowseElement".
bool returnAllProperties	If this flag is set, all properties of an item are obtained automatically. The properties are returned in the relevant "BrowseElement".
bool returnPropertyValues	If this flag is set, the values of the requested properties are also returned.
out BrowseElement[] browseElements	This array contains the child elements of the transferred element in the address space.
out bool moreElements	"moreElements" indicates whether all child elements were returned.

Return type	Functionality
ReturnCode	The return value indicates whether or not errors occurred relating to the transferred parameters. Based on the value of the enumerator, the "BrowseElement[]" should be checked.

The following code shows an example of how to find all items and folders at the root level without corresponding properties:

```

BrowseElement[] browseElements = null;

bool moreElements = false;
bool returnAllProperties = false;
bool returnPropertyValues = false;
string continuationPoint = null;

daServerMgt.Browse(string.Empty,
    string.Empty,
    ref continuationPoint,
    0,
    BrowseFilter.ALL,
    null,
    returnAllProperties,
    returnPropertyValues,
    out browseElements,
    out moreElements);

```

GetProperties method

Using the "GetProperties" method, properties of the OPC item can be obtained.

Parameters	Functionality
ref ItemIdentifier[] itemIdentifiers	The "itemIdentifiers" parameter describes the OPC items for which the properties will be found.
int[] propertyIDs	Here, you can transfer the IDs of the properties to be found when the "GetProperties" method is called. The properties are returned in the corresponding "ItemProperties" element.
bool returnAllProperties	If this flag is set, all properties of an item are obtained automatically. The properties are returned in the corresponding "ItemProperties" element.
out ItemProperties[] itemProperties	This array contains the requested "ItemProperties" objects. The properties in an "ItemProperties" object relate to the corresponding "ItemIdentifier" object with the same array index.

Return type	Functionality
ReturnCode	The return value indicates whether or not errors occurred relating to the transferred parameters. Based on the value of the enumerator, the "ItemIdentifier[]" should be checked.

The following code shows an example of how the properties "AccessRights" and "DataType" of the "Opcltem1" item can be obtained:

```
ItemProperties[] itemProperties = null;
ItemIdentifier[] itemIdentifiers = new ItemIdentifier[1];
itemIdentifiers[0] = new ItemIdentifier();
itemIdentifiers[0].ItemName = "S7:@LOCALSERVER]DB1,B0";
Int32[] propertyIDs = new Int32[2];
propertyIDs[0] = (Int32)PropertyID.ACCESSRIGHTS;
propertyIDs[1] = (Int32)PropertyID.DATATYPE;

daServerMgt.GetProperties(ref itemIdentifiers,
    ref propertyIDs,
    false,
    true,
```

```
    out itemProperties);
```

Note

All methods of the .NET API allow group calls, in other words, any number of items can be queried with one "GetProperties" call.

Where possible, group calls should be used because this strategy reduces the client-server communication to a minimum on the one hand and makes full use of the optimization mechanisms of the API on the other.

Subscribe method

Using the "Subscribe" method, items can be registered for monitoring value changes.

Before you create the first "Subscription", the application must register for the "DataChanged" event of the .NET API. This registration is described with the "DataChanged" event.

Parameters	Functionality
int clientSubscription	Using "clientSubscription", the application can specify an index for the "Subscription". This handle is required when several "Subscriptions" are registered and the application wants to distinguish the "Subscriptions" in the "DataChanged" event.
bool active	With the "active" flag, the subscription can be created as active or inactive.
int updateRate	The "UpdateRate" parameter specifies the cycle in which value changes are reported. The "UpdateRate" is specified in milliseconds.
out int revisedUpdateRate	This out parameter returns the "UpdateRate" set by the server. This can deviate from the requested "UpdateRate".
float deadband	The "Deadband" parameter specifies the minimum deviation which must be exceeded before a value change is reported. The value is a percentage deviation (value between 0.0 and 100.0) of the value range after which a change is reported. The range of values is specified with the properties "EULow" and "EUHigh" of the OPC item. The "Deadband" is normally only used with analog values.
ref ItemIdentifier[] itemIdentifiers	The OPC items to be inserted in the "Subscription" are specified with the "itemIdentifiers" array. Any item errors are returned in the "ResultID" object of the relevant "ItemIdentifiers". The parameter is an in/out parameter so that the .NET API can return error information and "ServerHandles" in the object. The objects are required to be able to remove individual items from the "Subscription".
out int serverSubscription	In the "serverSubscription" parameter, the .NET API returns the handle for the "Subscription" with which this can be deleted again using the "CancelSubscription" method.

Return type	Functionality
ReturnCode	The return code indicates whether or not errors occurred relating to the transferred parameters and whether or not the "UpdateRate" is supported by the server. Based on the value of the enumerator, the "ItemIdentifier[]" should be checked.

The following code shows an example of how a "Subscription" is created:

```
ItemIdentifier[] itemIdentifiers = new ItemIdentifier[10];

for (int i = 0; i < 10; i++)
{
    itemIdentifiers[i] = new ItemIdentifier();
    itemIdentifiers[i].ItemName = "S7:[@LOCALSERVER]DB1,B" + i.ToString();
    itemIdentifiers[i].ClientHandle = i;
}

int clientSubscription = 1;
bool active = false;
int updateRate = 500;
int revisedUpdateRate;
float deadBand = 0;
int serverSubscription;

daServerMgt.Subscribe(clientSubscription,
    active,
    updateRate,
    out revisedUpdateRate,
    deadBand,
    ref itemIdentifiers,
    out serverSubscription);
```

Note

To be able to process all "DataChanged" events, the "Eventhandler" of the application must be registered with .NET API before the first subscription is generated.

SubscriptionModify method

Using the "SubscriptionModify" method, properties of a subscription can be changed.

Parameters	Functionality
int serverSubscription	Identifies the subscription in the .NET API. This handle is returned when the subscription is created with the "Subscribe" method.
bool active	The subscription can be switched active or inactive with the "active" flag. If zero is transferred, the property is not changed.
int updateRate	The "updateRate" parameter specifies the cycle in which value changes are reported. The "UpdateRate" is specified in milliseconds. If zero is transferred, the property is not changed.
out int revisedUpdateRate	This out parameter returns the "UpdateRate" set by the server. This can deviate from the requested "UpdateRate".
float deadband	The "deadband" parameter specifies the minimum deviation which must be exceeded before a value change is reported. The value is a percentage deviation (value between 0.0 and 100.0) of the value range after which a change is reported. The range of values is specified with the properties "EULow" and "EUHigh" of the OPC item. The "Deadband" is normally only used with analog values. If zero is transferred, the property is not changed.

Return type	Functionality
ReturnCode	The return code indicates whether or not errors occurred relating to the transferred parameters and whether or not the "UpdateRate" is supported by the server. Based on the value of the enumerator, the "ItemIdentifier[]"

There are an additional 3 overloads of the "SubscriptionModify" method available in which the settings "Active", "UpdateRate" and "Deadband" of the "Subscription" can be changed individually.

SubscriptionAddItems method

Using the "SubscriptionAddItems" method, items can be added to an existing subscription.

Parameters	Functionality
int serverSubscription	Identifies the subscription in the .NET API. This handle is returned when the subscription is created with the "Subscribe" method.
ref ItemIdentifier[] itemIdentifiers	The OPC items to be inserted in the "Subscription" are specified with the "itemIdentifiers" array. Any item errors are returned in the "ResultID" object of the relevant "ItemIdentifiers". The parameter is an in/out parameter so that the .NET API can return error information and "ServerHandles" in the object. The objects are required to be able to remove individual items from the "Subscription".

Return type	Functionality
ReturnCode	The return value indicates whether or not errors occurred relating to the transferred parameters. Based on the value of the enumerator, the "ItemIdentifier[]" should be checked.

SubscriptionRemoveItems method

Using the "SubscriptionRemoveItems" method, items can be removed from an existing subscription.

Parameters	Functionality
int serverSubscription	Identifies the subscription in the .NET API. This handle is returned when the subscription is created with the "Subscribe" method.
ref ItemIdentifier[] itemIdentifiers	The OPC items to be removed from the "Subscription" are specified with the "itemIdentifiers" array. Any item errors are returned in the "ResultID" object of the relevant "ItemIdentifiers". The same objects that are to be returned by "Subscribe" or "SubscriptionAddItems" should be used here to identify the correct objects with the "ServerHandle". The parameter is an in/out parameter so that the .NET API can return error information in the object.

Return type	Functionality
ReturnCode	The return value indicates whether or not errors occurred relating to the transferred parameters. Based on the value of the enumerator, the "ItemIdentifier[]" should be checked.

SubscriptionCancel method

Using the "SubscriptionCancel" method, a subscription can be deregistered for "DataChanged". At the same time, the "serverSubscription" parameter contained in "Subscribe" must be transferred.

Parameters	Functionality
int serverSubscription	Identifies the subscription in the .NET API. This handle is returned when the subscription is created with the "Subscribe" method.

The following code shows an example of how a "Subscription" is canceled again:

```
daServerMgt.SubscriptionCancel(serverSubscription);
```

DataChanged event

Using the "DataChanged" event, the .NET API sends changed values for items of a "Subscription" to the application.

Parameter	Functionality
int clientSubscription	Handle transferred by the application when the subscription is created.
bool allQualitiesGood	Indicates that there are no values with the quality bad in the "ItemValueCallback" array.
bool noErrors	Indicates that there are no errors in the "ItemValueCallback" array.
ItemValueCallback[] itemValues	Array with the value, quality and timestamp for every changed item. The assignment is made using the "ClientHandle" in the "ItemValueCallback". If the value of "noErrors" is "false", the "ResultID" property of "ItemValueCallback" must also be checked.

To register for the "DataChanged" event of an OPC server, a "Handler" method must be registered for the event. The following code shows an example of how a "Handler" method is registered for the "DataChanged" event:

```
daServerMgt.DataChanged += new DaServerMgt.DataChangedEventHandler(MyDataChanged);
```

```
void MyDataChanged(int clientSubscription,
                   bool allQualitiesGood,
                   bool noErrors,
                   ItemValueCallback[] itemValues)
{
    if (noErrors)
    {
        if (allQualitiesGood)
        {
            // All items are correct

            // ToDo: Entry of the values in the application
        }
        else
        {
            // The quality is not good for at least one item

            // ToDo: Troubleshooting
        }
    }
    else
    {
```

```

    // At least one item has an error

    // ToDo: Troubleshooting

}

}

```

ServerStateChanged event

With the "ServerStateChanged" event, the .NET API informs the application of changes to the connection status.

Parameter	Functionality
int clientHandle	Handle transferred by the application with "Connect" for the connection to the OPC server.
ServerState state	Current status of the connection to the OPC server.

To register for the "ServerStateChanged" event of an OPC server, a "Handler" method must be registered for the event. The following code shows an example of a registration of a "Handler" method for the "ServerStateChanged" event:

```

daServerMgt.ServerStateChanged += new
DaServerMgt.ServerStateChangedEventHandler (MyServerStateChanged) ;

void MyServerStateChanged(int clientHandle,
                         ServerState state)
{
    // Identification of the connection
    // ToDo: Reaction to the new status of the server
}

```

5.2.1.3 Troubleshooting

Any errors that occur are reported to the user with exceptions. Here, two types of exceptions are used. Errors resulting from transferring bad parameters, null parameters or illegal combinations of parameters are returned by the .NET API system exceptions. Errors returned by the underlying OPC server are returned by the .NET API as OPC exceptions.

OPCExceptions

The "OpcException" class is derived from "ApplicationException". As an additional parameter, "OPCException" has a "ResultID".

SystemExceptions

The .NET OPC API generates three different classes of "SystemExceptions".

- ArgumentNullException – is used when a method does not permit a zero parameter.
- ArgumentOutOfRangeException – for example when a number > 100 is set as "Deadband".
- ArgumentException – for example when calling Write(), the arrays "ItemIdentifiers" and "Values" are not the same length.

```

int ErrorCode;
string ErrorName;
string ErrorDescription;
try
{
    daServerMgt.Write( ref itemIdentifiers,
    itemValues );
}
catch(OPCException opcex)
{
    // Error
    ErrorCode = opcex.ResultID.Code;
    ErrorName = opcex.ResultID.Name;
    ErrorDescription = opcex.ResultID.Description;

    // OPC-specific error handling
    // ToDo. . .
}
catch(System.ArgumentException sysarex)
{
    // Error
    ErrorDescription = sysarex.Message;
    // Special error handling for ArgumentException
    // ToDo. . .
}
catch(System.Exception)

{
    // Error handling for unexpected exception
    // ToDo. . .
}

```

5.2.2

Example of namespace SIMATICNET.OPCDACLIENT

This example describes how a client that uses the OPCDACLIENT API can connect to various OPC server types.

You will find detailed information in the sections shown.

Example of connection establishment to OPC DA and OPC UA servers (with/without encryption)

```

//Server instance initialization
//Information can be found on this in the section "DaServerMgt"
DaServerMgt m_Server = new DaServerMgt();

//Server identifier instance initialization
//The section "ServerIdentifier class" contains information
//about this
ServerIdentifier m_ServerId =
    new ServerIdentifier(m_EndpointUrl, m_EndpointIdentifier);

//Only connect to server if no connection exists yet
if (m_Server.IsConnected == false)
{
    //Fill in ConnectInfo instance
    //The section "ConnectInfo class" contains information
    //about this
    ConnectInfo connectInfo = new ConnectInfo();
    connectInfo.RetryAfterConnectionError = true;
    connectInfo.RetryInitialConnection = true;
    connectInfo.KeepAliveTime = 1000;

    //Set UA-specific parameters if the server to be connected
    //is a UA server
    if (m_ServerId.Category == ServerCategory.OPCUA)
    {
        //Fill in ConnectInfo instance
        //The section "ConnectInfo class" contains information
        //about this
        connectInfo.MessageSecurityMode =
            m_ServerId.Endpoint.MessageSecurityMode;
        connectInfo.SecurityPolicyUri =
            m_ServerId.Endpoint.SecurityPolicyUri;
        connectInfo.ServerCertificate =
            m_ServerId.Endpoint.ServerCertificate;

        //Setting parameters necessary for connection establishment
        //with encrypted UA servers
        if (1 < connectInfo.MessageSecurityMode)
        {
            //Fill in ConnectInfo instance
            //The section "ConnectInfo class" contains information
            //about this
            //Example of storage path in Windows store
            connectInfo.CertificateStoreName =
                Application.ProductName;
            connectInfo.CertificateStoreLocation =
                WinStoreLocation.CurrentUser;

            //Certificate handling is performed so that
            //each time the application starts
            //a new certificate does not need to be generated that in
            turn
            //needs to be accepted by the server.
    }
}

```

```

//To achieve this, the thumbprints of the certificates used
//must be stored, for example in the file system.

#region Certificate Handling Client

//The thumbprint is necessary to find the client
certificate
    //again in the Windows store
    byte[] clientThumbprint    = null;
    //Client certificate - Information on this can be found
    //in the section "PkiCertificate class"
    PkiCertificate cert_Client = null;

//ToDo: Reading out the stored thumbprint of the client
// "clientThumbprint" must occur here

//Fetch existing certificate from the Windows store
if (clientThumbprint != null)
{
    try
    {
        cert_Client =
            PkiCertificate.fromWindowsStoreWithPrivateKey(
                connectInfo.CertificateStoreLocation,
                connectInfo.CertificateStoreName,
                clientThumbprint);
    }
    catch
    {
        cert_Client = null;
    }
}

//If there is not yet a certificate in the Windows store
//it needs to be created
if (cert_Client == null)
{
    string Computername =
        System.Windows.Forms.SystemInformation.ComputerName;
    cert_Client = new PkiCertificate(
        Computername + ":CLIENTAPISAMPLE",
        "", Computername, 94608000,
        "CLIENTAPISAMPLE",
        "SIEMENS AG", "UNIT", "LOCAL",
        "COUNTRY", "LANG", 1024);
    cert_Client.toWindowsStoreWithPrivateKey(
        connectInfo.CertificateStoreLocation,
        connectInfo.CertificateStoreName);

    //ToDo: Save the thumbprint
    //"cert_Client.Thumbprint"
    //must occur here
}

#endregion

```

```

#region Certificate Handling Server
    //The thumbprint is necessary to find the server
certificate
    //again in the Windows store
    byte[] serverThumbprint      = null;
    //Server certificate - Information on this can be found
    //in the section "PkiCertificate class"
    PkiCertificate cert_Server = null;

    //ToDo: Reading out the stored server thumbprint
    //"serverThumbprint" must occur here

    //Fetch existing certificate from Windows store
    if (serverThumbprint != null)
    {
        try
        {
            cert_Server =
                PkiCertificate.fromWindowsStore(
                    connectInfo.CertificateStoreLocation,
                    connectInfo.CertificateStoreName,
                    serverThumbprint);
        }
        catch
        {
            cert_Server = null;
        }
    }

    //If there is not yet a certificate in the Windows store
    //the certificate of the server to be connected must
    //be stored in the same Windows Store as the client
    //certificate so that it counts as accepted
    if (cert_Server == null)
    {
        //Here it is possible to query whether the new
        //certificate will be accepted

        cert_Server =
            PkiCertificate.fromDER(connectInfo.ServerCertificate);

        cert_Server.toWindowsStore(
            connectInfo.CertificateStoreLocation,
            connectInfo.CertificateStoreName);

        //ToDo: Save the thumbprint
        //"cert_Server.Thumbprint" must occur here
    }
    else
    {
        //Here it is possible to query whether the current
        //server certificate and the stored certificate
        //match
    }
}

```

```

        #endregion

        //Set private key
        connectInfo.ClientPrivateKey = cert_Client.PrivateKey;

        //Encrypt certificate
        connectInfo.ClientCertificate = cert_Client.toDER();
    }
}

bool connectFailed;

//Send connection job to the server
m_Server.Connect(m_ServerId.Url, m_ClientHandle,
                  ref connectInfo, out connectFailed);

if (connectFailed)
{
    //Here, errors in the connection establishment to the server
    //can be handled
}
else
{
    //Server is already connected
}

```

5.2.3 SIMATICNET.OPCCMN namespace

The "SimaticNet.OpcCmn" namespace groups together general functionalities for OPC. In particular, this involves browsing for OPC servers and obtaining information necessary for connection establishment.

5.2.3.1 The interface of the OPCServerEnum object

Using the "OpcServerEnum" class, COM OPC servers can be searched for on a computer. Information can also be obtained that is required to establish the connection to the OPC server.

Generating the OPCServerEnum object

To be able to use the "OpcServerEnum" class, an instance of the class must first be generated.

The following code shows how an "OpcServerEnum" object is created:

```
OpcServerEnum opcServerEnum = new OpcServerEnum();
```

EnumComServers method

The "EnumComServers" method is used to find the OPC servers on a computer. Here it is possible to distinguish between the different OPC specifications, for example "OPC Data Access", "OPC Historical Data Access" or "OPC Alarms & Events".

Parameter	Functionality
string nodeName	The name or the IP address of the computer to be browsed for OPC servers. (for example localhost, PCTest, 192.168.0.120 etc.)
bool returnAllServers	Specifies whether or not all OPC servers on the computer should be returned. If this parameter is set to "true", the "serverCategories" array is not evaluated.
ServerCategory[] serverCategories	With this parameter, you can specify what types of OPC servers will be returned.
out ServerIdentifier[] serverIdentifiers	A list of "ServerIdentifier" objects. The objects contain the information required to establish the connection with the OPC server.

The following code returns all the OPC Data Access servers on the local computer:

```
ServerCategory[] serverCategories = new ServerCategory[1];
serverCategories[0] = ServerCategory.OPCDA;
ServerIdentifier[] serverIdentifier = null;

opcServerEnum.EnumComServer("localhost",
false,
serverCategories,
out serverIdentifier);
```

ClsidFromProgID method

The "ClidFromProgId" method obtains the corresponding "CLSID" of the OPC server depending on the computer name and "ProgID" parameters transferred.

Parameter	Functionality
string nodeName	The name or IP address of the computer on which the "CLSID" will be obtained. (for example localhost, PCTest, 192.168.0.120 etc.)
string progID	The "ProgID" for which the corresponding "CLSID" will be obtained.
out string clsID	The requested "CLSID"

The following code returns the "CLSID" for "ProgID OPC.SimaticNET" on the local computer:

```
string progID = "OPC.SimaticNET";
string clsID = null;
opcServerEnum.ClsidFromProgId("localhost",
progID,
```

```
    out clsID);
```

getCertificateForEndpoint method

The "getCertificateForEndpoint" method connects itself to the transferred endpoint URL and loads the server certificate, that is returned as Out parameter. The In parameters can be found prior to this by browsing on discovery server.

Parameter	Functionality
string endpointUrl	The URL of the UA endpoint (e.g. opc.tcp://localhost:4841)
string securityPolicyUri	The SecurityPolicy of the endpoint (e.g. "http://opcfoundation.org/UA/SecurityPolicy#Basic128Rsa15")
Byte messageSecurityMode	The MessageSecurityMode of the endpoint None, Sign or SignAndEncrypt
out byte[] serverCertificate	The certificate of the OPC UA server.

The following Code loads the certificate from the endpoint specified by the parameters:

```
string url = "opc.tcp://localhost:55101";
string securityPolicyUri =
"http://opcfoundation.org/UA/SecurityPolicy#Basic128Rsa";
byte messageSecurityMode = 3; // Sign and Encrypt
byte[] serverCertificate = null;
opcServerEnum.getCertificateForEndpoint( Url,
                                         securityPolicyUri,
                                         messageSecurityMode
                                         out serverCertificate);
```

5.2.3.2 ServerIdentifier class

The "ServerIdentifier" class returns all data required for the connection establishment to an OPC server.

Properties of the "ServerIdentifier" class:

- **string** Url
The full URL of the OPC server. This can, for example, be transferred with the "Connect" method. See also structure of the Url.
- **ServerCategory** Category
The OPC category in which the server will be classified, for example OPC DA, OPC DX, etc.
- **string** ProgID
The "ProgID" of the OPC server.
- **string** CLSID
The "CLSID" of the OPC server.
- **string** HostName
The name of the computer on which the OPC server was found.
- **EndpointIdentifier**
Additional information necessary for a connection to an OPC UA server.

5.2.3.3 ServerCategory enumerator

With the "ServerCategory" enumerator, the type of OPC server can be specified. It contains the following elements:

- OPCDA
Corresponds to the OPC specifications "OPC DA 2.05A" and "OPC DA 3.00"
- OPCUA
Corresponds to the OPC Unified Architecture specification
- OPCXMLDA
Corresponds to the OPC specification "OPC XML DA 1.01" (not currently supported)
- OPCDX
Corresponds to the OPC specification "OPC DX 1.00" (not currently supported)
- OPCAE
Corresponds to the OPC specification "OPC AE 1.10" (not currently supported)
- OPCHDA
Corresponds to the OPC specification "OPC HDA 1.10" (not currently supported)

Note

Since unlike COM OPC servers, OPC XML DA servers are not registered on a computer, these cannot be obtained using "OpcServerEnum".

To connect to an OPC XML DA server, the URL must be known.

5.2.3.4 EndpointIdentifier class

The "EndpointIdentifier" class returns all data required in addition to that for the connection establishment to a UA server.

Properties of the "EndpointIdentifier" class:

- string SecurityPolicy
The URI of the SecurityPolicy used to secure the connection.
- byte MessageSecurityMode
The type of connection security at message level.
- byte ServerCertificate
The X509 certificate of the OPC-UA server.
- string ApplicationUri
The URI of the OPC UA server application.
- string ProductUri
The product URI of the OPC UA server application.
- string ApplicationName
The name of the OPC UA server.

5.2.3.5 PkiCertificate class

The "PkiCertificate" class encapsulated an X509 certificate and allows simple generation of such a certificate. Methods for access to the "WindowsCertificateStore" for loading and saving of certificates are also made available. The fields of the certificate can only be read using Properties.

5.2.3.6 Creating a new certificate

The "PkiCertificate" class provides a constructor as a parameter that holds all the information required to generate the certificate.

Parameters	Functionality
string URI	The "ApplicationURI" of the certificate. A unique identifier for the certificate, for example: urn:<computername>:<company>:<productname>
string IP	The IP address of the computer on which the application runs. This is only used if no "domain name" is available.
string DNS	The "domainname" of the computer (computer name), on which the application runs.
int validTime	The time in seconds that the certificate will be valid.
string CommonName	The display name of the certificate.
string Organization	Organization or company e.g. Siemens
string OrganizationUnit	Department within the organization or company.
string Locality	Location of the issuer.
string State	State or national state of the issuer.
string Country	Country code e.g. DE, US, ...
int KeyStrength	Length of the key e.g. 1024 bits, 2048 bits ...

The following code shows an example of how a new "PkiCertificate" can be generated:

```
PkiCertificate clientCert = new PkiCertificate(
    "urn:my_machine:Siemens:SimaticNETSampleApplication",
    "",
    "Computer1",
    31536000, // 1 year = 31536000 seconds
    "SampleApplication",
    "Siemens",
    "SimaticNET",
    "Karlsruhe",
    "Baden-Württemberg",
    "DE",
    2048);
```

5.2.3.7 toDER method

With this method, you can generate a DER-coded byte array from a "PkICertificate". Certificates in the form of a DER-coded byte array are, for example, required in the EndpointIdentifier class.

Return type	Functionality
<code>byte[]</code>	Byte array that contains the DER-coded data of the certificate or zero if the conversion fails.

The following code shows an example of how the DER-coded byte array is generated from an existing PkICertificate:

```
// Load client certificate from Windows Store
PkICertificate clientCert;
...
byte[] clientCertificateDER = clientCertificate.toDER();
```

5.2.3.8 fromDER method

With this static method, a "PkICertificate" can be generated from a DER-coded byte array.

Parameters	Functionality
<code>byte[] DERData</code>	The DER-coded byte array that contains the data of an X509 certificate.

Return type	Functionality
<code>PkICertificate</code>	The newly created "PkICertificate" or null, if the conversion fails.

The following code shows an example of how "PkICertificate" can be generated from an existing byte[]:

```
Byte[] DERdata;
...
PkICertificate cert = PkICertificate.fromDER(DERdata);
```

5.2.3.9 toWindowsStore method

With this method the "PkICertificate" is saved in the "WindowsCertificateStore".

Parameters	Functionality
<code>WinStoreLocation location</code>	User context for saving in the "WindowsCertificateStore".
<code>string StoreName</code>	Name of the "Store" for saving in the "WindowsCertificateStore".

As an example, the following code shows how an existing "PkiCertificate" is saved in the "WindowsCertificateStore":

```
PkiCertificate cert;
...
Cert.toWindowsStore(WinStoreLocation.CurrentUser, "UA
Applications");
```

Note

We recommend that you store the certificates in the "WinStoreLocation.LocalMachine" under StoreName "UA Applications" since this is suggested by the OPC Foundation.

5.2.3.10 toWindowsStoreWithPrivateKey method

With this method the "PkiCertificate" including the private key is saved in the "WindowsCertificateStore".

Parameters	Functionality
WinStoreLocation location	User context for saving in the "WindowsCertificateStore".
string StoreName	Name of the "Store" for saving in the "WindowsCertificateStore".

As an example, the following code shows how an existing "PkiCertificate" is saved in the "WindowsCertificateStore":

```
PkiCertificate cert;
...
Cert.toWindowsStore(WinStoreLocation.CurrentUser, "MyStoreName");
```

5.2.3.11 fromWindowsStore method

With this method, a "PkiCertificate" is loaded from the "WindowsCertificateStore".

Parameters	Functionality
WinStoreLocation location	User context for loading from the "WindowsCertificateStore".
string StoreName	Name of the "Store" for loading from the "WindowsCertificateStore".
byte[] Thumbprint	The "Thumbprint" of the certificate. This can, for example, be loaded from a configuration file.

Return type	Functionality
PkiCertificate	The newly created "PkiCertificate" or null, if the loading fails.

As an example, the following code shows how a "PkiCertificate" is loaded from the "WindowsCertificateStore":

```
byte[] Thumbprint;
// ... load Thumbprint from configuration ...
PkiCertificate cert = PkiCertificate.fromWindowsStore(
    WinStoreLocation.CurrentUser,
    "MyStoreName",
    Thumbprint);
```

5.2.3.12 fromWindowsStoreWithPrivateKey method

With this method, a "PkiCertificate" and the associated private key is loaded from the "WindowsCertificateStore". The private key is a property of the "PkiCertificate".

Parameters	Functionality
WinStoreLocation location	User context for loading from the "WindowsCertificateStore".
string StoreName	Name of the "Store" for loading from the "WindowsCertificateStore".
byte[] Thumbprint	The "Thumbprint" of the certificate. This can, for example, be loaded from a configuration file.

Return type	Functionality
PkiCertificate	The newly created "PkiCertificate" or null, if the loading fails.

As an example, the following code shows how a "PkiCertificate" and the associated private key are loaded from the "WindowsCertificateStore".

```
byte[] Thumbprint;
// ... load Thumbprint from configuration ...
PkiCertificate cert = PkiCertificate.fromWindowsStoreWithPrivateKey(
    WinStoreLocation.CurrentUser,
    "MyStoreName",
    Thumbprint);
```

5.2.3.13 fromWindowsStoreWithPrivateKey method

With this method, a "PkiCertificate" and the associated private key is loaded from the "WindowsCertificateStore". The private key is a property of the "PkiCertificate".

Parameters	Functionality
WinStoreLocation location	User context for loading from the "WindowsCertificateStore".
string StoreName	Name of the "Store" for loading from the "WindowsCertificateStore".
string ApplicationURI	The "Thumbprint" of the certificate. This can, for example, be loaded from a configuration file.

Return type	Functionality
PkiCertificate	The newly created "PkiCertificate" or null, if the loading fails

As an example, the following code shows how a "PkiCertificate" and the associated private key are loaded from the "WindowsCertificateStore".

```
String sApplicationUri;
// ... load ApplicationUri from configuration ...
PkiCertificate cert = PkiCertificate.fromWindowsStoreWithPrivateKey(
    WinStoreLocation.CurrentUser,
    "MyStoreName",
    sApplicationUri);
```

5.2.3.14 Properties

The properties for this class are all read-only. They represent the different boxes of the X509 certificate maintained internally.

- PrivateKey [readonly]
- CommonName [readonly]
- Thumbprint [readonly]
- IPAddress [readonly]
- ApplicationURI [readonly]
- ValidFrom [readonly]
- ValidTo [readonly]
- SerialNumber [readonly]
- SignatureAlgorithm [readonly]
- CipherStrength [readonly]
- Subject [readonly]
- Issuer [readonly]

5.2.3.15 Enumerator WinStoreLocation

Using "Enumerator WinStoreLocation", you can specify the user context under which the certificates in the "WindowsCertificateStore" are loaded or saved.

It contains the following elements:

- LocalMachine
Certificates are stored at the following location in the registry:
"HKEY_LOCAL_MACHINE\Software\Microsoft\SystemCertificates"
- CurrentUser
Certificates are stored at the following location in the registry:
"HKEY_CURRENT_USER\Software\Microsoft\SystemCertificates"
- CurrentService
Certificates are stored at the following location in the registry:
"HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Services\<ServiceName>\SystemCertificates"
- Services
Certificates are stored at the following location in the registry:
"HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\Services\<ServiceName>\SystemCertificates"
- Users
Certificates are stored at the following location in the registry:
"HKEY_USERS\<UserName>\Software\Microsoft\SystemCertificates"

Sample programs

This section contains sample programs that use the Custom or Automation Interface.

After you have installed the SIMATIC NET PC software, you will find the sample programs in: "<installationpath>\SIEMENS\SIMATIC.NET\opc2\samples"

6.1 OPC Automation interface (synchronous communication) in VB.NET

This sample involves Visual Basic and uses the Automation Interface for Data Access V2.0 of OPC to read and write data synchronously.

The description is divided into the following sections:

6.1.1 Activating the simulation connection

Before you can run this program, you must activate a simulation connection that makes the demonstration variables used in the program available. Follow the steps outlined below:

1. Start the "Communication Settings" configuration program from the Start menu:
"Start" > "All Programs" > "Siemens Automation" > "SIMATIC" > "SIMATIC NET" > "Communication Settings".
Reaction: The "Communication Settings" configuration program opens.
2. In the left-hand navigation window, open the entry "OPC protocol selection".
"SIMATIC NET Settings" > "OPC Settings" > "OPC Protocol Selection".
3. Enable the check box for the protocol to be simulated.

This example uses the S7 protocol.

For this reason, select the check box under "Name: S7" and click on the arrow symbol next to the S7 protocol.

Reaction: The expanded parameter list is opened.

4. Select the "Provide virtual module (DEMO) for the simulation" check box.
5. Confirm with "Apply".
6. Close the "Communication Settings" configuration program.

Note

Before the changes take effect, you must first close all OPC clients!

6.1.2 Working with the sample program

The program is on your hard disk in
"<installationpath>\SIEMENS\SIMATIC.NET\opc2\samples\automation\sync.net"

When you start the program, only the "Start Sample" button is enabled:

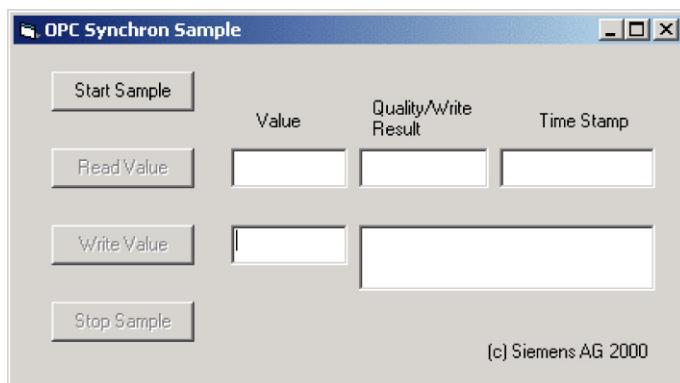


Figure 6-1 Dialog after starting the sample program for the OPC automation interface

After you have clicked the "Start Sample" button, the program generates the required OPC objects. You can then use the other buttons.

After you click the "Read Value" button, the values that were read are displayed in the relevant text boxes:

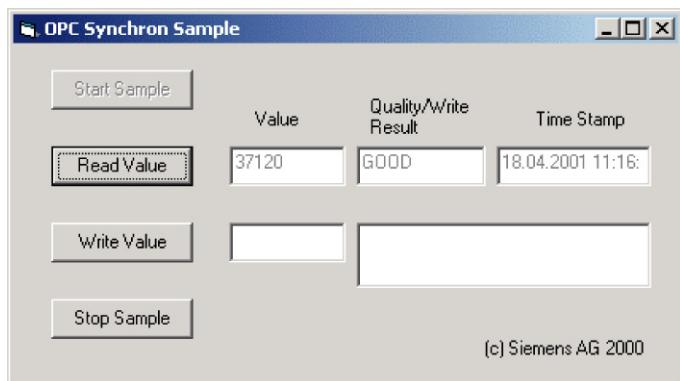


Figure 6-2 Dialog after clicking the "Read Value" button

As it stands, the sample program is designed to operate with a demonstration connection. If you want to run the sample program in a real environment, you must change the following line in the program code:

```
Set ItemObj = GroupObj.OPCItems.AddItem("S7:[DEMO]MW1", 1)
```

You can refer to several examples and section "OPC process variables for SIMATIC NET (Page 21)" for more information on the structure of the ItemIDs.

To write a value, this must be entered in the relevant text box. After clicking the "Write Value" button, the program displays a message about the result of this operation:

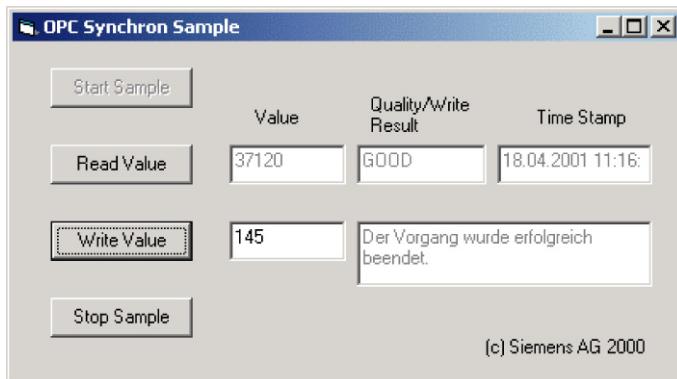


Figure 6-3 Dialog after writing a value to the text box and clicking the "Write Value" button

The "Stop Sample" button closes the program; in other words, all objects are deleted and the corresponding resources are released again.

6.1.3 How the program runs

Due to the class model of OPC, a specific order must be adhered to in the method calls of OPC objects. To be able to create an instance of the *OPCItem* class, an object of the *OPCGroup* class is necessary. This is, however, only possible when an instance of the *OPCServer* class exists and a connection to this server has been established.

The basic sequence of instructions for creating and deleting OPC objects is illustrated in the following graphic. The variable names of the sample program are used.

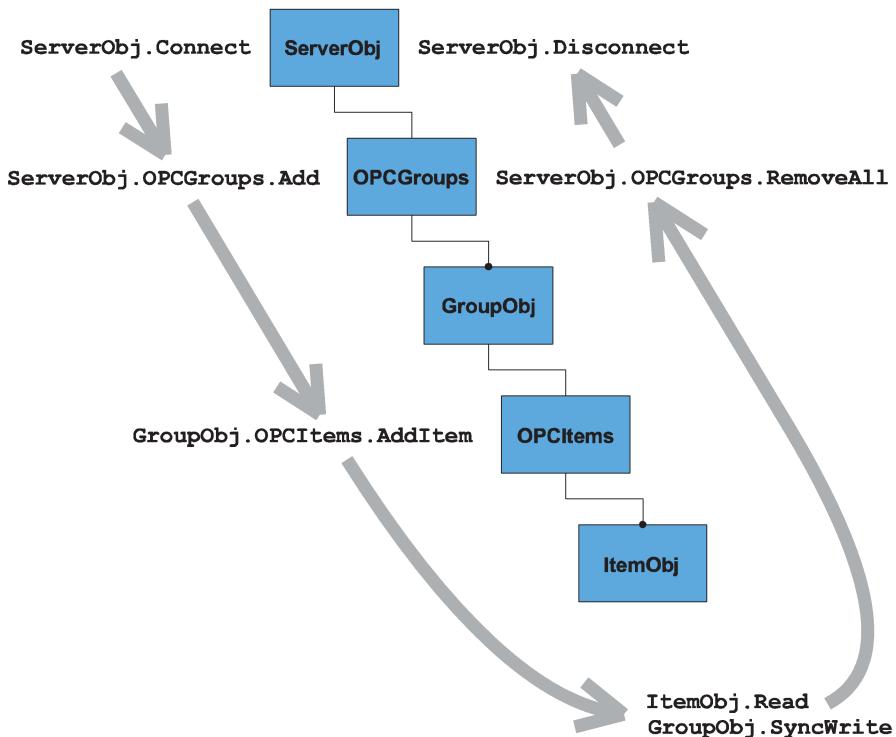


Figure 6-4 Sequence of commands for creating and deleting OPC objects

The sample program contains all the components that would normally exist in a typical client application. These include establishing the connection to the OPC server, creating a group with variables, and reading and writing values for an item.

6.1.4 Using the OPC automation interface with .NET-Framework

Introduction

This section describes the use and migration of the sample program for synchronous reading and writing via the OPC Automation interface using .NET-Framework for Visual Basic.NET.

After installing the SIMATIC NET software, you will find this program on your hard disk in the following folder:

"<installation path>\SIEMENS\SIMATIC.NET\opc2\samples\automation\sync.net"

How to work with the sample and how it executes is unchanged from the previous sample. The following section describes only the necessary changes.

Requirements for using the sample program

A .NET-Framework version 4.0 or higher must be installed on the computer on which you want the sample program to run.

In the existing VB.NET project, the COM component "Siemens OPC DAAutomation 2.0" has already been added:

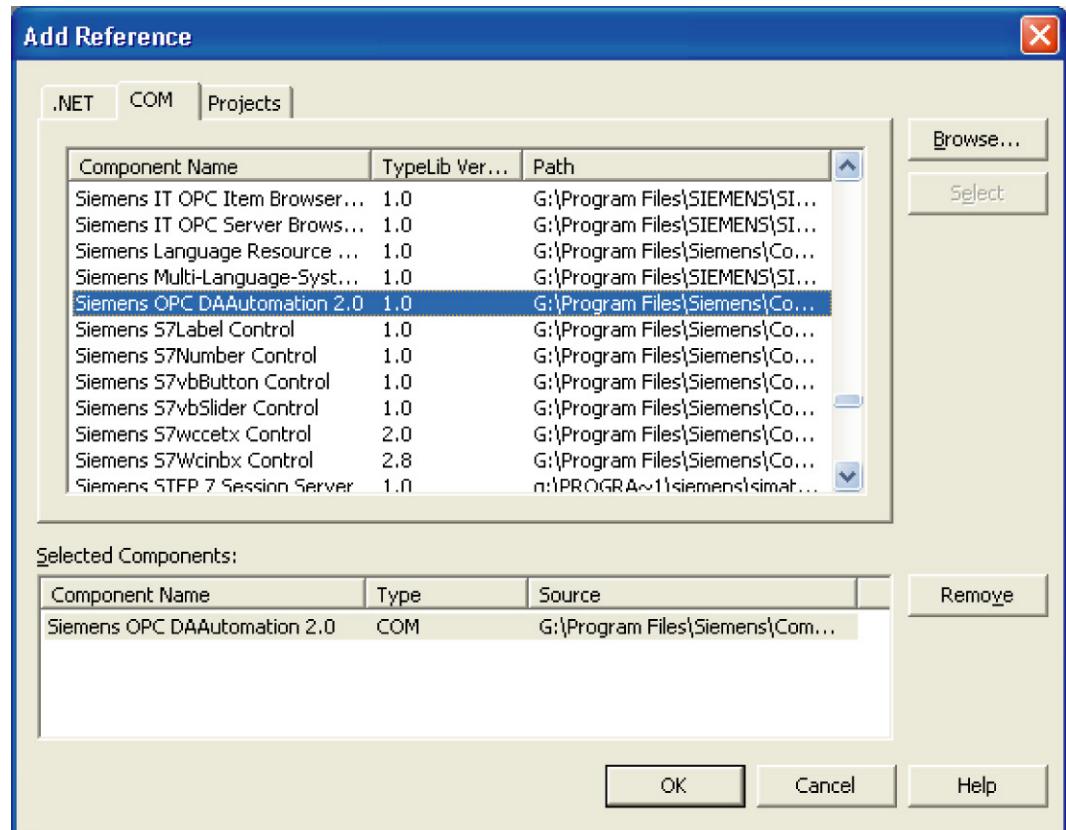


Figure 6-5 Adding the "Siemens OPC DAAutomation 2.0" COM component in the VB.NET project

With the command

```
Imports OPCSiemensDAAutomation
```

it is simple to use the namespace and methods of the OPC Automation interface with .NET Framework.

Establishing a connection to the OPC server, adding a group and an item

Using the *Connect*, *Add* and *AddItem* methods, the procedure is analogous to the previous sample:

```
ServerObj = New OPCServer
ServerObj.Connect(("OPC.SimaticNET"))

GroupsObj = ServerObj.OPCGroups
GroupObj = GroupsObj.Add("MyOPCGroup")
```

```
ItemObj = GroupObj.OPCItems.AddItem("S7:[DEMO]MW1", 1)
```

Read synchronously

To read values synchronously, the *Read* method of the OPCItem class is called:

```
ItemObj.Read(OPCDevice, myValue, myQuality, myTimeStamp)
```

Write synchronously

Note

Visual Basic .NET uses the value 0 as the minimum array index and not value 1 as in the previous Visual Basic and be adapted OPC Automation interface.

Array objects of the imported namespace *OPCSiemensDAAutomation* therefore begin with index 0. In the sample, special arrays have been created for the transfer value is with array limits of 1:

```
Dim Dims() As Integer = New Integer() {1}
Dim Bounds() As Integer = New Integer() {1}
Dim Serverhandles As Array =
    Array.CreateInstance(GetType(Integer),
                        Dims,
                        Bounds)

Dim MyErrors As Array =
    Array.CreateInstance(GetType(Integer),
                        Dims,
                        Bounds)

Dim MyValues As Array =
    Array.CreateInstance(GetType(Object),
                        Dims,
                        Bounds)
```

These variables must still be initialized. The server handle received its value 1 when the item was added:

```
Serverhandles.SetValue(ItemObj.ServerHandle, 1)
MyErrors.SetValue(0, 1)
```

The value of the Text property of the text box for user input is then assigned to the first component of the *MyValues* box:

```
MyValues.SetValue(Edit_WriteVal.Text, 1)
```

After all the required local variables have been declared and initialized, the *SyncWrite* method of the OPCGroup class is called:

```
GroupObj.SyncWrite(1, Serverhandles, MyValues, MyErrors)
```

6.2 OPC Custom interface (synchronous communication) in C++

This sample in Visual C++ uses the Microsoft Foundation Classes (MFC) and the Custom interface for Data Access V2.0 of OPC to read and write data synchronously.

The description is divided into the following sections:

- Activating the simulation connection
- Working with the sample program
- How the program runs
- Description of the program
- Notes on writing your own programs

6.2.1 Activating the simulation connection

Before you can run this program, you must activate a simulation connection that makes the demonstration variables used in the program available. Follow the steps outlined below:

1. Start the "Communication Settings" configuration program from the Start menu:
""Start" > "All Programs" > "Siemens Automation" > "SIMATIC" > "SIMATIC NET" > "Communication Settings".
Reaction: The "Communication Settings" configuration program opens.
2. In the left-hand navigation window, open the entry "OPC protocol selection".
"SIMATIC NET Configuration" > "OPC Settings" > "OPC Protocol Selection".
3. Click on the arrow symbol next to the S7 protocol and select the "make virtual module (DEMO) available for simulation" check box.
4. Confirm with "Apply".
5. Close the "Communication Settings" configuration program.

Note

Before the changes take effect, you must first close all OPC clients!

6.2.2

Working with the sample program

When you start the program, only the "Start Sample" button is enabled. After you have clicked this button, the program generates the required OPC objects. You can then use the other buttons.

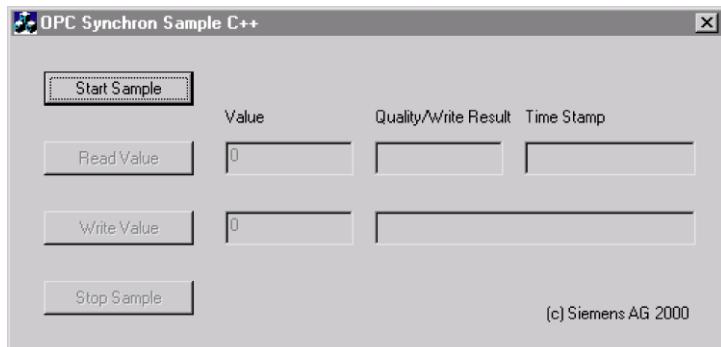


Figure 6-6 Dialog after starting the sample program for the OPC custom interface

After you click the "Read Value" button, the values that were read are displayed in the relevant text boxes. As it stands, the sample program is designed to operate with a demonstration connection. If you want to run the sample program in a real environment, you must change the following line in the "OpcSyncDlg.cpp" file:

```
LPWSTR szItemID = L"S7:[DEMO]MW1";
```

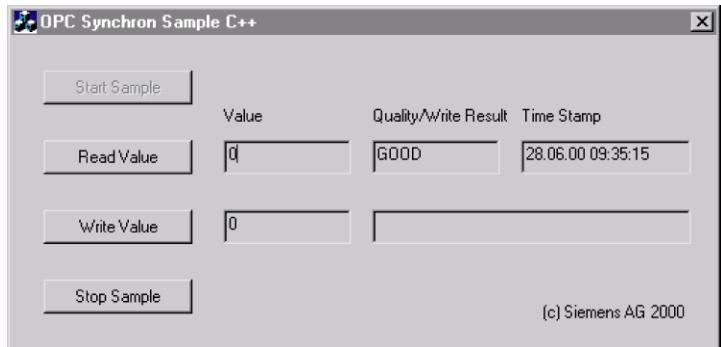


Figure 6-7 Display of the read values after clicking the "Read Value" button

To write a value, this must be entered in the relevant text box. After clicking the "Write Value" button, the program displays a message about the result of this operation:

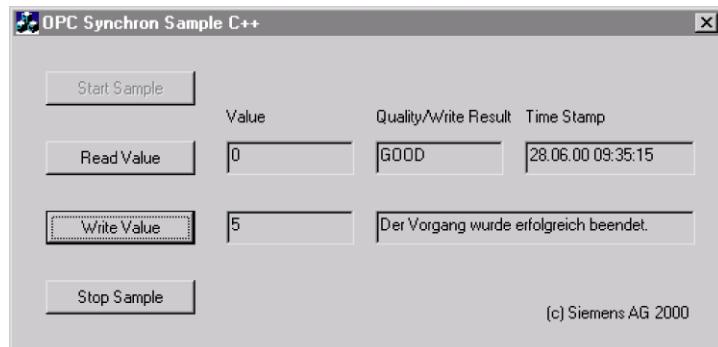


Figure 6-8 Display of the results after clicking the "Write Value" button

The "Stop Sample" button closes the program; in other words, all objects are deleted and the corresponding resources are released again.

6.2.3 How the program runs

Due to the class model of OPC, a specific order must be adhered to in the method calls of OPC objects. To be able to create an instance of the *OPCItem* class, an object of the *OPCGroup* class is necessary. This is, however, only possible when an instance of the *OPCServer* class exists and a connection to this server has been established.

The basic sequence of instructions for creating and deleting OPC objects is illustrated in the following graphic. The variable names of the sample program are used.

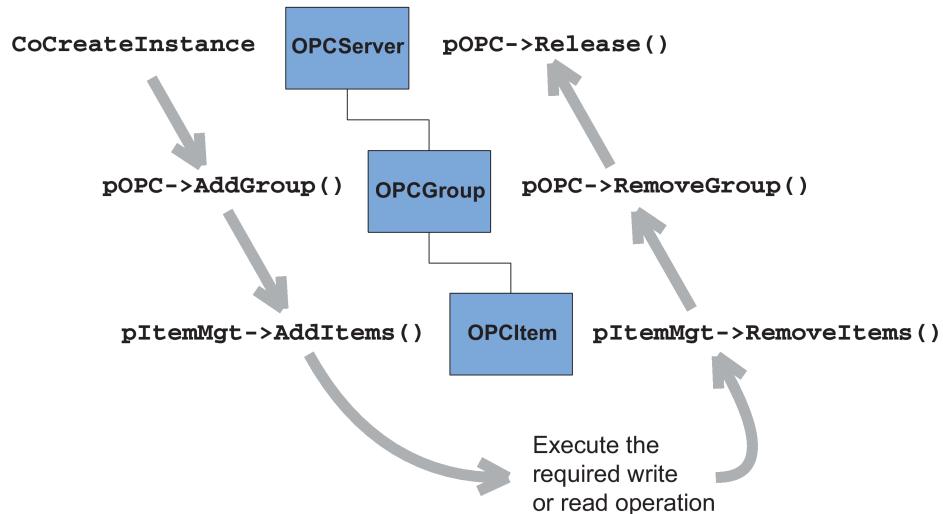


Figure 6-9 Sequence of commands for creating and deleting OPC objects

The sample program contains all the components that would normally exist in a typical client application. These include establishing the connection to the OPC server, creating a group

with variables, and reading and writing values for an item. Before describing the source code in detail, we will therefore have a look at the basic structure of an OPC application.

Step	Description
1	Register with COM
2	Convert the ProgID to a CLSID
3	Establish connection to the OPC server
4	Create an OPC group
5	Add items
6	Request an interface pointer for IOPCSyncIO
7	Delete objects and release memory

Step 1: Register with COM

Each program that wants to use functions of the COM library must first register itself with COM. This is the purpose of the *CoInitialize()* function:

```
HRESULT r1;
r1 = CoInitialize(NULL);
```

Step 2: Convert the ProgID to a CLSID

To identify it, each COM server has a *ProgID* that is assigned to a worldwide unique *CLSID* (128 bits code). Since the CLSID is required as a parameter for the later functions it must be obtained from the ProgID with the *CLSIDFromProgID()* function. The *ProgID* of the OPC server of SIMATIC NET is L"OPC.SimaticNET":

```
r1 = CLSIDFromProgID(L"OPC.SimaticNET", &clsid);
```

Step 3: Establish connection to the OPC server

The *CoCreateInstance()* function creates an instance of the class whose *CLSID* was specified.

```
r1 = CoCreateInstance(clsid, NULL, CLSCTX_LOCAL_SERVER ,
                     IID_IOPCServer, (void**)&m_pIOPCServer);
```

The result of this program section is an object of the OPC server class. *CoCreateInstance* also provides a pointer to the *IOPCServer* interface of the server object (*m_pIOPCServer* variable):

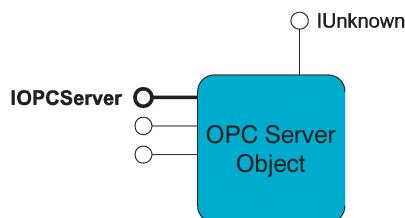


Figure 6-10 Object of the class OPC server with pointer to the *IOPCServer* interface

Step 4: Create an OPC group

The *IOPCServer* interface has the *AddGroup()* method for creating groups:

```
r1 = m_pIOPCServer->AddGroup(L"grp1", TRUE, 500, 1,
                               &TimeBias, &PercentDeadband, LOCALE_ID,
                               &m_GrpSrvHandle, &RevisedUpdateRate,
                               IID_IOPCItemMgt,
                               (LPUNKNOWN*)&m_pIOPCItemMgt);
```

The result of this program section is a group with the specified name and the required properties. In addition, a pointer to the requested interface of the group object exists as a return parameter, in this case, *IOPCItemMgt* (*m_pIOPCItemMgt* variable):

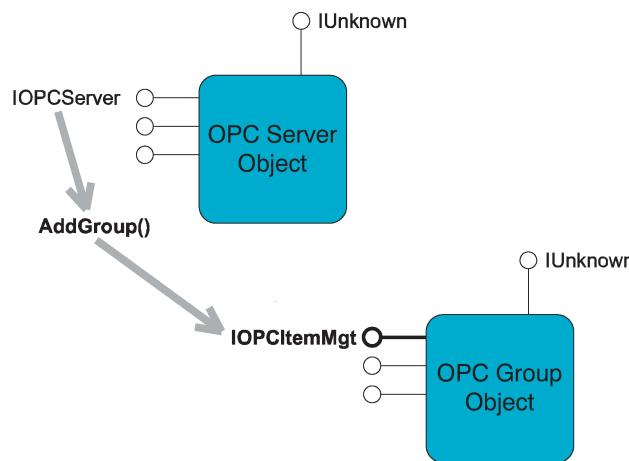


Figure 6-11 Generating an OPC group on the requested *IOPCItemMgt* interface of the group object

Step 5: Add items

The *IOPCItemMgt* interface has the *AddItems()* method for creating OPC items:

```
r1 = pItemMgt->AddItems(1, m_Items, &m_pItemResult, &m_pErrors);
```

The result of this step is the required number of items with the specified properties. The variables of the result structure *m_pItemResult* (server handle, data type of the item on the target system etc.) are also initialized.

Step 6: Request an interface pointer for *IOPCSyncIO*

A pointer to the *IOPCSyncIO* interface is necessary to allow use of the methods for synchronous reads and writes. It is requested using the already existing pointer to the *IOPCItemMgt* interface:

```
r1 = m_pIOPCItemMgt->QueryInterface(IID_IOPCSyncIO,
                                       (void**)&m_pIOPCSyncIO);
```

Using the *Read()* and *Write()* methods of this interface, values for items can be read or written:

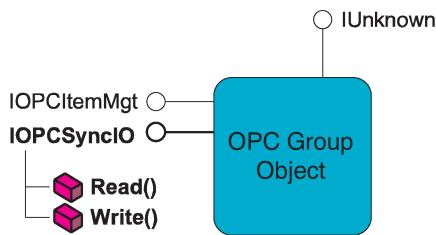


Figure 6-12 OPC group object with pointer to the *IOPCSyncIO* interface and its methods *Read()* and *Write()*

Step 7: Delete objects and release memory

Before exiting the program, the OPC objects that have been created must be deleted and the memory reserved for them must be released. The relevant functions are integral parts of the previously used interfaces.

```
r1 = m_pIOPCItemMgt->RemoveItems(1, phServer, &pErrors);
CoTaskMemFree(m_pItemResult);
m_pItemResult=NULL;
CoTaskMemFree(m_pErrors);
m_pErrors = NULL;
m_pIOPCSyncIO->Release();
m_pIOPCSyncIO = NULL;
m_pIOPCItemMgt->Release();
m_pIOPCItemMgt = NULL;
r1 = m_pIOPCServer->RemoveGroup(m_GrpSrvHandle, TRUE);
m_GrpSrvHandle = NULL;
m_pIOPCServer->Release();
m_pIOPCServer = NULL;
CoUninitialize();
```

6.2.4 Description of the OPCDA_SyncDlg.cpp program

The "OPCDA_SyncDlg.cpp" file can be divided into the following sections:

- Declaration of global variables
- Methods of the COPCDA_SyncDlg Class

Global variable

At the start of this module, a global variable is initialized with the item identifier. To ensure that the sample program functions correctly, this variable must be readable and writable.

```
const LPWSTR szItemID = L"S7:[DEMO]MW1";
```

You can refer to several examples and the manual for more information on the structure of the ItemIDs.

Methods of the COPCDA_SyncDlg Class

The main components of this module are the event procedures for the buttons of the main dialog that are called by MFC. Some settings must also be made in the *OnInitDialog* method:

- OnInitDialog
- OnStart
- OnRead
- OnWrite
- OnStop
- DestroyWindow

6.2.4.1 OnInitDialog

There are several class variables available for access to COM mechanisms that are initialized when the main dialog box is initialized. These variables can be put to the following uses:

- *m_pIOPCServer* is a pointer to the IOPCServer interface of the OPC server class.
- *m_pIOPCItemMgt* is a pointer to the IOPCItemMgt interface of the OPC group class.
- *m_pIOPCSyncIO* is a pointer to the IOPCSyncIO interface of the OPC group class.

```
m_pIOPCServer = NULL;
m_pIOPCItemMgt = NULL;
m_pIOPCSyncIO = NULL;
```

At the start of the program, the user can only click the "Start Sample" button. All other buttons are disabled. This makes sure that all required OPC objects are set up correctly before write or read operations are executed. This is achieved with the following program section:

```
m_CtrlStop.EnableWindow(FALSE);
m_CtrlRead.EnableWindow(FALSE);
m_CtrlWrite.EnableWindow(FALSE);
```

6.2.4.2 OnStart

OnStart sets up the OPC objects required for running the program. First, local variables are declared and initialized that are required for the methods of the OPC objects.

```
void COpcSyncDlg::OnStart()
{
    HRESULT r1;
    CLSID clsid;
    LONG TimeBias = 0;
    FLOAT PercentDeadband = 0.0;
    DWORD RevisedUpdateRate;
    LPWSTR ErrorStr;
    char str[100];
    Cstring szErrorText;
```

Then the following operations are executed:

CoInitialize

CoInitialize() initializes the COM library. The input parameter must always be *NULL*.

```
r1 = CoInitialize(NULL);  
if (r1 != S_OK)  
{    if (r1 == S_FALSE)  
    {        MessageBox("COM Library already initialized",  
                    "Error CoInitialize()",  
                    MB_OK+MB_ICONEXCLAMATION);  
    }  
    else  
    {        szErrorText.Format(  
            "Initialisation of COM Library failed. \\\n  
            Error Code= %4x", r1);  
        MessageBox(szErrorText, "Error CoInitialize()",  
                    MB_OK+MB_ICONERROR);  
        SendMessage(WM_CLOSE);  
        return;  
    }  
}
```

CLSIDFromProgID

CLSIDFromProgID() retrieves a worldwide unique *class identifier* or *CLSID* from a specified ProgID. This is required for the *CoCreateInstance* function.

This method is declared as follows:

```
HRESULT CLSIDFromProgID(LPCOLESTR lpszProgID, LPCLSID pclsid);
```

Parameter	Description
lpszProgID	Pointer to the ProgID [input parameter]
pclsid	Pointer to the retrieved <i>class identifier</i> [return value]

In the sample program, this method retrieves the class identifier of the OPC Server of SIMATIC NET:

```
r1 = CLSIDFromProgID(L"OPC.SimaticNET", &clsid);  
if (r1 != S_OK)  
{    MessageBox("Retrieval of CLSID failed",  
            "Error CLSIDFromProgID()",  
            MB_OK+MB_ICONERROR);  
    CoUninitialize();  
    SendMessage(WM_CLOSE);  
    return;  
}
```

CoCreateInstance

CoCreateInstance() creates an instance of the class whose class identifier was specified. This method is declared as follows:

```
STDAPI CoCreateInstance (REFCLSID rclsid,
                        LPUNKNOWN pUnkOuter,
                        DWORD dwClsContext,
                        REFIID riid,
                        LPVOID *ppv);
```

Parameter	Description
rclsid	The class identifier of the required object
pUnkOuter	NULL pointer if the object to be created is not part of an aggregated object. A pointer that is not a NULL pointer is interpreted as a pointer to the IUnknown interface of the aggregated object.
dwClsContext	Describes the runtime environment of the object to be created. This specifies whether or not the executable code that creates and manages objects of this class runs on the local machine and whether a separate process will be created for it. The constants that can be used for this are specified in the CLSCTX enumerated data type. In the sample, CLSCTX_LOCAL_SERVER is used; in other words, the executable code for managing the server objects runs on the same computer as the sample program but in its own process.
riid	Identifier of the interface over which communication with the object will take place, in the sample program <i>IOPCServer</i> .
ppv	Address of the pointer variable containing the required interface pointer if the method call is successful.

The following method call creates an object of the OPC server class and returns a pointer to the IOPCServer interface:

```
r1 = CoCreateInstance (clsid, NULL, CLSCTX_LOCAL_SERVER,
IID_IOPCServer, (void**)&m_pIOPCServer);
if (r1 != S_OK)
{
    MessageBox("Creation of IOPCServer-Object failed",
              "Error CoCreateInstance()", MB_OK+MB_ICONERROR);
    m_pIOPCServer = NULL;
    CoUninitialize();
    SendMessage(WM_CLOSE);
    return;
}
```

AddGroup

The *AddGroup()* method of the IOPCServer creates an OPC group and is declared as follows:

```
HRESULT AddGroup (LPWSTR szName,
                  BOOL bActive,
                  DWORD dwRequestedUpdateRate,
                  OPCHANDLE hClientGroup,
                  LONG *pTimeBias,
                  FLOAT *pPercentDeadband,
                  DWORD dwLCID,
                  OPCHANDLE *phServerGroup,
```

```
DWORD *pRevisedUpdateRate,
REFIID riid,
LPUNKNOWN *ppUnk);
```

Parameter	Description
szName	Group name that can be assigned freely by the client but must be unique within the client.
bActive	FALSE if the group is to be inactive when it is created. TRUE if the group is to be active when it is created.
dwRequestedUpdateRate	Specifies the shortest interval after which a client will be informed of changes to values or states of items. Specifying a suitable interval prevents information being sent to a client more quickly than it can process it.
hClientGroup	Code number that can be selected freely by the client and that is returned by the server with certain notifications. This allows the client to identify its data. The client identifies the group with this handle.
pTimeBias	Deviation of the server time from UTC (Universal Time Convention)
pPercentDeadband	Specifies the percentage of the bandwidth in which value changes do not lead to a notification. The upper and lower limit values of a value must be known to be able to calculate the bandwidth. In this sample program, this is not the case.
dwLCID	Selects the language to be used by the server when returning texts
phServerGroup	Handle assigned by the server that must be specified as a parameter with certain function calls (for example, <i>RemoveGroup</i>). The server requires this to identify this group.
pRevisedUpdateRate	Shortest interval returned by the server after which a client will be informed of changes to values or states of items.
riid	Pointer to the identifier of one of the interfaces of the OPC group object that will be available after the group is created. This parameter saves calling the <i>QueryInterface</i> method extra.
ppUnk	Pointer to the required interface.

When *AddGroup* is called in the sample program, the following settings are made:

- The *bActive* parameter is set to the value FALSE. Directly after it is created, the group is inactive; in other words, no OnDataChange callbacks are generated for this group.
- The duration of the required update interval is 500 milliseconds.
- Any client handle can be specified since only one group is being used.
- In the sample program, there is no provision for suppressing notification of value changes within a specific range. The value "0.0" is therefore entered for the *pPercentDeadband* parameter.
- AddGroup* will return a pointer to the *IOPCItemMgt* interface as the return value.

```
r1 = m_pIOPCServer->AddGroup(L"grpl",
                               TRUE,
                               500,
                               1,
                               &TimBias,
                               &PercentDeadband,
```

```

        LOCALE_ID,
        &m_GrpSrvHandle,
        &RevisedUpdateRate,
        IID_IOPCItemMgt,
        (LPUNKNOWN*)&m_pIOPCItemMgt);

if (r1 == OPC_S_UNSUPPORTEDRATE)
{
    szErrorText.Format ("Revised Update Rate %d is \
                        different from Requested Update Rate 500",
                        RevisedUpdateRate );
    AfxMessageBox(szErrorText);
}
else
if (FAILED(r1))
{
    MessageBox("Can't add Group to Server!", "Error
               AddGroup()", MB_OK+MB_ICONERROR);
    m_pIOPCServer->Release();
    m_pIOPCServer = NULL;
    CoUninitialize();
    SendMessage(WM_CLOSE);
    return;
}

```

AddItems

The *AddItems()* method of the *IOPCItemMgt* interface creates OPC items and is declared as follows:

```

HRESULT AddItems (DWORD dwNumItems,
                  OPCITEMDEF *pItemArray,
                  OPCITEMRESULT **ppAddResults,
                  HRESULT **ppErrors);

```

Parameter	Description
dwNumItems	Number of items to be inserted
pItemArray	Array with elements of the type OPCITEMDEF. Structure variables of this type contain all the information required by the server to create items.
ppAddResults	Array with elements of the type OPCITEMRESULT. The return values delivered by the OPC server are structure variables of this type.
ppErrors	Array with elements of the type HRESULT. These variables return an error code if items could not be created successfully or information about the successful method call.

Before calling *AddItems*, an array with elements of the type OPCITEMDEF must be created and initialized with valid values. When doing this, the following situations must be taken into account:

- An access path for the item is not necessary in the sample and an empty string is specified here.
- The item identifier was specified at the start of the module OPCDA_SyncDlg.cpp and assigned to the variable szItemID.

- The item will be activated after it is created.
- The sample program uses a client handle of 1.
- The OPC Server for SIMATIC NET does not require *BinaryLargeObjects* and the structure component dwBlobSize therefore has the value "0".
- For the item, the server must deliver the return value in a type corresponding to the original data type of the item.

```
m_Items[0].szAccessPath = L"";  
m_Items[0].szItemID = szItemID;  
m_Items[0].bActive = TRUE;  
m_Items[0].hClient = 1;  
m_Items[0].dwBlobSize = 0;  
m_Items[0].pBlob = NULL;  
m_Items[0].vtRequestedDataType = vtDataTypeItem;
```

The *m_pItemResult* pointer exists as a property of the *COPCDA_SyncDlg* class. The results returned by the server can be accessed using this variable. *m_pErrors* is a pointer to the error code:

```
r1 = m_pIOPCItemMgt->AddItems(1, m_Items, &m_pItemResult, &m_pErrors);
```

If the *AddItems* call was not successful, the program is aborted. First, however, the program must release the resources it is using:

```
if ( (r1 != S_OK) && (r1 != S_FALSE) )  
{    MessageBox("AddItems failed!", "Error AddItems()",  
            MB_OK+MB_ICONERROR);  
    m_pIOPCItemMgt->Release();  
    m_pIOPCItemMgt = NULL;  
    m_GrpSrvHandle = NULL;  
    m_pIOPCServer->Release();  
    m_pIOPCServer = NULL;  
    CoUninitialize();  
    SendMessage(WM_CLOSE);  
    return;  
}
```

GetErrorString

If the return value of *AddItems()* indicates that an error has occurred, the *GetErrorString()* method of the *IOPCServer* interface gets the corresponding error message. This method is declared as follows:

```
HRESULT GetErrorString (HRESULT dwError,  
                      LCID dwLocale,  
                      LPWSTR *ppString);
```

Parameter	Description
dwError	Error code returned by the server
dwLocale	The language identifier for the error message
ppString	Double pointer to a nullterminated string to which <i>GetErrorString</i> returns the error message

```

    else
    {
        m_pIOPCServer ->GetErrorString(m_pErrors[0], LOCALE_ID,
                                         &ErrorStr);
        sprintf(str, "%ls\n", ErrorStr);
        MessageBox(str, "Result AddItems()", MB_OK+MB_ICONEXCLAMATION);
        CoTaskMemFree(ErrorStr);
    }
}

```

QueryInterface

The *QueryInterface()* method of the *IUnknown* interface returns a pointer to an interface whose identifier is specified as the input parameter. *QueryInterface()* is declared as follows:

```
HRESULT QueryInterface (REFIID iid, void **ppvObject);
```

Parameter	Description
iid	Identifier of the required interface
ppvObject	Address of the pointer variable containing the required interface pointer if the method call is successful. If the object does not support this interface, an error code and the NULL pointer are returned.

QueryInterface gets a pointer to the *IOPCSyncIO* interface that provides methods for synchronous reading and writing:

```

r1 = m_pIOPCItemMgt->QueryInterface(IID_IOPCSyncIO,
                                       (void**)&m_pIOPCSyncIO);
if (r1 < 0)
{
    MessageBox("No IOPCSyncIO found!",
              "Error QueryInterface()", MB_OK+MB_ICONERROR);
    CoTaskMemFree(m_pItemResult);
    m_pIOPCItemMgt->Release();
    m_pIOPCItemMgt = NULL;
    m_GrpSrvHandle = NULL;
    m_pIOPCServer->Release();
    m_pIOPCServer = NULL;
    CoUninitialize();
    SendMessage(WM_CLOSE);
    return;
}

```

Enabling buttons

Once *OnButtonStart()* has set up all necessary OPC objects, it disables the "Start Sample" button. All other buttons are enabled. This strategy ensures that *OnButtonStart()* is executed once only. This saves additional queries in the program.

```
m_CtrlStop.EnableWindow(FALSE);
m_CtrlRead.EnableWindow(FALSE);
```

```
m_CtrlWrite.EnableWindow(TRUE);  
m_CtrlStart.EnableWindow(FALSE);
```

The OPCITEMDEF structure

OPCITEMDEF has the following structure:

```
typedef struct {  
    LPWSTR szAccessPath;  
    LPWSTR szItemID;  
    BOOL bActive;  
    OPCHANDLE hClient;  
    DWORD dwBlobSize;  
    BYTE *pBlob;  
    VARTYP vtRequestedDataType;  
    Word wReserved;  
} OPCITEMDEF;
```

Variables of OPCITEMDEF

Variable	Description
szAccessPath	Optional access path for the items (not required by SIMATIC NET)
szItemID	ItemID assigned by the client
bActive	TRUE if the client will be notified if there is a value change of the item in an active group; FALSE if the client will not be notified.
hClient	Handle for an item assigned by the client. The server transfers the client handle with calls to the client (for example, <i>OnDataChange</i>) so that the client can access the relevant variables in its structures.
dwBlobSize	Size of a memory area on the server in which additional information for faster access to the data of an item is stored.
pBlob	Pointer to the memory area described above
vtRequestedDataType	Data type requested by the client

The OPCITEMRESULT structure

OPCITEMRESULT has the following structure:

```
typedef struct {  
    OPCHANDLE hServer;  
    VARTYPE vtCanonicalDataType;  
    WORD wReserved;  
    DWORD dwAccessRights;  
    DWORD dwBlobSize;  
    BYTE *pBlob;  
} OPCITEMRESULT;
```

Variables of OPCITEMRESULT

Variable	Description
hServer	A handle for an item assigned by the server. The client transfers the server handle with calls to the server so that the server can access the relevant variables in its structures.
vtCanonicalDataType	The data type used by the server for an item
dwAccessRights	Information about whether an item can only be read, only be written, or read and written.
dwBlobSize	Size of a memory area on the server in which additional information for faster access to the data of an item is stored.
pBlob	Pointer to the memory area described above

6.2.4.3 OnRead

OnRead executes a synchronous read job. The following operations are executed:

Variable declaration

First, several local variables are declared:

```
void COpcSyncDlg::OnRead()
{
    OPCHANDLE *phServer;
    OPCITEMSTATE *pItemValue;
    HRESULT *pErrors;
    HRESULT r1;
    LPWSTR ErrorStr;
    char str[100];
    UINT qnr;
```

The server handle of the item for which the value will be read is required as a parameter for the *Read()* method. This identification number assigned by the server exists in the *hServer* component of the *pItemResult[0]* global variable. The *AddItem()* method enters its return values in this structure variable of the type OPCITEMRESULT.

```
phServer = new OPCHANDLE[1];
phServer[0] = m_pItemResult[0].hServer;
```

Read

Read reads values for OPC items synchronously. This method is declared as follows:

```
HRESULT Read (OPCDATASOURCE dwSource,
              DWORD dwNumItems,
              OPCHANDLE *phServer,
              OPCITEMSTATE **ppItemValues,
              HRESULT **ppErrors);
```

Parameter	Description
dwSource	Data source. If you use OPC_DS_CACHE, the data is read from the cache of the OPC server; with OPC_DS_DEVICE, a read job is executed over the network.
dwNumItems	Number of items for which values are read.
phServer	Array with server handles.
ppItemValues	Array with elements of the type OPCITEMSTATE for the read values and additional information about the read job.
ppErrors	Array with elements of the type HRESULT. These variables return an error code if Read() could not be called successfully or information about the successful method call.

Read is called with the previously initialized variables as parameters:

```
r1 = m_pIOPCSyncIO->Read(OPC_DS_DEVICE, 1, phServer, &pItemValue,
                            &pErrors);
```

The *Read* method provides the values that were read and additional information in the *pItemValue* array. If the read is successful, these data are displayed in the relevant text boxes of the main dialog box:

```
if (r1 == S_OK)
{
    m_ReadValue = pItemValue[0].vDataValue.lVal;
    qnr = pItemValue[0].wQuality;
    m_szReadQuality = GetQualityText(qnr);
    m_szTimeStamp =
        COleDateTime(pItemValue[0].ftTimeStamp).Format();
    UpdateData(FALSE);
}
```

The return value *S_FALSE* indicates that an error code was entered in the *pErrors* array. *GetErrorString* gets the corresponding error message that is displayed in a dialog box:

```
if (r1 == S_FALSE)
{
    m_pOPCServer->GetErrorString(pErrors[0], LOCALE_ID, &ErrorStr);
    sprintf(str, "%S\n", ErrorStr);
    MessageBox(str, "Error Read()", MB_OK+MB_ICONERROR);
    CoTaskMemFree(ErrorStr);
}
```

If there are no error messages and the method call was not successful, a suitable dialog box is displayed. Finally, the resources used by *Read* are released:

```
if (FAILED(r1))
{
    MessageBox("Read failed!", "Error Read()", MB_OK+MB_ICONERROR);
```

```

    }
else
{
    CoTaskMemFree(pErrors);
    CoTaskMemFree(pItemValue);
}

```

OPCITEMSTATE

```

typedef struct {
    OPCHANDLE hClient;
    FILETIME ftTimeStamp;
    WORD wQuality;
    WORD wReserved;
    VARIANT vDataValue;
} OPCITEMSTATE;

```

Variables of OPCITEMSTATE

Variable	Description
hClient	The client handle of the item.
ftTimeStamp	Time stamp (time at which the data was received from the OPC server).
wQuality	Information on the integrity of the data.
vDataValue	The value read for the item.

6.2.4.4 OnWrite

OnWrite executes a synchronous write job. The following operations are executed:

Variable declaration

First, several local variables are declared:

```

void COpcSyncDlg::OnWrite()
{
    OPCHANDLE *phServer;
    HRESULT *pErrors;
    VARIANT values[1];
    HRESULT r1;
    LPWSTR ErrorStr;
    CString szOut;
}

```

The server handle of the item for which the value will be written is required as a parameter for the *Write* method. This identification number assigned by the server exists in the *hServer* component of the *m_pItemResult[0]* global variable. The *AddItems* method enters its return values in this structure variable of the type OPCITEMRESULT:

```

phServer = new OPCHANDLE[1];
phServer[0] = m_pItemResult[0].hServer;

```

The *values* variable is intended for storing the value to be written. The *UpdateData* method of the *CWnd* class transfers the content of all controls to the relevant member variables. This

initializes the *iVal*/component of the *values* structure variable. *2byte integer* is specified as the data type:

```
UpdateData(TRUE);
values[0].vt = VT_I2;
values[0].iVal = m_WriteValue;
```

Write

Write writes values for OPC items synchronously. This method is declared as follows:

```
HRESULT Write (DWORD dwNumItems,
               OPCHANDLE *phServer,
               VARIANT *pItemValues,
               HRESULT **ppErrors);
```

Parameters	Description
dwNumItems	Number of items for which values are written.
phServer	Array with server handles
pItemValues	Array with the values to be written
ppErrors	Array with elements of the type <i>HRESULT</i> . These variables return an error code if <i>Write()</i> could not be called successfully or information about the successful method call.

Write is called with the previously initialized variables as parameters:

```
r1 = m_pIOPCSyncIO->Write(1, phServer, values, &pErrors);
```

GetErrorString gets the error message belonging to the *pErrors* return value that is assigned to the member variable *m_WriteRes*. *GetErrorString*, however, adds two special characters for the line break that must be removed before display in a text box.

Finally, the resources used by *Write* are released:

```
delete [] phServer;
if (FAILED(r1))
{
    szOut.Format("Method call IOPCSyncIO::Write \
                  failed with error code %x", r1);
    MessageBox(szOut, "Error Writing Item",
               MB_OK+MB_ICONERROR);
}
else
{
    m_pIOPCServer->GetErrorString(pErrors[0], LOCALE_ID,
                                    &ErrorStr);
    m_szWriteResult = ErrorStr;
    m_szWriteResult.Remove('\r');
    m_szWriteResult.Remove('\n');
    UpdateData(FALSE);
    CoTaskMemFree(pErrors);
    CoTaskMemFree(ErrorStr);
}
```

6.2.4.5 OnStop

OnStop removes the OPC objects used in the program and releases the corresponding resources. This method is called when the "Stop Sample" button is clicked, the *OnDestroy* method is executed or the WM_CLOSE message is sent (after clicking the button to close the dialog box or explicitly by calling *SendMessage*).

The resources are released in several steps:

- RemoveItems
- RemoveGroup
- Release
- CoUninitialize

RemoveItems

The *RemoveItems* method of the IOPCItemMgt interface removes OPC items and is declared as follows:

```
HRESULT RemoveItems (DWORD dwCount,
                     OPCHANDLE *phServer,
                     HRESULT **ppErrors);
```

Parameters	Description
dwCount	Number of items to be removed
phServer	Array with the server handles of the items to be removed
ppErrors	Array with elements of the type HRESULT. These variables return an error code if <i>RemoveItems()</i> could not be called successfully or information about the successful method call.

```
void COpcSyncDlg::OnStop()
{
    HRESULT r1;
    OPCHANDLE *phServer;
    HRESULT *pErrors;
    LPWSTR ErrorStr;
    char str[100];

    phServer = new OPCHANDLE[1];
    phServer[0] = m_pItemResult[0].hServer;
    r1 = m_pIOPCItemMgt->RemoveItems(1, phServer, &pErrors);
    if ( (r1 != S_OK) && (r1 != S_FALSE) )
    {
        MessageBox("RemoveItems failed!",
                  "Error RemoveItems()", MB_OK+MB_ICONERROR);
    }
    else
    {
        m_pIOPCServer->GetErrorString(pErrors[0], LOCALE_ID,
                                         &ErrorStr);
        sprintf(str, "%ls\n", ErrorStr);
        MessageBox(str, "Result RemoveItems()", MB_OK+MB_ICONEXCLAMATION);
    }
}
```

```
    CoTaskMemFree(ErrorStr);
}
```

RemoveGroup

RemoveGroup() removes a group from the server and is declared as follows:

```
HRESULT RemoveGroup (OPCHANDLE hServerGroup,
                      BOOL bForce);
```

Parameters	Description
hServerGroup	Server handle of the group to be removed.
bForce	Specifies whether or not a group can be removed when there is still a reference to it.

```
r1=m_pIOPCServer->RemoveGroup(m_GrpSrvHandle, TRUE);
if (r1 != S_OK)
{   MessageBox("RemoveGroup failed!",
            "Error Remove Group()", MB_OK+MB_ICONERROR);
}
```

Release

The *IOPCServer* and *IMalloc* interfaces have the *Release* method with which they release the resources used by the interface:

```
m_pIOPCSyncIO->Release();
m_pIOPCSyncIO = NULL;
m_pIOPCItemMgt->Release();
m_pIOPCItemMgt = NULL;
m_pIOPCServer->Release();
m_pIOPCServer = NULL;
```

In addition to this, all other requested resources must be released again:

```
delete[] phServer;
CoTaskMemFree(pErrors);
CoTaskMemFree(m_pItemResult);
m_pItemResult=NULL;
CoTaskMemFree(m_pErrors);
m_pErrors = NULL;
m_GrpSrvHandle = NULL;
```

All buttons apart from "Start Sample" are made inactive:

```
m_CtrlStart.EnableWindow(TRUE);
m_CtrlRead.EnableWindow(FALSE);
m_CtrlStop.EnableWindow(FALSE);
m_CtrlWrite.EnableWindow(FALSE);
```

CoUninitialize

The *CoUninitialize* method closes the COM library and releases all resources of the thread:

```
CoUninitialize();
}
```

6.2.4.6 DestroyWindow

This method of the CWnd class is overwritten so that all the "clean up jobs" can be done when the dialog window is closed. *OnButtonStop()* deletes all the OPC objects and releases the corresponding resources. Following this, *DestroyWindow* of the parent class is called to release the memory used by the window object:

```
BOOL COPCDA_SyncDlg::DestroyWindow()
{
    if (m_pIOPCServer)
        OnStop();
    return CDialog::DestroyWindow();
}
```

6.2.4.7 GetQualityText

GetQualityText provides a CString object as an error message for a specified error code:

```
CString GetQualityText(UINT qnr)
{
    CString qstr;
    switch(qnr)
    {
        case OPC_QUALITY_BAD:
            qstr = "BAD";
            break;
        case OPC_QUALITY_UNCERTAIN:
            qstr = "UNCERTAIN";
            break;
        case OPC_QUALITY_GOOD:
            qstr = "GOOD";
            break;
        case OPC_QUALITY_NOT_CONNECTED:
            qstr = "NOT_CONNECTED";
            break;
        case OPC_QUALITY_DEVICE_FAILURE:
            qstr = "DEVICE_FAILURE";
            break;
        case OPC_QUALITY_SENSOR_FAILURE:
            qstr = "SENSOR_FAILURE";
            break;
        case OPC_QUALITY_LAST_KNOWN:
            qstr = "LAST_KNOWN";
            break;
        case OPC_QUALITY_COMM_FAILURE:
            qstr = "COMM_FAILURE";
            break;
        case OPC_QUALITY_OUT_OF_SERVICE:
            qstr = "OUT_OF_SERVICE";
    }
}
```

```
        break;
    case OPC_QUALITY_LAST_USABLE:
        qstr = "LAST_USABLE";
        break;
    case OPC_QUALITY_SENSOR_CAL:
        qstr = "SENSOR_CAL";
        break;
    case OPC_QUALITY_EGU_EXCEEDED:
        qstr = "EGU_EXCEEDED";
        break;
    case OPC_QUALITY_SUB_NORMAL:
        qstr = "SUB_NORMAL";
        break;
    case OPC_QUALITY_LOCAL_OVERRIDE:
        qstr = "LOCAL_OVERRIDE";
        break;
    default: qstr = "UNKNOWN ERROR";
}
return qstr;
}
```

6.2.5

Notes on writing your own programs

Several requirements must be met before your own programs can use the Custom interface. Follow the steps outlined below:

1. Start the Visual C++ development environment.
2. Create a project of the required type using the MFC Class Wizard.
3. Copy the files "pre_opc.h" and "pre_opc.cpp" from the sample program to your project directory and add the files to the project ("Project" > "Add to project" > "Files...").

These files contain OPCspecific definitions and include statements.

4. Add the following statement to all the implementation files of your project (extension "*.cpp"): `#include pre_opc.h`
5. In the implementation files (extension ".*cpp") that use the *AddGroup()* or *GetErrorString()* methods, add the following statement: `#define LOCALE_ID0x409`
6. Arrange the required operator interface elements in the main dialog or create a suitable application window and program the corresponding event procedures.

6.3

OPC Custom interface (asynchronous communication) in C++

This example uses the Custom Interface for Data Access V2.0 of OPC.

6.3.1 Activating the simulation connection

Before you can run this program, you must activate a simulation connection that makes the demonstration variables used in the program available. Follow the steps outlined below:

1. Start the "Communication Settings" program from the Start menu:

"Start" > "All Programs" > "Siemens Automation" > "SIMATIC" > "SIMATIC NET" > "Configuration Console".

Reaction: The "Communication Settings" configuration program is opened.

2. In the left-hand navigation window, open the entry "OPC protocol selection":

"PC Station" > "Applications" > "OPC Settings" > "OPC Protocol Selection".

3. Enable the check box for the protocol to be simulated.

This example uses the S7 protocol.

For this reason, select the check box under "Name: S7" and click "Change details...".

Reaction: The "Protocol details" window opens.

4. Select the "Provide virtual module (DEMO) for the simulation" check box.

5. Confirm with "Apply".

6. Close the "Communication Settings" program.

Note

Before the changes take effect, you must first close all OPC clients!

6.3.2 Working with the sample program

The program includes several elements that the operator can use to trigger the following actions:

Operator Element	Effect
Start Sample	Start program
Read or Write Item 0	Read and write values
Group Active	Activate or deactivate group
Stop Sample	Stop the program

6.3.3 Start program

The program is located on your hard disk in:

"<installationpath>\SIEMENS\SIMATIC.NET\opc2\samples\custom\async."

When you start the program, only the "Start Sample" button is enabled. After you have clicked this button, the program generates the required OPC objects. You can then use the other buttons.

6.3.4 Read and write values

After you click the "Read Item 0" button, the value that was read and status information are displayed in the relevant text boxes. As it stands, the sample program is designed to operate with a demonstration connection. If you want to run the sample program in a real environment, you must change the following lines in the OPCDA_AsyncDlg.cpp program:

```
const LPWSTR szItemID0 = L"S7:[DEMO]MB1";  
const LPWSTR szItemID1 = L"S7:[DEMO]MW1";
```

To write a value, you enter it in the "Value" text box and click the "Write Value" button. The program displays a message about the result of the write operation.

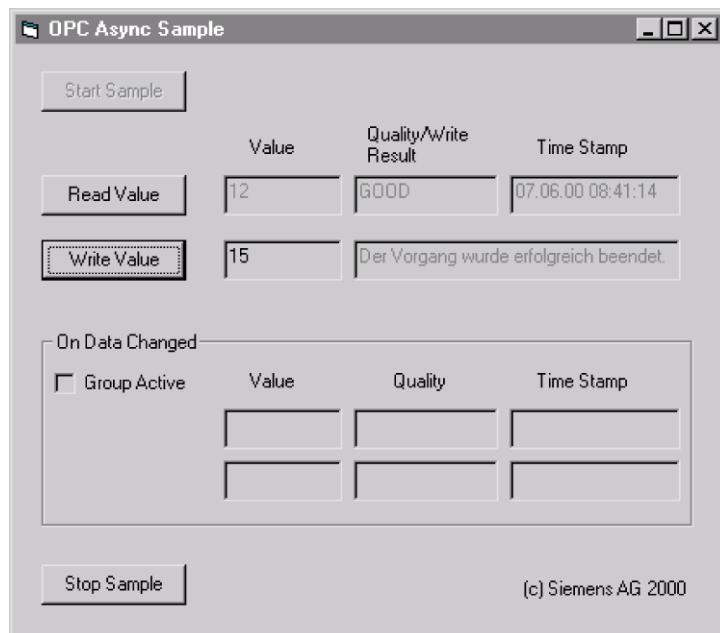


Figure 6-13 Dialog after clicking the "Write value" button

6.3.5 Activate group

Asynchronous operations do not return a result directly (for example as a function result or return parameter) but make use of events.

You can activate the OPC group in the sample program by selecting the "Group Active" check box. Among other things, the *OnDataChange* event is generated. A procedure that executes when this event occurs, displays values and status information in the text boxes of "On Data Changed".

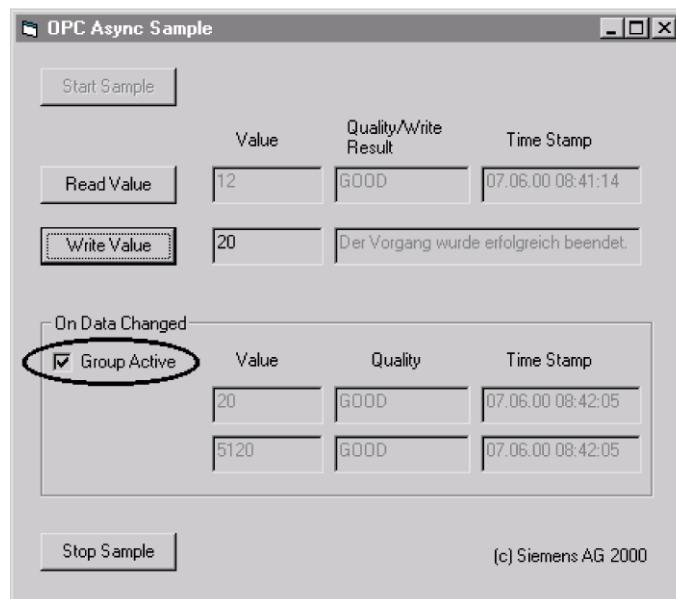


Figure 6-14 Display of the values and status information in the text boxes of the "On Data Changed" box after enabling the OPC group with the "Group Active" check box

Events for completion of write or read operations are also generated by inactive groups. This means that results can also be displayed for writes and reads even if this check box is not selected.

6.3.6 Stop the program

The "Stop Sample" button closes the program; in other words, all objects are deleted and the corresponding resources are released again.

6.3.7 How the program runs

Due to the class model of OPC, a specific order must be adhered to in the method calls of OPC objects. To be able to create an instance of the *OPCItem* class, an object of the *OPCGroup* class is necessary. This is, however, only possible when an instance of the *OPCServer* class exists and a connection to this server has been established.

The basic sequence of instructions for creating and deleting OPC objects is illustrated in the following graphic. The variable names of the sample program are used:

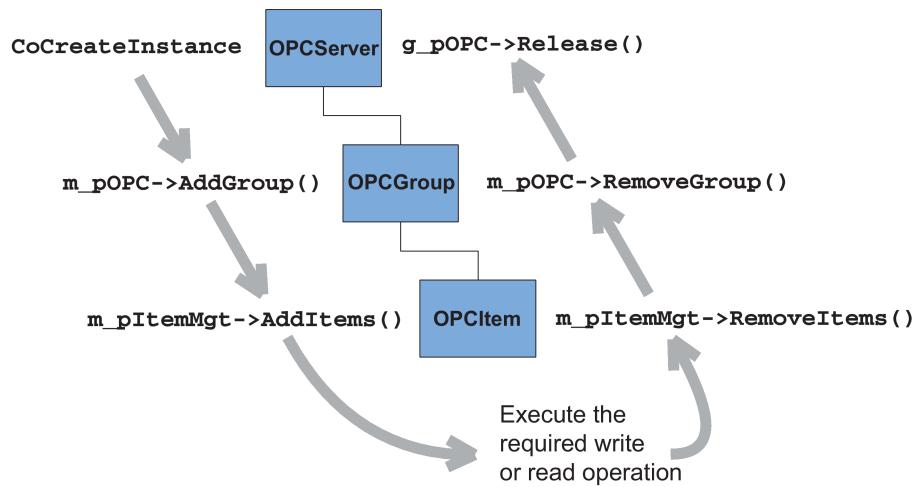


Figure 6-15 Sequence of commands for creating and deleting OPC objects

The sample program contains all the components that would normally exist in a typical client application. These include establishing the connection to the OPC server, creating a group with variables, and reading and writing values for an item. Before describing the source code in detail, we will first have a look at the basic structure of an OPC application.

Step	Description
1	Register with COM
2	Convert the ProgID to a CLSID
3	Establish connection to the OPC server
4	Create an OPC group
5	Add items
6	Request an interface pointer for <code>IOPCGroupStateMgt</code>
7	Request an interface pointer for <code>IOPCAsyncIO2</code>
8	Create callback object
9	Connect OPC server and callback object of the client
10	Execute required write and read operations
11	Receive notifications of the OPC server
12	Delete objects and release memory

Step 1: Register with COM

Each program that wants to use functions of the COM library must first register itself with COM. This is the purpose of the `CoInitialize` function:

```

HRESULT r1;
r1 = CoInitialize(NULL);
  
```

Step 2: Convert the ProgID to a CLSID

To identify it, each COM server has a *ProgID* that is assigned to a worldwide unique *CLSID*. This is obtained with the *CLSIDFromProgID()* function. The ProgID of the OPC server of SIMATIC NET is L"OPC.SimaticNET":

```
r1 = CLSIDFromProgID(L"OPC.SimaticNET", &clsid);
```

Step 3: Establish connection to the OPC server

The *CoCreateInstance()* function creates an instance of the class whose *CLSID* was specified.

```
r1 = CoCreateInstance(clsid, NULL, CLSCTX_LOCAL_SERVER,
                      IID_IOPCServer, (void**)&m_pIOPCServer);
```

The result of this program section is an object of the OPC server class. *CoCreateInstance* also provides a pointer to the *IOPCServer* interface of the server object (*m_pIOPCServer* variable):

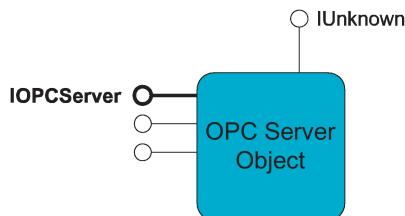


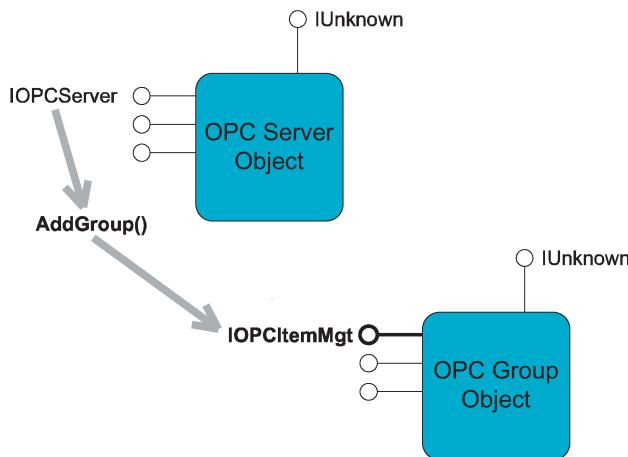
Figure 6-16 Object of the class OPC server with pointer to the *IOPCServer* interface

Step 4: Create an OPC group

The *IOPCServer* interface has the *AddGroup()* method for creating groups:

```
HRESULT r1;
r1 = m_pIOPCServer ->AddGroup(L"grp1", FALSE, 500, 1,
                                &TimBias, &PercentDeadband, LOCALE_ID, &m_GrpSrvHandle,
                                &RevisedUpdateRate, IID_IOPCItemMgt,
                                (LPUNKNOWN*) &m_pIOPCItemMgt);
```

The result of this program section is a group with the specified name and the required properties. *AddGroup* also returns a pointer to a requested interface of the group object as a return parameter, in this case, *IOPCItemMgt* (*m_pIOPCItemMgt* variable):

Figure 6-17 Generating an OPC group on the requested *IOPCItemMgt* interface of the group object

Step 5: Add items

The *IOPCItemMgt* interface has the *AddItems()* method for creating OPC items:

```

HRESULT r1;
r1 = m_pIOPCItemMgt->AddItems(2, m_Items, &m_pItemResult,
                                &m_pErrors);

```

The result of this program section is that the server adds two items with the properties specified in the *m_Items* parameter. The variables of the result structure *m_pItemResult* (server handle, data type of the item on the target system etc.) are also initialized.

Step 6: Request an interface pointer for *IOPCGroupStateMgt*

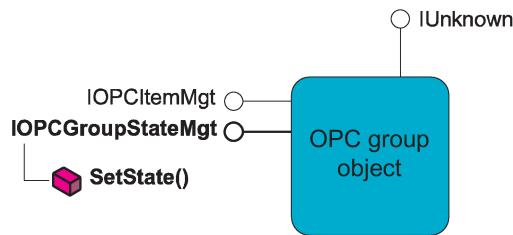
A pointer to the *IOPCGroupStateMgt* interface is necessary to allow use of the *SetState* methods. The *SetState* method is required later to activate and deactivate a group. This interface pointer is also required as a parameter for the *AtAdvise* and *AtUnadvise* methods:

```

HRESULT r1;
r1 = m_pIOPCItemMgt->QueryInterface(IID_IOPCGroupStateMgt,
                                         (void**) &m_pOPCGroupStateMgt);

```

The result of this program section is a pointer to the *IOPCGroupStateMgt* interface (*m_pOPCGroupStateMgt* variable) of the group object:

Figure 6-18 OPC group object with pointer to the *IOPCGroupStateMgt* interface and its *SetState()* method

Step 7: Request an interface pointer for IOPCAsyncIO2

In the same way, the program also requests the pointer to the IOPCAsyncIO2 interface (*m_pIOPCAsyncIO2* variable). This interface provides methods for asynchronous reading and writing of values:

```
r1 = m_pIOPCItemMgt ->QueryInterface(IID_IOPCAsyncIO2, (void**)&m_pIOPCAsyncIO2);
```

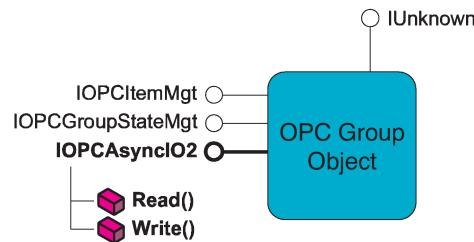


Figure 6-19 OPC group object with pointer to the *IOPCAsyncIO2* interface and its methods for asynchronous reading and writing of values

Step 8: Create callback object

To allow the OPC server to return a result of asynchronous operations, the IOPCDataCallback interface must be implemented on the client. In the sample program, this is handled by the *COPCDataCallback* class that is derived from *IOPCDataCallback* and *CComObjectRoot*.

The *CreateInstance* method creates an object of the *CComObject* template class:

```
CComObject<COPCDataCallback>* pCOPCDataCallback;
CComObject<COPCDataCallback>::CreateInstance
&pCOPCDataCallback;
```

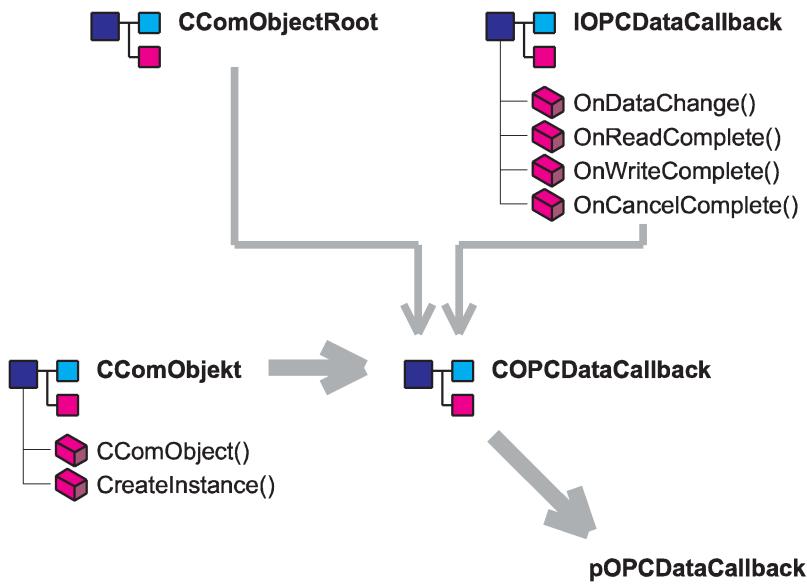


Figure 6-20 The *IOPCDataCallback* interface of the client; the `CreateInstance` method generates a template class object *CComObject*

Step 9: Connect OPC server and callback object of the client

The *AtAdvise()* method creates a connection between the OPC server and the callback object. The *GetUnknown()* method first gets a pointer to the *IUnknown* interface of the callback object: *AtAdvise* requires this pointer as an input parameter:

```

LPUNKNOWN pCbUnk;
pCbUnk = pCOPCDataCallback->GetUnknown();
HRESULT hRes = AtAdvise(m_pIOPCGroupStateMgt, pCbUnk,
                        IID_IOPCDataCallback, &m_dwAdvise);
  
```

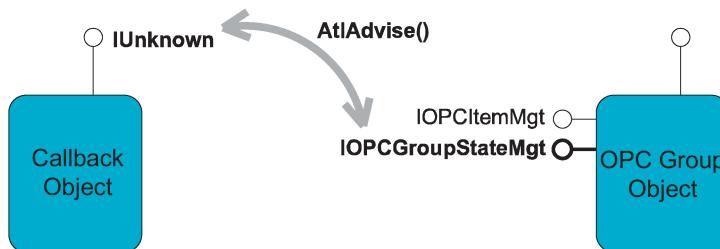


Figure 6-21 Creating a connection between OPC server and the callback object with the *AtAdvise()* method

Step 10: Execute required write and read operations

The *Read()* and *Write()* methods of the *IOPCAsyncIO2* interface can be accessed via the *m_pIOPCAsyncIO2* pointer that was created in step 7:

```
r1 = m_pIOPCAsyncIO2->Read(1, phServer, 1, &dwCancelID,
&pErrors);
```

Step 11: Receive notifications of the OPC server

If the data of an active item in an active group changes, the server calls the *OnDataChange* method of the callback object. On completion of a read operation, the server calls the method *OnReadComplete* etc. The server transfers the return values as a parameter to the relevant method.

```
STDMETHODIMP COPCDataCallback:: OnDataChange(
    WORD dwTransid,
    OPCHANDLE hGroup,
    HRESULT hrMasterquality,
    RESULT hrMastererror,
    DWORD dwCount,
    OPCHANDLE __RPC_FAR *phClientItems,
    VARIANT __RPC_FAR *pvValues,
    WORD __RPC_FAR *pwQualities,
    FILETIME __RPC_FAR *pftTimeStamps,
    HRESULT __RPC_FAR *pErrors)
```

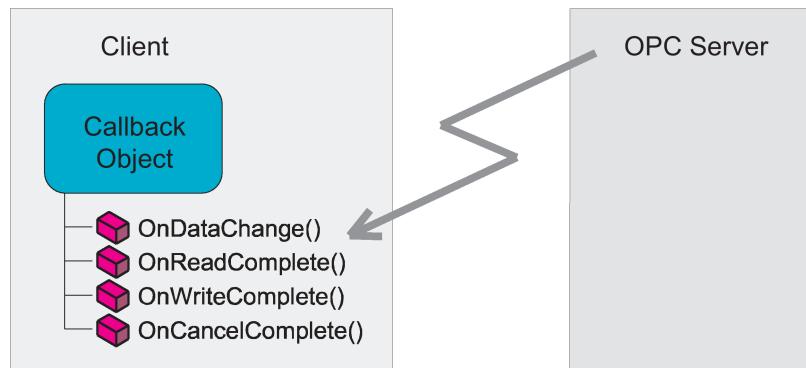


Figure 6-22 Calling methods (here: *OnDataChange()*) of the callback object by the OPC server

Step 12: Delete objects and release memory

Before exiting the program, the OPC objects that have been created must be deleted and the memory reserved for them must be released. The relevant functions are integral parts of the previously used interfaces. *AtlUnadvise* terminates the connection between the OPC server and the callback object:

```
HRESULT hRes = AtlUnadvise(m_pIOPCGroupStateMgt,
    IID_IOPCDataCallback, m_dwAdvise);
m_pIOPCGroupStateMgt->Release();
...
r1 = m_pIOPCItemMgt->RemoveItems(2, phServer,
    &pErrors);
...
CoTaskMemFree(pErrors);
```

```
CoTaskMemFree(m_pItemResult);
```

```
m_pItemResult=NULL;
```

```
...
```

```
...
```

Releasing memory

When using COM, it is sometimes necessary for the client to release memory that was requested by the server.

The rules for this are as follows:

- [out]: The memory for a parameter with this attribute will be requested from the server.
- [in, out]: The memory for such a parameter will be requested from the client. The server can release the memory again and reallocate it.
- [in]: The client will request the memory. The server is not responsible for releasing the memory.

In all three cases, the reserved memory will effectively be released by the client.

6.3.8 Description of the Program

This description explains the content of the following two files:

- "OPCDA_AsyncDlg.cpp" is the implementation file of the programspecific dialog box class. Here, you will find the event procedures for the various buttons. Their bodies may have been created by the MFC Class Wizard. The methods for initializing the dialog box and for displaying the data are also defined here.
- "Callback.cpp" is the implementation file of the program-specific callback class. All the methods that must be provided by a client so that it can receive notifications from an OPC server are defined in this module.

6.3.9 The "OPCDA_AsyncDlg.cpp" file

The "OPCDA_AsyncDlg.cpp" file can be divided into the following sections:

- Declaration of global variables
- The CAAboutDlg Class
- Methods of the COPCDA_AsyncDlg.cpp class

Declaration of the global variables for storing the ItemIDs

At the start of this module, two variables are initialized with the item identifiers:

```
const LPWSTR szItemID0 = L"S7:[DEMO]MB1";  
const LPWSTR szItemID1 = L"S7:[DEMO]MW1";
```

For the functionality of the program, it is important that the first variable can be both read and written, whereas it is adequate if the second variable can be read.

You can refer to several examples and the manual for more information on the structure of the ItemIDs.

If you select two variables that overlap in the memory map, both variables will change when you write to one of them.

The CAboutDlg Class

Following this, the *CAboutDlg* class is defined that displays an information box. Since there is no OPCspecific code in this class, it will not be described in any detail.

Methods of the COPCDA_AsyncDlg class

The *COPCDA_AsyncDlg* class has the following methods in addition to the constructor and destructor:

- DoDataExchange
- OnInitDialog
- OnSysCommand
- OnPaint
- OnQueryDragIcon
- OnButtonStart()
- OnButtonRead()
- OnButtonWrite()
- OnCheckActivategroup()
- OnButtonStop()
- OnDestroy
- DisplayData
- UpdateData

OnInitDialog

There are several class variables available for access to COM mechanisms that are initialized when the main dialog box is initialized. These variables can be put to the following uses:

- *m_pIOPCServer* is a pointer to the IOPCServer interface of the OPC server class.
- *m_pIOPCItemMgt* is a pointer to the IOPCItemMgt interface of the OPC group class.
- *m_pIOPCGroupStateMgt* is a pointer to the IOPCGroupStateMgt interface of the OPC group class.

```
m_pIOPCServer = NULL;
m_pIOPCItemMgt = NULL;
m_pIOPCGroupStateMgt = NULL;
```

At the start of the program, the user can only click the "Start Sample" button. All other buttons and the check box are disabled. This makes sure that all required OPC objects are set up correctly before write or read operations are executed.

A member variable was assigned to the relevant controls with which the status can be changed:

```
m_CtrlStop.EnableWindow(FALSE);  
m_CtrlRead.EnableWindow(FALSE);  
m_CtrlWrite.EnableWindow(FALSE);  
m_CtrlChkActive.EnableWindow(FALSE);
```

OnButtonStart()

OnButtonStart() sets up the OPC objects required for running the program. First, several local variables are declared that are required for the methods of the OPC objects:

```
void COPCDA_AsyncDlg::OnButtonStart()  
{  
    HRESULT r1;  
    CLSID clsid;  
    LONG TimBias = 0;  
    FLOAT PercentDeadband = 0.0;  
    DWORD RevisedUpdateRate;  
    LPWSTR ErrorStr1,ErrorStr2;  
    Cstring szErrorText;  
    m_pItemResult = NULL;  
    ...
```

Then the following operations are executed:

CoInitialize

CoInitialize() initializes the COM library. The input parameter must always be *NULL*.

```
r1 = CoInitialize(NULL);  
if (r1 != S_OK)  
{    if (r1 == S_FALSE)  
        {    MessageBox("COM Library already initialized",  
                    "Error CoInitialize()",  
                    MB_OK+MB_ICONEXCLAMATION);  
    }  
    else  
        {    szErrorText.Format("Initialisation of COM Library \\  
                                failed. ErrorCode= %4x", r1);  
            MessageBox(szErrorText, "Error CoInitialize()",  
                      MB_OK+MB_ICONERROR);  
            SendMessage(WM_CLOSE);  
            return;  
        }  
}
```

CLSIDFromProgID

CLSIDFromProgID() retrieves a worldwide unique *class identifier* or CLSID from a specified ProgID. This is required for the *CoCreateInstance* function.

This method is declared as follows:

```
HRESULT CLSIDFromProgID(LPCOLESTR lpszProgID,
                        LPCLSID pclsid);
```

Parameter	Description
lpszProgID	Pointer to the ProgID [input parameter].
pclsid	Pointer to the retrieved <i>class identifier</i> [return value].

In the sample program, this method retrieves the class identifier of the OPC Server of SIMATIC NET:

```
r1 = CLSIDFromProgID(L"OPC.SimaticNET", &clsid);
if (r1 != S_OK)
{
    MessageBox("Retrieval of CLSID failed",
              "Error CLSIDFromProgID()", MB_OK+MB_ICONERROR);
    CoUninitialize();
    SendMessage(WM_CLOSE);
    return;
}
```

CoCreateInstance

CoCreateInstance() creates an instance of the class whose class identifier was specified. This method is declared as follows:

```
STDAPI CoCreateInstance (REFCLSID rclsid,
                        LPUNKNOWN pUnkOuter,
                        DWORD dwClsContext,
                        REFIID riid,
                        LPVOID *ppv);
```

Parameter	Description
rclsid	The class identifier of the required object.
pUnkOuter	NULL pointer if the object to be created is not part of an aggregated object. A pointer that is not a NULL pointer is interpreted as a pointer to the IUnknown interface of the aggregated object.
dwClsContext	Describes the runtime environment of the object to be created. This specifies whether or not the executable code that creates and manages objects of this class runs on the local machine and whether a separate process will be created for it. The constants that can be used for this are specified in the CLSCTX enumerated data type. In the sample, CLSCTX_LOCAL_SERVER is used; in other words, the executable code for managing the server objects runs on the same computer as the sample program but in its own process.
riid	Identifier of the interface over which communication with the object will take place, in the sample program <i>IOPCServer</i> .
ppv	Address of the pointer variable containing the required interface pointer if the method call is successful.

The following method call creates an object of the OPC server class and returns a pointer to the IOPCServer interface:

```
r1 = CoCreateInstance (clsid, NULL, CLSCTX_LOCAL_SERVER,
                      IID_IOPCServer,
                      (void**)&m_pIOPCServer);
if (r1 != S_OK)
{   MessageBox("Creation of IOPCServer-Object failed",
            "Error CoCreateInstance ()",
            MB_OK+MB_ICONERROR);
    m_pIOPCServer = NULL;
    CoUninitialize();
    SendMessage(WM_CLOSE);
    return;
}
```

AddGroup

The *AddGroup()* method of the *IOPCServer* creates an OPC group and is declared as follows:

```
HRESULT AddGroup (LPWCSTR szName,
                  BOOL bActive,
                  DWORD dwRequestedUpdateRate,
                  OPCHANDLE hClientGroup,
                  LONG *pTimeBias,
                  FLOAT *pPercentDeadband,
                  DWORD dwLCID,
                  OPCHANDLE *phServerGroup,
                  DWORD *pRevisedUpdateRate,
                  REFIID riid,
                  LPUNKNOWN *ppUnk);
```

Parameter	Description
szName	Group name that can be assigned freely by the client but must be unique within the client.
bActive	FALSE if the group is to be inactive when it is created. TRUE if the group is to be active when it is created.
dwRequestedUpdateRate	Specifies the shortest interval after which a client will be informed of changes to values or states of items. Specifying a suitable interval prevents information being sent to a client more quickly than it can process it.
hClientGroup	Code number that can be selected freely by the client and that is returned by the server with certain notifications. This allows the client to identify its data. The client identifies the group with this handle.
pTimeBias	Deviation of the server time from UTC (Universal Time Convention)
pPercentDeadband	Specifies the deadband within which value changes do not lead to notification. This property is, however, only effective when you use analog values. The upper and lower limit values of a value must also be known. In this sample program, this is not the case.

Parameter	Description
dwLCID	Selects the language to be used by the server when returning texts
phServerGroup	Handle assigned by the server that must be specified as a parameter with certain function calls (for example, <i>RemoveGroup</i>). The server requires this to identify this group.
pRevisedUpdateRate	Shortest interval returned by the server after which a client will be informed of changes to values or states of items.
riid	Pointer to the identifier of one of the interfaces of the OPC group object that will be available after the group is created. This parameter saves calling the <i>QueryInterface</i> method extra.
ppUnk	Pointer to the required interface.

When *AddGroup* is called in the sample program, the following settings are made:

- The *bActive* parameter is set to the value FALSE. Directly after it is created, the group is inactive; in other words, no OnDataChange callbacks are generated for this group.
- The duration of the required update interval is 500 milliseconds.
- Any client handle can be specified since only one group is being used.
- In the sample program, there is no provision for suppressing notification of value changes within a specific range. The value "0.0" is therefore entered for the pPercentDeadband parameter.
- AddGroup will return a pointer to the IOPCItemMgt interface as the return value.

```
r1 = m_pIOPCServer->AddGroup(L"grp1",
                               FALSE,
                               500,
                               1,
                               &TimBias,
                               &PercentDeadband,
                               LOCALE_ID,
                               &m_GrpSrvHandle,
                               &RevisedUpdateRate,
                               IID_IOPCItemMgt,
                               (LPUNKNOWN*) &m_pIOPCItemMgt);
```

AddItems

The *AddItems()* method of the *IOPCItemMgt* interface creates OPC items and is declared as follows:

```
HRESULT AddItems (DWORD dwNumItems,
                  OPCITEMDEF *pItemArray,
                  OPCITEMRESULT **ppAddResults,
                  HRESULT **ppErrors);
```

Parameter	Description
dwNumItems	Number of items to be inserted.
pItemArray	Array with elements of the type OPCITEMDEF. Structure variables of this type contain all the information required by the server to create items.
ppAddResults	Array with elements of the type OPCITEMRESULT. The return values delivered by the OPC server are structure variables of this type.
ppErrors	Array with elements of the type HRESULT. These variables return an error code if items could not be created successfully or information about the successful method call.

Before calling *AddItems*, an array with elements of the type OPCITEMDEF must be created and initialized with valid values. When doing this, the following situations must be taken into account:

- An access path for the item is not necessary in the sample and an empty string is specified here.
- The item identifiers were specified at the start of the module OPCDA_AsyncDlg.cpp and assigned to the variables szItemID1 and szItemID2.
- The item will be activated after it is created.
- The sample program uses a client handle of 0 and 1.
- The OPC Server for SIMATIC NET does not require *BinaryLargeObjects* and the structure component dwBlobSize therefore has the value "0".
- For the first item, the server must deliver the return value in a type corresponding to the original data type of the item.
- For the second item, the server must deliver the return value as a twobyte integer.

```
m_Items[0].szAccessPath = L"";
m_Items[0].szItemID = szItemID1;
m_Items[0].bActive = TRUE;
m_Items[0].hClient = 0;
m_Items[0].dwBlobSize = 0;
m_Items[0].pBlob = NULL;
m_Items[0].vtRequestedDataType = 0;
m_Items[1].szAccessPath = L"";
m_Items[1].szItemID = szItemID2;
m_Items[1].bActive = TRUE;
m_Items[1].hClient = 1;
m_Items[1].dwBlobSize = 0;
m_Items[1].pBlob = NULL;
m_Items[1].vtRequestedDataType = VT_I2;
```

The *m_pItemResult* pointer exists as a property of the *COPCDA_AsyncDlg* class. The results

returned by the server can be accessed using this variable. *m_pErrors* is a pointer to the error code:

```
r1 = m_pIOPCItemMgt->AddItems(2, m_Items, &m_pItemResult, &m_pErrors);
```

If the *AddItems* call was not successful, the program is aborted. First, however, the program must release the resources it is using:

```
if ( (r1 != S_OK) && (r1 != S_FALSE) )
{
    MessageBox("AddItems failed!", "Error AddItems()", MB_OK+MB_ICONERROR);
    m_pIOPCItemMgt->Release();
    m_pIOPCItemMgt = NULL;
    m_GrpSrvHandle = 0;
    m_pIOPCServer->Release();
    m_pIOPCServer = NULL;
    CoUninitialize();
    SendMessage(WM_CLOSE);
    return;
}
```

GetErrorString

If the return value of *AddItems()* indicates that an error has occurred, the *GetErrorString()* method of the *IOPCServer* interface gets the corresponding error message. This method is declared as follows:

```
HRESULT GetErrorString (HRESULT dwError,
                        LCID dwLocale,
                        LPWSTR *ppString);
```

Parameter description:

Parameter	Description
dwError	Error code returned by the server.
dwLocale	The language identifier for the error message.
ppString	Double pointer to a nullterminated string to which <i>GetErrorString</i> returns the error message.

```
else
{
    m_pIOPCServer ->GetErrorString(m_pErrors[0], LOCALE_ID,
                                      &ErrorStr1);
    m_pIOPCServer ->GetErrorString(m_pErrors[1], LOCALE_ID,
                                      &ErrorStr2);
    CString szOut;
    szOut.Format("Item %ls :\n %ls\n\nItem %ls :\n %ls\n",
                 szItemID0,
                 ErrorStr1,
                 szItemID1,
                 ErrorStr2);
    MessageBox(szOut, "Result AddItems()", MB_OK+MB_ICONEXCLAMATION);
    CoTaskMemFree(ErrorStr1);
```

```
    CoTaskMemFree(ErrorStr2);
}
```

QueryInterface

The *QueryInterface()* method of the *IUnknown* interface returns a pointer to an interface whose identifier is specified as the input parameter. *QueryInterface()* is declared as follows:

```
HRESULT QueryInterface (REFIID iid, void **ppvObject);
```

Parameter description:

Parameter	Description
iid	Identifier of the required interface.
ppvObject	Address of the pointer variable containing the required interface pointer if the method call is successful. If the object does not support this interface, an error code and the NULL pointer are returned.

QueryInterface gets a pointer to the *IOPCAsyncIO2* interface that provides methods for asynchronous reading and writing:

```
r1 = m_pIOPCItemMgt ->QueryInterface(IID_IOPCAsyncIO2,
                                         (void**)& m_pIOPCAsyncIO2);

if (r1 < 0)
{   MessageBox("No IOPCAsyncIO found!",
            "Error QueryInterface()", 
            MB_OK+MB_ICONERROR);

    ...
return;
}
```

Create callback object

The *IOPCDataCallback* interface must be implemented in the client program so that notifications from the OPC server can be received. In the sample program, the implementation makes use of the Active Template Library (ATL). The *COPCDataCallback* class specific to the sample is used as a parameter for the *CComObject* template class.

The *CreateInstance* method creates an object of the *CComObject* class that is accessed via the *pOPCDataCallback* pointer:

```
CComObject<COPCDataCallback>* pCOPCDataCallback;
CComObject<COPCDataCallback>::CreateInstance(&pCOPCDataCallback);
```

Establish connection between dialog box and callback object

OnButtonStart() calls the *InformAboutDialog* method of the dialog box class and passes a pointer to the current dialog box to it (the *this* pointer). *InformAboutDialog* (in the *Callback.cpp* module) stores this pointer in the *m_pCDlgClass* member variable.

OnReadComplete(), *OnWriteComplete()* and *OnDataChange* use this pointer to call the *DisplayData* method of the dialog box. Without the information in *m_pCDlgClass*, there would be no reference to indicate the dialog box in which the data is to be displayed.

```
pCOPCDataCallback->InformAboutDialog(this);
```

The relationship between the class methods of COPCDA_AsyncDlg and IOpcDataCallback is illustrated in the following graphic:

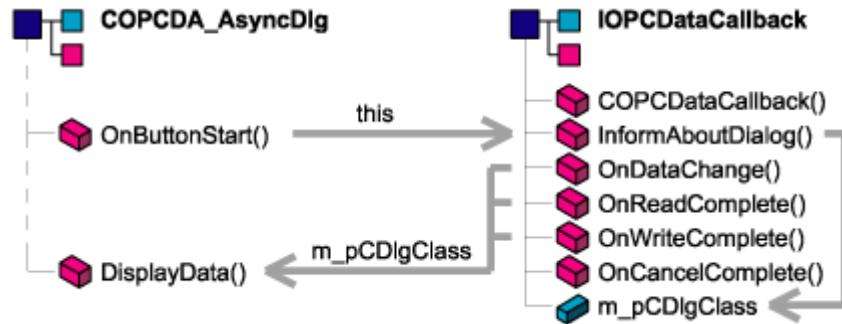


Figure 6-23 The relationship between the class methods of *COPCDA_AsyncDlg* and *IOpcDataCallback* for linking dialog box and callback object

Connect OPC server and callback object of the client

The *GetUnknown()* method gets a pointer to the IUnknown interface of the callback object:

```
LPUNKNOWN pCbUnk;
pCbUnk = pCOPCDataCallback->GetUnknown();
```

The *AtlAdvise()* method creates a connection between the OPC server and the callback object. AtlAdvise is declared as follows:

```
HRESULT AtlAdvise (IUnknown *pUnkCP,
                    IUnknown *pUnk,
                    const IID& iid,
                    LPDWORD pdw)
```

Parameter	Description
pUnkCP	Pointer to the IUnknown interface to which the client wants to establish a connection.
pUnk	Pointer to the IUnknown interface of the callback object.
iid	The identifier of the connection point. Normally the identifier of the interface that implements the callback functionality on the client is specified.
pdw	Return parameters. Pointer to a code number to identify the connection. This value is required if the connection is terminated with AtlUnadvise.

If *AtlAdvise* could not execute successfully, this is displayed in the dialog box:

```
HRESULT hRes = AtlAdvise(m_pIOPCGroupStateMgt, pCbUnk,
                           IID_IOPCDataCallback, &m_dwAdvise);
if (hRes != S_OK)
{   AfxMessageBox("Advise failed!");
    ...
    return;
}
```

Enabling buttons

Once *OnButtonStart()* has set up all necessary OPC objects, it disables the *Start Sample* button. All other buttons are enabled. This strategy ensures that *OnButtonStart()* is executed once only. This saves additional queries in the program.

```
m_CtrlStop.EnableWindow(TRUE);
m_CtrlRead.EnableWindow(TRUE);
m_CtrlWrite.EnableWindow(TRUE);
m_CtrlChkActive.EnableWindow(TRUE);
m_CtrlStart.EnableWindow(FALSE);
```

The OPCITEMDEF structure

OPCITEMDEF has the following structure:

```
typedef struct {
    LPWSTR szAccessPath;
    LPWSTR szItemID;
    BOOL bActive;
    OPCHANDLE hClient;
    DWORD dwBlobSize;
    BYTE *pBlob;
    VARTYP vtRequestedDataType;
    Word wReserved;
} OPCITEMDEF;
```

Variables of *OPCITEMDEF*:

Variable	Description
szAccessPath	Optional access path for the items (not required by SIMATIC NET).
szItemID	ItemID assigned by the client.
bActive	TRUE if the client will be notified if there is a value change of the item in an active group; FALSE if the client will not be notified.
hClient	Handle for an item assigned by the client. The server transfers the client handle with calls to the client (for example, <i>OnDataChange</i>) so that the client can access the relevant variables in its structures.
dwBlobSize	Size of a memory area on the server in which additional information for faster access to the data of an item is stored.
pBlob	Pointer to the memory area described above.
vtRequestedDataType	Data type requested by the client.

The OPCITEMRESULT structure

OPCITEMRESULT has the following structure:

```
typedef struct {
    OPCHANDLE hServer;
    VARTYPE vtCanonicalDataType;
    WORD wReserved;
    DWORD dwAccessRights;
    DWORD dwBlobSize;
    BYTE *pBlob;
} OPCITEMRESULT;
```

Variables of *OPCITEMRESULT*:

Variable	Description
hServer	A handle for an item assigned by the server. The client transfers the server handle with calls to the server so that the server can access the relevant variables in its structures.
vtCanonicalDataType	The data type used by the server for an item.
dwAccessRights	Information about whether an item can only be read, only be written, or read and written.
dwBlobSize	Size of a memory area on the server in which additional information for faster access to the data of an item is stored.
pBlob	Pointer to the memory area described above.

OnButtonRead()

OnButtonRead() starts an asynchronous read job. The following operations are executed:

Variable declaration and plausibility check

After the declaration of several local variables, the program checks whether an item exists. After successfully calling AddItems in the *OnButtonStart* method, the *m_pItemResult* variable is initialized. The *m_pErrors[0]* array component must have the value *S_OK*, otherwise the item was not created correctly.

```
void COPCDA_AsyncDlg::OnButtonRead()
{
    OPCHANDLE *phServer;
    DWORD dwCancelID;
    HRESULT *pErrors;
    HRESULT r1;
    LPWSTR ErrorStr;
    CString szOut

    if (m_pErrors[0] != S_OK)
    {
        MessageBox("OPC Item0 not available!",
                  "Error Read async",
                  MB_OK+MB_ICONERROR);
        return;
    }
}
```

Read

Read reads values for OPC items asynchronously and is declared as follows:

```

HRESULT Read (DWORD dwCount,
              OPCHANDLE * phServer,
              DWORD dwTransactionID,
              DWORD *pdwCancelID,
              HRESULT ** ppErrors);

```

Parameter	Description
dwCount	Number of items for which values are read.
phServer	Array with the server handles of the items for which values are read.
dwTransactionID	A code number assigned by the client to identify the asynchronous transaction. This is returned by the server with <i>OnReadComplete</i> calls.
pdwCancelID	A code number generated by the server that must be specified if the transaction is to be canceled.
ppErrors Array	Array with elements of the type HRESULT. These variables return an error code if <i>Read()</i> could not be called successfully or information about the successful method call.

The program sets up memory for an array with an element of the type OPCHANDLE and initializes the only array element with the structure component *hServer* of the *m_pItemResult* variable. The OPC server entered a server handle there for the item when *AddItems* was called:

```

phServer = new OPCHANDLE[1];
phServer[0] = m_pItemResult [0].hServer;
r1 = m_pIOPCAsyncIO2 ->Read(1, phServer, 1, &dwCancelID,
                             &pErrors);

```

The return value *S_FALSE* indicates that one or more items could not be read. In this case, the *GetErrorString* method gets the corresponding error message and displays it in a dialog box:

```

if (r1 == S_FALSE)
{   m_pOPCServer ->GetErrorString(pErrors[0], LOCALE_ID,
                                    &ErrorStr);
    szOut.Format ("%s", ErrorStr);
    MessageBox(szOut, "Error Reading Item0",
               MB_OK+MB_ICONERROR);
    CoTaskMemFree(ErrorStr);
}

```

Before the end of the program, the memory reserved by *OnButtonRead()* is released again. With *phServer* array, this is done with *delete* because the memory was created with *new*. The memory area of the *pErrors* array was organized by *CoTaskMemAlloc* and is therefore released with *CoTaskMemFree*:

```

delete[] phServer;
CoTaskMemFree(pErrors);
}

```

OnButtonWrite()

OnButtonWrite() starts an asynchronous write job. The following operations are executed:

Variable declaration and plausibility check

The variable declaration corresponds to the *OnButtonRead()* method. In addition, an array of the type VARIANT is declared for the result of the read operation. To check whether an item exists, the content of the *hServer* structure component is used as in *OnButtonRead()*:

```
void COPCDA_AsyncDlg::OnButtonWrite()
{
    OPCHANDLE *phServer;
    DWORD dwCancelID;
    VARIANT values[1];
    HRESULT *pErrors;
    HRESULT r1;
    LPWSTR ErrorStr;
    CString szOut;

    if (m_pErrors[0] != S_OK)
    {
        MessageBox("OPC Item not available!", "Error Write async",
                  MB_OK+MB_ICONERROR);
        return;
    }
    phServer = new OPCHANDLE[1];
    phServer[0] = m_pItemResult[0].hServer;
    ...
}
```

Write

Write writes values for OPC items asynchronously and is declared as follows:

```
HRESULT Write (DWORD dwCount,
               OPCHANDLE *phServer,
               VARIANT *pItemValues,
               DWORD dwTransactionID,
               DWORD *pdwCancelID
               HRESULT **ppErrors);
```

Parameter	Description
dwCount	Number of items for which values are written.
phServer	Array with the server handles of the items for which values are written.
pItemValues	Array with the values to be written.
dwTransactionID	A code number assigned by the client to identify the asynchronous transaction.
pdwCancelID	A code number generated by the server that must be specified if the transaction is to be canceled.
ppErrors	Array with elements of the type HRESULT. These variables return an error code if <i>Write()</i> could not be called successfully or information about the successful method call.

The sample does not use the *dwTransactionID* parameter, so any value is set:

```
r1 = m_pIOPCAsyncIO2 ->Write(1, phServer, values, 2,
                               &dwCancelID, &pErrors);
```

If *Write* returns *S_FALSE* an error occurred. The program gets the corresponding error message. If *Write* could not be executed, an error message to this effect is displayed in a dialog box.

delete[] phServer releases the memory reserved by *OnButtonWrite()*:

```
delete[] phServer;
if (r1 == S_FALSE)
{   m_pIOPCServer->GetErrorString(pErrors[0], LOCALE_ID,
                                    &ErrorStr);
    m_WriteResult = ErrorStr;
    UpdateData(FALSE);
    CoTaskMemFree(ErrorStr);
}
if (FAILED(r1))
{   szOut.Format("Method call IOPCAsyncIO2::Write failed \
               with error code %x", r1);
    MessageBox(szOut, "Error Writing Item0", MB_OK+MB_ICONERROR);
}
else
{   CoTaskMemFree(pErrors);
}
```

Applying values from the dialog box

The *UpdateData* method call with the TRUE parameter transfers the content of all control variables to the relevant member variables. The value of the *m_WriteVal* member variable of the IDC_EDIT_WRITEVAL text box is stored in the *iVa* component of the only element of *values[]*. The *vt* component is assigned the *VT_I2* type; in other words, *iVa* is to be interpreted as an integer with the length two bytes:

```
UpdateData(TRUE);
values[0].vt = VT_I2;
values[0].iVal = m_WriteVal;
```

OnCheckActivategroup()

OnCheckActivategroup is called when the "Group Active" check box is clicked. It activates and deactivates the group dependent on the status of the check box.

The following operations are executed:

Adopting values from the dialog box

The *UpdateData* method call with the TRUE parameter transfers the content of all control elements to the relevant member variables:

```
void COPCDA_AsyncDlg::OnCheckActivategroup()
{   UpdateData(TRUE);
```

SetState

SetState specifies several properties of a group and is declared as follows:

```
HRESULT SetState (DWORD * pRequestedUpdateRate,
                  DWORD * pRevisedUpdateRate,
```

```

    BOOL *pActive,
    LONG * pTimeBias,
    FLOAT * pPercentDeadband
    DWORD * pLCID,
    OPCHANDLE *phClientGroup);

```

Parameter	Description
pRequestedUpdateRate	Update interval required by the client for data and state changes of items.
pRevisedUpdateRate	The shortest update interval that can be implemented for the group by the server.
pActive	Status of the group: TRUE to activate the group. FALSE to deactivate the group.
pTimeBias	Deviation of the server time from UTC (Universal Time Convention)
pPercentDeadband	Specifies the deadband within which value changes do not lead to notification. This property is, however, only effective when you use analog values. The high and low limits of a value must also be known. In this sample program, this is not the case.
pLCID	Identification number for the local adaptation of the group (language etc.).
phClientGroup	New client handle for the group.

Only the group should be activated so only the *pActive* parameter is given a value. A suitable variable must also be available for the *pRevisedUpdateRate* return parameter:

```

DWORD dwRevUpdateRate;
HRESULT r1= m_pIOPCGroupStateMgt ->SetState
            (NULL,
             &dwRevUpdateRate,
             &m_bChkActivateGroup,
             NULL, NULL, NULL, NULL);

if (FAILED(r1))
{
    MessageBox("Set State failed", "Error",
               MB_OK+MB_ICONERROR);
    return;
}

```

OnButtonStop()

OnButtonStop() removes the OPC objects used in the program and releases the corresponding resources. This method is called when the "Stop Sample" button is clicked, the *OnDestroy* method is executed or the WM_CLOSE message is sent (after clicking the button to close the dialog box or explicitly by calling *SendMessage*).

The following operations are executed:

AtlUnadvise

AtlUnadvise terminates the connection between the OPC server and the callback object and is declared as follows:

```
HRESULT AtlUnadvise (IUnknown*pUnkCP,
                      const IID &iid,
                      DWORD dw);
```

Parameter	Description
pUnkCP	Pointer to the IUnknown interface of the object with which the client is connected.
iid	The identifier of the connection point. Normally the identifier of the interface that implements the callback functionality on the client is specified.
dw	Identification number with which the connection can be identified.

```
void COPCDA_AsyncDlg::OnButtonStop()
{
    HRESULT r1;
    OPCHANDLE *phServer;
    HRESULT *pErrors;
    HRESULT hRes = AtlUnadvise(m_pIOPCGroupStateMgt,
                               IID_IOPCDataCallback,
                               m_dwAdvise);
```

Release

Each COM interface has the *Release* method with which it releases the resources used by the interface. In actual fact, *Release* does not remove any objects from memory, but simply decrements the reference counter of the interface. The objects are actually deleted only when the interface is no longer referenced.

```
m_pIOPCGroupStateMgt ->Release();
```

RemoveItems

The *RemoveItems* method of the *IOPCItemMgt* interface removes OPC items and is declared as follows:

```
HRESULT RemoveItems (DWORD dwCount,
                     OPCHANDLE *phServer,
                     HRESULT **ppErrors);
```

Parameter	Description
dwCount	Number of items to be removed.
phServer	Array with the server handles of the items to be removed.
ppErrors	Array with elements of the type HRESULT. These variables return an error code if <i>RemoveItems()</i> could not be called successfully or information about the successful method call.

This program section sets up memory for an array with an element of the type OPCHANDLE and initializes the array element with the structure component *hServer* of the

m_pItemResult variable. The OPC server entered the server handles there for the items when *AddItems* was called. After the *RemoveItems* call, the memory for this array is released again:

```
phServer = new OPCHANDLE[2];
phServer[0] = m_pItemResult[0].hServer;
phServer[1] = m_pItemResult[1].hServer;
r1 = m_pIOPCItemMgt ->RemoveItems(2, phServer, &pErrors);
if ( (r1 != S_OK) && (r1 != S_FALSE) )
{
    MessageBox("RemoveItems failed!",
               "Error RemoveItems()", 
               MB_OK+MB_ICONERROR);
}
delete[] phServer;
```

The memory areas for *pErrors* and *m_pItemResult* are released with the *CoTaskMemFree* method because they were obtained with COM mechanisms. The resources of *m_pIOPCItemMgt* and *m_pIOPCAsyncIO2* are released by *Release*:

```
CoTaskMemFree(pErrors);
CoTaskMemFree(m_pItemResult);
m_pItemResult =NULL;
CoTaskMemFree(m_pErrors);
m_pErrors =NULL;
m_pIOPCAsyncIO2->Release();
m_pIOPCAsyncIO2 = NULL;
m_pIOPCItemMgt->Release();
m_pIOPCItemMgt = NULL;
```

RemoveGroup

RemoveGroup() removes a group from the server and is declared as follows:

```
HRESULT RemoveGroup (OPCHANDLE hServerGroup, BOOL bForce);
```

Parameter	Description
hServerGroup	Server handle of the group to be removed.
bForce	Specifies whether or not a group can be removed when there is still a reference to it.

```
r1 = m_pIOPCServer->RemoveGroup(m_GrpSrvHandle, TRUE);
if (r1 != S_OK)
{
    MessageBox("RemoveGroup failed!",
               "Error RemoveGroup()", 
               MB_OK+MB_ICONERROR);
}
m_GrpSrvHandle = NULL;
```

The resources of the OPC server object are released with *Release*, *CoUninitialize* closes the COM library:

```
m_pIOPCServer->Release();  
m_pIOPCServer = NULL;  
CoUninitialize();
```

Finally, the program disables all controls apart from the "Start Sample" button:

```
m_CtrlStart.EnableWindow(TRUE);  
m_CtrlStop.EnableWindow(FALSE);  
m_CtrlRead.EnableWindow(FALSE);  
m_CtrlWrite.EnableWindow(FALSE);  
m_CtrlChkActive.EnableWindow(FALSE);  
}
```

OnDestroy

This method of the CWnd class is overwritten so that all the "clean up jobs" can be done when the dialog window is closed. *OnButtonStop()* deletes all the OPC objects and releases the corresponding resources. Following this, *OnDestroy* of the parent class is called to release the memory used by the window object:

```
void COPCDA_AsyncDlg::OnDestroy()  
{  
    if (m_pIOPCServer)  
        OnButtonStop();  
    CDialog::OnDestroy();  
}
```

DisplayData

The dialog box class has three methods to display data after a message from the OPC server. Different methods are necessary because the various notifications from the server return different data.

The following methods are available:

- After data have changed (one method per item):

```
void DisplayData0 (CString ReadVal, CString ReadQuality, CString ReadTS) void  
DisplayData1 (CString ReadVal, CString ReadQuality, CString ReadTS)
```

- On completion of a read operation:

```
void DisplayData (long Value, CString Quality, CString TimeStamp)
```

- On completion of a write operation:

```
void DisplayData (HRESULT ErrorCode)
```

DisplayData0 (CString ReadVal, CString ReadQuality, CString ReadTS)

This version of *DisplayData* is called by the *OnDataChange* method of the Callback class and updates the display in the text boxes in the "On Data Changed" frame.

Note

In this case, the text boxes of Item0 are updated. To update the text boxes of Item1, the analogous *DisplayData1* method is implemented.

```
void COPCDA_AsyncDlg::DisplayData0(CString ReadVal,
                                    CString ReadQuality,
                                    CString ReadTS)
```

Parameter	Description
ReadVal	Value that was read.
ReadQuality	Information on the integrity of the data.
ReadTS	Time stamp (time at which the data was received from the OPC server).

The parameter values are assigned to the member variables of the dialog box. *UpdateData* transfers the new data from the member variables to the corresponding controls:

```
{
    m_Quality0 = ReadQuality;
    m_TimeStamp0 = ReadTS;
    m_Value0 = ReadVal;
    UpdateData(FALSE);
}
```

DisplayData (long Value, CString Quality, CString TimeStamp)

The *OnReadComplete()* method of the Callback class calls this method to update the display in the text boxes to the right of the "Read Value" button:

```
void COPCDA_AsyncDlg::DisplayData(long Value,
                                    CString Quality,
                                    CString TimeStamp)
```

Parameter	Description
Value	The value that was read.
Quality	Information on the integrity of the data.
TimeStamp	The time at which the data were received from the OPC server.

The parameters are assigned to the relevant member variables of the dialog box. *UpdateData* transfers the new data from the member variables to the corresponding controls:

```
{
    m_ReadValue = Value;
    m_QualityRead = Quality;
    m_TimeStampRead = TimeStamp;
```

```
    UpdateData(FALSE);  
}
```

DisplayData (HRESULT ErrorCode)

The *OnWriteComplete()* method of the Callback class calls this method to display a status message about the result of a write operation in text box to the right of the "Write Value" button:

```
void COPCDA_AsyncDlg::DisplayData(HRESULT ErrorCode)
```

Parameter	Description
ErrorCode	The error code returned by the OPC server.

The *GetErrorString()* method of the *IOPCServer* interface gets the error message belonging to the error code. This message is assigned to the member variable *m_WriteResult*. Before *UpdateData* displays this message in the text box, the line break characters must be removed with *Remove()*:

```
{    LPWSTR ErrorStr;  
    m_pIOPCServer->GetErrorString(ErrorCode, LOCALE_ID,  
                                    &ErrorStr);  
  
    m_WriteResult = ErrorStr;  
    m_WriteResult.Remove('\r');  
    m_WriteResult.Remove('\n');  
    UpdateData(FALSE);  
    CoTaskMemFree(ErrorStr);  
}
```

UpdateData

In certain situations, changing threads during access to a resource can lead to inconsistent data or undefined states. In such cases, a thread change must be prevented explicitly. The *CCriticalSection* class is available for this purpose. After creating an object of this class, it is only possible to change threads again when the *Unlock()* method is called.

The sample program uses this mechanism to update the controls without errors occurring. An object of the *CCriticalSection* class is created after the program starts. Immediately following the *UpdateData* method call of the parent class, *Unlock* releases the resources again for other threads:

```
BOOL COPCDA_AsyncDlg::UpdateData( BOOL bSaveAndValidate )  
{    BOOL bRes;  
    m_cCritSec.Lock();  
    bRes = CDialog::UpdateData( bSaveAndValidate );  
    m_cCritSec.Unlock();  
    return bRes;  
}
```

6.3.10 Callback.cpp and Callback.h

In the sample program, the *COPCDataCallback* class is the implementation of the IOPCDataCallback interface. This class provides the following methods which notifications can be received from the OPC server:

- OnDataChange()
- OnReadComplete()
- OnWriteComplete()
- InformAboutDialog

The file also includes the auxiliary function *GetQualityText*

OnDataChange

The OPC server calls this method when data changes:

```
STDMETHODIMP COPCDataCallback::OnDataChange(
    DWORD dwTransid,
    OPCHANDLE hGroup,
    HRESULT hrMasterquality,
    HRESULT hrMastererror,
    DWORD dwCount,
    OPCHANDLE __RPC_FAR *phClientItems,
    VARIANT __RPC_FAR *pvValues,
    WORD __RPC_FAR *pwQualities,
    FILETIME __RPC_FAR *pftTimeStamps,
    HRESULT __RPC_FAR *pErrors)
```

Parameter	Description
dwTransid	The code number for the type of method call is zero if a value change was the cause of the method call, if the code number is not zero, the method call was triggered by a refresh.
hGroup	The client handle of the group to allow the client to identify the group
hrMasterquality	Information on the integrity of the data: S_OK, if OPC_QUALITY_MASK for all items has the value OPC_QUALITY_GOOD, otherwise S_FALSE.
hrMastererror	S_OK, if the error message for all items is S_OK, otherwise S_FALSE.
dwCount	Number of items for which values exist.
phClientItems	Array with the client handles of the items whose values have changed.
pvValues	Array of the type VARIANT containing the values of the items that have changed.
pwQualities	Array with information about the integrity of the values of the individual items
pftTimeStamps	Array with time information for the individual items
pErrors	Array with error messages for the items

The task of this method is to convert the values transferred by the OPC server as parameters into objects of the type CString. The *DisplayData* method expects input parameters of this type.

The program creates three arrays for *CString* objects to store the values of the items, information on the quality of the values, and time information:

```
{    CString szReadVal;
    CString szReadQuality;
    CString szReadTS;
```

CComVariant is a wrapper class for the VARIANT data type. A *CComVariant* object is created for each item in a loop. For this, the constructor is used that contains a VARIANT value as parameter:

```
CComVariant *pvarOut;
DWORD i;
for (i = 0; i<dwCount; i++)
{    pvarOut = new CComVariant(pvValues[i]);
```

In the next step, the *ChangeType* method converts the value that was read into a string. Since this is still an object of the *CComVariant* class, it is possible to access the string via the *bstrVal* component of *pvarOut*. The result of the *Format* method is the required *CString* object:

```
pvarOut->ChangeType(VT_BSTR);
szReadVal.Format("%ls", pvarOut->bstrVal);
```

Since we are interested only in the *szReadVal* variable, the *pvarOut* object is deleted immediately after it has been used:

```
delete pvarOut;
```

GetQualityText returns quality information on the value that was read as a *CString* object. The parameter for this method is an identification number of the OPC server:

```
szReadQuality = GetQualityText(pwQualities[i]);
```

The OPC server returns the time stamps of the items as a variable of the type FILETIME. The result of the *Format* method of the *COleDateTime* class is the time as a *CString* object:

```
szReadTS = COleDateTime(pftTimeStamps[i]).Format();
}
```

Once all the results have been converted to *CString* objects, the program calls either *DisplayData0* method or the *DisplayData1* method of the *COPCDA_AsyncDlg* class:

```
switch (phClientItems[i])
{    case 0:
        m_pCDlgClass->DisplayData0(szReadVal,
                                     szReadQuality,
                                     szReadTS);
        break;
    case 1:
        m_pCDlgClass->DisplayData1(szReadVal,
                                     szReadQuality,
                                     szReadTS);
        break;
```

```

        default: ASSERT(0);
    }
}

return S_OK;
};

```

OnReadComplete()

The OPC server calls this method when an asynchronous read is completed. The parameters of *OnReadComplete()* match those of *OnDataChange*:

```

STDMETHODIMP COPCDataCallback::OnReadComplete(
    DWORD dwTransid,
    OPCHANDLE hGroup,
    HRESULT hrMasterquality,
    HRESULT hrMastererror,
    DWORD dwCount,
    OPCHANDLE __RPC_FAR *phClientItems,
    VARIANT __RPC_FAR *pvValues,
    WORD __RPC_FAR *pwQualities,
    FILETIME __RPC_FAR *pftTimeStamps,
    HRESULT __RPC_FAR *pErrors)

```

Parameter	Description
dwTransid	The code number for the type of method call is zero if a value change was the cause of the method call, if the code number is not zero, the method call was triggered by a refresh.
hGroup	The client handle of the group to allow the client to identify the group
hrMasterquality	Information on the integrity of the data: S_OK, if OPC_QUALITY_MASK for all items has the value OPC_QUALITY_GOOD, otherwise S_FALSE.
hrMastererror	S_OK, if the error message for all items is S_OK, otherwise S_FALSE.
dwCount	Number of items for which values exist.
phClientItems	Array with the client handles of the items whose values have changed.
pvValues	Array of the type VARIANT containing the values of the items that have changed.
pwQualities	Array with information about the integrity of the values of the individual items
pftTimeStamps	Array with time information for the individual items
pErrors	Array with error messages for the items

Before processing the data delivered by the OPC server, the program checks the *pErrors* variable. If the read was errorfree, the data is evaluated as in the *OnDataChange* program. In this case, however, the value of the item is converted to the *Long Integer* type:

```

if (pErrors[0] == S_OK)
{
    CComVariant * pvarOut;
    pvarOut = new CComVariant(pvValues[0]);
    pvarOut->ChangeType(VT_I4);
    CString readQuality = GetQualityText(pwQualities[0]);
    CString readTS = COleDateTime(pftTimeStamps[0]).Format();
}

```

```
    m_pCDlgClass->DisplayData(pvarOut->intval, readQuality, readTS);
    delete pvarOut;
}
```

If an error occurred, the *GetQualityText* method returns the message belonging to the error code:

```
else
{
    CString readQuality = GetQualityText(pErrors[0]);
    m_pCDlgClass->DisplayData(0, readQuality, "");
}
return S_OK;
};
```

OnWriteComplete()

The OPC server calls this method when an asynchronous write is completed.

```
STDMETHODIMP COPCDataCallback::OnWriteComplete(
    DWORD dwTransid,
    OPCHANDLE hGroup,
    HRESULT hrMastererr,
    DWORD dwCount,
    OPCHANDLE __RPC_FAR *pClienthandles,
    HRESULT __RPC_FAR *pErrors)
```

Parameter	Description
dwTransid	An identification number for the asynchronous write
hGroup	The client handle of the group
hrMastererr	S_OK, if the error message for all items is S_OK, otherwise S_FALSE.
dwCount	Number of items for which values were written.
pClienthandles	Array with the client handles of the items for which values were written.
pErrors	Array with error messages for the items

OnWriteComplete() calls the *DisplayData* method of the *COPCDA_AsyncDlg* class to display the error message of the OPC server:

```
{    m_pCDlgClass->DisplayData(pErrors[0]);
    return S_OK;
};
```

InformAboutDialog

InformAboutDialog returns a reference to the dialog box:

```
void InformAboutDialog (COPCDA_AsyncDlg *pCDlgClass)
{
    m_pCDlgClass = pCDlgClass;
}
```

GetQualityText

GetQualityText provides a CString object as an error message for a specified error code:

```
CString GetQualityText(UINT qnr)
{
    CString qstr;
    switch(qnr)
    {
        case OPC_QUALITY_BAD:
            qstr = "BAD";
            break;
        case OPC_QUALITY_UNCERTAIN:
            qstr = "UNCERTAIN";
            break;
        case OPC_QUALITY_GOOD:
            qstr = "GOOD";
            break;
        case OPC_QUALITY_NOT_CONNECTED:
            qstr = "NOT_CONNECTED";
            break;
        case OPC_QUALITY_DEVICE_FAILURE:
            qstr = "DEVICE_FAILURE";
            break;
        case OPC_QUALITY_SENSOR_FAILURE:
            qstr = "SENSOR_FAILURE";
            break;
        case OPC_QUALITY_LAST_KNOWN:
            qstr = "LAST_KNOWN";
            break;
        case OPC_QUALITY_COMM_FAILURE:
            qstr = "COMM_FAILURE";
            break;
        case OPC_QUALITY_OUT_OF_SERVICE:
            qstr = "OUT_OF_SERVICE";
            break;
        case OPC_QUALITY_LAST_USABLE:
            qstr = "LAST_USABLE";
            break;
        case OPC_QUALITY_SENSOR_CAL:
            qstr = "SENSOR_CAL";
            break;
        case OPC_QUALITY_EGU_EXCEEDED:
            qstr = "EGU_EXCEEDED";
            break;
        case OPC_QUALITY_SUB_NORMAL:
            qstr = "SUB_NORMAL";
            break;
        case OPC_QUALITY_LOCAL_OVERRIDE:
            qstr = "LOCAL_OVERRIDE";
            break;
        default: qstr = "UNKNOWN ERROR";
    }
    return qstr;
}
```

6.3.11 Notes on writing your own programs

Several requirements must be met before your own programs can use the Custom interface. Follow the steps outlined below:

1. Start the Visual C++ development environment.
2. Create a project of the required type using the MFC Class Wizard.
3. Copy the files "pre_opc.h" and "pre_opc.cpp" from the sample program to your project directory and add the files to the project ("Project" > "Add to project" > "Files...").
These files contain OPCspecific definitions and include statements.
4. Add the following statement to all the implementation files of your project (extension "*.cpp"): `#include pre_opc.h`
5. In the implementation files (extension ".*cpp") that use the *AddGroup()* or *GetErrorString()* methods, add the following statement: `#define LOCALE_ID 0x409`
6. Arrange the required operator interface elements in the main dialog or create a suitable application window and program the corresponding event procedures.
7. Implement a callback object if you want to use asynchronous operations over ATL. The best course is to define your own class with the required methods.

6.4 OPC Custom interface (asynchronous communication) in VB.NET

Requirements for using the sample program

After installing the SIMATIC NET software, you will find this program on your hard disk in the following folder:

"<installation path>\SIEMENS\SIMATIC.NET\opc2\samples\dotnet\vb\async.net"

A .NET-Framework version 4.0 or higher must be installed on the computer on which you want the sample program to run. You also need to activate a simulation connection so that the demonstration variables used in the program are available (see section "Activating the simulation connection (Page 647)").

Runtime callable wrapper

The sample uses the Custom interface for OPC Data Access V2.0 with .NET Framework. Access to the COM interface of the managed code of .NET applications uses the Runtime Callable Wrapper (RCW) of the OPC Foundation.

The primary function of Runtime Callable Wrappers is the marshaling of calls between a .NET client and an unmanaged COM object. Marshaling means making memory areas available.

6.4.1 Working with the sample program

The program includes several elements that the operator can use to trigger the following actions:

Operator Element	Effect
Start Sample	Start program
Read Value or Write Value	Read or write values
Group Active	Activate or deactivate group
Stop Sample	Stop the program

When you start the program, only the "Start Sample" button is enabled. After you have clicked this button, the program generates the required OPC objects. You can then use the other buttons.

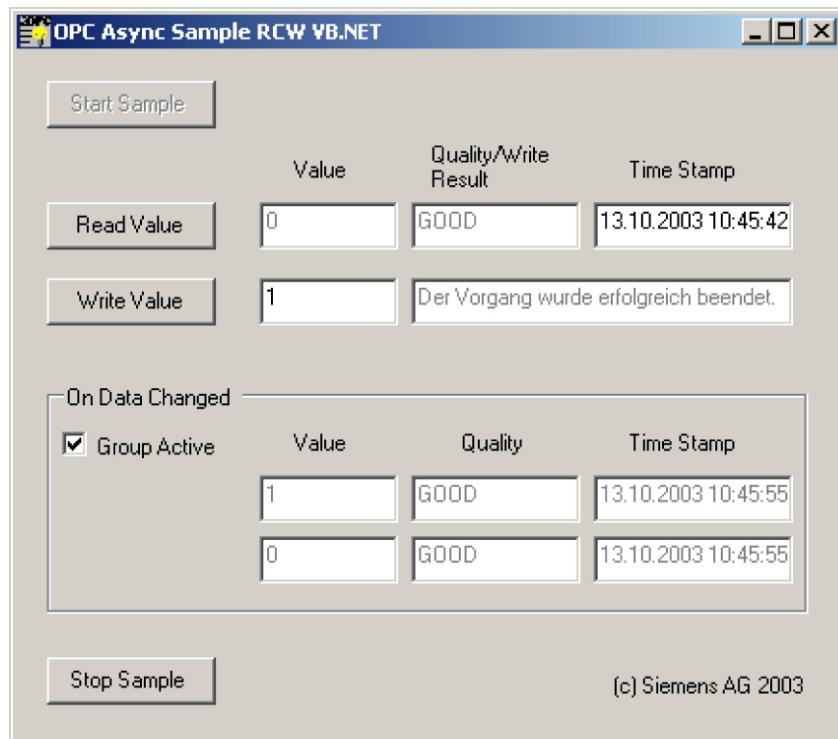


Figure 6-24 Start dialog of the sample program for the OPC custom interface (asynchronous communication) in VB.NET with .NET-Framework after clicking the "Start Sample" button

6.4.2 Description of the program

Introduction

The basic sequence of the individual programming steps corresponds to the previous example "OPC Custom interface (asynchronous communication) in C++ (Page 646)". These include establishing the connection to the OPC server, creating a group with variables, and reading and writing values for an item. The following table shows the steps to be executed by the program:

Step	Description
1	Select .NET components.
2	Convert the ProgID to a CLSID.
3	Establish connection to the OPC server.
4	Generate an OPC group.
5	Add Items.
6	Request the interface IConnectionPointContainer.
7	Request the interface IOPCAsyncIO2.
8	Create callback object.
9	Connect OPC server and callback object of the client.
10	Execute required write and read operation.
11	Receive notifications of the OPC server.
12	Delete objects and release memory.

Step 1: Select .NET components

To use the Custom Interface for OPC Data Access with .NET, you must use the Runtime Callable Wrapper (RCW) described above in your Visual Basic project. In the sample project, the components *OpcRcw.Da* and *OpcRcw.Comn* have already been added.

If these do not appear in the selection, you can also select them using the *Browse* button from the following folder

"<installation path>\SIEMENS\SIMATIC.NET\opc2\bin"

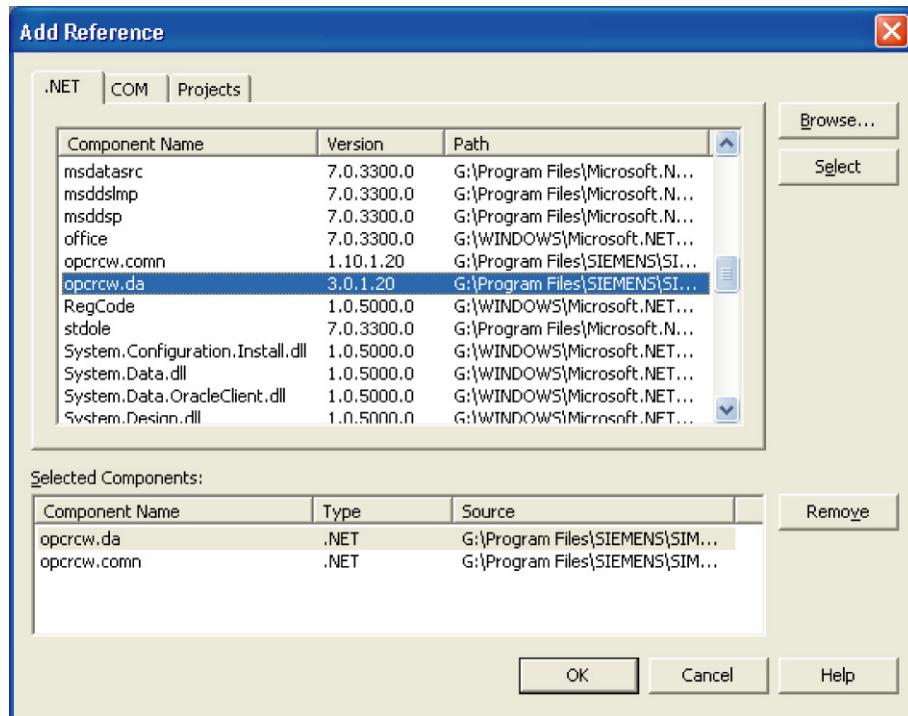


Figure 6-25 Selecting the "Runtime Callable Wrapper" (RCW) in the Visual Basic project

With the commands

```
Imports OpcRcw.Comm
Imports OpcRcw.Da
```

it is simple to use the namespace and methods of the OPC Custom interface with .NET Framework.

Step 2: Convert the ProgID to a type

To identify it, each COM server has a *ProgID* that is assigned to a worldwide unique type. This is obtained with the *GetTypeFromProgID()* function. The ProgID of the OPC server of SIMATIC NET is L"OPC.SimaticNET":

```
Dim typeOfOPCserver As Type =
Type.GetTypeFromProgID(L"OPC.SimaticNET")
```

Step 3: Establish connection to the OPC server

The *CreateInstance()* function of the *Activator* class generates an instance of the class with a previously specified type:

```
m_server = Activator.CreateInstance(typeOfOPCserver)
```

The result of this section of program is an interface variable *m_server* of the type *IOPCServer*.

Step 4: Create an OPC group

The IOPCServer interface has the *AddGroup()* method for creating groups:

```
m_server.AddGroup("MyOPCGroup",
    0,
    250,
    1,
    pTimeBias,
    pDeadband,
    LOCALE_ID,
    ServerGroup,
    RevisedUpdateRate,
    GetType(IOPCGroupStateMgt).GUID,
    m_group)
```

The result of this program section is a group with the specified name and the required properties. *AddGroup()* also returns a variable *m_group* as return parameter, an interface to a group object, in this case *IOPCGroupStateMgt*. The *IOPCGroupStateMgt* interface is necessary to allow use of the *SetState()* methods. The *SetState()* method is required later to activate and deactivate a group.

By means of a simple assignment, the *m_item* variable of the type *IOPCItemMgt* can be derived from the *m_group* interface variable.

```
m_item = m_group
```

Step 5: Add items

The IOPCItemMgt interface has the *AddItems()* method for creating OPC items:

```
m_item.AddItems(2, itemdefs, Results, pErrors)
```

The result of this program section is that the server adds two items with the properties specified in the *itemdefs* parameter. The variables of the result structure *Results* (server handle, data type of the item on the target system etc.) are also initialized.

Since the results are written to unmanaged memory by the underlying COM interface, the access from managed .NET code must make use of the marshaling functions:

```
result = Marshal.PtrToStructure(pos, GetType(OPCITEMRESULT))
ServerHandle1 = result.hServer
pos = New IntPtr(pos.ToInt32() +
    Marshal.SizeOf(GetType(OPCITEMRESULT)))
result = Marshal.PtrToStructure(pos, GetType(OPCITEMRESULT))
ServerHandle2 = result.hServer
```

The client must also make sure that the unmanaged memory is released:

```
Marshal.FreeCoTaskMem(Results)
Marshal.FreeCoTaskMem(pErrors)
```

Step 6: Request the interface *IConnectionPointContainer*

The *m_ConnectionContainer* variable of the type *IConnectionPointContainer* can be derived from the *m_group* variable.

```
m_ConnectionContainer = m_group
```

The interface is required to locate the *IConnectionPoint* interface.

Step 7: Request the *IOPCAsyncIO2* interfaces

The *m_asyncIO2* variable of the type *IOPCAsyncIO2* can be derived from the *m_group* variable.

```
m_asyncIO2 = m_group
```

This interface provides methods for asynchronous reading and writing of values.

Step 8: Create callback object

To allow the OPC server to return a result of asynchronous operations, the *IOPCDataCallback* interface must be implemented on the client.

```
Implements IOPCDataCallback
```

The connection of the *IConnectionPoint* interface and *IOPCDataCallback* of the OPC server is established using the *FindConnectionPoint()* method of the *IConnectionPointContainer* interface. This generates a callback object for asynchronous calls.

```
m_ConnectionContainer.FindConnectionPoint(
    GetType(IOPCDataCallback).GUID,
    m_ConnectionPoint)
```

Step 9: Connect OPC server and callback object of the client

The connection between the callback object of the OPC server and the client application is established by the *Advise()* method. The return variable *m_dwCookie* is an ID for the connection.

```
m_ConnectionPoint.Advise(Me, m_dwCookie)
```

Step 10: Execute required write or read operation

The *Read()* and *Write()* methods that were generated in Step 7 can be accessed over the *IOPCAsyncIO2* interface:

```
m_asyncIO2.Read(1, pServer, Transaction, CancelID, pErrors)
```

Step 11: Receive notifications of the OPC server

If the data of an active item in an active group changes, the server calls the *OnDataChange()* method of the callback object. On completion of a read operation, the server calls the *OnReadComplete()* method, on completion of a write operation, the *OnWriteComplete()* method is called. The server passes the return values as parameters to the method.

```
Overridable Sub OnDataChange(
    ByVal dwTransid As Integer,
    ByVal hGroup As Integer,
    ByVal hrMasterquality As Integer,
    ByVal hrMastererror As Integer,
    ByVal dwCount As Integer,
    ByVal phClientItems() As Integer,
    ByVal pvValues() As Object,
    ByVal pwQualities() As Short,
    ByVal pftTimeStamps() As OpcRcw.Da.FILETIME,
    ByVal pErrors() As Integer) Implements
    IOPCDataCallback.OnDataChange
```

Step 12: Delete objects and release memory

Before exiting the program, the OPC objects that have been created must be deleted and the memory reserved for them must be released. The relevant functions are integral parts of the previously used interfaces. *Unadvise()* terminates the connection between the OPC server and the callback object:

```
m_ConnectionPoint.Unadvise(m_dwCookie)
Marshal.ReleaseComObject(m_ConnectionPoint)
m_item.RemoveItems(2, pItems, pErrors)
....
Marshal.FreeCoTaskMem(pErrors)
m_server.RemoveGroup(ServerGroup, True)
```

Releasing memory

When using COM, it is sometimes necessary for the client to release memory that was requested by the server. The rules for this are as follows:

- [out]: The memory for a parameter with this attribute will be requested from the server.
- [in, out]: The memory for such a parameter will be requested from the client. The server can release the memory again and reallocate it.
- [in]: The client will request the memory. The server is not responsible for releasing the memory.

In all three cases, the reserved memory of unmanaged COM objects should be released by the client.

6.5 OPC Custom interface (synchronous communication) in C#

Requirements for using the sample program

After installing the SIMATIC NET software, you will find this program on your hard disk in the following folder:

"<installation path>\SIEMENS\SIMATIC.NET\opc2\samples\dotnet\C#\sync.net"

A .NET-Framework version 4.0 or higher must be installed on the computer on which you want the sample program to run. You also need to activate a simulation connection so that the demonstration variables used in the program are available (see section "Activating the simulation connection (Page 647)").

Runtime callable wrapper

The sample uses the Custom interface for OPC Data Access V2.0 with .NET Framework. Access to the COM interface of the managed code of .NET applications uses the Runtime Callable Wrapper (RCW) of the OPC Foundation.

The primary function of Runtime Callable Wrappers is the marshaling of calls between a .NET client and an unmanaged COM object. Marshaling means making memory areas available.

6.5.1 Working with the sample program

The application includes several elements that the operator can use to trigger the following actions:

Operator Element	Effect
Start Sample	Start application
Read item or write value	Read item or write value
Stop Sample	Exit application

When you start the program, only the "Start Sample" button is enabled. After you have clicked this button, the application generates the required OPC objects. The other buttons are then enabled.

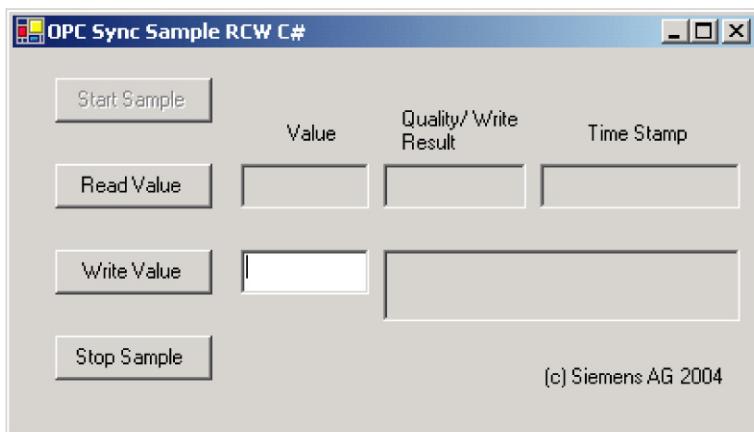


Figure 6-26 Start dialog of the sample program for the OPC custom interface (synchronous communication) in C# with .NET-Framework after clicking the "Start Sample" button

6.5.2 Description of the program

Introduction

The basic sequence of the individual programming steps corresponds to the previous example "OPC Custom interface (synchronous communication) in C++ (Page 625)". These include establishing the connection to the OPC server, creating a group with variables, and reading and writing values for an item. The following table shows the steps to be executed by the program:

Step	Description
1	Select .NET components
2	Convert the ProgID to a CLSID
3	Establish connection to the OPC server
4	Create an OPC group
5	Add items
6	Request the interface IOPCSyncIO
7	Execute required write and read operations
8	Delete objects and release memory

Step 1: Select .NET components

See section "Description of the program (Page 684)" step 1

Step 2: Convert the ProgID to a CLSID

To identify it, each COM server has a *ProgID* that is assigned to a worldwide unique type. This is obtained with the *GetTypeFromProgID()* function. The ProgID of the OPC server of SIMATIC NET is L"OPC.SimaticNET":

```
Type svrComponenttyp = Type.GetTypeFromProgID("OPC.SimaticNET");
```

Step 3: Establish connection to the OPC server

The *CreateInstance()* function of the Activator class generates an instance of the class with a previously specified type:

```
pIOPCServer =
    (IOPCServer)Activator.CreateInstance(svrComponenttyp);
```

The result of this section of program is an interface variable *pIOPCServer* of the type IOPCServer.

Step 4: Create an OPC group

The IOPCServer interface has the *AddGroup()* method for creating groups:

```
pIOPCServer.AddGroup(GROUP_NAME,
    0,
    dwRequestedUpdateRate,
    hClientGroup,
    hTimeBias.AddrOfPinnedObject(),
    hDeadband.AddrOfPinnedObject(),
    dwLCID,
    out pSvrGroupHandle,
    out pRevUpdateRate,
    ref out pobjGroup1)
```

Note

The memory allocation for hDeadband and hTimeBias is made by the GCHandle.Alloc() function.

```
GCHandle hTimeBias;
hTimeBias = GCHandle.Alloc(timebias,GCHandleType.Pinned);

GCHandle hDeadband;
hDeadband = GCHandle.Alloc(deadband,GCHandleType.Pinned);
```

These functions handle the unmanaged memory of the transfer variables timebias and deadband (Percent Deadband).

The management handles must be released again as shown below when they are no longer required:

```
if (hTimeBias.IsAllocated) hTimeBias.Free();
if (hDeadband.IsAllocated) hDeadband.Free();
```

The result of this program section is a group with the specified name and the required properties. *AddGroup()* also returns a variable *pobjGroup1* as the return parameter, an interface to a group object.

With the type adaptation call *IOPCGroupStateMgt*, the interface can be obtained simply from the returned group interface *pobjGroup1*. This call simplifies the COM method

QueryInterface().

The *SetState()* method of the IOPCGroupStateMgt interface is required later to activate and deactivate a group.

```
pIOPCGroupStateMgt = (IOPCGroupStateMgt)pobjGroup1;
```

Step 5: Add items

The *IOPCItemMgt* interface has the *AddItems()* method for creating OPC items:

```
((IOPCItemMgt)pobjGroup1).AddItems(1, ItemDefArray, out pResults, out  
pErrors);
```

The result of this program section is that the server adds an item with the properties specified in the *itemdefs* parameter. The variables of the result structure *OPCITEMRESULT* (server handle, data type of the item on the target system etc.) are also initialized.

Since the results are written to unmanaged memory by the underlying COM interface, the access from managed .NET code must make use of the marshaling functions:

```
OPCITEMRESULT result =  
    OPCITEMRESULT)Marshal.PtrToStructure(pResults,  
    typeof(OPCITEMRESULT));
```

The client must also make sure that the unmanaged memory is released:

```
Marshal.FreeCoTaskMem(Results)  
Marshal.FreeCoTaskMem(pErrors)
```

Step 6: Request the interface IOPCSyncIO

The *pobjGroup1* variable of the type *pIOPCSyncIO* can be derived from the *pIOPCSyncIO* variable.

```
pIOPCSyncIO2 = (IOPCSyncIO2)pobjGroup1;
```

The interface provides methods for synchronous writing and reading of values.

Step 7: Execute required write or read operation

With the *Read()* and *Write()* methods, values of OPC items can be accessed over the IOPCSyncIO interface:

```
pIOPCSyncIO.Read(OPCDATASOURCE.OPC_DS_DEVICE, 1, nItemSrvID, out  
pItemValues, out pErrors);  
pIOPCSyncIO.Write(1, nItemSrvID, values, out pErrors);
```

Since the results are written to unmanaged memory by the underlying COM interface, the access from managed .NET code must make use of the marshaling functions:

```
OPCITEMSTATE pItemState =
    (OPCITEMSTATE)Marshal.PtrToStructure(pItemValues,
    typeof(OPCITEMSTATE));
```

The client must also make sure that the unmanaged memory is released:

```
Marshal.FreeCoTaskMem(pItemValues);
Marshal.FreeCoTaskMem(pErrors);
```

Step 8: Delete objects and release memory

Before exiting the program or if Stop is clicked, the generated OPC object must be deleted and the occupied memory released.

```
Marshal.ReleaseComObject(pIOPCSyncIO);
pIOPCServer.RemoveGroup(nSvrGroupID, 0);
Marshal.ReleaseComObject(pobjGroup1);
Marshal.ReleaseComObject(pIOPCServer);
```

Releasing memory

When using COM, it is sometimes necessary for the client to release memory that was requested by the server. The rules for this are as follows:

- [out]: The memory for a parameter with this attribute will be requested from the server.
- [in, out]: The memory for such a parameter will be requested from the client. The server can release the memory again and reallocate it.
- [in]: The client will request the memory. The server is not responsible for releasing the memory.

In all three cases, the reserved memory of unmanaged COM objects should be released by the client.

6.6 OPC Custom interface (asynchronous communication) in C#

Requirements for using the sample program

After installing the SIMATIC NET software, you will find this program on your hard disk in the following folder:

"<installation path>\SIEMENS\SIMATIC.NET\opc2\samples\dotnet\C#\async.net"

A .NET-Framework version 4.0 or higher must be installed on the computer on which you want the sample program to run. You also need to activate a simulation connection so that the demonstration variables used in the program are available (see section "Activating the simulation connection (Page 647)").

Runtime callable wrapper

The sample uses the Custom interface for OPC Data Access V2.0 with .NET Framework. Access to the COM interface of the managed code of .NET applications uses the Runtime Callable Wrapper (RCW) of the OPC Foundation.

The primary function of Runtime Callable Wrappers is the marshaling of calls between a .NET client and an unmanaged COM object. Marshaling means making memory areas available.

6.6.1 Working with the sample program

The program includes several elements that the operator can use to trigger the following actions:

Operator Element	Effect
Start Sample	Start program
Read item or write value	Read item or write value
Group Active	Activate or deactivate group
Stop Sample	Stop the program

When you start the program, only the "Start Sample" button is enabled. After you have clicked this button, the program generates the required OPC objects. The other buttons are then enabled.

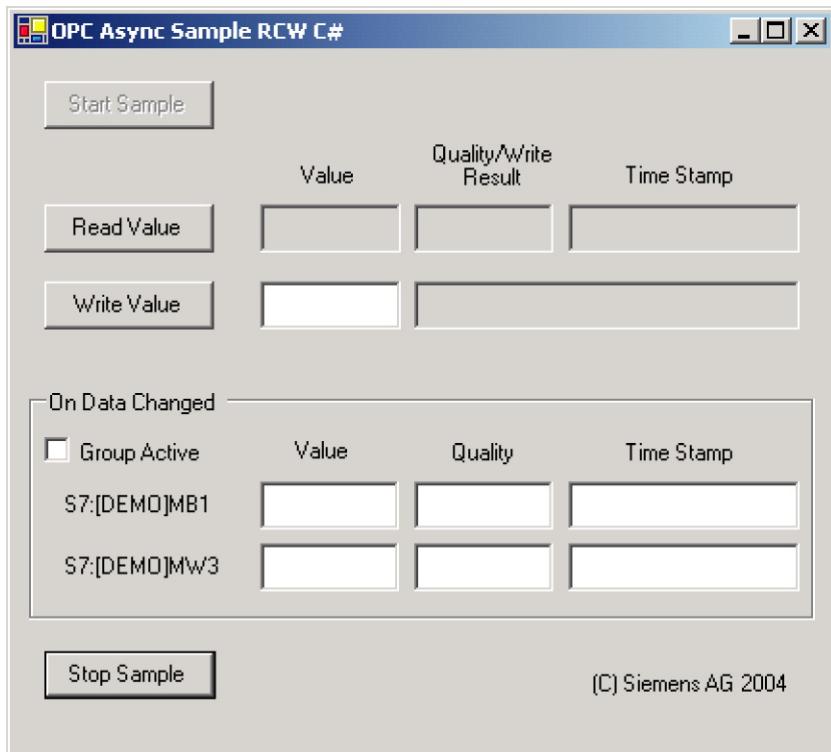


Figure 6-27 Start dialog of the sample program for the OPC custom interface (asynchronous communication) in C# with .NET-Framework after clicking the "Start Sample" button

6.6.2 Description of the program

Introduction

The basic sequence of the individual programming steps corresponds to the previous example "OPC Custom interface (asynchronous communication) in VB.NET (Page 682)". These include establishing the connection to the OPC server, creating a group with variables, and reading and writing values for an item. The following table shows the steps to be executed by the program:

Step	Description
1	Select .NET components
2	Convert the ProgID to a CLSID
3	Establish connection to the OPC server
4	Create an OPC group
5	Add items
6	Request the interface <code>IConnectionPointContainer</code>
7	Request the interface <code>IOPCAsyncIO2</code>
8	Create callback object
9	Connect OPC server and callback object of the client

Step	Description
10	Execute required write and read operations
11	Receive notifications of the OPC server
12	Delete objects and release memory

Step 1: Select .NET components

See section "Description of the program (Page 684)" step 1

Step 2: Convert the ProgID to a CLSID

To identify it, each COM server has a *ProgID* that is assigned to a worldwide unique type. This is obtained with the *GetTypeFromProgID()* function. The ProgID of the OPC server of SIMATIC NET is L"OPC.SimaticNET":

```
Type svrComponenttyp = Type.GetTypeFromProgID(L"OPC.SimaticNET");
```

Step 3: Establish connection to the OPC server

The *CreateInstance()* function of the Activator class generates an instance of the class with a previously specified type:

```
pIOPCServer =
    (IOPCServer)Activator.CreateInstance(svrComponenttyp);
```

The result of this section of program is an interface variable *pIOPCServer* of the type *IOPCServer*.

Step 4: Create an OPC group

The IOPCServer interface has the *AddGroup()* method for creating groups:

```
pIOPCServer.AddGroup(GROUP_NAME,
    0,
    dwRequestedUpdateRate,
    hClientGroup,
    hTimeBias..AddrOfPinnedObject(),
    hDeadband.AddrOfPinnedObject(),
    dwLCID,
    out pSvrGroupHandle,
    out pRevUpdateRate,
    ref out pobjGroup1)
```

Note

The memory allocation for *hDeadband* and *hTimeBias* is made by the *GCHandle.Alloc()* function.

```
GCHandle hTimeBias;
hTimeBias = GCHandle.Alloc(timebias,GCHandleType.Pinned);
```

```
GCHandle hDeadband;
hDeadband = GCHandle.Alloc(deadband, GCHandleType.Pinned);
```

These functions handle the unmanaged memory of the transfer variables timebias and deadband (Percent Deadband).

The management handles must be released again as shown below when they are no longer required:

```
if (hTimeBias.IsAllocated) hTimeBias.Free();
if (hDeadband.IsAllocated) hDeadband.Free();
```

The result of this program section is a group with the specified name and the required properties. *AddGroup()* also returns a variable *pobjGroup1* as the return parameter, an interface to a group object.

With the type adaptation call *IOPCGroupStateMgt*, the interface can be obtained simply from the returned group interface *pobjGroup1*. In simplified form, this call corresponds to the COM method *QueryInterface()*.

The *SetState()* method of the *IOPCGroupStateMgt* interface is required later to activate and deactivate a group.

```
pIOPCGroupStateMgt = (IOPCGroupStateMgt)pobjGroup1;
```

Step 5: Add items

The *IOPCItemMgt* interface has the *AddItems()* method for creating OPC items:

```
((IOPCItemMgt)pobjGroup1).AddItems(2, ItemDefArray, out
                                     pResults, out pErrors);
```

The result of this program section is that the server adds two items with the properties specified in the *itemdefs* parameter. The variables of the result structure *OPCITEMRESULT* (server handle, data type of the items on the target system etc.) are also initialized.

Since the results are written to unmanaged memory by the underlying COM interface, the access from managed .NET code must make use of the marshaling functions:

```
OPCITEMRESULT result = (OPCITEMRESULT)Marshal.PtrToStructure(pos,
                                                               typeof(OPCITEMRESULT));
ItemSvrHandleArray[0] = result.hServer;
pos = new IntPtr(pos.ToInt32() +
                  Marshal.SizeOf(typeof(OPCITEMRESULT)));
OPCITEMRESULT result = (OPCITEMRESULT)Marshal.PtrToStructure
                           (pos, typeof(OPCITEMRESULT));
ItemSvrHandleArray[1] = result.hServer;
```

The client must also make sure that the unmanaged memory is released:

```
Marshal.FreeCoTaskMem(pResults);
Marshal.FreeCoTaskMem(pErrors);
```

Step 6: Request the interface *IConnectionPointContainer*

The *pIConnectionPointContainer* variable of the type *IConnectionPointContainer* can be derived from the *m_group_1* variable.

```
pIConnectionPointContainer =  
    (IConnectionPointContainer)pobjGroup1;
```

The interface is required to locate the *IConnectionPoint* interface.

Step 7: Request the *IOPCAsyncIO2* interfaces

The *pIOPCAsyncIO2* variable of the type *pIOPCAsyncIO2* can be derived from the *pobjGroup1* variable.

```
pIOPCAsyncIO2 = (IOPCAsyncIO2)pobjGroup1;
```

This interface provides methods for asynchronous reading and writing of values.

Step 8: Create callback object

To allow the OPC server to return a result of asynchronous operations, the *IOPCDataCallback* interface must be implemented on the client.

```
public class OPCAsync : System.Windows.Forms.Form ,  
    IOPCDataCallback
```

The connection of the *IConnectionPoint* interface and *IOPCDataCallback* of the OPC server is established using the *FindConnectionPoint()* method of the *IConnectionPointContainer* interface. This generates a callback object for asynchronous calls.

```
Guid iid = typeof(IOPCDataCallback).GUID;  
pIConnectionPointContainer.FindConnectionPoint(ref iid,  
                                              out pIConnectionPoint);
```

Step 9: Connect OPC server and callback object of the client

The connection between the callback object of the OPC server and the client application is established by the *Advise()* method. The return variable *dwCookie* is an ID for the connection.

```
pIConnectionPoint.Advise(this,out dwCookie);
```

Step 10: Execute required write or read operation

The *Read()* and *Write()* methods that were generated in Step 7 can be accessed over the *IOPCAsyncIO2* interface:

```
pIOPCAsyncIO2.Read(1,ItemSrvHandleArray,nTransactionID+1,  
                   out nCancelid,out pErrors);
```

Step 11: Receive notifications of the OPC server

If the data of an active item in an active group changes, the server calls the *OnDataChange()* method of the callback object. On completion of a read operation, the server calls the *OnReadComplete()* method, on completion of a write operation, the *OnWriteComplete()* method is called. The server passes the return values as parameters to the method.

```
virtual void OnDataChange(
    Int32 dwTransid,
    Int32 hGroup,
    Int32 hrMasterquality,
    Int32 hrMastererror,
    Int32 dwCount,
    int[] phClientItems,
    object[] pvValues,
    short[] pwQualities,
    FILETIME[] pftTimeStamps,
    int[] pErrors,
)
```

Step 12: Delete objects and release memory

Before exiting the program or if Stop is clicked, the generated OPC object must be deleted and the occupied memory released.

```
pIConnectionPoint.Unadvise(dwCookie);
Marshal.ReleaseComObject(pIConnectionPoint);
Marshal.ReleaseComObject(pIConnectionPointContainer);
Marshal.ReleaseComObject(piOPCAsyncIO2);
Marshal.ReleaseComObject(piOPCGroupStateMgt);
Marshal.ReleaseComObject(pobjGroup1);
Marshal.ReleaseComObject(piOPCServer);
```

Releasing memory

When using COM, it is sometimes necessary for the client to release memory that was requested by the server. The rules for this are as follows:

- [out]: The memory for a parameter with this attribute will be requested from the server.
- [in, out]: The memory for such a parameter will be requested from the client. The server can release the memory again and reallocate it.
- [in]: The client will request the memory. The server is not responsible for releasing the memory.

In all three cases, the reserved memory of unmanaged COM objects should be released by the client.

6.7 OPC XML interface in C#

This sample in the C# programming language uses the XML interface for OPC data access. It contains methods to establish a connection to a Web service and to execute read and write jobs.

Requirements for using the sample program

On the computer on which you want to run the sample program, you will first have to activate the S7 simulation module and the S7 protocol.

You will also have to set up the Web service.

6.7.1 Working with the sample program

Start program

After starting the program, all the buttons except for "Start Sample" are disabled. Click on this button to establish a connection to the OPC XML Web service. If the Web service is available, a dialog box displays information about the server. Otherwise, an error message is displayed.

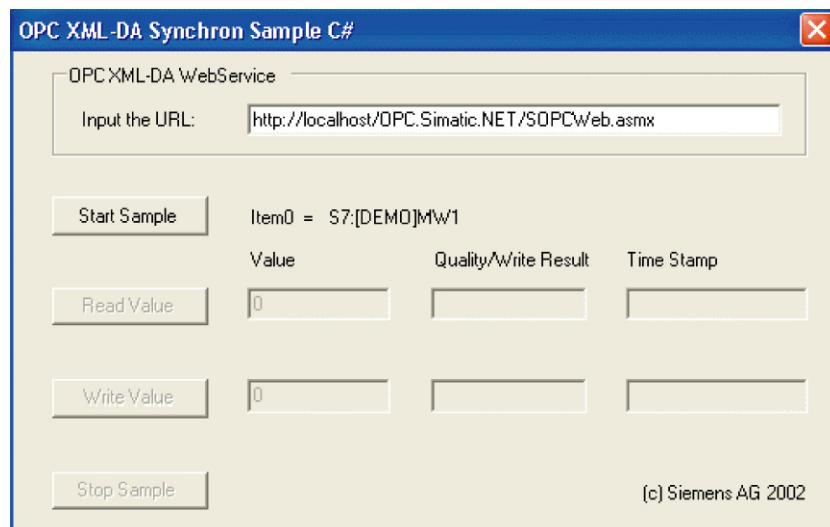


Figure 6-28 Start dialog of the sample program (C#) for OPC data access over the XML interface before clicking the "Start Sample" button

Reading values

Click the "Read Value" button to read the value of the *S7:[DEMO]MW1* item. If the read job executed successfully, the program displays the value of the item, information on the quality, and a time stamp in the text boxes beside the "Read Value" button:

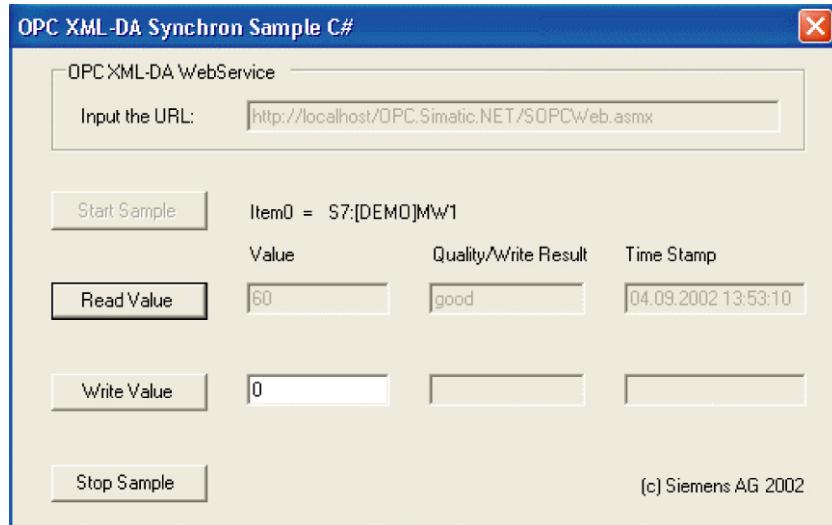


Figure 6-29 Display of the read values after clicking the "Read Value" button

Writing values

Enter the value you want to be written in the text box beside the "Write Value" button and then click the "Write Value" button. If the write job executes correctly, the program displays information on the quality of the value in the "Quality/Write Result" text box and the time stamp in the "TimeStamp" text box.

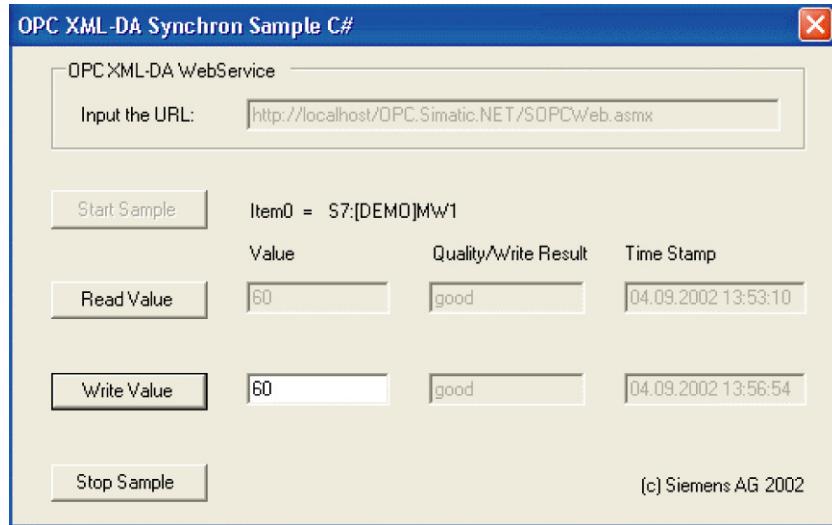


Figure 6-30 Displays of the quality of the value and time stamp of the written values after clicking the "Write Value" button

Stop the program

Close the program by clicking the "Stop Sample" button. This disables the "Read Value", "Write Value" and "Stop Sample" buttons. The "Start Sample" button and the text box for the URL are enabled again.

6.7.2 Add Web service to project

Settings in Visual Studio .NET

To allow the sample program to use the Web service of the OPC XML interface, a "Web Reference" was included in the project in Visual Studio .NET.

If you want to create programs yourself with an OPC XML interface, you will need to take the following steps:

1. Call the "Project" > "Add Web Reference" menu in "Microsoft Visual Studio".
2. Click the "Advanced" button in the "Add Web Reference" dialog.
3. Click the "Add Web Reference" button in the "Service Reference Settings" dialog

4. Enter the Web address of the SIMATIC NET OPC XML Web service in the "URL" input box. This URL is as follows:
"http://<Computer address>/<Name of the OPC SIMATIC NET Web service>/SOPCWeb.asmx?wsdl"

5. Click the "Add Reference" button.

In the "OPCXML_DataAccess Description" window, you will find the methods that are available.

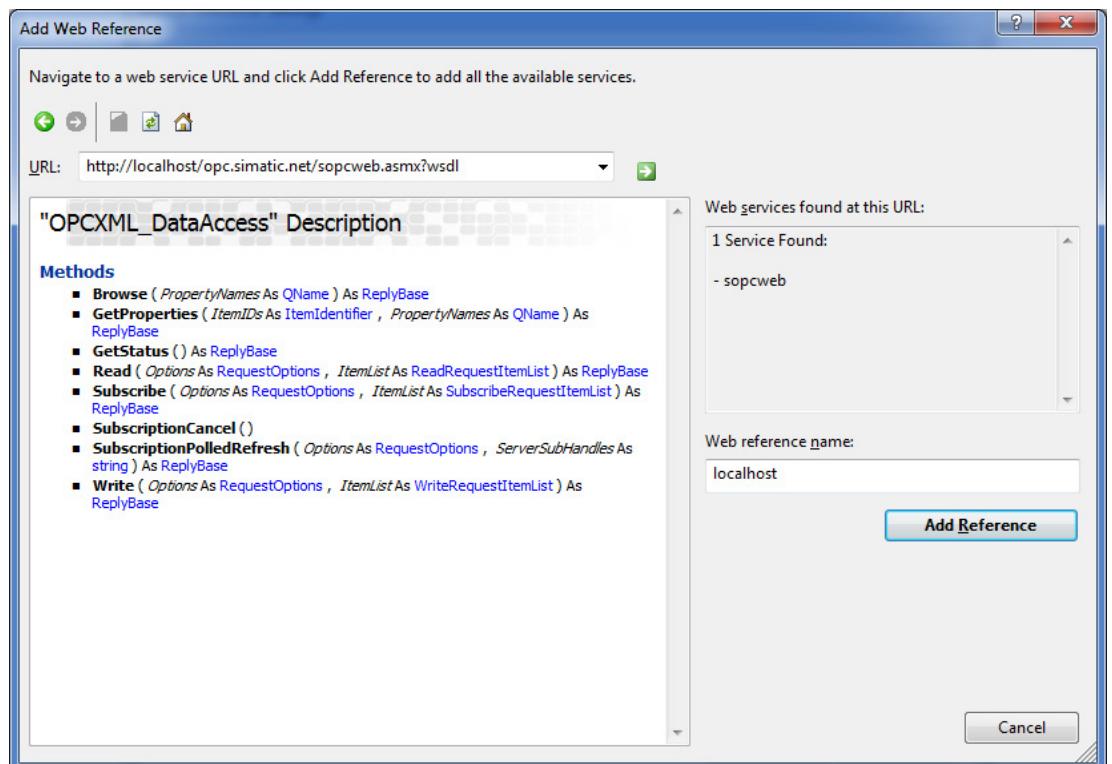


Figure 6-31 The "Add Web Reference" window with the listing of the methods defined in the WSDL

6.7.3 The MainForm class

Program code

All the methods of the sample program are defined in the *MainForm* class. At the beginning, the elements of the dialog box are declared.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Net;
namespace opcxml_da_sync
```

```
{  
    using localhost;  
    /// <summary>  
    /// Summary description for Form1.  
    /// </summary>  
    public class MainForm : System.Windows.Forms.Form  
    {  
        private System.Windows.Forms.Button Button_Start_Sample;  
        private System.Windows.Forms.Button Button_Stop_Sample;  
        private System.Windows.Forms.Button Button_Read_Value;  
        private System.Windows.Forms.Button Button_Write_Value;  
        private System.Windows.Forms.TextBox Edit_URL;  
        private System.Windows.Forms.Label Label_URL;  
        private System.Windows.Forms.Label Label_Item0;  
        private System.Windows.Forms.Label Label_Item_Value;  
        private System.Windows.Forms.Label Label_Siemens;  
        private System.Windows.Forms.Label Label_Value;  
        private System.Windows.Forms.Label Label_Quality;  
        private System.Windows.Forms.Label Label_Timestamp;  
        private System.Windows.Forms.TextBox Edit_Read_Value;  
        private System.Windows.Forms.TextBox Edit_Write_Value;  
        private System.Windows.Forms.TextBox Edit_Read_Quality;  
        private System.Windows.Forms.TextBox Edit_Write_Quality;  
        private System.Windows.Forms.TextBox Edit_Read_TimeStamp;  
        private System.Windows.Forms.TextBox Edit_Write_TimeStamp;  
        private System.Windows.Forms.GroupBox GroupBox_URL;  
        /// <summary>  
        /// Required designer variable.  
        /// </summary>  
        private System.ComponentModel.Container components = null;  
    }
```

The Web service is an instance of the OPCXML_DataAccess class:

```
// proxy class for the webservice  
private OPCXML_DataAccess m_OPCTML_DataAccess;
```

The name of the item is stored in the m_strItemName variable:

```
// ItemID used in this sample  
private string m_strItemName = "S7:[DEMO]MW1";
```

6.7.4 The constructor of MainForm and the Dispose method

Program code

The constructor of the dialog box class calls the method *InitializeComponent()*. All the elements of the dialog box are created here. The content of *InitializeComponent()* is created by Visual Studio .NET automatically when the dialog box is created with the Form Designer.

```

public MainForm()
{
    // Required for Windows Form Designer support
    //
    InitializeComponent();
    //
    // TODO: Add any constructor code after
    // InitializeComponent call
}

```

The item name appears in the dialog box because an instance of the class *System.Windows.Forms.Label* is initialized with the content of the *m_strItemName* variable:

```

    Label_Item_Value.Text = m_strItemName;
}

```

The *Dispose()* method releases all used resources at the end of the program:

```

/// <summary>
/// Clean up any resources being used.
/// </summary>

protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

```

6.7.5 Creating the dialog box elements

Program code

The following section is created automatically by Visual Studio .NET when the user creates the main dialog box of the application with the Form Designer. This contains information on all elements of the dialog box, such as size, location, and labeling of buttons. This section also specifies which events procedures are executed when a button is clicked.

```

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.Button_Start_Sample = new System.Windows.Forms.Button();
    this.Button_Stop_Sample = new System.Windows.Forms.Button();
    this.Button_Read_Value = new System.Windows.Forms.Button();
    this.Button_Write_Value = new System.Windows.Forms.Button();
}

```

```
Button();
this.Edit_URL = new System.Windows.Forms.TextBox();
this.Label_URL = new System.Windows.Forms.Label();
this.Label_Item0 = new System.Windows.Forms.Label();
this.Label_Item_Value = new System.Windows.Forms.Label();
this.Label_Siemens = new System.Windows.Forms.Label();
this.Label_Value = new System.Windows.Forms.Label();
this.Label_Quality = new System.Windows.Forms.Label();
this.Label_Timestamp = new System.Windows.Forms.Label();
this.Edit_Read_Value = new System.Windows.Forms.TextBox();
this.Edit_Write_Value = new System.Windows.Forms.TextBox();
this.Edit_Read_Quality = new System.Windows.Forms.TextBox();
this.Edit_Write_Quality = new System.Windows.Forms.TextBox();
this.Edit_Read_TimeStamp = new System.Windows.Forms.TextBox();
this.GroupBox_URL = new System.Windows.Forms.GroupBox();
this.Edit_Write_TimeStamp = new System.Windows.Forms.TextBox();
this.SuspendLayout();
//
// Button_Start_Sample
//
this.Button_Start_Sample.Location = new System.Drawing.
    Point(24, 88);
this.Button_Start_Sample.Name = "Button_Start_Sample";
this.Button_Start_Sample.Size =
    new System.Drawing. Size(96, 24);
this.Button_Start_Sample.TabIndex = 0;
this.Button_Start_Sample.Text = "Start Sample";
this.Button_Start_Sample.Click += new System.EventHandler(
    this.Button_Start_Sample_Click);
//
// Button_Stop_Sample
//
this.Button_Stop_Sample.Enabled = false;
this.Button_Stop_Sample.Location = new System.Drawing.
    Point(24, 256);
this.Button_Stop_Sample.Name = "Button_Stop_Sample";
this.Button_Stop_Sample.Size = new System.Drawing. Size(96, 23);
this.Button_Stop_Sample.TabIndex = 1;
this.Button_Stop_Sample.Text = "Stop Sample";
this.Button_Stop_Sample.Click += new System.EventHandler(
    this.Button_Stop_Sample_Click);
//
// Button_Read_Value
//
this.Button_Read_Value.Enabled = false;
this.Button_Read_Value.Location = new System.Drawing.
    Point(24, 144);
this.Button_Read_Value.Name = "Button_Read_Value";
this.Button_Read_Value.Size = new System.Drawing. Size(96, 23);
this.Button_Read_Value.TabIndex = 2;
this.Button_Read_Value.Text = "Read Value";
this.Button_Read_Value.Click += new System.EventHandler(
```

```
this.Button_Read_Value_Click);
//
// Button_Write_Value
//
this.Button_Write_Value.Enabled = false;
this.Button_Write_Value.Location = new System.Drawing.
    Point(24, 200);
this.Button_Write_Value.Name = "Button_Write_Value";
this.Button_Write_Value.Size = new System.Drawing. Size(96, 23);
this.Button_Write_Value.TabIndex = 3;
this.Button_Write_Value.Text = "Write Value";
this.Button_Write_Value.Click += new System.EventHandler(
this.Button_Write_Value_Click);
//
// Edit_URL
//
this.Edit_URL.Location = new System.Drawing.Point(144, 32);
this.Edit_URL.Name = "Edit_URL";
this.Edit_URL.Size = new System.Drawing.Size(328, 20);
this.Edit_URL.TabIndex = 4;
this.Edit_URL.Text =
"http://localhost/OPC.Simatic.NET/SOPCWeb.asmx";
//
// Label_URL
//
this.Label_URL.Location = new System.Drawing.Point(40, 35);
this.Label_URL.Name = "Label_URL";
this.Label_URL.Size = new System.Drawing.Size(96, 24);
this.Label_URL.TabIndex = 5;
this.Label_URL.Text = "Input the URL:";
//
// Label_Item0
//
this.Label_Item0.Location = new System.Drawing. Point(144, 94);
this.Label_Item0.Name = "Label_Item0";
this.Label_Item0.Size = new System.Drawing.Size(48, 23);
this.Label_Item0.TabIndex = 6;
this.Label_Item0.Text = "Item0 = ";
//
// Label_Item_Value
//
this.Label_Item_Value.Location = new System.Drawing.
    Point(192, 94);
this.Label_Item_Value.Name = "Label_Item_Value";
this.Label_Item_Value.Size = new System.Drawing. Size(296, 23);
this.Label_Item_Value.TabIndex = 7;
//
// Label_Siemens
//
this.Label_Siemens.Location = new System.Drawing.
    Point(384, 262);
this.Label_Siemens.Name = "Label_Siemens";
```

```
this.Label_Siemens.Size = new System.Drawing.Size(120, 16);
this.Label_Siemens.TabIndex = 8;
this.Label_Siemens.Text = "(c) Siemens AG 2003";
//
// Label_Value
//
this.Label_Value.Location = new System.Drawing. Point(144, 120);
this.Label_Value.Name = "Label_Value";
this.Label_Value.Size = new System.Drawing.Size(40, 16);
this.Label_Value.TabIndex = 9;
this.Label_Value.Text = "Value";
//
// Label_Quality
//
this.Label_Quality.Location = new System.Drawing.
                               Point(256, 120);
this.Label_Quality.Name = "Label_Quality";
this.Label_Quality.Size = new System.Drawing.Size(104, 23);
this.Label_Quality.TabIndex = 10;
this.Label_Quality.Text = "Quality/Write Result";
//
// Label_Timestamp
//
this.Label_Timestamp.Location = new System.Drawing.
                               Point(374, 120);
this.Label_Timestamp.Name = "Label_Timestamp";
this.Label_Timestamp.TabIndex = 11;
this.Label_Timestamp.Text = "Time Stamp";
//
// Edit_Read_Value
//
this.Edit_Read_Value.Enabled = false;
this.Edit_Read_Value.Location = new System.Drawing.
                               Point(144, 144);
this.Edit_Read_Value.Name = "Edit_Read_Value";
this.Edit_Read_Value.Size = new System.Drawing. Size(88, 20);
this.Edit_Read_Value.TabIndex = 12;
this.Edit_Read_Value.Text = "0";
//
// Edit_Write_Value
//
this.Edit_Write_Value.Enabled = false;
this.Edit_Write_Value.Location = new System.Drawing.
                               Point(144, 200);
this.Edit_Write_Value.Name = "Edit_Write_Value";
this.Edit_Write_Value.Size = new System.Drawing. Size(88, 20);
this.Edit_Write_Value.TabIndex = 13;
this.Edit_Write_Value.Text = "0";
//
// Edit_Read_Quality
//
this.Edit_Read_Quality.Enabled = false;
```

```
this.Edit_Read_Quality.Location = new System.Drawing.
    Point(256, 144);
this.Edit_Read_Quality.Name = "Edit_Read_Quality";
this.Edit_Read_Quality.Size = new System.Drawing. Size(96, 20);
this.Edit_Read_Quality.TabIndex = 14;
this.Edit_Read_Quality.Text = "";
// 
// Edit_Write_Quality
//
this.Edit_Write_Quality.Enabled = false;
this.Edit_Write_Quality.Location = new System.Drawing.
    Point(256, 200);
this.Edit_Write_Quality.Name = "Edit_Write_Quality";
this.Edit_Write_Quality.Size = new System.Drawing. Size(96, 20);
this.Edit_Write_Quality.TabIndex = 15;
this.Edit_Write_Quality.Text = "";
// 
// Edit_Read_TimeStamp
//
this.Edit_Read_TimeStamp.Enabled = false;
this.Edit_Read_TimeStamp.Location = new System.Drawing.
    Point(376, 144);
this.Edit_Read_TimeStamp.Name = "Edit_Read_TimeStamp";
this.Edit_Read_TimeStamp.Size = new System.Drawing.
    Size(112, 20);
this.Edit_Read_TimeStamp.TabIndex = 16;
this.Edit_Read_TimeStamp.Text = "";
// 
// GroupBox_URL
//
this.GroupBox_URL.Location = new System.Drawing. Point(24, 8);
this.GroupBox_URL.Name = "GroupBox_URL";
this.GroupBox_URL.Size = new System.Drawing.Size(464, 56);
this.GroupBox_URL.TabIndex = 17;
this.GroupBox_URL.TabStop = false;
this.GroupBox_URL.Text = "OPC XML-DA WebService";
// 
// Edit_Write_TimeStamp
//
this.Edit_Write_TimeStamp.Enabled = false;
this.Edit_Write_TimeStamp.Location = new System.Drawing.
    Point(376, 200);
this.Edit_Write_TimeStamp.Name = "Edit_Write_TimeStamp";
this.Edit_Write_TimeStamp.Size = new System.
    Drawing.Size(112, 20);
this.Edit_Write_TimeStamp.TabIndex = 18;
this.Edit_Write_TimeStamp.Text = "";
// 
// MainForm
//
this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(506, 293);
```

```
        this.Controls.Add(this.Edit_Write_TimeStamp);
        this.Controls.Add(this.Edit_Read_TimeStamp);
        this.Controls.Add(this.Edit_Write_Quality);
        this.Controls.Add(this.Edit_Read_Quality);
        this.Controls.Add(this.Edit_Write_Value);
        this.Controls.Add(this.Edit_Read_Value);
        this.Controls.Add(this.Edit_URL);
        this.Controls.Add(this.Label_Timestamp);
        this.Controls.Add(this.Label_Quality);
        this.Controls.Add(this.Label_Value);
        this.Controls.Add(this.Label_Siemens);
        this.Controls.Add(this.Label_Item_Value);
        this.Controls.Add(this.Label_Item0);
        this.Controls.Add(this.Label_URL);
        this.Controls.Add(this.Button_Write_Value);
        this.Controls.Add(this.Button_Read_Value);
        this.Controls.Add(this.Button_Stop_Sample);
        this.Controls.Add(this.Button_Start_Sample);
        this.Controls.Add(this.GroupBox_URL);
        this.FormBorderStyle =
            System.Windows.Forms.FormBorderStyle.FixedDialog;
        this.MaximizeBox = false;
        this.MinimizeBox = false;
        this.Name = " MainForm";
        this.Text = "OPC XML-DA Synchron Sample C#";
        this.ResumeLayout(false);
    }
#endregion
```

6.7.6 The Main method

Program code

The Main method is the starting point for the sample program. An instance of the dialog box classes generated and started by calling Application.Run.

```
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
static void Main()
{
    Application.Run(new MainForm());
}
```

6.7.7 The Button_Start_Sample_Click method

Program code

This method is executed when you click the "Start Sample" button. The OPC XML Web service is represented by an instance of the OPCXML_DataAccess class.

```
/*-----
 | Name:  Button_Start_Sample_Click
 | Desc:  Handler is called, when button "Start Sample"
 |        is pressed
 | Notes: Create an instance of the proxy class of the
 |        webservice
-----*/
private void Button_Start_Sample_Click(object sender, System.EventArgs e)
{
    try
    {
        if (m_OPCTML_DataAcess == null)
        {
            m_OPCTML_DataAcess = new OPCTML_DataAccess ();
            m_OPCTML_DataAcess.Timeout = 10000; //Timeout 10 sec.
        }
    }
}
```

The program checks the connection to the Web service by calling methods of the *WebRequest* class. If the Web service is not available, an exception is triggered.

```
// Checking the connection to the WebService
// The WebRequest class throws a WebException when
// errors occur while accessing an Internet resource

WebRequest myRequest = WebRequest.Create(Edit_URL.Text);
WebResponse myResponse = myRequest.GetResponse();
myResponse.Close();
```

Using the *GetStatus* method, the program queries the server status. This is the first time that a method of the XML interface is used.

```
// Assigning the url to the proxy class for the
// web service
m_OPCTML_DataAcess.Url = Edit_URL.Text;

//Checking the webservice status
ServerStatus Status;
ReplyBase replay = m_OPCTML_DataAcess.GetStatus ("en", "1", out
Status);
```

If the server is running correctly, the program displays a dialog box with a variety of information.

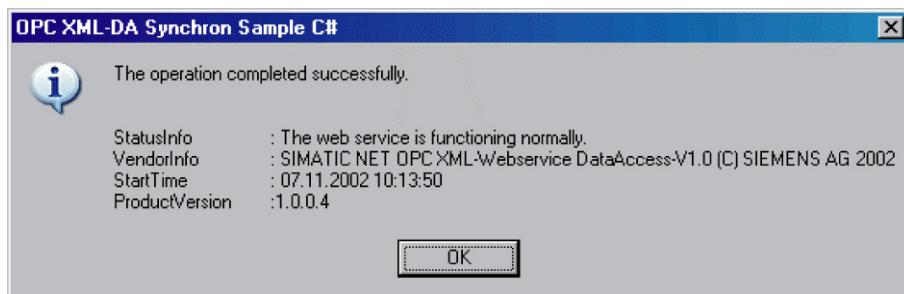


Figure 6-32 Information dialog after successful startup of a Web service

```
if (replay.ServerState == serverState.running)
{
    string strText =
        ""The operation completed successfully.\n\n";
    if (Status.StatusInfo != null)
        { strText+= "\nStatusInfo\t: " + Status.StatusInfo;
        }
    if (Status.VendorInfo != null)
        { strText+= "\nVendorInfo\t: " + Status.VendorInfo;
        }
    strText+= "\nStartTime\t\t: " +
    Status.StartTime.ToString();
    if (Status.ProductVersion != null)
        { strText+= "\nProductVersion\t:"+
        + Status.ProductVersion;
        }
    MessageBox.Show (this, strText, this.Text,
    MessageBoxButtons.OK, MessageBoxIcon.Information);
```

The text box for entering the URL and the "Start Sample" button are disabled. All remaining buttons are enabled.

```
Edit_URL.Enabled = false;
Button_Start_Sample.Enabled = false;
Edit_Write_Value.Enabled = true;
Button_Read_Value.Enabled = true;
Button_Write_Value.Enabled = true;
Button_Stop_Sample.Enabled = true;
}
```

If an error has occurred in the Web service, this is displayed in a dialog box:

```
else
{
    string strText =
        "The operation is not completed successfully.\n\n";
    MessageBox.Show (this, strText, this.Text,
    MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
}
}
```

If an error has occurred during execution of the program, this is handled within a catch block.

```
    catch (Exception excep)
    {
        MessageBox.Show (this, excep.Message, this.Text,
                        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}
```

6.7.8 The Button_Stop_Sample_Click method

Program code

This method is executed when you click the "Stop Sample" button. Only the appearance of the dialog box changes, the text box for the URL can be edited again and the "Start Sample" button can be clicked. All other buttons are disabled.

```
/*-----
| Name: Button_Stop_Sample_Click
| Desc: Handler is being called, when button "Stop Sample"
|       has been pressed
-----*/
private void Button_Stop_Sample_Click(object sender, System.EventArgs e)
{
    Edit_URL.Enabled      = true;
    Button_Start_Sample.Enabled = true;
    Edit_Write_Value.Enabled = false;
    Button_Read_Value.Enabled = false;
    Button_Write_Value.Enabled = false;
    Button_Stop_Sample.Enabled = false;
}
```

6.7.9 The Button_Read_Value_Click method

Program code

This method is executed when you click the "Read Value" button. The program first generates all the objects necessary for the read job, an array of the type *ReadRequestItemLists* and an array of the type *ReadRequestItem* each with one element. In addition to this, an instance of the *RequestOptions* class and the *OPCError* class are required as return parameters.

```
/*-----
| Name: Button_Read_Value_Click
| Desc: Handler is called, when button "Read Value"
|       is pressed
| Notes: initiates a sync read request
-----*/
private void Button_Read_Value_Click(object sender,
```

```

System.EventArgs e)
{
    try
    {
        Edit_Read_Value.Text = "0";
        Edit_Read_Quality.Text = "";
        Edit_Read_TimeStamp.Text = "";
        /// <summary>
        /// Make a new ItemList for a ReadRequest
        /// </summary>
        ReadRequestItemList ItemLists = new ReadRequestItemList();
        ItemLists.Items = new ReadRequestItem[1];
        ItemLists.Items[0] = new ReadRequestItem();
        ItemLists.Items[0].ItemPath = "";
        ItemLists.Items[0].ItemName = m_strItemName;
        RequestOptions opt = new RequestOptions();
        ReplyItemList ItemValues;
        OPCError[] Errors;
    }
}
```

The *Read* method of the XML interface is called with previously defined parameters. If a value could be read, the dialog box displays this in the "Value" text box. Otherwise "0" is displayed.

```

m_OPXML_DataAccess.Read (opt, ItemLists, out ItemValues,
                         out Errors);
/// <summary>
/// Assign the returned values to the TextBoxes
/// </summary>
if(ItemValues.Items[0].Value != null)
{
    Edit_Read_Value.Text = ItemValues.Items[0].Value.ToString();
}

else
{
    Edit_Read_Value.Text = "0";
}
```

The values for the time stamp and the quality of the values are also displayed in the appropriate text boxes.

```

if(ItemValues.Items[0].TimestampSpecified)
{
    Edit_Read_TimeStamp.Text =
        ItemValues.Items[0].Timestamp.ToString();
}
else
{
    Edit_Read_TimeStamp.Text = "";
}

if(ItemValues.Items[0].Quality!=null)
{
    Edit_Read_Quality.Text =
        ItemValues.Items[0].Quality.QualityField.ToString();
}
```

```

        else
        {
            Edit_Read_Quality.Text = "";
        }
        /// <summary>
        /// Show Errors in a Message Box
        /// </summary>
        if(Errors.Length>0)
        {
            MessageBox.Show (this,
                Errors[0].Text,
                this.Text,
                MessageBoxButtons.OK,
                MessageBoxIcon.Exclamation);
        }
    }
}

```

Program errors are handled in a catch block.

```

    catch(Exception excep)
    {
        MessageBox.Show (this, excep.Message, this.Text,
            MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}

```

6.7.10 The Button_Write_Value_Click method

Program code

This method is executed when you click the "Write Value" button. The program first generates all the objects necessary for the write job, an array of the type *WriteRequestItemList* and an array of the type *ItemValue* each with one element. The read value is returned in a parameter of the type *ReplyItemList*. In addition to this, an instance of the *RequestOptions* class and the *OPCError* class are required as return parameters.

```

/*
| Name: Button_Write_Value_Click
| Desc: Handler is called, when button "Write Value"
|       is pressed
| Notes: initiates a sync read request
-----
private void Button_Write_Value_Click(object sender,
                                      System.EventArgs e)
{
    try
    {
        Edit_Write_Quality.Text = "";
        /// <summary>
        /// Make a new ItemList for a WriteRequest
        /// </summary>
        WriteRequestItemList ItemLists =
            new WriteRequestItemList();
        ItemLists.Items = new ItemValue[1];
        ItemLists.Items[0] = new ItemValue();
        ItemLists.Items[0].ItemPath = "";
    }
}

```

```
ItemLists.Items[0].ItemName = m_strItemName;
ItemLists.Items[0].Value = System.Convert.ToInt32(Edit_Write_
    Value.Text);
ItemLists.Items[0].TimestampSpecified = false;
RequestOptions opt = new RequestOptions();
ReplyItemList ItemValues;
OPCError[] Errors;
```

The *Write* method of the XML interface is called with previously defined parameters. If the value could be written, the dialog box displays this in the "Value" text box. Otherwise "0" is displayed.

```
ReplyBase replay = m_OPCTXML_DataAccess.Write(
    opt,
    ItemLists,
    true,
    out ItemValues,
    out s);

/// <summary>
/// Assign the returned values to the TextBoxes
/// </summary>
if (ItemValues.Items[0].Value != null)
{   Edit_Write_Value.Text =
        ItemValues.Items[0].Value.ToString();
}
else
{   Edit_Write_Value.Text = "0";
}
```

The returned values for the time stamp and the quality of the values are also displayed in the appropriate text boxes.

```
if (ItemValues.Items[0].TimestampSpecified)
{   Edit_Write_TimeStamp.Text =
        ItemValues.Items[0].Timestamp.ToString();
}
else
{   Edit_Write_TimeStamp.Text = "";
}
if (ItemValues.Items[0].Quality!=null)
{   Edit_Write_Quality.Text =
        ItemValues.Items[0].Quality.QualityField.ToString();
}
else
{   Edit_Write_Quality.Text = "";
}
/// <summary>
/// Show Errors in a Message Box
/// </summary>
if (Errors.Length>0)
{   MessageBox.Show (this,
    Errors[0].Text,
```

```

        this.Text,
        MessageBoxButtons.OK,
        MessageBoxIcon.Exclamation);
    }
}

```

Program errors are handled in a catch block.

```

catch (Exception excep)
{
    MessageBox.Show (this, excep.Message, this.Text,
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Exclamation);
}
}

```

6.8 OPC Alarms & Events Custom interface in C++

Introduction

This section introduces the basic steps that a client must perform to be able to process events.

Step 1: Initialize the COM library

Every program that wants to use the COM library must first initialize it. This is done using the *CoInitializeEx* function that has an additional parameter compared with *CoInitialize* for the required thread model. This allows you to decide whether the COM object is created in the single-thread mode or in the multi-thread mode.

```
HRESULT hr = CoInitializeEx(NULL, COINIT_MULTITHREADED);
```

Step 2: Convert the ProgID to a CLSID

To identify it, each COM server has a *ProgID* that is assigned to a worldwide unique *CLSID*. This is obtained with the *CLSIDFromProgID()* function. The ProgID of the OPC Alarms & Events server of SIMATIC NET is L"OPC.SimaticNETAlarms":

```
hr = CLSIDFromProgID(L"OPC.SimaticNETAlarms",
                      &clsidOPCEventServer);
```

Step 3: Create a server object

The *CoCreateInstance()* function creates an instance of the class whose *CLSID* was specified.

```
hr = CoCreateInstance (clsidOPCEventServer,
                      NULL,
                      CLSCTX_LOCAL_SERVER,
```

```
IID_IOPCEventServer,  
(void**)&gpIOPCEventServer);
```

The result of this program section is an object of the OPC server class. *CoCreateInstance* also provides a pointer to the IOPCServer interface of the server object (*gpIOPCEventServer* parameter):

Step 4: Register with the event server

The client must be registered with the server to obtain notification of events. The *CreateEventSubscription* method of the OPCEventServer class as an object of the *OPCEventSubscription* class to ES server and returns a pointer to the *IOPCEventSubscriptionMgt* interface (*pgIOPCEventSubscrMgt* parameter).

```
hr = gpIOPCEventServer->CreateEventSubscription  
    (TRUE,  
     dwBufferTime,  
     dwMaxSize,  
     hClientSubscription,  
     IID_IOPCEventSubscriptionMgt,  
     (LPUNKNOWN*) &pgIOPCEventSubscrMgt,  
     &dwRevisedBufferTime,  
     &dwRevisedMaxSize );
```

Step 5: Create callback object

To allow the event server to send a notification to the client when events occurred, the IOPCEventCallback interface must be implemented on the client. In the sample program there is an object of the *COPCEventCallback* class for this purpose that once derived from *CObjectRoot* and *IOPCEventSink*.

```
CComObject<COPCEventCallback>::CreateInstance  
(&pgCOPCEventCallback);
```

The *pgCOPCEventCallback* parameter is a pointer to the callback object.

Step 6: Connect the OPC event server and callback object

The *AtAdvise()* method creates a connection between the OPC server and the callback object. The first parameter is a pointer to the IUnknown interface of the object to which the client wants to connect. The second parameter is a pointer to the IUnknown interface of the callback object:

```
hr = AtlAdvise(pgiOPCEventSubscrMgt,  
               pgCOPCEventCallback->GetUnknown(),  
               IID_IOPCEventSink,  
               &dwAdviseEvent);
```

Step 7: Receive notification from the Alarms & Events server

If an alarm occurs, the server calls the *OnEvent* method of the callback object. In the sample program, *OnEvent* is implemented so that the *displayEvent* method of the *CReceiveAnAlarm* dialog box class displays detailed information on the event.

```
STDMETHODIMP COPCEventCallback::OnEvent(
    OPCHANDLE hClientSubscription,
    BOOL bRefresh,
    BOOL bLastRefresh,
    DWORD dwCount,
    ONEVENTSTRUCT __RPC_FAR *pEvents)
```

The *pEvents* parameter is a pointer to a structure in which the information about the events is stored.

Step 8: Acknowledge received event

The client acknowledges the notification of the server with the *AckCondition* method. The *pszSource* parameter contains the source of the event and *pszConditionName* the condition whose status change must be acknowledged. Both parameters are array addresses. In the example, only one event is acknowledged. Calling *AckCondition* can, however, and acknowledge several events.

The Alarms & Events server provides the values for *pszSource* and *pszConditionName* when it calls *OnEvent* as the components *szSource* and *szConditionName* of the *ONEVENTSTRUCT* structure.

```
hr = gpIOPCEventServer->AckCondition(1,
                                         L"Me",
                                         L"NoComment",
                                         &pszSource,
                                         &pszConditionName,
                                         &ftActiveTime,
                                         &dwCookie,
                                         &pErrors);
```

Step 9: Delete objects and release memory

Before exiting the program, the OPC objects that have been created must be deleted and the memory reserved for them must be released. *AtlUnadvise* terminates the connection between the OPC server and the callback object.

```
hr = AtlUnadvise(pgIOPCEventSubscrMgt,
                  IID_IOPCEventSink,
                  dwAdviseEvent);
```

Each COM interface has the *Release* method with which the reference counter of the interface is decremented. When the interface is no longer referenced, the occupied resources are released.

```
hr=pgIOPCEventSubscrMgt->Release();
```

The *CoUninitialize* method closes the COM library for the relevant thread and releases the resources it was using. For each successful *CoInitializeEx* call, there must be a corresponding *CoUninitialize* call.

```
CoUninitialize();
```

6.9 OPC UA interface in C

These examples use the OPC UA interface in simple C.

You will find these in:

- "<installationpath>\SIEMENS\SIMATIC.NET\opc2\samples\ua\c\read.c\"
- "<installationpath>\SIEMENS\SIMATIC.NET\opc2\samples\ua\c\alarm.c\"
- "<installationpath>\SIEMENS\SIMATIC.NET\opc2\samples\ua\c\publish.c\"

The "Read.exe" program displays the function "Synchronous reading". Only unencrypted communication is used.

The "Alarm.exe" program shows the function "Asynchronous monitoring of data variables and S7 alarms". Encrypted communication is used.

The "Publish.exe" program shows the functions "Reading, writing and asynchronous monitoring of data variables and S7 alarms". Both unencrypted and encrypted communication can be used. User authentication is also possible.

The remainder of the description refers to the more extensive example "Publish" that also includes the functionality of the simpler examples "Read" and "Alarm".

6.9.1

Activating the simulation connection

To make the program functional, you need to activate a simulation connection that makes the demonstration variable used in the program available. Follow these steps:

1. Start the "Communication Settings" program from the Start menu.
"Start" > "Siemens Automation" > "SIMATIC" > "SIMATIC NET" > "Communication Settings"
2. In the left navigation window, open and the properties page for the OPC protocol selection by opening the levels "SIMATIC NET Configuration" > "OPC Protocol Selection".

3. Enable the check box for the protocol to be simulated.

This example uses the S7 protocol.

You therefore activate the DEMO connections for the S7 protocol.

4. Close the "Communication Settings" program.

Note

For the changes to take effect, all OPC clients must first be shut down and the OPC server restarted!

6.9.2 Importing the client certificates

To be able to establish a secure connection to the server, the server must accept the client certificate transferred to it during the exchange of certificates. To ensure this, the certificate is imported into the certificate store of the server.

Follow these steps to import a client certificate:

1. Start the "Communication Settings" configuration program from the start menu.
"Start" > "SIMATIC" > "SIMATIC NET" > "Settings" > "Communication Settings"
2. In the left navigation window, open the properties page for the certificates by opening the levels "SIMATIC NET Configuration" > "Applications" > "OPC Settings" and clicking on the "OPC UA Certificates" icon.
3. In the shortcut menu, select the menu command "Import Client Certificate ...".
This is on your hard disk under
<installationpath>\SIEMENS\SIMATIC.NET\opc2\samples\ua\c\pki\ca\certs\Opc.Publish.C.Sample.der and then click "Open".
4. Close the "Communication Settings" program.

6.9.3 Working with the sample program

The program contains two menus. The first is the main menu that appears when you start the program. If a connection is established to the server, the connection menu appears. The following actions are triggered by the menus:

Menu command	Action
mainmenu	
c (connect)	Establish an unsecured connection to the server.
s (secure connect)	Establish a secure connection to the server.
q (quit)	Exit the program.
connectionmenu	
r (read)	Read the value of a variable.
w (write)	Write a value to a variable.
m (monitor)	Monitor a variable for value changes and alarms.

Menu command	Action
d (disconnect)	Terminate the connection to the server.
q (quit)	Terminate the connection to the server and exit the program.

6.9.4 Start program

The program is located on your hard disk in:

"<installationpath>\SIEMENS\SIMATIC.NET\opc2\samples\ua\c\publish.c\"

When the program starts, the main menu appears. By pressing the "c" key on the keyboard, an unsecured connection is established to the server. If instead, you press "s", there is an exchange of certificates between the client and server during which both ends must confirm the other certificate. Following this, a secure connection is established to the server.

After connection establishment, the connection menu is displayed that provides the options for reading, writing and monitoring variables.

6.9.5 Reading and writing values

After selecting the "read" action by pressing the "r" key on the keyboard, the value of the variable is read and displayed on the console. The connection menu then appears.

If you press the "w" key on the keyboard and then enter an integer between 0 and 255, this is written to the variable. Once again, the connection menu is displayed on completion of the action.

As it stands, the sample program is designed to operate with a demonstration connection. If you want to run the sample program in a real environment, you will need to adapt the code to the real variables (see section "Notes on converting to real variables (Page 726)").

6.9.6 Monitoring variables

If you press the "m" key on the keyboard, value changes and alarms of a variable are output continuously on the console. Monitoring is stopped with the "q" key and the connection menu appears again.

As it stands, the sample program is designed to operate with a demonstration connection. To view value changes on the console, these therefore need to be generated manually or using a generation mode.

Follow these steps:

1. Start the OPC Scout V10 program from the start menu.
."Start" > "Siemens Automation" > "SIMATIC" > "SIMATIC NET" > "OPC Scout V10"
2. In the left navigation window, open the levels.
"UA Server" > "opc.tcp://<hostname>:55101" > "Objects" > "Server" >
"S7:" > "DEMO" > "blocks" > "db" > "db20"
3. Drag the variable "db20.0,b" to the DA view 1.

4. Enter the new value in the "New value" box to change the value manually and then click the "Write" button. When you click the "Reading or Monitoring ON" button, the current value of the variable appears in the "Value" box.
5. Enter the required generation in the "Generation Mode" box to generate values for setting the variable, for example [0...255]+1 to continuously increment the values from 0 to 255.
6. Activate generation of the value with the "Generate Values ON" button.

Note

If you want to run the sample program in a real environment, you will need to adapt the code to the real variables (see section "Notes on converting to real variables (Page 726)").

6.9.7 Stop the program

The sample program is closed by pressing the "q" key in the main or connection menu. If the program is in the connection menu, the connection to the server is first terminated.

6.9.8 Description of the program sequence

6.9.8.1 Connection establishment

There are two ways in which a connection can be established to the server. On the one hand by establishing a secure connection and exchanging certificates and a secure endpoint, and on the other, via an unsecured endpoint.

The steps for a secure connection establishment

1. Create a channel.
To establish a connection to the server, a channel must first be created. To do this, you use the "OpcUa_Channel_Create()" function.
2. Open a channel to an endpoint with the security mode "None".
To establish a connection to an endpoint, use the function "OpcUa_Channel_Connect()". This transfers the security requirement of the endpoint, in this case "None". This specifies that no security functions are required of the server. As a result, this connection is not secure.
3. Read out the endpoints of the server.
The "OpcUa_ClientApi_GetEndpoints()" function informs the client of the endpoints of the server. The server certificate is also transferred to the client that is required for connection establishment with an endpoint with the security mode "Sign&Encrypt".
4. Accept the server certificate.
The server certificate must be accepted by the client. Following this, the "ValidateCertificate()" function checks whether the certificate is already located in the

certificate store of the client. If this is not the case, it is copied to the client store with the "SaveCertificate()" function.

5. Close the unsecured channel.

To close a connection to an endpoint, use the "OpcUa_ClientApi_Disconnect()" function.

6. Open a secure channel.

As in step 2, the "OpcUa_Channel_Connect()" function is used again to establish a connection to an endpoint. The endpoint used here has the security mode "Sign&Encrypt" which means that the messages between the client and server need to be encrypted and signed.

7. Create a session.

To create a session, the "OpcUa_ClientApi_CreateSession()" function is used. The reply of the server consists of a unique number known as the "ServerNonce" that is necessary to activate the session in step 8.

8. Activate the session.

To activate the session created in step 7, the "OpcUa_ClientApi_ActivateSession()" function is used. This transfers a signature made up of the "ServerNonce" received in step 7 and the server certificate.

The steps for an unsecured connection establishment

1. Create a channel.

To establish a connection to the server, a channel must first be created. To do this, you use the "OpcUa_Channel_Create()" function.

2. Open a channel to an endpoint with the security mode "None".

To establish a connection to an endpoint, use the function "OpcUa_Channel_Connect()". This transfers the security requirement of the endpoint, in this case "None". This specifies that no security functions are required of the server. As a result, this connection is not secure.

3. Create a session.

To create a session, the "OpcUa_ClientApi_CreateSession()" function is used. The reply of the server consists of a unique number known as the "ServerNonce" that is necessary to activate the session in step 8.

4. Activate the session.

To activate the session created in step 7, the "OpcUa_ClientApi_ActivateSession()" function is used. This transfers a signature made up of the "ServerNonce" received in step 7 and the server certificate.

6.9.8.2 **Reading and writing variables**

Reading the value of the variable

To read the value of a variable, the "OpcUa_ClientApi_Read()" function is used. This returns not only the value but also a status code specifying whether or not the function could be completed successfully.

Writing a value to a variable

To write a value to a variable, the "OpcUa_ClientApi_Write()" function is used. The output is a status code that specifies whether or not the function was successful.

6.9.8.3 Monitoring variables and alarms

Follow the steps below to monitor variables and alarms:

1. Create a subscription.
To create a subscription, use the "OpcUa_CreateSubscription()" function. As the transfer value, the function transfers the publish interval among other things.
2. Create the monitored items for monitoring values and alarms.
The monitored items for monitoring data changes and alarms are created with the "OpcUa_ClientApi_CreateMonitoredItems()" function. For the monitored item for monitoring alarms, you need to set an event filter.
3. Send a publish request to the server.
To receive alarms or data changes from the server, the clients needs to call the "OpcUa_ClientApi_BeginPublish()" function. This transfers the callback function which is called for the server to send its response.
4. Evaluate the response to the publish request.
The "pfnUaServer_ClientChannelRequestComplete()" callback function receives the server's response to the publish request. Here, the alarms contained are sorted and the relevant screen display generated. Following this, a further publish request is sent to the server.
5. Delete the monitored items.
The monitored items are deleted with the "OpcUa_ClientApi_DeleteMonitoredItems()" function.
6. Delete the subscription.
The "OpcUa_ClientApi_DeleteSubscription()" function deletes the subscription. Here, the callback function can be called with an error if a pending publish response appears only after the subscription has been deleted.

6.9.8.4 Connection termination

Follow the steps below to terminate the connection:

1. Close the session.
The session is closed with the "OpcUa_ClientApi_CloseSession()" function.
2. Close the channel.
The "OpcUa_Channel_Close()" function closes the channel.
3. Delete the channel.
The channel is deleted with the "OpcUa_Channel_Delete()" function.

6.9.9 Notes on converting to real variables

The following lines must be adapted to the variables to convert the sample program to a real variable:

- g_Nodeld_Variable.NamespaceIndex = NAMESPACE_S7;
- g_Nodeld_Variable.IdentifierType = OpcUa_IdentifierType_String;
- g_Nodeld_Variable.Identifier.String.strContent = NODE_IDENTIFIER_STRING;
- g_Nodeld_Variable.Identifier.String.uLength =
OpcUa_StrLenA(NODE_IDENTIFIER_STRING);

Note

You will find more detailed information on the Nodeld and the namespace of the server in the section "S7 communication with OPC UA (Page 159)".

6.10 OPC UA interface (asynchronous communication) in C#

You can download thoroughly described examples of programming the OPC UA interface in C# from our Internet pages:

Programming an OPC UA .NET client with C# for the SIMATIC NET OPC UA server
(<http://support.automation.siemens.com/WW/view/en/42014088>)

The OPC UA client in the PC station is implemented here in two stages of complexity. A very simple client (Simple OPC UA Client) shows you all the basic functions to get you started quickly in OPC UA. A more complex client (OPC UA .NET Client) with a convenient user interface demonstrates professional use of OPC UA implemented with reusable classes under .NET in the C# programming language.

The following scenarios are explained in terms of the program in both sample clients:

- Logging on, logging off and authenticating with the OPC UA server
- Browsing the namespace of variables
- Reading, writing and monitoring variables
- Reading and writing when using S7 block services
- Use of absolute and symbolic addressing
- Simple error handling

Reference Automation Interface

Introduction

This section contains the Specification of the OPC Automation Interface.

It describes the properties, the methods and, where applicable, the events of the individual objects.

7.1 General information

This introduction to the OPC Automation Interface explains the most important terminology that is used often in the specification. It is essential to understanding the Automation Interface.

The following terms are explained in detail:

- Interface
- Interface types
- COM/OLE objects
- Collection objects
- Object model
- Data synchronization
- Exceptions
- Events
- Arrays
- Parameters
- Type library

This description relates to version 2.02 of the OPC Automation Interface specification.

7.1.1 What is an interface?

Objects can have several interfaces and are described fully by the interfaces. The methods (object functions) can only be executed over the interfaces and it is only possible to access object over the interfaces. Objects can be used by several applications. An interface consists of a table of pointers to the actual functions.

Structure of an interface

In OLE 2.0, a pointer to an interface is a pointer to a jump table with function pointers. The caller uses a pointer to the required interface. This in turn contains a pointer to a list of function pointers. These now reference the actual methods in the COM/OLE objects.

7.1.2 The two Interface types of OPC

COM/OLE interfaces

A COM/OLE component provides objects and their methods for other components or applications. These objects are accessed via the COM/OLE interfaces. An interface in the sense of COM/OLE is a group of logically related functions.

The OPC server for SIMATIC NET supports two different types of COM/OLE interface:

- the Automation and
- the custom interface.

Both interfaces are used for the communication between objects. The automation and custom interfaces differ from each other in the way in which the methods of an interface are called internally. This means that there are also two different interface specifications for the OPC server.

When is which interface used?

Client applications based on a script language such as Visual Basic or VBA must use the automation interface. Applications in the C/C++ programming language should use the custom interface to achieve maximum performance. It is, nevertheless, also possible to use the automation interface in C/C++.

7.1.3 COM/OLE objects

COM/OLE objects are units in Windows that provide other objects with defined functions via their interfaces. COM/OLE objects provide their services via defined interfaces. The content of the object, data and code, remains hidden to the object user. COM/OLE objects are defined by their interfaces. The term object in the sense of OLE is not the same as the object definition in object-oriented programming languages. COM/OLE objects, for example, do not support inherit functions.

Structure of a COM/OLE object

The object is accessed via only one of the interfaces. It is not possible to access the actual object as a whole, nor the data or code it contains. The interfaces hide the methods assigned to them.

7.1.4 Collection objects

A collection object is used to generate and manage subunits. The corresponding subunits can only be created when a collection object already exists. The collection objects specify default values for the relevant subunits.

COM/OLE automation collections are objects that support the *Item* method, the *Count* property and the hidden property *_NewEnum*. Each object that has these properties as part of the interface is a collection object.

In VB, there are two methods that run through the elements of these collections.

The first method explicitly uses *Count* and *Item* to identify the collection elements.

```
For I = 1 To object.Count
element = object.Item ( I )

' or...

element = object( I )
Next I
```

In the second option, the available items are run through by using the hidden function *_NewEnum*:

```
For Each element In object
...
Next element
```

The method with *For Each* runs through a collection faster than when the *Item* property is used.

With *Item*, it is also possible to access a specific index such as *Item(3)*. The use of this property is therefore not restricted to loops.

Two types of collection objects

In OPC, there are two different collection objects: The OPCGroups collection object can be used to create and/or edit objects of the OPCGroup class. The OPCGroups collection object has seven properties, seven methods and one event.

The OPCItems collection object can be used to create and/or edit objects of the OPCItem class. The OPCItems collection object has five properties and nine methods, but no events.

7.1.5 Object model

Objects of the OPCServer class

Objects of the OPCServer class are created by the client. The properties of an OPC server contain general information about the server. When an OPCServer object is created, an OPCGroup collection is also created as a property of the OPCServer object.

Objects of the OPCGroups class

The OPCGroups object is a collection object for creating and managing OPCGroup objects. The default properties of OPCGroups specify default values for creating all OPCGroup objects.

Note

Public groups are not supported by the OPC Server for SIMATIC NET.

Objects of the OPCGroup class

The OPCGroup class manages the individual process variables, the OPC items. Using the OPCGroup objects, a client can form meaningful units of OPCItem objects and execute operations with them.

Objects of the OPCItems class

The OPCItems object is a collection object for creating and managing objects of the OPCItem class. The default properties of OPCItems specify default values for all OPCItem objects to be created.

Objects of the OPCItem class

An object of the OPCItem class represents a connection to a process variable, for example to an input module of a programmable controller. A process variable is a writable and/or readable data item of the process I/O such as the temperature of a tank. Each process variable is associated with a value (data type VARIANT), a quality and a timestamp.

The following schematic illustrates the hierarchy on which the object model is based:

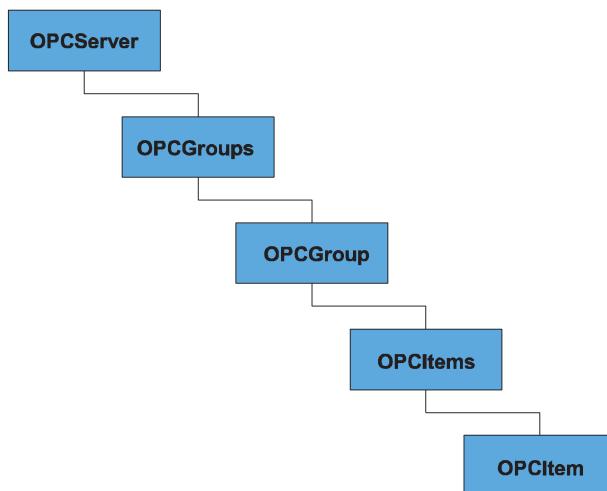


Figure 7-1 The hierarchy underlying in the object model

7.1.6 Data synchronization

The VB client must be capable of reading and receiving data so that the value, quality and timestamp are kept synchronized. The client must be certain that the data quality and timestamp match the values.

If the client obtains values with a read method, the value, quality and timestamp are synchronized.

If a client obtains data using the *DataChange* event, the value, timestamp and quality are synchronized within the scope of the event handling routine.

If these two methods of obtaining data are not kept strictly separate, the client cannot tell whether or not the item properties are exactly synchronized; this can, for example, be the case when the properties of the data change while the client is accessing the properties.

7.1.7 Exceptions

Most of the properties and events described here communicate with an OPC custom server. With OLE automation, it is not easy to return an error when a property is accessed. If there is an error in the corresponding data source, the automation server generates an exception. This means that the client must have an exception handling routine to be able to deal with errors.

Errors occurring when a property is written are indicated by the *Error* object of Visual Basic.

7.1.8 Events

The automation interface supports the notification mechanism for events of Visual Basic 5.0.

The automation server triggers events according to the requirements of the method calls *AsyncRefresh*, *AsyncRead* and *AsyncWrite* and also when data changes according to the specifications of the client.

The implementation assumes that the client has a suitable event handling routine.

7.1.9 Arrays

The numbering of arrays begins with 1. If an array is used in a function and the array is longer than the count or *NumItems* parameter, only the number of array elements specified by the parameter is used (the index begins at 1).

This relates only to parameters for functions and events within the automation interface; it does not affect item values for which the data type itself is an array.

To avoid errors, the VB code *Option Base 1* should be used.

7.1.10 Parameters

Optional parameters

Optional parameters are indicated by *Optional*. Optional parameters do not necessarily need to be included in a method call if the default behavior is adequate. In OLE Automation, the optional parameters must be declared as *Variant* even if they contain a string or an array etc.

Method parameters

Method parameters are passed on as values unless they are specified as a reference with the *ByRef* supplement. *ByRef* parameters are assigned values by methods and returned.

7.1.11 Type library

In VB, the "OPC Automation Type Library" is used to define the following interfaces. Make sure that "OPC Automation 2.0" is activated.

7.2 The OPCServer object

Description

A client generates the OPCServer automation object and then *connects* it with the OPC Data Access custom interface (see *Connect* method). Using the OPCServer object, general information about an OPC server can be called up and an OPCGroups collection object can be created and processed.

Syntax

OPCServer

Remarks

The effect of a declaration with *WithEvents* is that an object supports the events specified for this object type. The OPC server object has only one event, the *ServerShutDown* event. The OPCGroup Object has all events relating to *DataChange* and the support of asynchronous methods of the OPCGroup object.

Example

```
Dim WithEvents AnOPCServer As OPCServer  
Set AnOPCServer = New OPCServer
```

7.2.1 Properties of the OPCServer object

StartTime
CurrentTime
LastUpdateTime
MajorVersion
MinorVersion
BuildNumber
VendorInfo
ServerState
LocaleID
Bandwidth
OPCGroups
PublicGroupNames
ServerName
ServerNode
ClientName

7.2.1.1 StartTime

Description

(Read-only) This property indicates the time at which the server selected by the client was started. Multiple clients connected to the same OPC server all read the same time for this property.

Syntax

```
StartTime As Date
```

Remarks

The automation server uses the custom interface *GetStatus()* to obtain the values both for this property as well as for many other properties of the OPCServer object. Errors occur when the client did not connect to the OPC server with the *Connect* method.

Example

```
Dim AnOPCServerTime As Date
AnOPCServerTime = AnOPCServer.StartTime
```

7.2.1.2 CurrentTime

Description

(Read-only) This property returns the current time from the server. If you access the *CurrentTime* property, you obtain the value that the automation server received from the custom server via the *GetStatus* interface.

Syntax

```
CurrentTime As Date
```

Remarks

Errors occur when the client did not connect to the OPC server with the *Connect* method.

Example

```
Dim AnOPCServerTime As Date  
AnOPCServerTime = AnOPCServer.CurrentTime
```

7.2.1.3 LastUpdateTime

Description

(Read-only) This property indicates the time at which the server sent the last data update. If you access the *LastUpdateTime* property, you obtain the value that the automation server received from the custom server via the GetStatus interface.

Syntax

```
LastUpdateTime As Date
```

Remarks

This property indicates the time at which the server last sent data to an application client. Errors occur when the client did not connect to the OPC server with the *Connect* method.

Example

```
Dim AnOPCServerTime As Date  
AnOPCServerTime = AnOPCServer.LastUpdateTime
```

7.2.1.4 MajorVersion

Description

(Read-only) This property indicates the major version number of the server (for example 1 for version 1.32). If you access the *MajorVersion* property, you obtain the value that the automation server received from the custom server via the GetStatus interface.

Syntax

```
MajorVersion As Integer
```

Remarks

Errors occur when the client did not connect to the OPC server with the *Connect* method.

Example

```
Dim AnOPCServerMajorVersion As String  
AnOPCServerMajorVersion = Str(AnOPCServer.MajorVersion)
```

7.2.1.5 MinorVersion

Description

(Read-only) This property indicates the minor version number of the server (for example 32 for version 1.32). If you access the *MinorVersion* property, you obtain the value that the automation server received from the custom server via the GetStatus interface.

Syntax

```
MinorVersion As Integer
```

Remarks

Errors occur when the client did not connect to the OPC server with the *Connect* method.

Example

```
Dim AnOPCServerMinorVersion As String  
AnOPCServerMinorVersion = Str(AnOPCServer.MinorVersion)
```

7.2.1.6 BuildNumber

Description

(Read only) This property specifies the build number of the server. If you access the *BuildNumber* property, you obtain the value that the automation server received from the custom server via the GetStatus interface.

Syntax

```
BuildNumber As Integer
```

Remarks

Errors occur when the client did not connect to the OPC server with the *Connect* method.

Example

```
Dim BuildNumber as Integer  
BuildNumber = AnOPCServer.BuildNumber
```

7.2.1.7 **VendorInfo**

Description

(Read only) This property provides information on the vendor of the server (as a string). If you access the *VendorInfo* property, you obtain the value that the automation server received from the custom server via the GetStatus interface.

Syntax

```
VendorInfo As String
```

Remarks

Errors occur when the client did not connect to the OPC server with the *Connect* method.

Example

```
Dim info As String  
info = AnOPCServer.VendorInfo
```

7.2.1.8 **ServerState**

Description

(Read only) This property indicates the state of the server as a constant of the type OPCServerState.

Syntax

```
ServerState As Long
```

Settings

OPC_STATUS_RUNNING

The server is running normally. This is the normal state of the server.

OPC_STATUS_FAILED

A vendor-specific error has occurred on the server and it cannot operate correctly. The elimination of the error is vendor-specific. Every other method of the server outputs the error code E_FAIL.

OPC_STATUS_NOCONFIG

Although the server is operating, it has no information about the configuration and therefore cannot operate correctly.

Note: The server requires information about the configuration so that it can operate! Servers that do not require information about the configuration do not indicate this status.

OPC_STATUS_SUSPENDED

At times, the server has no connection via a vendor-specific method and it does not send or receive data. The information on the quality is as follows:

OPC_QUALITY_OUT_OF_SERVICE.

OPC_STATUS_TEST

The server is in test mode. Data output is disconnected from the hardware, otherwise the server behaves normally. Depending on the vendor-specific implementation, the input data is either actually present or is simulated. Information on the quality is output normally.

Remarks

The above status information is described in the specification of the Data Access Custom interface and is returned by an OPC server via the Custom interface; for more detailed information, see *IOPCServer::GetStatus()* in the *OPC Data Access Custom Interface* specification. The automation server has adopted this information from the custom server; they are connected together via the *GetStatus ()* interface. Errors occur when the client did not connect to the OPC server with the *Connect* method.

Example

```
Dim ServerState As Long
ServerState = AnOPCServer.ServerState
```

7.2.1.9 LocaleID**Description**

(Read and write) This property identifies the locality; using *LocaleID*, strings returned by the server can be adapted to specific languages. This *LocaleID* is used by the *GetErrorString* Method of this interface.

Syntax

```
LocaleID As Long
```

Remarks

The *LocaleID* should be set as default in all functions of the server that are affected by the *LocaleID*.

Errors occur when the client did not connect to the OPC server with the *Connect* method.

Example

```
' Get the property:  
Dim LocaleID As Long  
LocaleID = AnOPCServer.LocaleID  
  
' Specify the property:  
AnOPCServer.LocaleID = LocaleID
```

7.2.1.10 Bandwidth

Description

(Read-only) *Bandwidth* specifies the bandwidth of the server. This property is server-specific. It is advisable to specify the bandwidth of the server as a percentage of the available bandwidth. The value is hFFFFFFF if the server cannot calculate bandwidth. If you access the this property, you obtain the value that the automation server received from the custom server via the GetStatus interface.

Syntax

```
Bandwidth As Long
```

Remarks

Errors occur when the client did not connect to the OPC server with the *Connect* method.

Example

```
Dim Bandwidth As Long  
Bandwidth = AnOPCServer.Bandwidth
```

7.2.1.11 OPCGroups

Description

(Read-only) *OPCGroups* is a collection of objects of the *OPCGroup* class. This is a default property of the *OPCServer* object.

Syntax

```
OPCGroups As OPCGroups
```

Example

```
' To specify the property precisely:  
Dim groups As OPCGroups  
Set groups = AnOPCServer.OPCGroups  
  
' Default specification:  
Dim groups As OPCGroups  
Set groups = AnOPCServer
```

7.2.1.12 PublicGroupNames

Description

(Read-only) This property specifies the name of the PublicGroups provided by the server. These names can be used in ConnectPublicGroup. The names are output as an array of strings.

Syntax

```
PublicGroupNames As Variant
```

Remarks

Errors occur when the client did not connect to the OPC server with the *Connect* method. If the OPC server does not support any public groups or if no public groups are defined, an empty list is output.

Example

```
Dim AllPublicGroupNames As Variant  
AllPublicGroupNames = AnOPCServer.PublicGroupNames
```

7.2.1.13 ServerName

Description

(Read-only) This property outputs the name of the server with which the client is connected via the *Connect* method.

Syntax

```
ServerName As String
```

Remarks

The automation server returns the value that it finds locally in the cache. If no value was output, the client is not connected to a Data Access server.

Example

```
Dim info As String  
info = AnOPCServer.ServerName
```

7.2.1.14 ServerNode

Description

(Read-only) This property outputs the computer name of the node in the network on which the OPC server operates. If you access this property, you obtain the value that the automation server finds locally in the cache.

Syntax

```
ServerNode As String
```

Remarks

If the client is not connected to a Data Access server, no value is output. If no host name was specified in the *Connect* method, the string is also empty.

Example

```
Dim info As String  
info = AnOPCServer.ServerNode
```

7.2.1.15 ClientName

Description

(Read and write) Here, the client has the option of registering a ClientName with the server. This procedure is used mainly in troubleshooting. It is advisable for the client to specify its node and EXE name here.

Syntax

```
ClientName As String
```

Remarks

It is advisable to enter both the node and client name in the string separated by a semicolon (;).

Example

```
' To get the property:  
Dim info As String  
info = AnOPCServer.ClientName  
  
' To specify the property:  
AnOPCServer.ClientName = "NodeName;c:\programfiles\vendor\someapplication.exe"
```

7.2.2 Methods of the OPCServer object

- GetOPCServers
- Connect
- Disconnect
- CreateBrowser
- GetErrorString
- QueryAvailableLocaleIDs
- QueryAvailableProperties
- GetItemProperties
- LookupItemIDs

7.2.2.1 GetOPCServers

Description

This property outputs the names (ProgIDs) of the registered OPC servers. Use one of these ProgIDs in the *Connect* method. The names are output as an array of strings.

Syntax

```
GetOPCServers (Optional Node As Variant) As Variant Node
```

Node

With the node name, it is possible to specify the network nodes for which the automation server will output all registered OPC servers.

Remarks

For information on registering custom servers, refer to the standard of the OPC Data Access Custom interface.

The use of a node is optional. If a node name is used, access to another computer is achieved with DCOM. Permitted node names are UNC names and DNS names (server.com, www.vendor.com or 180.151.19.75).

Example

```

' Get the registered OPC servers (that actually
' exist that will be added to a VB
' list box):
Dim AllOPCServers As Variant
AllOPCServers = AnOPCServer.GetOPCServers
For i = LBound(AllOPCServers) To UBound(AllOPCServers)
    listbox.AddItem AllOPCServers(i)
Next i

```

7.2.2.2 Connect

Description

If you require any connection to an OPC data access server (that provides the custom interface), this method must be called.

Syntax

```
Connect (ProgID As String,
         Optional Node As Variant)
```

ProgID:

ProgID is a string that uniquely identifies the registered OPC Data Access server (that provides the custom interface).

Node:

With the node name, you can specify that the connection to another computer is established using DCOM.

Remarks

Each instance of an OPC automation server is connected to an OPC Data Access server (that provides the Custom interface).

The use of a node is optional. If a node name is used, access to another computer is achieved with DCOM. Permitted node names are UNC names and DNS names (server.com, www.vendor.com or 180.151.19.75).

With this method, the automation wrapper calls *CoCreateInstanceEx* and sets up a Data Access custom server (specified by the ProgID) in the specified computer (StrNodeName).

If this method is called a second time without terminating the connection previously by calling the *Disconnect method*, the automation wrapper automatically cancels the previous connection.

Use the *GetOPCServers* method to obtain valid ProgIDs.

Example

```

' Establish a connection to the first registered
' OPC Server found by the "GetOPCServers"
' method:
Dim AllOPCServers As Variant
AllOPCServers = AnOPCServer.GetOPCServers
AnOPCServer.Connect(AllOPCServers(1))

' Connection to a specific server or a
' network computer:
Dim ARealOPCServer As String
Dim ARealOPCNodeName As String
ARealOPCServer = "Vendorname.DataAccessCustomServer"
ARealOPCNodeName = "AComputername"
AnOPCServer.Connect (ARealOPCServer, ARealOPCNodeName)

```

7.2.2.3 Disconnect

Description

This method terminates the link to an OPC server.

Syntax

```
Disconnect()
```

Remarks

The connection to a server can be terminated using this method; afterwards, either a new connection to another server can be established or the server object can be deleted.

It is good programming style if the client application with the corresponding Automation methods terminates all objects it created (including all OPCGroup(s) and OPCItem objects). The *Disconnect* method removes all group objects and references to the relevant OPC Custom server.

Example

```
AnOPCServer.Disconnect
```

7.2.2.4 CreateBrowser

Description

This method generates an object of the OPCBrowser class.

Syntax

```
CreateBrowser() As OPCBrowser
```

Remarks

The OPC Browser interface is optional and does not necessarily need to be supported by an OPC Custom interface server. This means that an OPC Custom interface server that does not implement the Browser interface also does not return an object of the OPC Browser class.

Example

```
Dim ARealOPCServer As String
Dim ARealOPCNodeName As String
ARealOPCServer = "Vendorname.DataAccessCustomServer"
ARealOPCNodeName = "AComputername"
AnOPCServer.Connect(ARealOPCServer, ARealOPCNodeName)
Dim AnOPCServerBrowserObject As OPCBrowser
Set AnOPCServerBrowserObject = AnOPCServer.CreateBrowser
```

7.2.2.5 GetErrorString

Description

This method converts an error message into a readable string. The server outputs the string in the localization specified in the LocaleID property of the server.

Syntax

```
GetErrorString (ErrorCode As Long) As String
```

ErrorCode

ErrorCode is a server-specific error code that the client application has received back from an interface function of the server. For this error code, the client queries the text representation.

Example

```

Dim AnOPCServerErrorString As String
' this example assumes that while adding
' several items, an item is detected as being invalid; the code
' is not given completely here to simplify matters
AnOPCItemCollection.Add AddItemCount,
AnOPCItemIDs,
AnOPCItemServerHandles,
AnOPCItemErrors
' Get and display the error message to inform the user
' why the item could not
' be added
AnOPCServerErrorString =
    AnOPCServer.GetErrorString(AnOPCItemErrors (index))
' further code
ErrorBox.Text = AnOPCServerErrorString
' further code

```

7.2.2.6 QueryAvailableLocaleIDs

Description

This method outputs the available LocaleIDs for the existing server-client connection as an array of long values.

Syntax

```
QueryAvailableLocaleIDs () As Variant
```

Example

```

Dim LocaleID As Variant
Dim AnOPCTextString as String
AnOPCServerLocaleID = AnOPCServer.QueryAvailableLocaleIDs()
For i = LBound(LocaleID) To UBound(LocaleID)
    AnOPCTextString = LocaleIDToString(LocaleID(i))
    listbox.AddItem AnOPCTextString
Next i

```

7.2.2.7 QueryAvailableProperties

Description

This method outputs a list of descriptions and ID codes of the available properties of the relevant ItemID. This list can differ depending on the ItemID. It should be relatively stable for a particular ItemID. The means that it may be affected by changes to the system configuration.

Syntax

```
QueryAvailableProperties (ItemID As String,
                           ByRef Count As Long,
                           ByRef PropertyIDs () as Long,
                           ByRef Descriptions() As String,
                           ByRef DataTypes() As Integer)
```

ItemID

ItemID is the ItemID for which the available properties should be queried.

Count

Count indicates the number of properties found.

PropertyIDs

PropertyIDs are IDs of the type DWORD for the properties found. These IDs can be passed on the *GetItemProperites* or *LookupItemIDs*.

Descriptions

Descriptions is a brief description of each property provided by the vendor.

Note: LocaleID has no influence on the language of this description.

DataTypes

DataTypes is the data type that *GetItemProperties* outputs for this property.

Example

```
' Get the available properties:
Dim OPCItemID As String
Dim ItemCount As Long
Dim PropertyIDs() As Long
Dim Descriptions() As String
Dim DataTypes() As Integer
Dim AnOPCTextString As String
OPCItemID = "SomeOPCDataAccessItem"
AnOPCServer.QueryAvailableProperties (OPCItemID, ItemCount,
                                       PropertyIDs,
                                       Descriptions,
                                       DataTypes)
For i = 1 To ItemCount
    AnOPCTextString = Str(PropertyIDs(i)) + " "
    + Descriptions(i)
    listbox.AddItem AnOPCTextString
Next I
```

7.2.2.8 **GetItemProperties**

Description

This method outputs a list of the current values of the properties for the ItemID entered.

Syntax

```
QueryAvailableProperties (ItemID As String,
                         ByRef Count As Long,
                         ByRef PropertyIDs () as Long,
                         ByRef Descriptions() As String,
                         ByRef DataTypes() As Integer)
```

ItemID

ItemID is the ItemID for which a list of properties is requested.

Count

Count indicates the number of returned properties.

PropertyIDs

PropertyIDs are IDs of the type DWORD for the required properties. These IDs were output by *QueryAvailableProperties*.

PropertyValues

PropertyValues is an array with the size *Count* with variables of the type VARIANT that is output by the server; this array contains the current values of the required properties.

Errors

Errors is an array of error codes that indicate whether each property was returned.

Example

```
Dim OPCItemID as String
Dim ItemCount As Long
Dim PropertyIDs(3) as Long
Dim Data() as Variant
Dim Errors() as Long
Dim AnOPCTextString As String

' Specifies the values for ItemCount and PropertyIDs...
AnOPCServer.GetItemProperties (OPCItemID, ItemCount,
                                 PropertyIDs, Data, Errors)
For i = 1 To ItemCount
    AnOPCTextString = Str(PropertyIDs(i)) + " " + Data(i)
    listbox.AddItem AnOPCTextString
Next i
```

7.2.2.9 LookupItemIDs

Description

This method returns a list of the ItemIDs (if applicable) corresponding to the transferred PropertyIDs. The ItemIDs can be inserted in OPCGroups and used for more effective access to data of the relevant item properties. An error within an error array can indicate that the returned property ID is not defined for the relevant item.

Syntax

```
LookupItemIDs (ItemID As String,
               Count As Long,
               PropertyIDs() as Long,
               ByRef NewItemIDs() As String,
               ByRef Errors () As Long)
```

ItemID

ItemID shows the ItemID for which a list of properties is requested.

Count

Count corresponds to the number of returned properties.

PropertyIDs

PropertyIDs are IDs of the type DWORD for the required properties. These IDs were output by *QueryAvailableProperties*.

NewItemIDs

NewItemIDs contains the returned list of ItemIDs.

Errors

Errors is an array of error codes that indicate whether ItemIDs were returned.

Example

```
Dim OPCItemID as String
Dim Count As Long
Dim PropertyIDs(1) as Long
Dim NewItemIDs () as String
Dim Errors() as Long
Dim AnOPCTextString As String
OPCItemID = "OneOPCDataAccessItem"
Count = 1
PropertyIDs(1) = 5;
AnOPCServer.LookupItemIDs (OPCItemID, Count, PropertyIDs,
                           NewItemIDs, Errors)
For i = 1 To Count
    AnOPCTextString = Str(PropertyIDs(i)) + " " +
                      NewItemIDs(i)
    listbox.AddItem AnOPCTextString
Next i
```

7.2.3

Events of the OPCServer object

The only event that exists for the OPC server is the ServerShutDown event.

7.2.3.1 ServerShutDown

Description

This event is triggered when the server is to be shut down and all active clients should terminate their connection to the server. The client application supports this command so that the client terminates the connection to the server at the command of the server and clears all groups and items.

Syntax

```
ServerShutDown (Reason As String)
```

ServerReason

This optional text string of the server indicates why it is shutting down.

Example

```
Dim WithEvents AnOPCServer As OPCServer
Dim ARealOPCServer As String
Dim ARealOPCNodeName As String
Set AnOPCServer = New OPCServer
' this example is necessary
' to make creation of an object with the addition "WithEvents"
' easier
ARealOPCServer = "Vendorname.DataAccessCustomServer"
ARealOPCNodeName = "AComputername"
AnOPCServer.Connect(ARealOPCServer, ARealOPCNodeName)
Private Sub AnOPCServer_ServerShutDown
    (ByRef aServerReason As String)
    ' Program code to terminate the connection to the server
End Sub
```

7.3 The collection object OPCGroups

Description

The OPCGroups object is a collection of objects of the OPCGroup class and the methods with which this is created, organized and removed.

This object has properties for OPCGroup defaults. If an object of the OPCGroup class is added, its initial status is specified with the default property DefaultGroupXXXX. The defaults can be changed to create OPCGroups with a different initial status. Changing the defaults has no effect on existing group objects. As soon as a group object has been added, its properties can be changed. This reduces the number of parameters necessary to call the *Add* method.

Example

The following code is required to allow the following VB examples to work.

```

Dim WithEvents AnOPCServer As OPCServer
Dim ARealOPCServer As String
Dim ARealOPCNodeName As String
Dim AnOPCServerBrowser As OPCBrowser
Dim MyGroups As OPCGroups
Dim DefaultGroupUpdateRate As Long
Dim OneGroup As OPCGroup
Dim AnOPCItemCollection As OPCItems
Dim AnOPCItem As OPCItem
Dim ClientHandles(100) As Long
Dim AnOPCItemIDs(100) As String
Dim AnOPCItemServerHandles() As Long
Dim AnOPCItemServerErrors() As Long
Set AnOPCServer = New OPCServer
ARealOPCServer = "Vendorname.DataAccessCustomServer"
ARealOPCNodeName = "OneComputername"
AnOPCServer.Connect(ARealOPCServer, ARealOPCNodeName)
Set MyGroups = AnOPCServer.OPCGroups
MyGroups.DefaultGroupIsActive = True
Set OneGroup = MyGroups.Add("OneOPCGroupName")
Set AnOPCItemCollection = OneGroup.OPCItems

```

7.3.1 Properties of the OPCGroups object

- Parent
- DefaultGroupIsActive
- DefaultGroupUpdateRate
- DefaultGroupDeadband
- DefaultGroupLocaleID
- DefaultGroupTimeBias
- Count

7.3.1.1 Parent

Description

(Read-only) This property returns a reference to the higher-level OPCServer object.

Syntax

```
Parent As OPCServer
```

7.3.1.2 DefaultGroupIsActive

Description

(Read and write) This property sets the initial value for the *IsActive* property of a newly generated OPCGroup.

Syntax

```
DefaultGroupIsActive As Boolean
```

Remarks

The default setting of this property is *True*.

Example

```
' Example of VB syntax (getting the property):
Dim DefaultGroupIsActive As Boolean
DefaultGroupIsActive = MyGroups.DefaultGroupIsActive

' Example of VB syntax (specifying the property):
MyGroups.DefaultGroupIsActive = FALSE
```

7.3.1.3 DefaultGroupUpdateRate

Description

(Read and write) This property sets the initial value for the *UpdateRate* property of a newly generated OPCGroup in milliseconds, default: 1000 ms (= 1 second).

Syntax

```
DefaultGroupUpdateRate As Long
```

Example

```
' Example of VB syntax (getting the property):
Dim DefaultGroupUpdateRate As Long
DefaultGroupUpdateRate = MyGroups.DefaultGroupUpdateRate

' Example of VB syntax (specifying the property):
MyGroups.DefaultGroupUpdateRate = 250
```

7.3.1.4 DefaultGroupDeadband

Description

(Read and write) This property sets the initial value for the *Deadband* property as a percentage for OPCGroups created with *Add*. The *Deadband* property is specified as a percentage of the entire width (valid values from 0 to 100%).

Syntax

```
DefaultGroupDeadband As Single
```

Remarks

The default setting of this property is "0". If the value is >100 or <0, an error message is output.

Example

```
' Example of VB syntax (getting the property):
Dim DefaultGroupDeadband As Single
DefaultGroupDeadband = MyGroups.DefaultGroupDeadband

' Example of VB syntax (specifying the property:
MyGroups.DefaultGroupDeadband = 10
```

7.3.1.5 DefaultGroupLocaleID

Description

(Read and write) This property sets the initial value for the *LocaleID* property for an OPCGroup that was created with *Add*.

Syntax

```
DefaultGroupLocaleID As Long
```

Remarks

The default setting for this property does not match the settings of the LocaleID of the server.

Example

```
' Example of VB syntax (getting the property):
Dim DefaultGroupLocaleID As Long
DefaultGroupLocaleID = MyGroups.DefaultGroupLocaleID

' Example of VB syntax (setting the property):
MyGroups.DefaultGroupLocaleID =
    ConvertLocaleIdStringToLocaleIdLong("English")
```

7.3.1.6 DefaultGroupTimeBias

Description

(Read and write). This property sets the initial value in minutes for the *TimeBias* property (= time offset) for objects of the OPCGroup class created with *Add*.

Syntax

```
DefaultGroupTimeBias As Long
```

Remarks

The default setting for this property is *0 minutes*.

Example

```
' Example of VB syntax (getting the property):
Dim DefaultGroupTimeBias As Long
DefaultGroupTimeBias = MyGroups.DefaultGroupTimeBias

' Example of VB syntax (specifying the property):
MyGroups.DefaultGroupTimeBias = 60
```

7.3.1.7 Count

Description

(Read-only) This is the number of groups. This property is required for collection objects.

Syntax

```
Count As Long
```

Example

```
' Example of VB syntax:  
For index = 1 to MyGroups.Count  
    ' further code  
Next index
```

7.3.2 Methods of the OPCGroups object

Item
Add
GetOPCGroup
Remove
RemoveAll
ConnectPublicGroup
RemovePublicGroup

7.3.2.1 Item

Description

This method returns a reference to the specified item. The item can be specified by its name or an index (beginning at 1). GetOPCGroup is used to reference according to ServerHandle. With OPCGroups, *Item* is the default method.

Syntax

```
Item (ItemSpecifier As Variant) As OPCGroup
```

ItemSpecifier

Item Specifier is either the ServerHandle of the OPCGroup or the name of an OPCGroup. To reference with an index, the Item method is used.

Example

```
' Example of VB syntax:  
Dim AnOPCGroup As OPCGroup  
Set AnOPCGroup = MyGroups.Item(3)  
  
' or  
Set AnOPCGroup = MyGroups("Group3")
```

7.3.2.2 Add

Description

This method creates a new OPCGroup object and adds it to the collection. The properties of this new group are specified by the current default settings of the OPCServer object. After a group has been added, its properties can be modified as necessary.

Syntax

```
Add (Optional Name As Variant) As OPCGroup
```

Name

Name specifies the name of the group. The name must be unique within all the groups created by this client. If no name is specified, the name generated by the server is unique in all existing groups.

Remarks

If no name is specified, the server itself generates a unique name. If a name is specified and this is not unique, no OPCGroup object will be generated and VB outputs and a message if there is an attempt to call the object that was not created.

For more detailed information on errors and exceptions, refer to the Automation Interface referenced in the appendix.

Example

```
MyGroups.DefaultGroupIsActive = True
Set OneGroup = MyGroups.Add("OneOPCGroupName")
```

7.3.2.3 GetOPCGroup

Description

This method returns the reference to an OPCGroup identified by the name or the server handle.

Syntax

```
GetOPCGroup (ItemSpecifier As Variant) As OPCGroup
```

ItemSpecifier

ItemSpecifier is either the ServerHandle of the OPCGroup or the name of an OPCGroup. To reference with an index, the Item method is used.

Example

```
' The example assumes that  
' "OneOPCGroupName" has already been added  
Set OneGroup = MyGroups.GetOPCGroup("OneOPCGroupName")
```

7.3.2.4 Remove

Description

This method removes an OPCGroup on the server.

Syntax

```
Remove (ItemSpecifier As Variant)
```

ItemSpecifier

ItemSpecifier is either the ServerHandle of the OPCGroup or the name of an OPCGroup. To reference with an index, the *Item* method is used.

Remarks

If the OPCGroup is a *PublicGroup*, this method cannot be used.

For more detailed information on errors and exceptions, refer to the Automation Interface referenced in the appendix.

Example

```
Set OneGroup = MyGroups.Add("OneOPCGroupName")  
' further code  
MyGroups.Remove("OneOPCGroupName")  
  
' or  
Set OneGroup = MyGroups.Add("OneOPCGroupName")  
' further code  
MyGroups.Remove(OneGroup.ServerHandle)
```

7.3.2.5 RemoveAll

Description

Removes all existing objects of the OPCGroup and OPCItem classes to prepare for shutting down the server.

Syntax

```
RemoveAll()
```

Remarks

This method simplifies the complete release of subobjects by the client if the server object is deleted. *RemoveAll* corresponds to the *Remove* method when this is called for all objects of the OPCGroup and OPCItem that still exist. Because objects of the OPCBrowser class are not subobjects of the server, they cannot be removed with the *RemoveAll* method.

Example

```
Set OneGroup = MyGroups.Add("EinOPCGroupName")
Set OneGroup = MyGroups.Add("EinOPCGroupName1")
Set OneGroup = MyGroups.Add("EinOPCGroupName2")
' further code
MyGroups.RemoveAll
```

7.3.2.6 ConnectPublicGroup

Description

PublicGroups are default groups of a server. Connections can be established to these groups without first adding them to the server.

Syntax

```
ConnectPublicGroup (Name As String) As OPCGroup
```

Name

Name is the name of the group to be connected.

Remarks

If the name is invalid or if PublicGroups is not supported, this method cannot be used. You will find further information on errors and exceptions in the appendix of the Automation Interface reference.

Example

```
Set OneGroup = MyGroups.ConnectPublicGroup ("AnOPCServerDefinedPublicGroup")
```

7.3.2.7 RemovePublicGroup

Description

This method removes the OPCGroup specified as the parameter.

Syntax

```
RemovePublicGroup (ItemSpecifier As Variant)
```

ItemSpecifier

ItemSpecifier is either the ServerHandle of the OPCGroup returned by *ConnectPublicGroup* or the name of an OPCGroup.

Remarks

If *PublicGroups* is not supported or if the group was not connected using the *ConnectPublicGroup* method, *RemovePublicGroup* cannot be used.

For more detailed information on errors and exceptions, refer to the Automation Interface referenced in the appendix.

Example

```
Set OneGroup = MyGroups.ConnectPublicGroup ("OneOPCGroupName")
' further code
MyGroups.RemovePublicGroup ("OneOPCGroupName")

'or
Set OneGroup = MyGroups.ConnectPublicGroup ("OneOPCGroupName")
' further code
MyGroups.RemovePublicGroup (OneGroup.ServerHandle)
```

7.3.3 Events of the OPCGroups object

The OPCGroups collection object only has the GlobalDataChange event.

7.3.3.1 GlobalDataChange

Description

GlobalDataChange simplifies the processing of events throughout all the groups of the collection by notifying values and status changes of all items in all groups.

Syntax

```
GlobalDataChange (TransactionID As Long,
                  GroupHandle As Long,
                  NumItems As Long,
                  ClientHandles() As Long,
                  ItemValues() As Variant,
                  Qualities() As Long,
                  TimeStamps() As Date)
```

TransactionID

TransactionID is the transaction ID specified by the client. If the value is not equal to 0, this

event occurred due to an *AsyncRefresh*. If the value is equal to 0, this event occurred due to normal event handling.

GroupHandle

GroupHandle is the ClientHandle of the OPCGroup to which the changed data belongs.

NumItems

NumItems specifies the number of items output.

ClientHandles

ClientHandles is an array of client handles for the items.

Item Values

ItemValues is an array with values.

Qualities

Qualities is an array of the quality of the value of each item.

TimeStamps

TimeStamps is an array with UTC timestamps for the value of each item.

Remarks

Please note that the *OnDataChange* event should be used normally with an OPCGroup. This event allows an EventHandle to be set up so that it can process data changes from several objects of the OPCGroups class. Normally, an application has an EventHandle for each group to allow data changes to be received and processed. This means that you only need one EventHandle and by using the *GroupHandle*, it is clear for which group the event was triggered.

The GlobalDataChange event is triggered for every object of the OPCGroup class containing items whose values or status has changed since the time at which the event was last triggered. The relevant event of the OPCGroup object is also triggered. If the application uses both global and group-specific events, it receives two notifications: Once for the global and once for the group-specific event.

Example

```
Dim WithEvents AnOPCGroupCollection As OPCGroups
Private Sub AnOPCGroupCollection_GlobalDataChange
    (TransactionID As Long,
     GroupHandle As Long,
     MasterQuality As Long,
     MasterError As Long,
     NumItems As Long,
     ClientHandles() As Long,
     ItemValues() As Variant,
     Qualities() As Long,
     TimeStamps() As Date)

    ' Here, enter the client code for processing changed
    ' values
    . . .
End Sub
```

7.4 The OPCGroup object

Description

A client can manage data using OPCGroup. The group can, for example, represent items on a particular user interface or in a report. Data can be read and written. Checkbacks from the items in the group to the client can be enabled and disabled when necessary. The client can configure the update rate of the data change in the OPC server.

Syntax

```
OPCGroup
```

Example

The following code is required to allow the following VB examples to work.

```
Dim WithEvents AnOPCServer As OPCServer
Dim ARealOPCServer As String
Dim ARealOPCNodeName As String
Dim AnOPCServerBrowser As OPCBrowser
Dim MyGroups As OPCGroups
Dim DefaultGroupUpdateRate As Long
Dim WithEvents OneGroup As OPCGroup
Dim AnOPCItemCollection As OPCItems
Dim AnOPCItem As OPCItem
Dim ClientHandles(100) As Long
Dim AnOPCItemIDs(100) As String
Dim AnOPCItemServerHandles() As Long
Dim AnOPCItemServerErrors() As Long
Set AnOPCServer = New OPCServer
ARealOPCServer = "Vendorname.DataAccessCustomServer"
ARealOPCNodeName = "OneComputername"
AnOPCServer.Connect(ARealOPCServer, ARealOPCNodeName)
Set MyGroups = AnOPCServer.OPCGroups
MyGroups.DefaultGroupIsActive = True
Set OneGroup = MyGroups.Add("OneOPCGroupName")
Set AnOPCItemCollection = OneGroup.OPCItems
For x = 1 To AddItemCount
    ClientHandles(x) = x + 1
    AnOPCItemID(x) = "Register_" & x
Next x
AnOPCItemCollection.AddItems AddItemCount, AnOPCItemIDs,
    AnOPCItemServerHandles, AnOPCItemServerErrors
```

7.4.1 Properties of the OPCGroup object

Parent
Name
IsPublic
IsActive
IsSubscribed
ClientHandle
ServerHandle
LocaleID
TimeBias
DeadBand
UpdateRate
OPCItems

7.4.1.1 Parent

Description

(Read-only) This property returns the reference to the higher-level OPCServer object.

Syntax

```
Parent As OPCServer
```

7.4.1.2 Name

Description

(Read and write) *Name* contains the name of the group.

Syntax

```
Name As String
```

Name

Name contains the name of the group. The name must be unique within all the groups created by the client.

Remarks

As an option, groups can be given a name. The user can assign a name using this property. The name must be unique. If no name is specified, the server generates a unique name for the group using the *Add* method of the OPCGroups object.

Example

```
' Example of VB syntax: (getting the property):
Dim CurrentValue As String
Set OneGroup = MyGroups.Add("OneOPCGroupName")
CurrentValue = OneGroup.Name

' Example of VB syntax (specifying the property):
Set OneGroup = MyGroups.Add("OneOPCGroupName")
OneGroup.Name = "OneName"
```

7.4.1.3 IsPublic

Description

(Read-only) This property returns *True* if a PublicGroup is involved, otherwise *False*.

Syntax

```
IsPublic As Boolean
```

Example

```
Dim CurrentValue As Boolean
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
Set CurrentValue = OneGroup.IsPublic
' this obtains the value
```

7.4.1.4 IsActive

Description

(Read and write) This property specifies whether all not a group is active. A group that is active contains data. An inactive group does not normally contain any data unless it is necessary for read and write operations.

Syntax

```
IsActive As Boolean
```

Remarks

The default setting for this property is the default value for the value of the OPCGroups object adds the time *Add()*.

Example

```
' Example of VB syntax: (getting the property):
Dim CurrentValue As Boolean
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
Set CurrentValue = OneGroup.IsActive
' this obtains the value

' Example of VB syntax (specifying the property):
Dim CurrentValue As Boolean
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
OneGroup.IsActive = True
```

7.4.1.5 IsSubscribed

Description

(Read and write) this property checks asynchronous messages to the group. A group that is subscribed on the server receives notifications from the server about changed data.

Syntax

```
IsSubscribed As Boolean
```

Remarks

The default setting for this property is the default value for the value of the OPCGroups object adds the time *Add()*.

Example

```
' Example of VB syntax: (getting the property):
Dim CurrentValue As Boolean
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
Set CurrentValue = OneGroup.IsSubscribed
' this obtains the value

' Example of VB syntax (specifying the property):
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
OneGroup.IsSubscribed = True
' this specifies the value
```

7.4.1.6 ClientHandle

Description

(Read and write) *ClientHandle* is a long value that stands for the group. This property allows the client to identify the recipient of the data quickly. This value is normally an index or similar and is returned to the client along with data or information on the status.

Syntax

```
ClientHandle As Long
```

Example

```
' Example of VB syntax (getting the property):
Dim CurrentValue As Long
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup =
    MyGroups.ConnectPublicGroup ("OneOPCGroupName")
    ' further code
Set CurrentValue = OneGroup.ClientHandle
    ' this obtains the value

' Example of VB syntax (specifying the property):
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup =
    MyGroups.ConnectPublicGroup ("OneOPCGroupName")
    ' further code
OneGroup.ClientHandle = 1975
    ' this obtains the value
```

7.4.1.7 ServerHandle

Description

(Read-only) this is the handle that the server assigned to the group. The value of this property is specified as a long value; this allows the group to be identified uniquely. The client must transfer this handle to several methods as a parameter when they operate with the objects of the OPCGroup class (for example *OPCGroups.Remove*).

Syntax

```
ServerHandle As Long
```

Example

```
' Example of VB syntax (getting the property):
Dim CurrentValue As Long
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
Set CurrentValue = OneGroup.ServerHandle
' this obtains the value
```

7.4.1.8 LocaleID

Description

(Read and write) This property identifies the language ID; strings returned by the server can be localized with this property. The default of this property depends on the value defined in the OPCGroups collection (*DefaultGroupLocaleID*).

Syntax

LocaleID As Long

Example

```
' Example of VB syntax (getting the property):
Dim CurrentValue As Long
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
Set CurrentValue = OneGroup.LocaleID

' Example of VB syntax (specifying the property):
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
OneGroup.LocaleID = StringToLocaleID("English")
```

7.4.1.9 TimeBias

Description

(Read and write) This property is required to convert the timestamp of the data to the local time of the device. Unit: Milliseconds.

Syntax

TimeBias As Long

Example

```
' Example of VB syntax (getting the property):
Dim CurrentValue As Long
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
Set CurrentValue = OneGroup.TimeBias

' Example of VB syntax (specifying the property):
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
OneGroup.TimeBias = 100
```

7.4.1.10 DeadBand

Description

(Read and write) The value of this property is specified as a percentage of the entire bandwidth (valid values from 0 through 100). The default of this property depends on the value defined in the OPCGroups collection (*DefaultGroupDeadband*).

Syntax

```
DeadBand As Single
```

Example

```
' Example of VB syntax (getting the property):
Dim CurrentValue As Single
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
Set CurrentValue = OneGroup.DeadBand

' Example of VB syntax (specifying the property):
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
OneGroup.DeadBand = 5
```

7.4.1.11 UpdateRate

Description

(Read and write) The maximum frequency (in ms) at which the DataChange events are triggered. A slow process may lead to data changes occurring less often than this frequency, however, never more often. The default of this property depends on the value defined in the OPCGroups collection (*DefaultGroupUpdateRate*). New values for this property are specified with a query for a new value. It is possible that the server does not support a particular frequency and that the result is a different frequency when the property is read in (the server uses the next higher frequency than the requested frequency).

Syntax

```
UpdateRate As Long
```

Example

```
' Example of VB syntax (getting the property):
Dim CurrentValue As Long
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("EinOPCGroupName")
' further code
DefaultGroupUpdateRate = OneGroup.UpdateRate

' Example of VB syntax (specifying the property):
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("EinOPCGroupName")
' further code
OneGroup.UpdateRate = 50
```

7.4.1.12 OPCItems

Description

OPCItems is a collection of OPCItem objects; this is the default property of the OPCGroup object.

Syntax

```
OPCItems As OPCItems
```

Example

```
' Example of VB syntax (getting the property):
Dim AnOPCItemCollection As OPCItems
Set MyGroups = AnOPCServer.OPCGroups
Set OneGroup = MyGroups.ConnectPublicGroup("OneOPCGroupName")
' further code
Set AnOPCItemCollection = OneGroup.OPCItems
```

7.4.2 Methods of the OPCGroup object

SyncRead
 SyncWrite
 AsyncRead
 AsyncWrite
 AsyncRefresh
 AsyncCancel

7.4.2.1 SyncRead

Description

This function reads values, quality and timestamp for an item or for multiple items of a group.

Syntax

```
SyncRead (Source As Integer,
          NumItems As Long,
          ServerHandles() As Long,
          ByRef Values() As Variant,
          ByRef Errors() As Long,
          Optional ByRef Qualities As Variant,
          Optional ByRef TimeStamps As Variant)
```

Source

Source is the data source; OPC_DS_CACHE or OPC_DS_DEVICE

NumItems

NumItems contains the number of items to be read.

ServerHandles

ServerHandles is an array of server handles for the items to be read.

Values

Values is an array of values.

Errors

Errors is an array of long values that specify whether the relevant items were read successfully, in other words, whether using the *Read* method, a defined value, a quality and a timestamp were received.

note: If *FAILED* is output in the error code, the values, quality and timestamp are not defined (UNDEFINED)!

Qualities

Qualities contains a value of the type VARIANT that contains an integer array with values relating to the quality.

TimeStamps

TimeStamps contains a value of the type VARIANT that contains a data array with UTC timestamps. If the device cannot return a timestamp, it is created by the server.

Remarks

The function is completed before the result is output. The data can be read from the CACHE; in this case, the data should conform with the UpdateRate and the *Deadband* property of the group. The data can also be read from the DEVICE; in this case, the data should actually be read from the real device. The precise implementation of the CACHE and DEVICE read methods is not dealt with in greater detail here.

The data read from the cache is only valid when both the group and the item are active. If one of these two elements is inactive, instead of the quality, the constant OPC_QUALITY_OUT_OF_SERVICE is output.

The status of Group or Item (active/inactive) has no influence on the DEVICE read method.

Example

```

Private Sub ReadButton_Click()
Dim Source As Integer
Dim NumItems As Long
Dim ServerIndex As Long
Dim ServerHandles(10) As Long
Dim Values() As Variant
Dim Errors() As Long
Dim Qualities() As Variant
Dim TimeStamps() As Variant
Source = OPC_DS_DEVICE
NumItems = 10
For ServerIndex = 1 to NumItems
    ' specifies which item will be read
    ServerHandles(ServerIndex) = AnOPCItemServerHandles(ServerIndex)
Next ServerIndex
OneGroup.SyncRead Source,
    NumItems,
    ServerHandles,
    Values,
    Errors,
    Qualities,
    TimeStamps
For ServerIndex = 1 to NumItems
    ' processes the values
    TextBox(ServerIndex).Text = Values(ServerIndex)
Next ServerIndex
End Sub

```

7.4.2.2 SyncWrite

Description

This method writes values for one or more items of a group. The function is executed completely. The values are written to DEVICE; this means that the function should not be returned until it is certain that the device has accepted the data (or rejected the data).

Syntax

```

SyncWrite (NumItems As Long,
          ServerHandles() As Long,
          Values() As Variant,
          ByRef Errors() As Long)

```

NumItems

NumItems specifies the number of items to be written.

ServerHandles

ServerHandles is an array of server handles for the items to be written.

Values

Values is an array of values.

Errors

Errors is an array of long values that specify whether the items were written successfully.

Remarks

The status of the group or item (active/inactive) has no influence on the write method.

Example

```
Private Sub WriteButton_Click()
Dim Source As Integer
Dim NumItems As Long
Dim ServerIndex As Long
Dim ServerHandles() As Long
Dim Values() As Variant
Dim Errors() As Long
NumItems = 10
For ServerIndex = 1 to NumItems
    ' specifies which item will be written
    ServerHandles(ServerIndex) = AnOPCItemServerHandles(ServerIndex)
    Values(ServerIndex) = ServerIndex * 2
    ' the value of the item can be any value here
Next ServerIndex
OneGroup.SyncWrite NumItems, ServerHandles, Values, Errors
For ServerIndex = 1 to NumItems
    ' processes the errors
    TextBox(ServerIndex).Text = Errors(ServerIndex)
Next ServerIndex
End Sub
```

7.4.2.3 AsyncRead**Description**

This method reads an item or several items of a group. The results are returned with the *AsyncReadComplete* event assigned to the OPCGroup object.

Syntax

```
AsyncRead (NumItems As Long,
           ServerHandles() As Long,
           ByRef Errors() As Long,
           TransactionID As Long,
           ByRef CancelID As Long)
```

NumItems

NumItems contains the number of items to be read.

ServerHandles

ServerHandles is an array of server handles for the items to be read.

Errors

Errors is an array of long values that specifies the status of the item to be read.

TransactionID

TransactionID is the transaction ID specified by the client. This ID is contained in the relevant event.

CancelID

CancelID is a transaction ID generated by the server with which the client can cancel the transaction.

Remarks

The *AsyncRead* method assumes that the object of the OPCGroup class is declared with events (Dim WithEvents xxx As OPCGroup) so that the result of *AsyncRead* can be returned to the client application. The *AsyncReadComplete* event that is assigned to the OPCGroup object is triggered by the automation server with the result from *AsyncRead*.

The status of the group or item (active/inactive) has no influence on the DEVICE read method.

For more detailed information, refer to *IOPCAsyncIO2::Read* in the specification *OPC Data Access Custom Interface*.

Example

```

Private Sub AsyncReadButton_Click()
    Dim NumItems As Long
    Dim ServerIndex As Long
    Dim ServerHandles(10) As Long
    Dim Values() As Variant
    Dim Errors() As Long
    Dim ClientTransactionID As Long
    Dim ServerTransactionID As Long
    Dim Qualities() As Variant
    Dim TimeStamps() As Variant
    NumItems = 10
    ClientTransactionID = 1975
    For ServerIndex = 1 to NumItems
        ' specifies which item will be read
        ServerHandles(ServerIndex) = AnOPCItemServerHandles(ServerIndex)
    Next ServerIndex
    OneGroup.AsyncRead NumItems, ServerHandles, Errors,
                      ClientTransactionID, ServerTransactionID
End Sub

```

7.4.2.4 AsyncWrite

Description

This method writes values for one or more items in a group. The results are returned with the *AsyncWriteComplete* event assigned to the OPCGroup object.

Syntax

```
AsyncWrite (NumItems As Long,
           ServerHandles() As Long,
           Values() As Variant,
           ByRef Errors() As Long,
           TransactionID As Long,
           ByRef CancelID As Long)
```

NumItems

NumItems specifies the number of items to be written.

ServerHandles

ServerHandles is an array of server handles for the items to be written.

Values

Values is an array of values.

Errors

Errors is an array of long values that specifies the status of the item to be written.

TransactionID

TransactionID is the transaction ID specified by the client. This ID is contained in the relevant event.

CancelID

CancelID is a transaction ID generated by the server with which the client can cancel the transaction.

Remarks

The *AsyncWrite* method assumes that the object of the OPCGroup class was dimensioned with events (Dim WithEvents xxx As OPCGroup) so that the result of the *AsyncWrite* operation can be returned to the automation client application. The *AsyncWriteComplete* event that is assigned to the OPCGroup object is triggered by the automation server with the results from the *AsyncWrite* operation.

For more detailed information, refer to *IOPCAsyncIO2::Write* in the specification *OPC Data Access Custom Interface*.

Example

```

Private Sub AsyncWriteButton_Click()
Dim NumItems As Long
Dim ServerIndex As Long
Dim ServerHandles(10) As Long
Dim Values() As Variant
Dim Errors() As Long
Dim ClientTransactionID As Long
Dim ServerTransactionID As Long
NumItems = 10
For ServerIndex = 1 to NumItems
    ClientTransactionID = 1957
    ' specifies which item will be written
    ServerHandles(ServerIndex) = AnOPCItemServerHandles(ServerIndex)
    Values(ServerIndex) = ServerIndex * 2
    ' the value of the item can be any value here
Next ServerIndex
OneGroup.AsyncWrite NumItems, ServerHandles, Values, Errors,
                    ClientTransactionID, ServerTransactionID
End Sub

```

7.4.2.5 AsyncRefresh

Description

This method creates an event for all active items of a group (indicates whether or not they have changed). Inactive items are not executed. The result is output both with the *DataChange* event assigned to the OPCGroup object and with the *GlobalDataChange* event that is assigned to the OPCGroups object.

Syntax

```

AsyncRefresh (Source As Integer,
             TransactionID As Long,
             ByRef CancelID As Long)

```

Source

Source is the data source; OPC_DS_CACHE or OPC_DS_DEVICE

TransactionID

TransactionID is the transaction ID specified by the client. This ID is contained in the relevant event.

CancelID

CancelID is a transaction ID generated by the server with which the client can cancel the transaction.

Remarks

The *AsyncRefresh* method assumes that the object of the OPCGroup class was dimensioned with events (Dim WithEvents xxx As OPCGroup) so that the result of the *AsyncRefresh* operation is returned to the automation client application. The *DataChange* event that is assigned to the OPCGroup object is triggered by the automation server with the results from the *AsyncRefresh* operation.

If the client application declares the object of the OPCGroups class with the *WithEvents* supplement (Dim WithEvents xyz As OPCGroups), the *GlobalDataChange* event for this group is triggered with the results of the data update.

For more detailed information, refer to *IOPCAsyncIO2::Refresh* in the specification *OPC Data Access Custom Interface*.

Note

The call of the *Async2RefreshDevice* function of the OPC automation interface returns all items, even "write-only".

Example

```
Dim MyGroups As OPCGroups
Dim DefaultGroupUpdateRate As Long
Dim WithEvents OneGroup As OPCGroup
Private Sub AsyncRefreshButton_Click()
    Dim ServerIndex As Long
    Dim Source As Long
    Dim ClientTransactionID As Long
    Dim ServerTransactionID As Long
    ClientTransactionID = 2125
    Source = OPC_DS_DEVICE
    OneGroup.AsyncRefresh Source, ClientTransactionID
                           ServerTransactionID
End Sub
```

7.4.2.6 AsyncCancel

Description

With this method, the server is requested to cancel all unfinished transactions. The *AsyncCancelComplete* event indicates whether the cancel took place.

Syntax

```
AsyncCancel (CancelID As Long)
```

CancelID

CancelID is a CancelID created by the server that was returned by one of the methods *AsyncRead*, *AsyncWrite* or *AsyncRefresh*.

Remarks

The *AsyncCancel*/method assumes that the object of the OPCGroup class was declared with events (Dim WithEvents xxx As OPCGroup) so that the result of the *AsyncCancel*/operation is returned to the automation client application. The *AsyncCancelComplete* event that is assigned to the OPCGroup object is triggered by the automation server with the results from the *AsyncCancel*/operation. The TransactionID specified by the client is returned to the automation client application with the *AsyncCancelComplete* event.

For more detailed information, refer to *IOPCAsyncIO2::Cancel*/in the specification *OPC Data Access Custom Interface*.

Example

```
Private Sub AsyncCancelButton_Click()
    Dim ServerIndex As Long
    Dim CancelID As Long
    CancelID = 1 ' a transaction ID
    ' from one of the "Async" calls
    ' such as AsyncRead, AsyncWrite or AsyncRefresh
    OneGroup.AsyncCancel CancelID
End Sub
```

7.4.3 Events of the OPCGroup object

- DataChange
- AsyncReadComplete
- AsyncWriteComplete
- AsyncCancelComplete

7.4.3.1 DataChange

Description

This event can be triggered when the value or the quality of a value of an item has changed within a group. The event is not triggered faster than specified by the update rate of the group. This means that the values of items are buffered by the server until the current time + update rate are greater than the time of the previous update. This behavior also depends on whether or not the group and the items are active. Before values can be sent to the client in an event, both the items and the corresponding group must be active.

Syntax

```
DataChange (TransactionID As Long,
            NumItems As Long,
            ClientHandles() As Long,
            ItemValues() As Variant,
            Qualities() As Long,
            TimeStamps() As Date)
```

TransactionID

TransactionID is the transaction ID specified by the client. If a value that is not equal to 0 is output, the call was generated as a result of the *AsyncRefresh* method. If the value is equal to 0, the call was generated as the result of normal event handling.

NumItems

NumItems specifies the number of items output.

ClientHandles

ClientHandles is an array of client handles for the items.

Item Values

ItemValues is an array with values.

Qualities

Qualities is an array of the quality of the value of each item.

TimeStamps

TimeStamps is an array with UTC timestamps for the value of each item. If the device cannot return a timestamp, this is handled by the server.

Remarks

If the value of the item changes faster than the update rate, only the last current value for each item is buffered and returned to the client in the event.

Example

```
Dim WithEvents AnOPCGroup As OPCGroup
Private Sub AnOPCGroup_DataChange (TransactionID As Long,
                                  NumItems As Long, ClientHandles() As Long,
                                  ItemValues() As Variant, Qualities() As Long,
                                  TimeStamps() As Date)
    ' insert your program code here,
    ' to process the changed data
End Sub
```

7.4.3.2 AsyncReadComplete**Description**

This event is triggered when the *AsyncRead* method is completed.

Syntax

```
AsyncReadComplete (TransactionID As Long,
                  NumItems As Long,
                  ClientHandles() As Long,
                  ItemValues() As Variant,
                  Qualities() As Long,
                  TimeStamps() As Date,
                  Errors() As Long)
```

TransactionID

TransactionID is the transaction ID specified by the client.

NumItems

NumItems specifies the number of items output.

ClientHandles

ClientHandles is an array of client handles for the items.

ItemValues

ItemValues is an array with values.

Qualities

Qualities is an array of the quality of the value of each item.

TimeStamps

TimeStamps is an array with UTC timestamps for the value of each item. If the device does not return a timestamp, this is handled by the server.

Errors

Errors is an array of long values that specify whether the relevant items were read successfully, in other words, whether a defined value, a quality and a timestamp were obtained using the *Read* method. If *FAILED* is output in the area code, the values, quality and timestamp are not defined (*UNDEFINED*)!

Example

```
Dim WithEvents AnOPCGroup As OPCGroup
Private Sub AnOPCGroup_AsyncReadComplete (
    TransactionID As Long,
    NumItems As Long,
    ClientHandles() As Long,
    ItemValues() As Variant,
    Qualities() As Long,
    TimeStamps() As Date)
    ' insert your program code here,
    ' to process the changed data
End Sub
```

7.4.3.3 AsyncWriteComplete**Description**

This event is triggered when the *AsyncWrite* method is completed.

Syntax

```
AsyncWriteComplete (TransactionID As Long,
    NumItems As Long,
    ClientHandles() As Long,
    Errors() As Long)
```

TransactionID

TransactionID is the transaction ID specified by the client.

NumItems

NumItems specifies the number of items output.

ClientHandles

ClientHandles is an array of client handles for the items.

Errors

Errors is an array of long values that specify whether the items were written successfully.

Example

```
Dim WithEvents AnOPCGroup As OPCGroup
Private Sub AnOPCGroup_AsyncWriteComplete (TransactionID As Long,
    NumItems As Long, ClientHandles() As Long, ItemValues() As Variant,
    Qualities() As Long, TimeStamps() As Date)
    ' insert your program code here,
    ' to process the errors
End Sub
```

7.4.3.4 AsyncCancelComplete**Description**

This event is triggered when the AsyncCancel method is completed.

Syntax

```
AsyncCancelComplete (TransactionID As Long)
```

TransactionID

TransactionID is the transaction ID specified by the client.

Example

```
Dim WithEvents AnOPCGroup As OPCGroup
Private Sub AnOPCGroup_AsyncCancelComplete(TransactionID As Long)
    ' Enter your program code here ,
    ' that will be executed after the operation is canceled
End Sub
```

7.5 The collection object OPCItems

Description

This collection object has properties for the default values for objects of the OPCItem class. If such an item is added, the properties of the item are given these default values (DefaultXXXX) as their initial status. You can change these defaults if you want to add items with a different initial status. It is, of course, also possible to change the properties of an item after adding it. This reduces the number of parameters necessary to call the *AddItems* method.

Syntax

OPCItems

Example

The following code is required to allow the following VB examples to work:

```
Dim AnOPCServer As OPCServer
Dim ARealOPCServer As String
Dim ARealOPCNodeName As String
Dim AnOPCServerBrowser As OPCBrowser
Dim MyGroups As OPCGroups
Dim DefaultGroupUpdateRate As Long
Dim OneGroup As OPCGroup
Dim AnOPCItemCollection As OPCItems
Dim AnOPCItem As OPCItem
Dim ClientHandles(100) As Long
Dim AnOPCItemIDs(100) As String
Dim AnOPCItemServerHandles(10) As Long
Dim AnOPCItemServerErrorErrors() As Long
Set AnOPCServer = New OPCServer
ARealOPCServer = "Vendorname.DataAccessCustomServer"
ARealOPCNodeName = "OneComputername"
AnOPCServer.Connect(ARealOPCServer, ARealOPCNodeName)
Set MyGroups = AnOPCServer.OPCGroups
MyGroups.DefaultGroupIsActive = True
Set OneGroup = MyGroups.Add("OneOPCGroupName")
Set AnOPCItemCollection = OneGroup.OPCItems
```

7.5.1

Properties of the OPCItems collection object

- Parent
- DefaultRequestedDataType
- DefaultAccessPath
- DefaultIsActive
- Count

7.5.1.1 Parent

Description

(Read-only) This property returns the reference to the higher-level OPCGroup object.

Syntax

```
Parent As OPCGroup
```

7.5.1.2 DefaultRequestedDataType

Description

(Read and write) *DefaultRequestedDataType* is the requested data type that is used in *Add* calls (*AddItem*, *AddItems*); in other words, the initial value for the *RequestedDataType* property of newly added items. The default setting for this property is VT_EMPTY (in other words, the server sends data in its own data format).

Syntax

```
DefaultRequestedDataType As Integer
```

Remarks

Each valid VARIANT type can be specified as a requested data type.

For more detailed information on errors and exceptions, refer to the Automation Interface referenced in the appendix.

Example

```
' Example of VB syntax (getting the property):
Dim CurrentValue As Integer
Dim SomeValue As Integer
CurrentValue = AnOPCItemCollection.DefaultRequestedDataType

' Example of VB syntax (specifying the property):
AnOPCItemCollection.DefaultRequestedDataType = SomeValue
```

7.5.1.3 DefaultAccessPath

Description

(Read and write) *DefaultAccessPath* is the default access path used in *Add* calls (*AddItem*, *AddItems*). The default setting for this property is "" (an empty string).

Syntax

```
DefaultAccessPath As String
```

Example

```
' Example of VB syntax (getting the property):  
Dim CurrentValue As String  
Dim SomeValue As String  
CurrentValue = AnOPCItemCollection.DefaultAccessPath  
  
' Example of VB syntax (specifying the property):  
AnOPCItemCollection.DefaultAccessPath = SomeValue
```

7.5.1.4 DefaultIsActive

Description

(Read and write) This property sets the initial value for the *ActiveState* property of newly added items. The default for this property is *True*.

Syntax

```
DefaultIsActive As Boolean
```

Example

```
' Example of VB syntax (getting the property):  
Dim CurrentValue As Boolean  
Dim SomeValue As Boolean  
CurrentValue = AnOPCItemCollection.DefaultIsActive  
  
' Example of VB syntax (specifying the property):  
AnOPCItemCollection.DefaultIsActive = SomeValue
```

7.5.1.5 Count

Description

(Read-only) This property is required for collection objects.

Syntax

```
Count As Long
```

Example

```
' Example of VB syntax (getting the property):
Dim CurrentValue As Long
Dim SomeValue As Long
CurrentValue = AnOPCItemCollection.Count
```

7.5.2 Methods of the collection object OPCItems

Item
 GetOPCItem
 AddItem
 AddItems
 Remove
 Validate
 SetActive
 SetClientHandles
 SetDataTypes

7.5.2.1 Item

Description

This method is required for collection objects.

Syntax

`Item (ItemSpecifier As Variant) As OPCItem`

ItemSpecifier

ItemSpecifier is the index beginning with 1 within the collection and it returns an object of the OPCItem class using an ItemSpecifier.

Remarks

Item returns a reference to the item of the collection described by the *ItemSpecifier* index. The ItemSpecifier is the index (beginning with 1) for the collection. *GetOPCItem* references an item using a ServerHandle.

The automation property *Item* is often confused with the *OPCItem* object. The automation property *Item* is a special, reserved property that is used by automation collections to reference the contained items. The *OPCItem* object is an object type specific to OPC automation that can be contained in a collection of OPC items.

7.5.2.2 GetOPCItem

Description

This method returns the OPCItem that belongs to a server handle. The server handle is provided by an *Add* method (*AddItem*, *AddItems*). With the *Item* property, referencing is according to index.

Syntax

```
GetOPCItem (ServerHandle As Long) As OPCItem
```

ServerHandle

This is the ServerHandle for objects of the OPCItem class. With */item*, referencing is according to index.

Example

```
Dim AnOPCItem as OPCItem  
Set OPCItem = GetOPCItem(SomeItemServerHandle)
```

7.5.2.3 AddItem

Description

This method creates a new object of the OPCItem class and adds it to the collection. The properties of this new object are decided by the current defaults in the OPCItems collection object. If an object of the OPCItem class is added, its properties can be changed.

Syntax

```
AddItem (ItemID As String,  
ClientHandle As Long)
```

ItemID

/itemID is the full ItemID.

ClientHandle

This is the ClientHandle of the corresponding group.

Remarks

With this property, only one item at a time is added to the collection. If you want to add several items at one time, you should use the *AddItems* method.

For more detailed information on errors and exceptions, refer to the Automation Interface referenced in the appendix.

Example

```
Dim AnOPCItemID as String
Dim AnClientHandle as Long
AnOPCItemID = "N7:0"
AnClientHandle = 1975
AnOPCItemCollection.AddItem AnOPCItemID AnClientHandle
```

7.5.2.4 AddItems

Description

This method creates new objects of the OPCItem class and adds them to the collection. The properties of this new object are decided by the current defaults in the OPCItems collection object. If an object of the OPCItem class is added, its properties can be changed.

Syntax

```
AddItems (Count As Long,
          ItemIDs() As String,
          ClientHandles() As Long,
          ByRef ServerHandles() As Long,
          ByRef Errors() As Long,
          Optional RequestedDataTypes As Variant,
          Optional AccessPaths As Variant)
```

Count

Count indicates the number of items involved.

ItemIDs

ItemIDs contains an array with full ItemIDs.

ClientHandles

ClientHandles is an array of the client handles for the processed items.

ServerHandles

ServerHandles is an array of server handles of the processed items.

Errors

Errors is an array of long values that indicate the success of each individual item operation.

RequestedDataTypes

RequestedDataTypes is an optional variable of the type VARIANT that contains an integer array for the requested data types.

AccessPaths

AccessPaths is an optional parameter of the type VARIANT that contains an array of strings.

Remarks

For more detailed information on errors and exceptions, refer to the Automation Interface referenced in the appendix.

Example

```

Dim AddItemCount as long
Dim AnOPCItemIDs() as String
Dim AnOPCItemServerHandles as long
Dim AnOPCItemServerErrors as long
Dim AnOPCRequestedDataTypes as variant
Dim AnOPCAccessPathss as variant
For x = 1 To AddItemCount
    ClientHandles(x) = x + 1
    AnOPCItemID(x) = "Register_" & x
Next x
AnOPCItemCollection.AddItems AddItemCount,
                               AnOPCItemIDs,
                               ClientHandles,
                               AnOPCItemServerHandles,
                               AnOPCItemServerErrors,
                               AnOPCRequestedDataTypes,
                               AnOPCAccessPaths
' Code for the error handling
' individual errors are listed in the "Errors" array

```

7.5.2.5 Remove

Description

This method removes an object of the OPCItem class.

Syntax

```
Remove (Count As Long,
        ServerHandles() As Long,
        ByRef Errors() As Long)
```

Count

Count specifies the number of items to be removed.

ServerHandles

This is an array of server handles of the items involved.

Errors

Errors is an array of long values that indicate the success of each individual item operation.

Example

```

AnOPCItemCollection.Remove AnOPCItemServerHandles,
                           AnOPCItemServerErrors
' Code for the error handling
' individual errors are listed in the "Errors" array

```

7.5.2.6 Validate

Description

This method checks whether it was possible to create an object or several objects of the OPCServer class successfully with an *Add* method (*AddItem*, *AddItems*) but does not add any objects.

Syntax

```
Validate (Count As Long,  
          ItemIDs() As String,  
          ByRef Errors() As Long,  
          Optional RequestedDataTypes As Variant,  
          Optional AccessPaths As Variant)
```

Count

Count indicates the number of items involved.

ItemIDs

ItemIDs is an array with full ItemIDs.

Errors

Errors is an array of long values that indicate the success of each individual item operation.

RequestedDataTypes

RequestedDataTypes is a variant that contains an integer array of the requested data types.

AccessPaths

This is a variant that contains an array of strings of AccessPaths.

Remarks

For more detailed information on errors and exceptions, refer to the Automation Interface referenced in the appendix.

Example

```

Dim AddItemCount as long
Dim AnOPCItemIDs() as String
Dim AnOPCItemServerHandles as long
Dim AnOPCItemServerErrors as long
Dim AnOPCRequestedDataTypes as variant
Dim AnOPCAccessPathss as variant
For x = 1 To AddItemCount
    ClientHandles(x) = x + 1
    AnOPCItemID(x) = "Register_" & x
Next x
AnOPCItemCollection.Validate AddItemCount,
                               AnOPCItemIDs,
                               AnOPCItemServerErrors,
                               AnOPCRequestedDataTypes,
                               AnOPCAccessPaths
' Code for the error handling
' individual errors are listed in the "Errors" array

```

7.5.2.7 SetActive

Description

Using this method, individual objects of the OPCItem class can be activated or deactivated in an OPCItems collection.

Syntax

```

SetActive (Count As Long,
           ServerHandles() As Long,
           ActiveState As Boolean,
           ByRef Errors() As Long)

```

Count

Count indicates the number of items involved.

ServerHandles

ServerHandles is an array of server handles of the items involved.

ActiveState

ActiveState outputs the value *True* if the item should be active, *False* if the item should be inactive.

Errors

An array of long values that indicate the success of each individual item operation.

Remarks

For more detailed information on errors and exceptions, refer to the Automation Interface referenced in the appendix.

Example

```

' Activate items (assign corresponding
' property the value TRUE)
AnOPCItemCollection.SetActive ItemCount,
                           AnOPCItemServerHandles,
                           TRUE,
                           AnOPCItemServerErrors

' Code for the error handling
' individual errors are listed in the "Errors" array

```

7.5.2.8 SetClientHandles

Description

Changes the ClientHandles of one or more items of a group.

Syntax

```
SetClientHandles (Count As Long,
                  ServerHandles() As Long,
                  ClientHandles() (As Long,
                  ByRef Errors() As Long)
```

Count

Count indicates the number of items involved.

ServerHandles

ServerHandles is an array of server handles of the processed items.

ClientHandles

ClientHandles is an array of new client handles to be stored. The client handles do not need to be unique.

Errors

Errors is an array of long values that indicate the success of each individual item operation.

Example

```

For x = 1 To ItemCount
    ClientHandles(x) = x + 1975
Next x
AnOPCItemCollection.SetClientHandles ItemCount,
                                      AnOPCItemServerHandles,
                                      ClientHandles,
                                      AnOPCItemServerErrors

```

7.5.2.9 SetDataTypes

Description

This method changes the required data type for one or more items.

Syntax

```
SetDataTypes (Count As Long,
              ServerHandles() As Long,
              RequestedDataTypes() As Long,
              ByRef Errors() As Long)
```

Count

Count indicates the number of items involved.

ServerHandles

ServerHandles is an array of server handles of the processed items.

RequestedDataTypes

RequestedDataTypes is an array containing the newly specified data types to be stored.

Errors

Errors is an array of long values that indicate the success of each individual item operation.

Remarks

For more detailed information on errors and exceptions, refer to the Automation Interface referenced in the appendix.

Example

```
Dim RequestedDataTypes(100) As Long
For x = 1 To ItemCount
    RequestedDataTypes (x) = "a VB integer"
Next x
AnOPCItemCollection.SetDataTypes ItemCount,
                                  AnOPCItemServerHandles,
                                  RequestedDataTypes,
                                  AnOPCItemServerErrors
```

7.6 The OPCItem object

An object of the OPCItem class represents a connection to data sources within the server. A value, a quality and a timestamp belong to each item. The value is specified as a variable of the type VARIANT and has a quality.

7.6.1 Properties of the OPCItem object

Parent
ClientHandle
ServerHandle
AccessPath
AccessRights
ItemID
IsActive
RequestedDataType
Value
Quality
TimeStamp
CanonicalDataType
EUType
EUInfo

7.6.1.1 Parent

Description

(Read-only) This property returns the reference to the higher-level OPCGroup object.

Syntax

```
Parent As OPCGroup
```

7.6.1.2 ClientHandle

Description

(Read and write) This property returns a value of the type long belonging to the OPCItem. Based on this property, the client recognizes the client as the target of the data. A ClientHandle is typically an index or similar, It is returned to the client when data or status changes occur along with events of the OPCGroup object.

Syntax

```
ClientHandle As Long
```

Example

```
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(EinItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As Long
Dim SomeValue As Long
CurrentValue = AnOPCItem.ClientHandle

' Example of VB syntax (specifying the property):
AnOPCItem.ClientHandle = SomeValue
```

7.6.1.3 ServerHandle

Description

(Read-only) This is the handle assigned to the OPCItem by the server; and OPCItem can be identified uniquely with this. The client must make this handle available to several methods when they operate with the objects of the OPCItem class (for example *OPCItems.Remove*).

Syntax

```
ServerHandle As Long
```

Example

```
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As Long
Dim SomeValue As Long
CurrentValue = AnOPCItem.ServerHandle
```

7.6.1.4 AccessPath

Description

(Read-only) *AccessPath* provides the access path that the client specify with an *Add* call (*AddItem*, *AddItems*).

Syntax

```
AccessPath As String
```

Example

```

Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As String
Dim SomeValue As String
CurrentValue = AnOPCItem.AccessPath

```

7.6.1.5 AccessRights

Description

(Read-only) *AccessRights* outputs the access rights of the item.

Syntax

```
AccessRights As Long
```

Remarks

This property indicates whether or not read-only or write-only or both read and write permissions exist for the corresponding item.

Example

```

Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As Long
Dim SomeValue As Long
CurrentValue = AnOPCItem.AccessRights

```

7.6.1.6 ItemID

Description

(Read-only) *ItemID* is the unique identification of the item.

Syntax

```
ItemID As String
```

Example

```
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(EinItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As String
Dim SomeValue As String
CurrentValue = AnOPCItem.ItemID
```

7.6.1.7 IsActive

Description

(Read and write) This property specifies whether or not notification events are generated for this item.

Syntax

```
IsActive As Boolean
```

Remarks

If the item is active, the value *True* is output, if the value is inactive, the *False* is output.

Example

```
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As Boolean
Dim SomeValue As Boolean
CurrentValue = AnOPCItem.IsActive

' Example of VB syntax (specifying the property):
AnOPCItem.IsActive = SomeValue
```

7.6.1.8 RequestedDataType

Description

(Read and write) *RequestedDataType* is the data type in which the value of the item is output.

Note: If the requested data type was rejected, the item becomes invalid until the requested data type has a valid value.

Syntax

```
RequestedDataType As Integer
```

Remarks

For more detailed information on errors and exceptions, refer to the Automation Interface referenced in the appendix.

Example

```
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As Integer
Dim SomeValue As Integer
CurrentValue = AnOPCItem.RequestedDataType

' Example of VB syntax (specifying the property):
AnOPCItem.RequestedDataType = SomeValue
```

7.6.1.9 Value

Description

(Read-only) This property outputs the last value read from the server. This is the default property of *OPCItem*.

Syntax

```
Value As Variant
```

Example

```
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As Variant
Dim SomeValue As Variant
CurrentValue = AnOPCItem.Value
```

7.6.1.10 Quality

Description

(Read-only) This property outputs the last value for quality read from the server.

Syntax

```
Quality As Long
```

Example

```
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As Long
Dim SomeValue As Long
CurrentValue = AnOPCItem.Quality
```

7.6.1.11 TimeStamp

Description

(Read-only) This property outputs the last value for timestamp read from the server.

Syntax

```
TimeStamp As Date
```

Example

```
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(EinItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As Date
Dim SomeValue As Date
CurrentValue = AnOPCItem.TimeStamp
```

7.6.1.12 CanonicalDataType

Description

(Read-only) This property outputs the original data type of the item. This value may differ from the requested data type (RequestedDataType).

Syntax

```
CanonicalDataType As Integer
```

Remarks

For more detailed information on errors and exceptions, refer to the Automation Interface referenced in the appendix.

Example

```
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As Integer
Dim SomeValue As Integer
CurrentValue = AnOPCItem.CanonicalDataType
```

7.6.1.13 EUType

Description

(Read-only) The *EUType* property specifies the type of EU information included in the EUInfo property (EU=Engineering Unit).

Syntax

```
EUType As Integer
```

Remarks

0 means that no EU information is available (the EUInfo property has the value VT_EMPTY).

1 means *Analog*, the EU information contains a SAFEARRAY of exactly two variants of the type double (VT_ARRAY | VT_R8), that correspond to the minimum and maximum value.

2 means *Enumeration*; the EU information contains a SAFEARRAY of strings (VT_ARRAY | VT_BSTR) that contains a list of strings (for example *OPEN*, *CLOSE*, *IN TRANSIT* etc.) that correspond to consecutive numeric values (0, 1, 2 etc.).

For more information, refer to the specification *OPC Data Access Custom Interface* in *OPCItem Attributes*.

Example

```
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As Integer
Dim SomeValue As Integer
CurrentValue = AnOPCItem.EUType
```

7.6.1.14 EUInfo

Description

(Read-only) *EUInfo* contains a variable with information on the unit (EU = Engineering Unit).

Syntax

```
EUInfo As Variant
```

Remarks

For more information, refer to the specification *OPC Data Access Custom Interface* in *OPCItem Attributes*.

Example

```
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)

' Example of VB syntax (getting the property):
Dim CurrentValue As Variant
Dim SomeValue As Variant
CurrentValue = AnOPCItem.EUInfo
```

7.6.2 Methods of the OPCItem object

Read
Write

7.6.2.1 Read

Description

An item can be read from the server with this method. *Read* can only be called with one source (either OPCCache or OPCDevice) to update values, quality and timestamp of the item. If the value, quality and timestamp need to be synchronized, the optional parameters of this method return values that were updated at the same time.

Syntax

```
Read (Source As Integer,
      Optional ByRef Value As Variant,
      Optional ByRef Quality As Variant,
      Optional ByRef TimeStamp As Variant)
```

Source

Source means the data source; i.e. OPC_DS_CACHE or OPC_DS_DEVICE

Value

Value outputs the last value read from the server.

Quality

Quality outputs the last value for the quality read from the server.

Timestamp

Timestamp outputs the last timestamp read from the server.

Example

```
Private Sub ReadButton_Click()
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)
Dim Source As Integer
Dim Value As Variant
Dim Quality As Variant
Dim TimeStamp As Variant
Source = OPC_DS_DEVICE
AnOPCItem.Read Source,
               ServerHandles,
               Value,
               Quality,
               TimeStamp
' processing of the values
TextBox.Text = Value
End Sub
```

7.6.2.2 Write

Description

A value can be written in the OPC server with this method.

Syntax

```
Write (Value As Variant)
```

Value

Value contains the value to be written.

Example

```
Private Sub WriteButton_Click()
Dim AnOPCItem as OPCItem
Set OPCItem = GetOPCItem(OneItemServerHandle)
Dim Value As Variant
Value = 1975
AnOPCItem.Write Value
End Sub
```

7.7 Definitions

Here, you will find definitions of constants relating to the state of the server and explanations of error messages.

7.7.1 State of the server

OPCRunning

The server is operating correctly, this is the normal server state.

OPCFailed

A vendor-specific error has occurred on the server, the server is no longer operating correctly. How to eliminate the problem in this case depends on the vendor. Normally, the error code *E_FAIL* is output by one of the other server methods.

OPCNoconfig

The server is working but has not loaded any information on the configuration; in other words, the server is not operating correctly.

Servers that do not require this type of information do not output this status message.

OPCSuspended

Operation of the server was temporarily interrupted by a method specific to a vendor, in other words, the server is not sending or receiving data. The information on the quality is as follows: OPC_QUALITY_OUT_OF_SERVICE.

OPCTest

The server is in test mode. Data output is disconnected from the hardware, otherwise the server responds normally. Depending on the vendor implementation, data input is based on values that either really exist or are simulated. Information on the quality is output normally.

OPCDisconnected

The automation server object is not connected to an OPC custom interface server.

7.7.2 Error messages

OPCInvalidHandle

0xC0040001L

The value of this handle is invalid. A client must not pass an invalid handle to a server. If this error occurs, there is a programming error on the client or possibly on the server.

OPCBadType

0xC0040004L

The server cannot convert the data between the required data type and the original data type.

OPCPublic

0xC0040005L

The required operation cannot be executed for a group declared as *public*.

OPCBadRights

0xC0040006L

The access rights of the item do not allow the desired operation.

OPCUknownItemID

0xC0040007L

The ItemID is not defined in the address space of the server or no longer exists there (for read and write operations).

OPCInvalidItemID

0xC0040008L

The ItemID does not match the syntax of the server.

OPCInvalidFilter

0xC0040009L

The string of the filter is invalid.

OPCUncorrectedPath

0xC004000AL

The access path of the item is unknown to the server.

OPCRage

0xC004000BL

The value is outside the valid range.

OPCDuplicateName

0xC004000CL

DuplicateName is invalid.

OPCUncorrectedRate

0x0004000DL

The server does not support the required data rate, however it uses a rate that is as close as possible.

OPCClamp

0x0004000EL

A write value was accepted but was not output.

OPCInuse

0x0004000FL

The operation cannot be executed because there is still a reference to the object.

OPCInvalidConfig

0xC0040010L

The file format of the server configuration is invalid.

OPCNotFound**0xC0040011L**

The required object (for example a *PublicGroup*) could not be found.

OPCInvalidPID**0xC0040203L**

The PropertyID of the item that was passed on is invalid.

7.8 Appendix on the Automation Interface reference

Error

If a runtime error occurs, the error is identified unequivocally by the *Err* object of Visual Basic.

If VB does not handle errors with the *On Err* mechanism, an exception is generated and, depending on the context (Visual Basic in debug mode or an executable program), a message box appears with the following information:

Runtime error: Error code in decimal numbers (in hexadecimal characters)

Method X of Object Y Failed (if you are working with an executable application, no values are specified for *X* and *Y*).

The OPC Foundation therefore strongly advises that the application takes suitable measures to detect OPC automation errors that can occur when specifying properties or when executing methods.

Error handler is a routine that detects errors in the application and reacts to them. An OPC automation client should provide an error handler for every application functionality if the application specifies properties or calls a method.

An error handler consists of the following three parts:

1. Creating or activating a branch for error situations that specifies where the application should jump to (which routine will be used for error handling) if an error occurs. The statement in *On Error* activates the branch and brings the application to the label that indicates the start of the routine.
2. Write an error handling routine that handles errors resulting from specifying properties or using methods.
3. Exiting the routine.

It must be clear which measures the application takes if an error occurs. If, for example, a group is being added whose name (specified by the user) is not unique, the user can be informed that the group was not added; the user can also be prompted to enter a different name. On the other hand, the application could also attempt to add the group again (with " ") and the name will be adopted as generated by the server.

Locating errors

In the event of an error, a reference that specifies an error handler is activated when VB executes an On Error statement. The branch remains active as long as the procedure containing this branch is itself active; in other words until *Exit Sub*, *Exit Function*, *Exit Property*, *End Sub*, *End Function* or *End Property* is run.

To create a branch that jumps to an error handling routine, use the *On Error Go To Line* statement, where *Line* is the label that identifies the code of the error handling.

Handling errors

A label is first set at the start of the routine for handling errors. The name of the label should be descriptive and must end with a colon.

The main part contains the code for error handling either in the form of *If(...Then...)Else(...)* loops or as a *Case* statement. Here, the measures to deal with the most common errors are executed.

The *Number* property of the *Err* object contains a numeric code that represents runtime errors that have occurred most often in the recent past. The error number of this property contains the value with which *GetErrorString* is called to convert the error number to a readable string.

Example

```
Dim AnOpcServer As OPCServer
Private Sub Command1_Click()
On Error GoTo testerror
Set AnOpcServer = New OPCServer
' if "fuzz" does not exist so that the connection
' cannot be established, there is a jump to the label
' "testerror"
AnOpcServer.Connect ("fuzz")
Time = AnOpcServer.CurrentTime
Debug.Print Time
testerror:
Debug.Print Err.Number
End Sub
```

Related literature

You want to read the specifications of the OPC Foundation or require additional information on a particular topic.

Your Area of Interest	is dealt with in:
You want more detailed information on a specific topic?	Other Available Literature
You would like detailed information on OPC Data Access and OPC Alarms & Events?	OPC Specifications

Finding the SIMATIC NET documentation

- **Catalogs**

You will find the order numbers for the Siemens products of relevance here in the following catalogs:

- SIMATIC NET Industrial Communication / Industrial Identification, catalog IK PI
- SIMATIC Products for Totally Integrated Automation and Micro Automation, catalog ST 70

You can request the catalogs and additional information from your Siemens representative.

You can go to the Industry Mall on the Internet at the following address:

<https://mall.industry.siemens.com>

- **Documentation on the Internet**

You will find SIMATIC NET manuals on the Internet pages of Siemens Automation Customer Support:

<http://support.automation.siemens.com>

Go to the required product group and make the following settings:

"Entry list" tab, Entry type "Manuals / Operating Instructions"

- **Documentation from the STEP 7 installation**

Manuals that are included in the online documentation of the STEP 7 installation on your PG/PC can be found in the start menu ("Start" > "All Programs" > "Siemens Automation" > "Documentation").

See also

SIMATIC NET manuals: (<http://support.automation.siemens.com/WW/view/en/10805878>)

SIMATIC documentation: (<http://www.siemens.com/simatic-docu>)

8.1 OPC Specifications

SIMATIC NET
PROFIBUS Network Manual
Siemens AG
(SIMATIC NET Manual Collection)

On the Internet under the following entry ID: 35222591
(<http://support.automation.siemens.com/WW/view/en/35222591>)

SIMATIC NET
Industrial Ethernet Network Manual
Siemens AG
(SIMATIC NET Manual Collection)

On the Internet under the following entry ID: 27069465
(<http://support.automation.siemens.com/WW/view/en/27069465>)

SIMATIC NET
Programming Interface DP Base for CP 5613/CP 5614
Siemens AG
(SIMATIC NET Manual Collection)

On the Internet under the following entry ID: 1653531
(<http://support.automation.siemens.com/WW/view/en/1653531>)

SIMATIC NET
IO Base User Interface
Siemens AG
(SIMATIC NET Manual Collection)

On the Internet under the following entry ID: 19779901
(<http://support.automation.siemens.com/WW/view/en/19779901>)

8.1 OPC Specifications

What OPC specifications are there?

The first and most fundamental specification of the OPC Foundation was the OPC Data Access Specification. This provides management of process variables and various ways of accessing these variables.

The OPC Alarms & Events Specification covers the transfer of process alarms and events. It is independent of the Data Access Specification.

Up to now, the OPC Foundation has published the following specifications also included on the SIMATIC NET Manual Collection:

- **OPC overview**

Version 1.0, October 27, 1998

A brief introduction to the basic concept of OPC

- **OPC Common Definition and Interfaces**

Version 1.0, October 27, 1998

This document describes important interfaces available on all OPC servers.

- **Data Access Custom Interface**

Version 1.1, September 11, 1997

The specification of the Custom Interface for Data Access, Version 1.1

- **Data Access Custom Interface**

Version 2.04, September 5, 2000

The specification of the Custom Interface for Data Access, Version 2.04

- **Data Access Custom Interface**

Version 2.05, December 17, 2001

The specification of the Custom Interface for Data Access, Version 2.05

- **Data Access Custom Interface**

Version 3.00, March 4, 2003

The specification of the Custom Interface for Data Access, Version 3.00

- **Data Access Automation Interface**

Version 2.02, February 4, 1999

The specification of the Automation Interface for Data Access, Version 2.02

- **OPC Alarms and Events Custom Interface**

Version 1.10, October 2, 2002

A description of the OPC Alarms & Events server as well as the specification of the Custom Interface of this server

- **Alarm & Events Automation Interface**

Version 1.01, December 15, 1999

The specification of the Automation Interface of the OPC Alarms & Events server

8.1 OPC Specifications

- **OPC XML-DA Specification**

Version 1.01, December 21, 2004

The specification of the OPC XML interface for Data Access

- **OPC Unified Architecture**

Version 1.0 or later

<http://www.opcfoundation.org/UA/>

Part 1 - Concepts

Part 2 - Security Model

Part 3 - Address Space Model

Part 4 - Services

Part 5 - Information Model

Part 6 - Service Mappings

Part 7 - Profiles

Part 8 - Data Access

Part 9 - Alarms and Conditions

Part 10 - Programs

Part 11 - Historical Access

Part 12 - Discovery

The specification of OPC UA.

Additional specifications for tasks of the automation will follow.