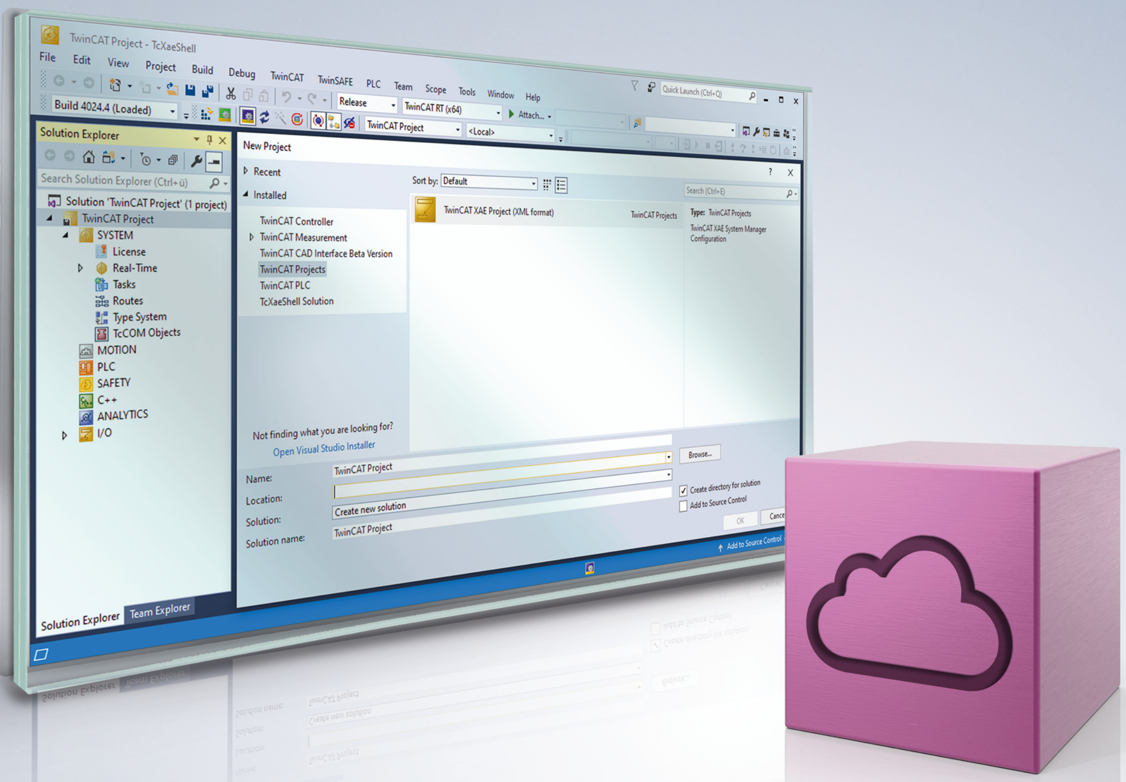
BECKHOFF New Automation Technology

Manual | EN

TF6770

TwinCAT 3 | IoT WebSockets



2024-4-15 | Version: 1.0.0

Table of contents

1 Foreword **5**

1.1 Notes on the documentation 5

1.2 For your safety 5

1.3 Notes on information security 7

2 Overview **8**

3 Installation **9**

3.1 System requirements 9

3.2 Installation 9

3.3 Licensing 9

4 Technical introduction **12**

4.1 WebSocket 12

4.2 Compression 12

4.3 Security 12

4.3.1 Transport layer 12

4.3.2 Application layer 19

5 PLC API **20**

5.1 Function blocks 20

5.1.1 FB_IotWebSocketClient 20

5.2 Data types 26

5.2.1 ETcIotWebSocketStatus 26

5.2.2 ETcIotWebSocketContentType 27

6 Samples **28**

7 Appendix **29**

7.1 ADS Return Codes 29

1 Foreword

1.1 Notes on the documentation

This description is intended exclusively for trained specialists in control and automation technology who are familiar with the applicable national standards.
For installation and commissioning of the components, it is absolutely necessary to observe the documentation and the following notes and explanations.
The qualified personnel is obliged to always use the currently valid documentation.

The responsible staff must ensure that the application or use of the products described satisfies all requirements for safety, including all the relevant laws, regulations, guidelines, and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without notice.
No claims to modify products that have already been supplied may be made on the basis of the data, diagrams, and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered and licensed trademarks of Beckhoff Automation GmbH.
If third parties make use of designations or trademarks used in this publication for their own purposes, this could infringe upon the rights of the owners of the said designations.

Patents

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
and similar applications and registrations in several other countries.

EtherCAT® is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.
The distribution and reproduction of this document as well as the use and communication of its contents without express authorization are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event that a patent, utility model, or design are registered.

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation, and drive technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER
Hazard with high risk of death or serious injury.

⚠ WARNING
Hazard with medium risk of death or serious injury.

⚠ CAUTION
There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE
The environment, equipment, or data may be damaged.

Information on handling the product

ℹ This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Overview

The function blocks of the PLC library Tc3_IotBase can be used to establish a WebSocket connection with a server as a client and exchange data.

Product components

The TF6770 IoT WebSockets function consists of the following components, which can be used from TwinCAT version 3.1.4026.x:

- **Driver:** TcIotDrivers.sys (included in the TwinCAT.Standard.XAR installation in the TwinCAT.XAR.DriversBase package)
- **PLC library:** Tc3_IotBase (included in the installation of TF6770.IotWebSockets.XAE)

3 Installation

3.1 System requirements

Technical data	Description
Operating system	Windows 7/10, Windows Embedded Standard 7, TwinCAT/BSD
Target platform	PC architecture (x86, x64 or ARM)
TwinCAT version	TwinCAT 3.1 Build 4026.3 or higher
Required TwinCAT setup level	TwinCAT 3 XAE, XAR
Required TwinCAT license	TF6770 TC3 IoT WebSockets

3.2 Installation

TwinCAT Package Manager

If you are using TwinCAT 3.1 Build 4026 (and higher) on the Microsoft Windows operating system, you can install this function via the TwinCAT Package Manager, see installation documentation.

Normally you install the function via the corresponding workload; however, you can also install the packages contained in the workload individually. This documentation briefly describes the installation process via the workload.

Command line program TcPkg

You can use the TcPkg **C**ommand **L**ine **I**nterface (CLI) to display the available workloads on the system:

```
tcpkg list -t workload
```

You can use the following command to install the workload of the TF6770 IoT Websockets function.

```
tcpkg install TF6770.IotWebSockets.XAE
```

TwinCAT Package Manager UI

You can use the **U**ser **I**nterface (UI) to display all available workloads and install them if required.
To do this, follow the corresponding instructions in the interface.

3.3 Licensing

The TwinCAT 3 function can be activated as a full version or as a 7-day test version. Both license types can be activated via the TwinCAT 3 development environment (XAE).

Licensing the full version of a TwinCAT 3 Function

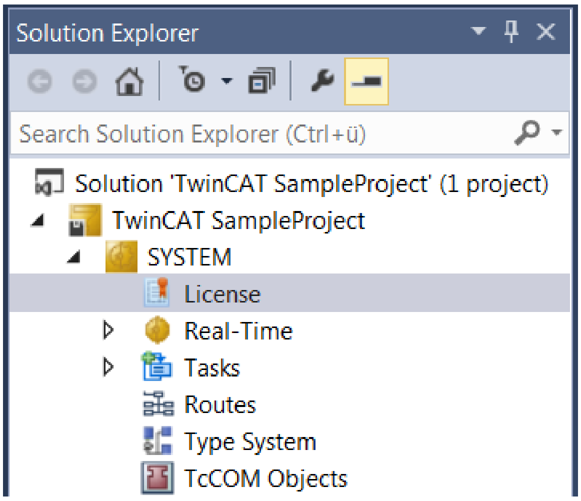
A description of the procedure to license a full version can be found in the Beckhoff Information System in the documentation "TwinCAT 3 Licensing".

Licensing the 7-day test version of a TwinCAT 3 Function

A 7-day test version cannot be enabled for a TwinCAT 3 license dongle.

1. Start the TwinCAT 3 development environment (XAE).
2. Open an existing TwinCAT 3 project or create a new project.
3. If you want to activate the license for a remote device, set the desired target system. To do this, select the target system from the **Choose Target System** drop-down list in the toolbar.
   ⇨ The licensing settings always refer to the selected target system. When the project is activated on the target system, the corresponding TwinCAT 3 licenses are automatically copied to this system.

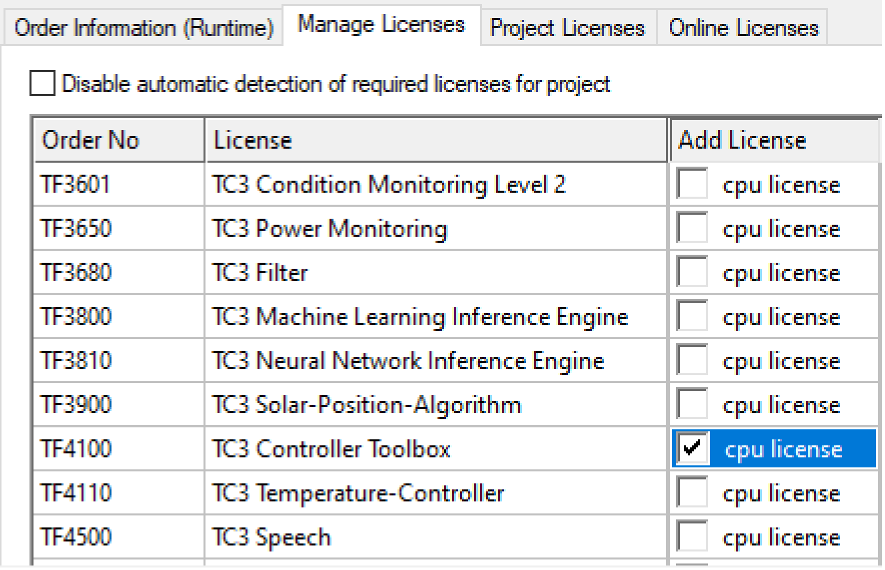
4. In the **Solution Explorer**, double-click **License** in the **SYSTEM** subtree.



- Solution 'TwinCAT SampleProject' (1 project)
  - TwinCAT SampleProject
    - SYSTEM
      - License
      - Real-Time
      - Tasks
      - Routes
      - Type System
      - TcCOM Objects

⇨ The TwinCAT 3 license manager opens.

5. Open the **Manage Licenses** tab. In the **Add License** column, check the check box for the license you want to add to your project (e.g. "TF4100 TC3 Controller Toolbox").



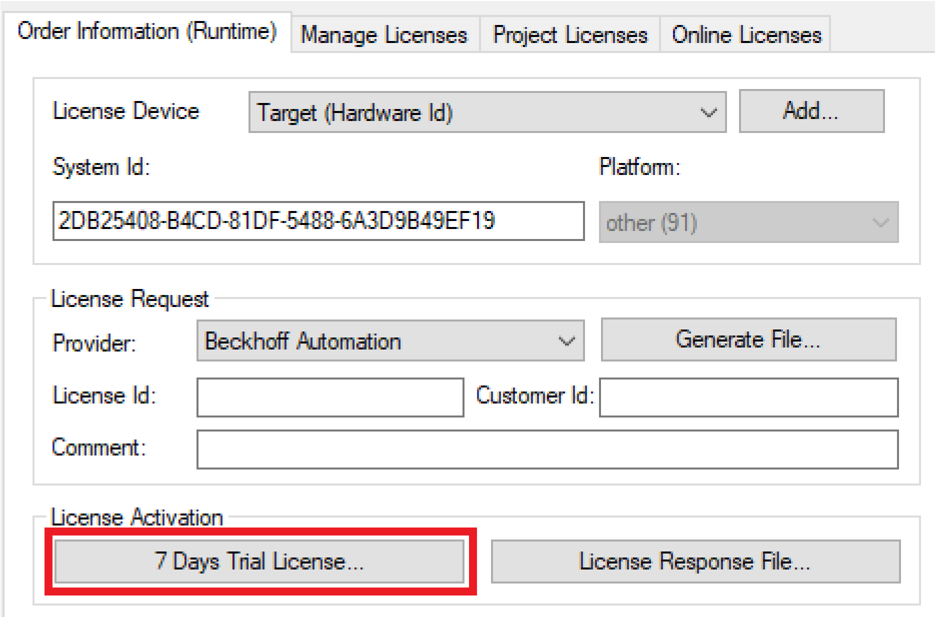
□ Disable automatic detection of required licenses for project

Order No	License	Add License
TF3601	TC3 Condition Monitoring Level 2	□ cpu license
TF3650	TC3 Power Monitoring	□ cpu license
TF3680	TC3 Filter	□ cpu license
TF3800	TC3 Machine Learning Inference Engine	□ cpu license
TF3810	TC3 Neural Network Inference Engine	□ cpu license
TF3900	TC3 Solar-Position-Algorithm	□ cpu license
TF4100	TC3 Controller Toolbox	☑ cpu license
TF4110	TC3 Temperature-Controller	□ cpu license
TF4500	TC3 Speech	□ cpu license

6. Open the **Order Information (Runtime)** tab.

⇨ In the tabular overview of licenses, the previously selected license is displayed with the status “missing”**.**

7. Click **7-Day Trial License...** to activate the 7-day trial license.



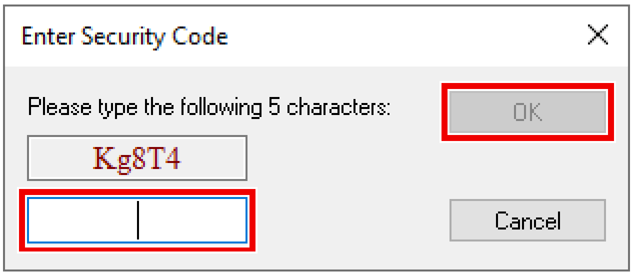
License Device: Target (Hardware Id)

System Id: 2DB25408-B4CD-81DF-5488-6A3D9B49EF19

Platform: other (91)

Provider: Beckhoff Automation

⇨ A dialog box opens, prompting you to enter the security code displayed in the dialog.



Please type the following 5 characters: Kg8T4

8. Enter the code exactly as it is displayed and confirm the entry.
9. Confirm the subsequent dialog, which indicates the successful activation.
   ⇨ In the tabular overview of licenses, the license status now indicates the expiry date of the license.
10. Restart the TwinCAT system.

⇨ The 7-day trial version is enabled.

4 Technical introduction

4.1 WebSocket

WebSocket is a TCP-based network protocol. In contrast to the stateless HTTP protocol, WebSocket enables a permanent bidirectional connection between server and client.

WebSocket Client
WebSocket
WebSocket Server

A WebSocket connection is created from a WebSocket handshake. In this WebSocket handshake, an HTTP GET request is first sent to the server, which contains information about an update of this connection. If the server supports WebSocket connections and accepts the request, an HTTP response with the status code 101 Switching Protocols is sent back to the client. Once the handshake is complete, the system switches from HTTP to WebSocket.

In contrast to the HTTP protocol, both the client and the server can send data to each other without prior request once a connection has been established. Both communication participants can also terminate the connection again.

The WebSocket protocol is used, for example, in chat applications, online games or live sports tickers. In the example of the live sports ticker, the server can communicate updates to the connected client without the client always having to send a request as with HTTP.

4.2 Compression

In general, the term "data compression" refers to the ability to reduce the number of bits needed to represent data. One way of dealing with this is to provide recurring strings with a reference to the first of these strings through a compression algorithm. Appropriate compression must occur without loss of information.

The RFC 7692 specification defines the "permessage-deflate" extension for the compression of WebSocket messages. The compression option can be enabled in FB_IotWebSocketClient [▸ 20] using the variable bPerMessageDeflate. Compression can then be enabled or disabled for each message using the SendMessage [▸ 25]() method.

4.3 Security

When considering protection of data communication, a distinction can be made between two levels: protection of the transport channel [▸ 12] and protection at application layer.

4.3.1 Transport layer

The worldwide common standard Transport Layer Security (TLS) is used in the TwinCAT IoT driver for the secure transmission of data. The following chapter describes the TLS communication flow, taking TLS version 1.2 as an example.

The TLS standard combines symmetric and asymmetric cryptography to protect transmitted data from unauthorized access and manipulation by third parties. In addition, TLS supports authentication of communication devices for mutual identity verification.

Contents of this chapter

The information in this chapter refers to the general TLS communication flow, without specific reference to the implementation in TwinCAT. They are only intended to provide a basic understanding in order to better comprehend the reference to the TwinCAT implementation explained in the following sub-chapters.

Supported functions

The TwinCAT IoT driver enables the use of the following TLS functions.

Function	Description
Self-signed client certificates	Use a self-signed client certificate to authenticate to the message broker.
CA-signed client certificates	Use of a CA-signed client certificate to authenticate to the message broker. The CA certificate can also be specified to establish a trust relationship.
Certificate revocation lists	Use of certificate revocation lists (CRL).
Pre-Shared Key (PSK)	Use of a pre-shared key (PSK) to authenticate to the message broker.

Cipher suite definition

A cipher suite is by definition a composition of algorithms (key exchange, authentication, encryption, MAC) for encryption. The client and server agree on these during the TLS connection establishment. For more information on cipher suites please refer to the relevant technical literature.

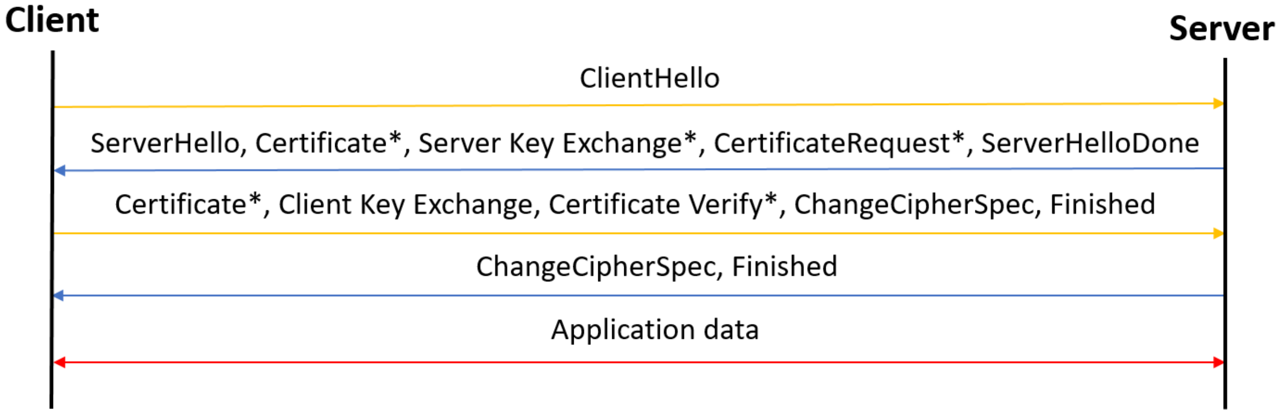
TLS communication flow

Communication with TLS encryption starts with a TLS handshake between server and client. During the handshake asymmetric cryptography is used; after successful completion of the handshake the server and client communicate based on symmetric cryptography, because this is many times faster than asymmetric cryptography.

There are three different types of authentication for the TLS protocol:

- The server identifies itself through a certificate (see Server certificate [▶ 15])
- The client and server identify themselves through a certificate (see Client/Server certificate [▶ 16])
- Pre-shared keys (see Pre-shared keys [▶ 16])

Please refer to the relevant technical literature for information about the advantages and disadvantages of the different authentication types.


Client
Server
ClientHello
ServerHello, Certificate*, Server Key Exchange*, CertificateRequest*, ServerHelloDone
Certificate*, Client Key Exchange, Certificate Verify*, ChangeCipherSpec, Finished
ChangeCipherSpec, Finished
Application data

Exemplary explanation based on RSA

All messages marked with * are optional, i.e. not mandatory. The following steps refer to the RSA procedure and are not generally valid for other procedures.

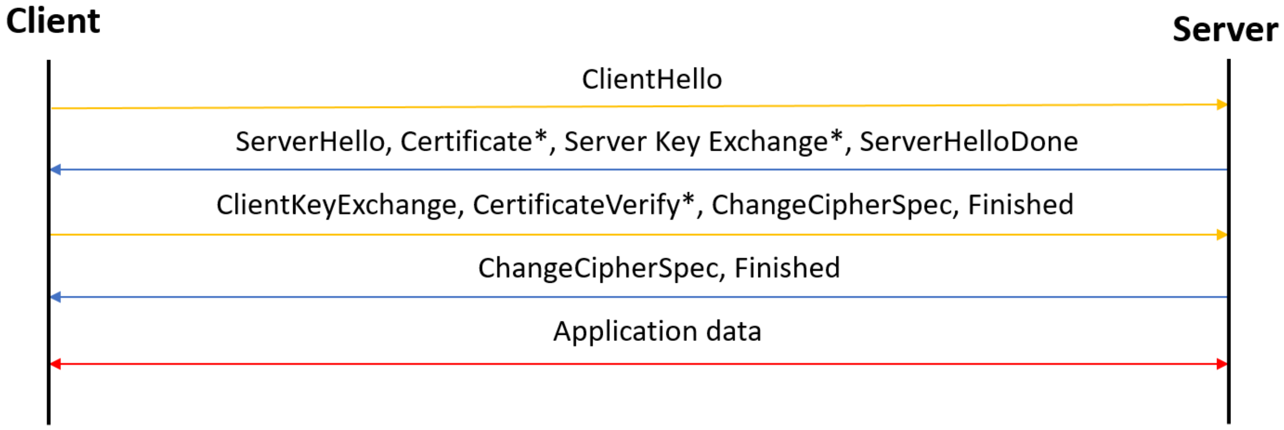
The following table explains the individual steps from the communication flow shown above.

Step	Description
ClientHello	The client initiates a connection to the server. The TLS version used, a random sequence of bytes (client random) and the cipher suites supported by the client are transmitted, among other parameters.
ServerHello	The server selects one of the cipher suites offered by the client and specifies it for the communication. If there is no intersection between the cipher suites supported by the client and server, the TLS connection establishment is aborted. In addition, the server also communicates a random sequence of bytes (server random).
Certificate	The server presents its certificate to the client to enable the client to verify that the server is the expected server. If the client does not trust the server certificate, the TLS connection establishment is aborted. The server certificate also contains the server's public key.
ServerKeyExchange	For certain key exchange algorithms, the information from the certificate is not sufficient for the client to generate the so-called pre-master secret. In this case the missing information is transferred using Server Key Exchange.
CertificateRequest	The server requests a certificate from the client to verify the identity of the client.
ServerHelloDone	The server notifies the client that sending of the initial information is complete.
Certificate	The client communicates its certificate, including the public key, to the server. The procedure is the same as in the opposite direction: If the server does not trust the certificate sent by the client, the connection establishment is aborted.
ClientKeyExchange	The client generates an encrypted pre-master secret and uses the server's public key to send the secret to the server using asymmetric encryption. This pre-master secret, the "server random" and the "client random" are then used to calculate the symmetric key that is used for communication after the connection has been established.
CertificateVerify	The client signs the previous handshake messages with its private key. Since the server has obtained the client's public key by sending the certificate, it can verify that the certificate presented really "belongs" to the client.
ChangeCipherSpec	The client notifies the server that it is switching to symmetric cryptography. From here on every message from the client to the server is signed and encrypted.
Finished	The client notifies the server in encrypted form that the TLS connection establishment on its side is complete. The message contains a hash and a MAC relating the previous handshake messages.

Step	Description
ChangeCipherSpec	The server decrypts the pre-master secret that the client encrypted with its public key. Since only the server has its private key, only the server can decrypt this pre-master secret. This ensures that the symmetric key is only known to the client and the server. The server then calculates the symmetric key from the pre-master secret and the two random sequences and notifies the client that it too is now communicating using symmetric cryptography. From here on every message from the server to the client is signed and encrypted. By generating the symmetric key, the server can decrypt the client's Finished message and verify both hash and MAC. If this verification fails, the connection is aborted.
Finished	The server notifies the client that the TLS connection establishment on its side is also finished. As with the client, the message contains a hash and a MAC relating to the previous handshake messages. On the client side, the same verification is then performed as on the server. Here too, if the hash and MAC are not successfully decrypted, the connection is aborted.
ApplicationData	Once the TLS connection establishment is complete, client and server communicate using symmetric cryptography.

4.3.1.1 Server certificate

This section covers a situation where the client wants to verify the server certificate but the server does not want to verify the client certificate. In this case the communication flow described in chapter Transport layer [▸ 12] is shortened as follows.


Client
Server
ClientHello
ServerHello, Certificate*, Server Key Exchange*, ServerHelloDone
ClientKeyExchange, CertificateVerify*, ChangeCipherSpec, Finished
ChangeCipherSpec, Finished
Application data

Verification of the server certificate

The server certificate is verified on the client side. A check is performed to ascertain whether it is signed by a particular certificate authority. If this is not the case, the client aborts the connection, since it does not trust the server.

Application in TwinCAT

In TwinCAT, the file path to the CA certificate (.PEM or .DER file) or the content of the .PEM file is specified as a string. The certificate presented by the server is then checked in the IoT driver. If the certificate chain is not signed by the specified CA, the connection to the server is aborted. The following code illustrates the described connection parameters as an example. The sample code refers to the HTTP client, the MQTT client and the WebSocket client. The HTTP client is used as an example.

```
PROGRAM MAIN
VAR
    fbClient : FB_IotHttpClient;
END_VAR
```

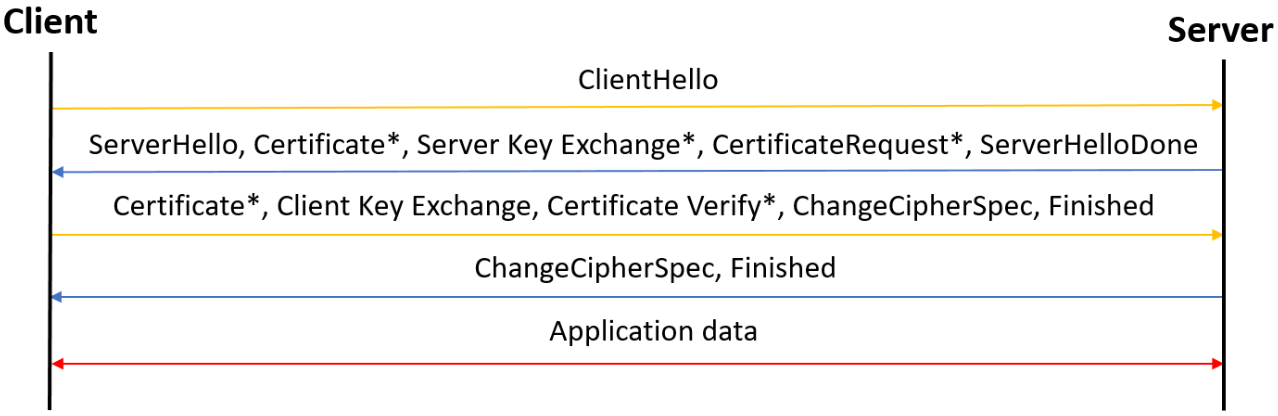
```
fbClient.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\someCA.pem';
```

If the user does not have the CA certificate, a connection can still be established. A boolean variable is available for this purpose, which prevents TwinCAT from verifying the server certificate. Although the connection is encrypted with the public key of the unverified server certificate, it is more vulnerable to man-in-the-middle attacks.

```
fbClient.stTLS.sCA.bNoServerCertCheck:= TRUE;
```

4.3.1.2 Client/Server certificate

This section considers the case where both the client certificate and the server certificate are verified. The slightly modified communication flow (compared to the Server certificate [▶ 15] chapter) is visualized in the following diagram. The individual steps of the TLS connection establishment are described in chapter Transport layer [▶ 12].


Client
Server
ClientHello
ServerHello, Certificate*, Server Key Exchange*, CertificateRequest*, ServerHelloDone
Certificate*, Client Key Exchange, Certificate Verify*, ChangeCipherSpec, Finished
ChangeCipherSpec, Finished
Application data

Application in TwinCAT

If a client certificate is used, in TwinCAT the file path (.PEM or .DER file) or the content of the .PEM file is passed as a string, just as for the CA certificate. TwinCAT as the client then presents this certificate to the server. For Certificate Verify the client's private key must also be referenced. Optionally, in the case of password protection for the private key, this password can also be transferred. The sample code refers to the HTTP client, the MQTT client and the WebSocket client. The HTTP client is used as an example.

```
PROGRAM MAIN
VAR
    fbClient : FB_IotHttpClient;
END_VAR
```

```
fbClient.stTLS.sCA:= 'C:\TwinCAT\3.1\Config\Certificates\someCA.pem';
fbClient.stTLS.sCert:= 'C:\TwinCAT\3.1\Config\Certificates\someCRT.pem';
fbClient.stTLS.sKeyFile:= 'C:\TwinCAT\3.1\Config\Certificates\someprivatekey.pem.key';
fbClient.stTLS.sKeyPwd:= 'yourkeyfilepasswordhere';
```

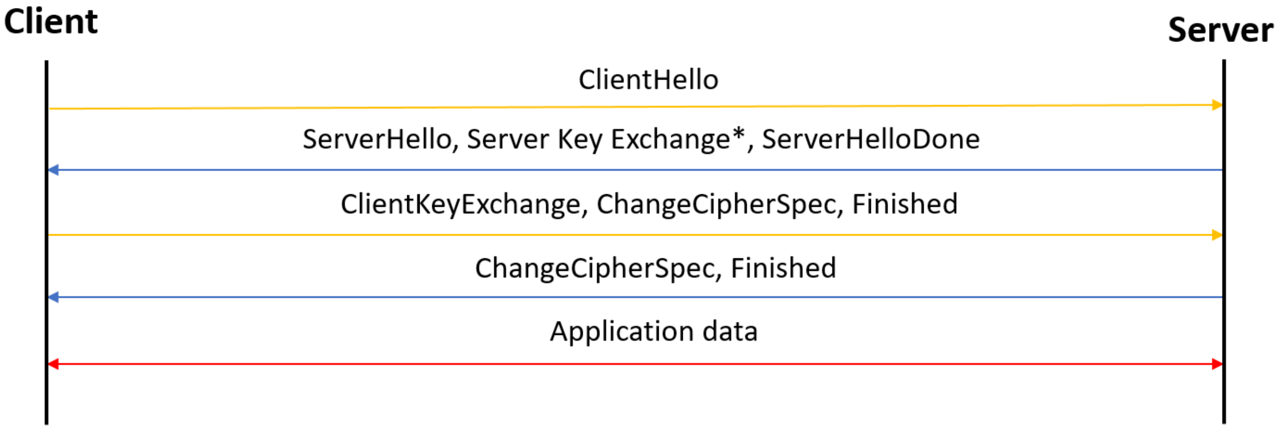
If a client certificate is set, a CA certificate must also be set to validate the server certificate. This is due to the behavior of the security framework used in the IoT driver.

If the validation of the server certificate is to be shutdown in this case, an additional flag can be set to skip the validation. However, it is not possible to omit the CA certificate.

4.3.1.3 Pre-shared keys

By default, asymmetric key pairs are used for the TLS connection establishment. Asymmetric cryptography requires more computing power, so using Pre-Shared Keys (PSK) may be an option in situations where CPU power is limited. Pre-shared keys are previously shared symmetric keys.

Compared to the communication flow with asymmetric encryption, the certificate is omitted when using PSK. Client and server must agree on a PSK via the so-called identity. By definition the PSK is known in advance to both parties.


Client
Server
ClientHello
ServerHello, Server Key Exchange*, ServerHelloDone
ClientKeyExchange, ChangeCipherSpec, Finished
ChangeCipherSpec, Finished
Application data

Server Key Exchange: In this optional message, the server can give the client a hint about the identity of the PSK used.

Client Key Exchange: The client specifies the identity of the PSK to be used for encryption.

Application in TwinCAT

In TwinCAT the identity of the PSK is specified as a string; the PSK itself is stored as a byte array in the controller. The length of the PSK is also specified. The sample code refers to the HTTP client, the MQTT client and the WebSocket client. The HTTP client is used as an example.

```
PROGRAM MAIN
VAR
    fbClient : FB_IotHttpClient;
    cMyPskKey : ARRAY[1..64] OF BYTE := [16#1B, 16#D0, 16#6F, 16#D2, 16#56, 16#16, 16#7D, 16#C1, 16#
E8, 16#C7, 16#48, 16#2A, 16#8E, 16#F5, 16#FF];
END_VAR
```

```
fbClient.stTLS.sPskIdentity:= identityofPSK';
fbClient.stTLS.aPskKey:= cMyPskKey;
fbClient.stTLS.nPskKeyLen:= 15;
```

4.3.1.4 Supported cipher suites

The TwinCAT IoT driver supports secure data transmission using the TLS standard. Below you will find an overview of all cipher suites supported by the IoT driver, depending on the TwinCAT version.

TwinCAT 3.1 Build 4024.x

Cipher suite
AES128-GCM-SHA256
AES128-SHA
AES128-SHA256
AES256-SHA
AES256-SHA256
DES-CBC3-SHA
DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES128-SHA
DHE-RSA-AES128-SHA256
DHE-RSA-AES256-SHA
DHE-RSA-AES256-SHA256
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-SHA
ECDHE-ECDSA-DES-CBC3-SHA
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-SHA
ECDHE-RSA-DES-CBC3-SHA
EDH-RSA-DES-CBC3-SHA
PSK-3DES-EDE-CBC-SHA
PSK-AES128-CBC-SHA
PSK-AES128-CBC-SHA256
PSK-AES128-GCM-SHA256
PSK-AES256-CBC-SHA

TwinCAT 3.1 Build 4026.x

Cipher suite
AES128-GCM-SHA256
AES128-SHA
AES128-SHA256
AES256-GCM-SHA384
AES256-SHA
AES256-SHA256
DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES128-SHA
DHE-RSA-AES128-SHA256
DHE-RSA-AES256-GCM-SHA384
DHE-RSA-AES256-SHA
DHE-RSA-AES256-SHA256
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA384
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES256-SHA
ECDHE-RSA-AES256-SHA384
PSK-AES128-CBC-SHA
PSK-AES128-CBC-SHA256
PSK-AES128-GCM-SHA256
PSK-AES256-CBC-SHA
PSK-AES256-CBC-SHA384
PSK-AES256-GCM-SHA384

4.3.2 Application layer

Various safety mechanisms are also available at the application layer. These safety mechanisms are described below.

4.3.2.1 JSON Web Token (JWT)

JSON Web Token (JWT) is an open standard (based on RFC 7519) that defines a compact and self-describing format for securely transmitting information between communication devices in the form of a JSON object. The authenticity of the transmitted information can be verified and ensured, since a JWT is provided with a digital signature. The signature can involve a shared secret (via an HMAC algorithm) or a public/private key (via RSA).

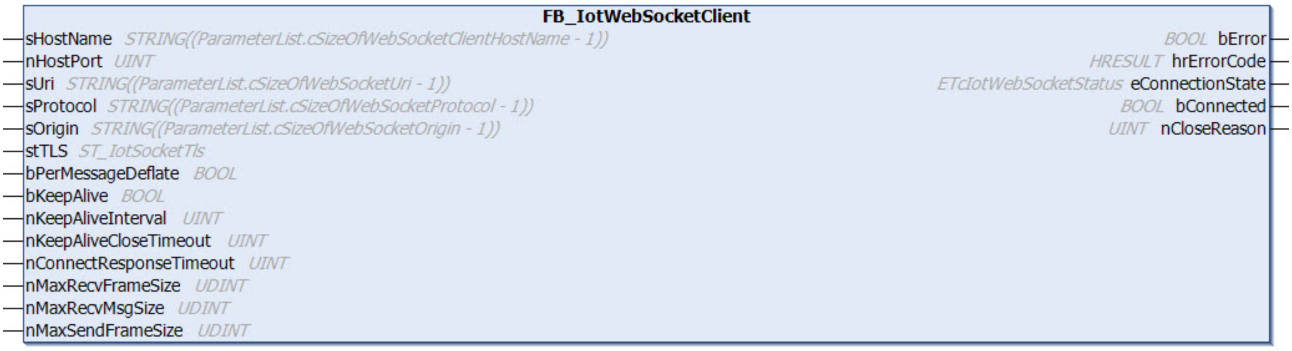
The most common application example for JWT is the authorization of a device or user for a service. Once a user has logged into the service, all further requests to the service include the JWT. Based on the JWT, the service can then decide which additional services or resources the user may access. This means, for example, that single sign-on solutions can be implemented in cloud services.

The PLC library Tc3_JsonXml provides an option to create and sign a JWT via the method FB_JwtEncode.

5 PLC API

5.1 Function blocks

5.1.1 FB_IotWebSocketClient


FB_IotWebSocketClient
sHostName STRING((ParameterList.cSizeOfWebSocketClientHostName - 1))
nHostPort UINT
sUri STRING((ParameterList.cSizeOfWebSocketUri - 1))
sProtocol STRING((ParameterList.cSizeOfWebSocketProtocol - 1))
sOrigin STRING((ParameterList.cSizeOfWebSocketOrigin - 1))
stTLS ST_IotSocketTls
bPerMessageDeflate BOOL
bKeepAlive BOOL
nKeepAliveInterval UINT
nKeepAliveCloseTimeout UINT
nConnectResponseTimeout UINT
nMaxRecvFrameSize UDINT
nMaxRecvMsgSize UDINT
nMaxSendFrameSize UDINT
BOOL bError
HRESULT hrErrorCode
ETcIotWebSocketStatus eConnectionState
BOOL bConnected
UINT nCloseReason

This function block enables communication with a WebSocket server. A client can manage a connection to exactly one WebSocket server. The Execute [▶ 23]() method must be called cyclically as background communication to the server.

All input parameters are only processed when a connection is established. All parameters of type STRING expect the UTF-8 format. This corresponds to the STRING format for the 7-bit ASCII characters.

Syntax

```
FUNCTION BLOCK FB_IotWebSocketClient
VAR_INPUT
    sHostName               : STRING;
    nHostPort               : UINT;
    sUri                    : STRING;
    sProtocol               : STRING;
    sOrigin                 : STRING;
    stTLS                   : ST_IotSocketTls;
    bPerMessageDeflate      : BOOL;
    bKeepAlive              : BOOL;
    nKeepAliveInterval      : UINT;
    nKeepAliveCloseTimeout  : UINT;
    nConnectResponseTimeout: UINT;
    nMaxRecvFrameSize       : UDINT;
    nMaxRecvMsgSize         : UDINT;
    nMaxSendFrameSize       : UDINT;
END_VAR
VAR_OUTPUT
    bError                  : BOOL;
    hrErrorCode             : HRESULT;
    eConnectionState        : ETcIotWebSocketStatus;
    bConnected              : BOOL;
    nCloseReason            : UINT;
END_VAR
```

Inputs

Name	Type	Description
sHostName	STRING	sHostName can be specified as name or as IP address. If no information is provided, the local host is used.
nHostPort	UINT	The host port is specified here. The default is 80.
sUri	STRING	Optional parameter to specify a URI for the WebSocket opening handshake.
sProtocol	STRING	Optional parameter to specify a sub-protocol for the WebSocket opening handshake.
sOrigin	STRING	Optional parameter to specify an origin for the WebSocket opening handshake.
stTLS	ST_IotSocketTls	If the server offers a TLS-secured connection, the required configuration can be implemented here.
bPerMessageDeflate	BOOL	Enables the permessage deflate compression extension.
bKeepAlive	BOOL	Enables keep-alive ping messages sent by the client. The default setting is TRUE.
nKeepAliveInterval	UINT	Specifies how many seconds after no messages have been received from the server the client should send a ping message.
nKeepAliveCloseTimeout	UINT	Specifies how many seconds the client should wait for the ping response.
nConnectResponseTimeout	UINT	Specifies how many seconds the client should wait for the server's handshake response.
nMaxRecvFrameSize	UDINT	Maximum receivable frame size. Standard size is 16#10000.
nMaxRecvMsgSize	UDINT	Maximum receivable message size. Standard size is 16#100000.
nMaxSendFrameSize	UDINT	Maximum frame size that can be sent. Standard size is 16#4000.

Outputs

Name	Type	Description
bError	BOOL	Becomes TRUE when an error situation occurs.
hrErrorCode	HRESULT	Returns an ADS return code. An explanation of the possible ADS return codes can be found in the Appendix.
eConnectionState	ETcIotWebSocketStatus	Specifies the state of the connection from the client to the server as the enumeration ETcIotWebSocketState.
bConnected	BOOL	TRUE if the connection to the host is established and the WebSocket handshake was successful.
nCloseReason	UINT	WebSocket status code as defined in RFC 6455.

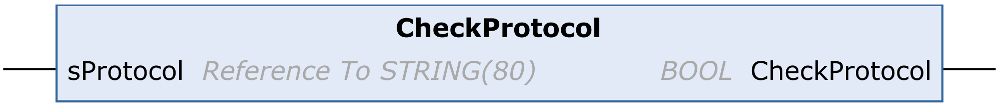
Methods

Name	Description
CheckProtocol [▶ 22]	This method can optionally be overridden to check the protocol returned by the server when the connection is established.
Connect [▶ 22]	Method for establishing the connection to the WebSocket server.
Disconnect [▶ 23]	Method for disconnecting the connection to the WebSocket server.
Execute [▶ 23]	Must be called cyclically as background communication to the WebSocket server. If messages have been received, the Callback method is triggered once for each message.
OnWebSocketClose [▶ 23]	Callback method that is triggered when the WebSocket connection is closed.
OnWebSocketMessage [▶ 24]	Callback method that is triggered when a message is received.
SendMessage [▶ 25]	Method for sending a message.

Requirements

Development environment	Target platform	PLC libraries to include
TwinCAT v3.1.4026.x	IPC or CX (x86, x64, ARM)	Tc3_IotBase (3.5.1 or higher)

5.1.1.1 CheckProtocol



The method must be overridden if the protocol returned by the WebSocket server is to be checked. If the check fails, the value FALSE must be returned.

Syntax

```
METHOD CheckProtocol: BOOL
VAR_IN_OUT CONSTANT
    sProtocol        : STRING;
END_VAR
```

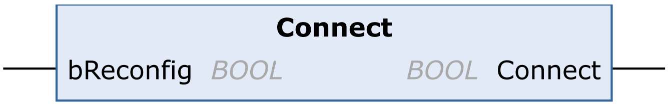
Return value

Name	Type	Description
CheckProtocol	BOOL	TRUE if the check of the protocol returned by the WebSocket server was successful. FALSE if the check was not successful.

/ Inputs/Outputs

Name	Type	Description
sProtocol	STRING	The protocol to be checked.

5.1.1.2 Connect



This method is called when a connection is to be established from the client to the server.

Syntax

```
METHOD Connect: BOOL
VAR_IN
    bReconfig : BOOL;
END_VAR
```

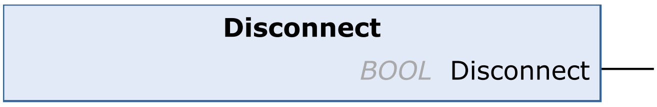
Return value

Name	Type	Description
Connect	BOOL	If the connection is successfully established, the method returns TRUE.

Inputs

Name	Type	Description
bReconfig	BOOL	This parameter must be set to TRUE if the connection parameters were changed after the initial connection was established and should be used the next time the connection is established.

5.1.1.3 Disconnect



This method is called when a connection from the client to the server is to be closed.

Syntax

```
METHOD Disconnect: BOOL
```

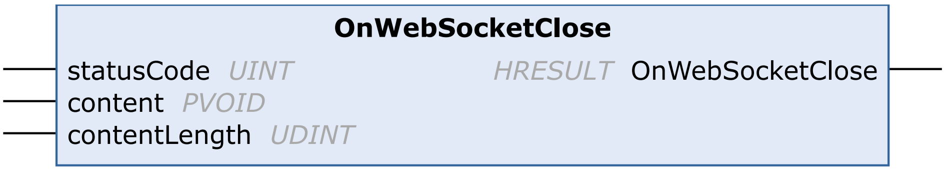
Return value

Name	Type	Description
Disconnect	BOOL	If the connection is successfully closed, the method returns TRUE.

5.1.1.4 Execute

This method must be called cyclically as background communication to the WebSocket server. If messages have been received, the Callback method OnWebSocketMessage [▶ 24]() is triggered once for each message.

5.1.1.5 OnWebSocketClose



This method must not be called by the user. Instead, you can derive from the function block FB_IotWebSocketsClient and override this method. When the Execute [▶ 23]() method is called, the responsible TwinCAT driver has the option of calling the OnWebSocketClose() method in case of an incoming Close Frame.

Syntax

```
METHOD OnWebSocketClose: HRESULT
VAR_IN
    statusCode     : UINT;
    content        : PVOID;
    contentLength  : UDINT;
END_VAR
```

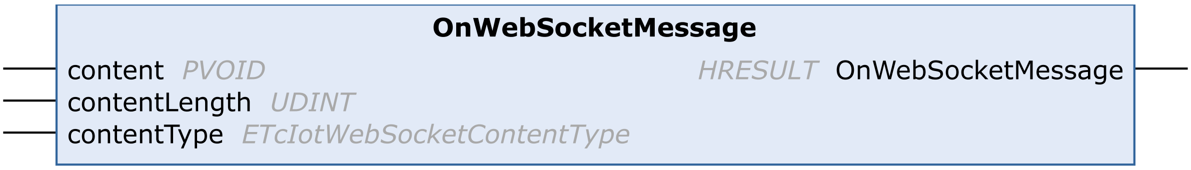
Return value

Name	Type	Description
OnWebSocketClose	HRESULT	This return value can be freely selected.

Inputs

Name	Type	Description
statusCode	UINT	WebSocket status code as defined in RFC 6455.
content	PVOID	Pointer to the content.
contentLength	UDINT	Size of the content in bytes.

5.1.1.6 OnWebSocketMessage


OnWebSocketMessage
content PVOID
contentLength UDINT
contentType ETcIotWebSocketContentType
HRESULT OnWebSocketMessage

This method must not be called by the user. Instead, you can derive from the function block FB_IotWebSocketsClient and override this method. While the Execute() method is called, the responsible TwinCAT driver can call the OnWebSocketMessage() method in the event of new incoming messages. In the event of several incoming messages the callback method is called several times, once per message. This must be taken into account when the method is implemented.

```
METHOD OnWebSocketMessage: HRESULT
VAR_IN
    content              : PVOID;
    contentLength        : UDINT;
    contentType          : ETcIotWebSocketContentType;
END_VAR
```

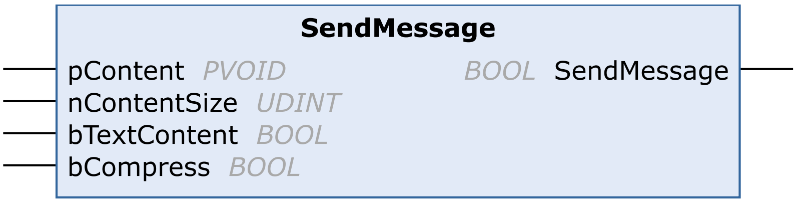
Return value

Name	Type	Description
OnWebSocketMessage	HRESULT	The return value of the method should be S_OK, if the message was accepted. If the message is to be issued again in the context of the next Execute() call, the return value can be assigned S_FALSE.

Inputs

Name	Type	Description
content	PVOID	Pointer to the content.
contentLength	UDINT	Size of the content in bytes.
contentType	ETcIotWebSocketContentType	Specifies whether the content is binary or text.

5.1.1.7 SendMessage



This method is called when a message is to be sent to the WebSocket server.

Syntax

```
METHOD SendMessage: BOOL
VAR_IN
    pContent      : PVOID;
    nContentSize  : UDINT;
    bTextContent  : BOOL;
    bCompress     : BOOL;
END_VAR
```

Return value

Name	Type	Description
SendMessage	BOOL	If the message is sent successfully, the method returns the value TRUE.

Inputs

Name	Type	Description
pContent	PVOID	Pointer to the content.
nContentSize	UDINT	Size of the content in bytes.
bTextContent	BOOL	TRUE: Content is text, FALSE: Content is binary.
bCompress	BOOL	If set to TRUE, compression is used.

5.1.1.8 ST_IotSocketTls

The following type contains the TLS security settings for the HTTP client and the WebSocket client. Either CA (Certificate Authority) or PSK (PreSharedKey) can be used.

Syntax

Definition:

```
TYPE ST_IotSocketTls :
STRUCT
    sCA                 : STRING(255*);
    sCert               : STRING(255*);
    sKeyFile            : STRING(255*);
    sKeyPwd             : STRING(255*);
    sCrl                : STRING(255*);
    sCiphers            : STRING(255*);
    sVersion            : STRING(80)  := 'tlsv1.2';
    bNoServerCertCheck  : BOOL  := FALSE;

    sPskIdentity        : STRING(255*);
    aPskKey             : ARRAY[1..64*] OF BYTE;
    nPskKeyLen          : USINT;
END_STRUCT
END_TYPE
```

Parameter

Name	Type	Description
sCA	STRING(255)	Certificate of the certificate authority (CA)
sCert	STRING(255)	Client certificate for server authentication
sKeyFile	STRING(255)	Private key of the client
sKeyPwd	STRING(255)	Password of the private key, if applicable
sCrl	STRING(255)	Path to the certificate revocation list, which may be present in PEM or DER format
sCiphers	STRING(255)	Cipher suites to be used, specified in OpenSSL string format
sVersion	STRING(80)	TLS version to be used
bNoServerCertCheck	BOOL	Disables verification of the server certificate validity. If communication is to take place without TLS encryption (HTTP/WebSockets), this value must remain FALSE.
sPskIdentity	STRING(255)	PreSharedKey identity for TLS PSK connection
aPskKey	ARRAY[1..64] OF BYTE	PreSharedKey for TLS PSK connection
nPskKeyLen	USINT	Length of the PreSharedKey in bytes

All strings and arrays marked with an * are initialized with the value in brackets. These values can be accessed and changed via the parameter list. This is not possible at runtime, but only before the code is compiled.

5.2 Data types

5.2.1 ETcIotWebSocketStatus

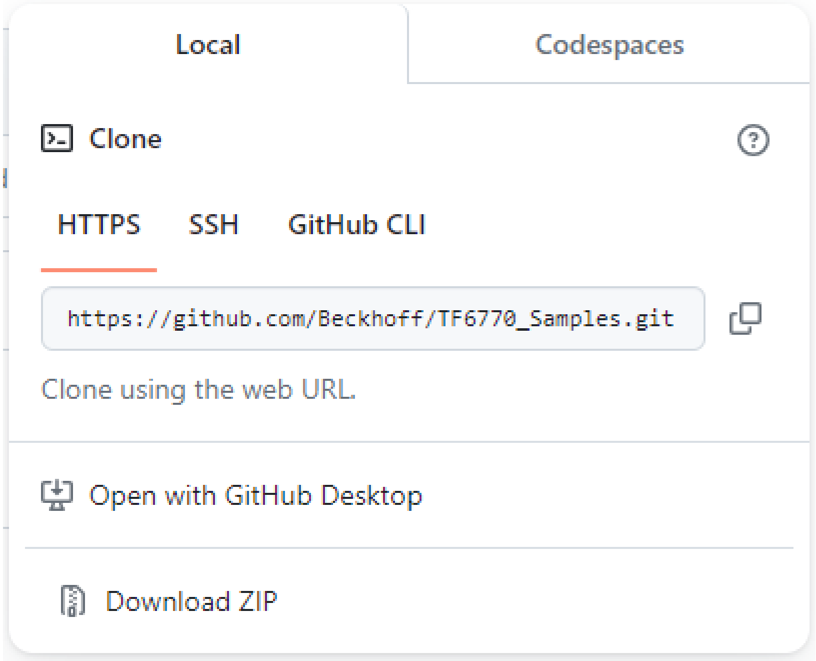
```
TYPE ETcIotWebSocketStatus :
(
    WS_STATUS_BUSY:=-1,
    WS_STATUS_SUCCESS:=0,
    WS_STATUS_SOCK_NOMEM:=1,
    WS_STATUS_SOCK_ERR_CREATE_PROTOCOL:=2,
    WS_STATUS_SOCK_CONN_INVAL:=3,
    WS_STATUS_SOCK_NO_CONN:=4,
    WS_STATUS_SOCK_CONN_REFUSED:=5,
    WS_STATUS_SOCK_NOT_FOUND:=6,
    WS_STATUS_SOCK_CONN_LOST:=7,
    WS_STATUS_SOCK_ERR_TLS:=8,
    WS_STATUS_SOCK_NOT_SUPPORTED:=10,
    WS_STATUS_SOCK_ERR_AUTH:=11,
    WS_STATUS_SOCK_ERRACL_DENIED:=12,
    WS_STATUS_SOCK_ERR_UNKNOWN:=13,
    WS_STATUS_SOCK_ERRNO:=14,
    WS_STATUS_SOCK_ERR_EAI:=15,
    WS_STATUS_SOCK_ERR_PROXY:=16,
    WS_STATUS_TLS_CA_NOTFOUND:=17,
    WS_STATUS_TLS_CERT_NOTFOUND:=18,
    WS_STATUS_TLS_KEY_NOTFOUND:=19,
    WS_STATUS_TLS_CA_INVALID:=20,
    WS_STATUS_TLS_CERT_INVALID:=21,
    WS_STATUS_TLS_KEY_INVALID:=22,
    WS_STATUS_TLS_VERIFY_FAIL:=23,
    WS_STATUS_TLS_SETUP:=24,
    WS_STATUS_TLS_HANDSHAKE_FAIL:=25,
    WS_STATUS_TLS_CIPHER_INVALID:=26,
    WS_STATUS_TLS_VERSION_INVALID:=27,
    WS_STATUS_TLS_PSK_INVALID:=28,
    WS_STATUS_TLS_CRL_NOTFOUND:=29,
    WS_STATUS_TLS_CRL_INVALID:=30,
    WS_STATUS_FINALIZE_DISCONNECT:=31,
    WS_STATUS_SOCK_ERR_BIND:=32,
    WS_STATUS_SOCK_BIND_ADDR_INUSE:=33,
    WS_STATUS_SOCK_BIND_ADDR_INVAL:=34,
    WS_STATUS_SOCK_ERR_CREATE:=35,
    WS_STATUS_SOCK_ERR_CREATE_TYPE:=36,
    WS_STATUS_SOCK_CONN_FAILED:=37,
    WS_STATUS_SOCK_CONN_TIMEDOUT:=38,
    WS_STATUS_SOCK_CONN_HOSTUNREACH:=39,
    WS_STATUS_TLS_CERT_EXPIRED:=40,
    WS_STATUS_TLS_CN_MISMATCH:=41,
    WS_STATUS_INTERNAL_ERROR:=1000,
    WS_STATUS_CONNECT_REQ_INTERNAL_ERROR:=1001,
    WS_STATUS_CONNECT_REQ_SEND_ERROR:=1002,
    WS_STATUS_CONNECT_RES_PARSE_ERROR:=1003,
    WS_STATUS_CONNECT_RES_INVALID:=1004,
    WS_STATUS_CONNECT_RES_INVALID_STATUS:=1005,
    WS_STATUS_CONNECT_RES_ACCEPT_INVALID:=1006,
    WS_STATUS_CONNECT_RES_ACCEPT_INVALID_HASH:=1007,
    WS_STATUS_CONNECT_RES_INVALID_EXTENSION:=1008,
    WS_STATUS_CONNECT_RES_REJECT:=1009,
    WS_STATUS_CONNECT_RES_TIMEDOUT:=1010,
    WS_STATUS_KEEP_ALIVE_TIMEDOUT:=1011,
    WS_STATUS_NOMEMORY:=1012,
    WS_STATUS_INVALID_MSG_SIZE:=1013,
    WS_STATUS_PROTOCOL_ERROR:=1014,
    WS_STATUS_RCV_QUEUE_FULL:=1015,
    WS_STATUS_DECOMPRESS_ERROR:=1016
) DINT;
END_TYPE
```

5.2.2 ETcIotWebSocketContentType

```
TYPE ETcIotWebSocketContentType :
(
    WS_CONTENT_CONTINUATION:=0,
    WS_CONTENT_TEXT:=1,
    WS_CONTENT_BINARY:=2
) DINT;
END_TYPE
```

6 Samples

Sample code and configurations for this product can be obtained from the corresponding repository on GitHub: https://github.com/Beckhoff/TF6770_Samples. There you have the option to clone the repository or download a ZIP file containing the sample.



```
https://github.com/Beckhoff/TF6770_Samples.git
```

Clone using the web URL.

7 Appendix

7.1 ADS Return Codes

Grouping of error codes:
Global error codes: ADS Return Codes [▶ 29]... (0x9811_0000 ...)
Router error codes: ADS Return Codes [▶ 29]... (0x9811_0500 ...)
General ADS errors: ADS Return Codes [▶ 30]... (0x9811_0700 ...)
RTime error codes: ADS Return Codes [▶ 32]... (0x9811_1000 ...)

Global error codes

Hex	Dec	HRESULT	Name	Description
0x0	0	0x98110000	ERR_NOERROR	No error.
0x1	1	0x98110001	ERR_INTERNAL	Internal error.
0x2	2	0x98110002	ERR_NORTIME	No real time.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Allocation locked – memory error.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Wrong HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Target port not found – ADS server is not started or is not reachable.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Target computer not found – AMS route was not found.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unknown command ID.
0x9	9	0x98110009	ERR_BADTASKID	Invalid task ID.
0xA	10	0x9811000A	ERR_NOIO	No IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unknown AMS command.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 error.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port not connected.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Invalid AMS length.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Invalid AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installation level is too low –TwinCAT 2 license error.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	No debugging available.
0x12	18	0x98110012	ERR_PORTDISABLED	Port disabled – TwinCAT system service not started.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port already connected.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 error.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync error.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	No index map for AMS Sync available.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Invalid AMS port.
0x19	25	0x98110019	ERR_NOMEMORY	No memory.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP send error.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host unreachable.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Invalid AMS fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS send error – secure ADS connection failed.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Access denied – secure ADS access denied.

Router error codes

Hex	Dec	HRESULT	Name	Description
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Locked memory cannot be allocated.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	The router memory size could not be changed.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	The mailbox has reached the maximum number of possible messages.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	The Debug mailbox has reached the maximum number of possible messages.
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	The port type is unknown.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	The router is not initialized.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	The port number is already assigned.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	The port is not registered.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	The maximum number of ports has been reached.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	The port is invalid.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	The router is not active.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	The mailbox has reached the maximum number for fragmented messages.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	A fragment timeout has occurred.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	The port is removed.

General ADS error codes

Hex	Dec	HRESULT	Name	Description
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	General device error.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service is not supported by the server.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Invalid index group.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Invalid index offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Reading or writing not permitted.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parameter size not correct.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Invalid data values.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Device is not ready to operate.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Device is busy.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Insufficient memory.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Invalid parameter values.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Not found (files, ...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax error in file or command.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objects do not match.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Object already exists.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol not found.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Invalid symbol version. This can occur due to an online change. Create a new handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Device (server) is in invalid state.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode not supported.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification handle is invalid.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification client not registered.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	No further handle available.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Notification size too large.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Device not initialized.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Device has a timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface query failed.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Wrong interface requested.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class ID is invalid.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object ID is invalid.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Request pending.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Request is aborted.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal warning.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Invalid array index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol not active.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Access denied.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Missing license.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	License expired.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	License exceeded.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Invalid license.
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	License problem: System ID is invalid.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	License not limited in time.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Licensing problem: time in the future.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	License period too long.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception at system startup.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	License file read twice.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Invalid signature.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Invalid certificate.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public key not known from OEM.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	License not valid for this system ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo license prohibited.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Invalid function ID.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Outside the valid range.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Invalid alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Invalid platform level.

Hex	Dec	HRESULT	Name	Description
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Context – forward to passive level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Context – forward to dispatch level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Context – forward to real time.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Client error.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Service contains an invalid parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling list is empty.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var connection already in use.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	The called ID is already in use.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Error in Win32 subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Invalid client timeout value.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port not open.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	No AMS address.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Internal error in Ads sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Hash table overflow.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Key not found in the table.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	No symbols in the cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Invalid response received.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port is locked.
0x756	1878	0x98110756	ADSERR_CLIENT_REQUESTCANCELLED	The request was cancelled.

RTime error codes

Hex	Dec	HRESULT	Name	Description
0x1000	4096	0x98111000	RTERR_INTERNAL	Internal error in the real-time system.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer value is not valid.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task pointer has the invalid value 0 (zero).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack pointer has the invalid value 0 (zero).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	The request task priority is already assigned.
0x1005	4101	0x98111005	RTERR_NOMORETCB	No free TCB (Task Control Block) available. The maximum number of TCBs is 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	No free semaphores available. The maximum number of semaphores is 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	No free space available in the queue. The maximum number of positions in the queue is 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	An external synchronization interrupt is already applied.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	No external sync interrupt applied.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Application of the external synchronization interrupt has failed.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Call of a service function in the wrong context
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x extension is not supported.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x extension is not enabled in the BIOS.
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Missing function in Intel VT-x extension.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Activation of Intel VT-x fails.

Specific positive HRESULT Return Codes:

HRESULT	Name	Description
0x0000_0000	S_OK	No error.
0x0000_0001	S_FALSE	No error. Example: successful processing, but with a negative or incomplete result.
0x0000_0203	S_PENDING	No error. Example: successful processing, but no result is available yet.
0x0000_0256	S_WATCHDOG_TIMEOUT	No error. Example: successful processing, but a timeout occurred.

TCP Winsock error codes

Hex	Dec	Name	Description
0x274C	10060	WSAETIMEDOUT	A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond.
0x274D	10061	WSAECONNREFUSED	Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running.
0x2751	10065	WSAEHOSTUNREACH	No route to host - a socket operation referred to an unavailable host.
More Winsock error codes: Win32 error codes			

More Information:
www.beckhoff.com/tf6770/

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com