OPC UA SDK
Client + Server Development

Server Development Introduction

Tested? You want it?

License Model Prices Quotation Order Now

The Server

Start

This is happening on calling 'Start':

1. Checking, if an **address was set** (Address Property).
2. The Server changes its status (**OpcServer.State** property) to the value ***Starting***.
3. The Server **checks its configuration** for validity and conclusiveness.
4. Next the Server tries to **create a host for every endpoint description**.
5. What follows is the **start of all Managers** (NodeManager, SessionManager, ...)
6. Finally, **every created host** responsible for the endpoint-specific communication with the Client gets **started**.
7. The Server changes its status (**OpcServer.State** property) to the value ***Started***.

Stop

This is happening on calling 'Stop':

1. The Server changes its status (**OpcServer.State** property) to the value ***Stopping***.
2. **Closing all Managers** (NodeManager, SessionManager, ...)
3. The Server **releases** all **gathered resources**.
4. **Closing** the **hosts of endpoint descriptions**.
5. The Server changes its status (**OpcServer.State** property) to the value ***Stopped***.

Parameters

In order for the Server to be able to give the Clients access to its OPC UA Services, the right parameters have to be determined. **In general** the **server address** (**OpcServer.Address** property) **is needed**. The Uri instance (= Uniform Resource Identifier) supplies all Clients the primarily necessary information via the server. For example, the server address “opc.tcp://192.168.0.80:4840” contains the information of the concept “opc.tcp” (possible are “http”, “https”, “opc.tcp”, “net.tcp” and “net.pipe”), which determines via which protocol the data shall be exchanged. Generally, “opc.tcp” is advisable for OPC UA Servers in a local network. Servers out of the local network should use “http”, even better “https”. Furthermore the address defines that the server is carried out on the computer with the IP address “192.168.0.80” and listens to requests via the port with the number “4840” (which is default for OPC UA, customized port numbers are also possible). Instead of the static IP address the DNS name of the computer can also be used, so instead of “127.0.0.1” you could also use “localhost”.

If the server shall provide **no endpoint** for data exchange with savety mode **“None”** (also additionally possible are “Sign” and “SignAndEncrypt”) as its policy, **at least one Endpoint-Policy has to be** manually **configured** (**OpcServer.Security.EndpointPolicies** property). If, however, an **endpoint with the Property “None”** is supplied by the Server, a Client can select this one automatically for an easy and quick connection. When a policy level (a number) is **assigned according to the OPC Foundation** to the separate Endpoint Policies during the definition of the endpoints, Clients can handle those appropriately. Here the OPC intends that the higher the level of the Policy of an endpoint, the “better” that endpoint (note that this is merely a neither watched nor imposed guideline).

If the Server shall use an access control, for example via an ACL (= Access Control List), **the user data has to be determined for identification of possible / valid identities of users** of the Server (this also works in a running system). Here it is possible to determine the identities of users through a username-password pair (**OpcServerIdentity** class) or through a certificate (**OpcCertificateIdentity** class). Those identities have then to be **communicated to the Server** (**OpcServer.Security.UserNameAcl/CertificateAcl** property). Those access control lists have to be activated in order for the server to recognize them (**OpcServer.Security.UserNameAcl/CertificateAcl.IsEnabled** property).

Endpoints

Endpoints result from the crossing of the used Base-Addresses of the Server and the security strategies supported by the Server. The results are the Base-Addresses of every scheme-port pair supported, while several schemes (possible are “http”, “https”, “opc.tcp”, “net.tcp” and “net.pipe”) can be determined for data exchange on different ports. The hereby linked policies determine the procedure during the data exchange. Consisting of the Policy Level, the Security-Mode and the Security-Algorithm, every policy determines the kind of secure data exchange.

For example, when two Security-Policies are followed, they can be defined as follows:

- Security-Policy A: Level=0, Security-Mode=None, Security-Algorithm=None
- Security-Policy B: Level=1, Security-Mode=Sign, Security-Algorithm=Basic256

When furthermore, for example, three Base-Addresses are combined for different schemes as follows:

- Base-Address A: "https://mydomain.com/"
- Base-Address B: "opc.tcp://192.168.0.123:4840/"
- Base-Address C: "opc.tcp://192.168.0.123:12345/"

The results will be the following endpoint descriptions through the crossing:

- Endpoint 1: Address="https://mydomain.com/", Level=0, Security-Mode=None, Security-Algorithm=None
- Endpoint 2: Address="https://mydomain.com/", Level=1, Security-Mode=Sign, Security-Algorithm=Basic256
- Endpoint 3: Address="opc.tcp://192.168.0.123:4840/", Level=0, Security-Mode=None, Security-Algorithm=None
- Endpoint 4: Address="opc.tcp://192.168.0.123:4840/", Level=1, Security-Mode=Sign, Security-Algorithm=Basic256
- Endpoint 5: Address="opc.tcp://192.168.0.123:12345/", Level=0, Security-Mode=None, Security-Algorithm=None
- Endpoint 6: Address="opc.tcp://192.168.0.123:12345/", Level=1, Security-Mode=Sign, Security-Algorithm=Basic256

Here the address part of the endpoint is always determined by the Server (via constructor or via **Address** property). While the Server defines an endpoint with the Security-Mode “None” by default, the policy of the endpoint has to be configured manually (**OpcServer.Security.EndpointPolicies** property) when none of this kind or additional ones shall be used.

Information about Certificates

Certificates in OPC UA

Certificates are used to **ensure** the **authenticity** and **integrity** of Client and Server applications. Therefore they act as a kind of identity card for Client as well as Server application. This “identification card” has to be stored somewhere as it exists as a form or data. The decision on where certificates are stored is individual.

There are different types of Stores for certificates:

- **Store for user certificates**
  The Store also called **Application Certificate Store** exclusively contains certificates of those applications that use this Store as an Application Certificate Store. Here a Client / Server application saves its own certificate.
- **Store for certificates from trustworthy certificate issuers**
  The Store also called **Trusted Issuer Certificate Store** exclusively contains certificates from certificate issuers that issues further certificates. Here a Client / Server application saves all certificates from issuers whose certificates shall be treated as trusted by default.
- **Store for trustworthy certificates**
  The Store also called **Trusted Peer Store** exclusively contains certificates treated as trusted. Here a Client saves the **certificates from trusted Servers** and a Server saves the **certificates from trusted Clients**.
- **Store for rejected certificates**
  The Store also called **Rejected Certificate Store** exclusively contains certificates that are decreed as not trusted. Here a Client saves the **certificates from not trusted Servers** and a Server saves the **certificates from not trusted Clients**.

Regardless of the Store being located somewhere in the system or in the data system via a list, generally certificates in the **Trusted Store** are **trusted** and certificates in the **Rejected Store** are untrusted. Certificates not belonging to eeither of the former are automatically saved in the Trusted Store, if the certificate of the certificate issuer mentioned in the certificate is deposited in the Trusted Issuer Store; otherwise it is automatically saved in the Rejected Store. Even if a trustworthy certificate has expired or if its deposited information cannot be successfully verified through the certification center the certificate is graded as not trustworthy and saved in the Rejected Store. During this process it also is removed from the Trusted Peer Store. A certificate can also expire when it is listed in a CRL (=Certificate Revocation List), which can be kept separately in the concerning store.

A certificate that the Client receives from the Server or the other way around is **for the moment always** classified as *unknown* and therefore also treated as **untrusted**. In order for a certificate to be treated as trusted it must be declared as such. This happens by saving the certificate of the Client in the Trusted Store of the Server and the certificate of the Server in the Trusted Store of the Client.

Dealing with a Server certificate at the Client:

1. The Client establishes the certificate on the Server on whose Endpoint it shall connect with.
2. The Client verifies the certificate of the Server.
    1. Is the certificate valid?
        1. Has the effective date expired?
        2. Is the issuer's certificate valid?
    2. Does the certificate exist in the Trusted Peer Store?

        1. Is it listed in a CRL?
    3. Does the certificate exist in the Rejected Store?
3. When the certificate is trusted, the Client establishes a connection to the server.

Dealing with a Client certificate at the Server:

1. The Server receives the Client's certificate from the Client while connecting.
2. The Server verifies the certificate of the Client.
    1. Is the certificate valid?
        1. Has the effective date expired?
        2. Is the issuer's certificate valid?
    2. Does the certificate exist in the Trusted Peer Store?
        1. Is it listed in a CRL?
    3. Does the certificate exist in the Rejected Store?
3. When the certificate is trusted, the Server allows the connection of the Client and operates it.

In case the verification of the certificate fails at the respective counterpart the verification can be extended by custom mechamisms and still decided on user scale, if the certificate gets accepted or not.

Types of Certificates

General: Self-Signed Certificates vs. Signed Certificates

A certificate is comparable to a document. A document can be issued by everybody and can also be signed by everybody. However, the main difference here is, if the signee of a document really vouches for its correctness (like a notary) or if the signee is the owner of the document itself. Especially documents of the latter are not really inspiring confidence because no (legally) recognized instance as e.g. a notary vouches for the owner of the document.

As certificates are comparable to documents and also have to show a (digital) signature, the situation here is the same. The signature of a certificate has to tell the recipient of the certificate copy, who vouches for this certificate. Herefore it always applies that the issuer of a certificate also signs it. When the **issuer of a certificate equals the subject** of the certificate, you call this a **self-signed certificate** (subject equals issuer). When the **issuer of a certificate does not equal the subject** of the certificate, you call this a **(simple / normal / signed) certificate** (subject does not equal issuer).

As certificates are used especially in the context of the OPC UA authentication of an identity (of a certain Client or Server application), signed certificates should be used as application certificates for the own application. If, however, the issuer of the certificate also its owner, this self-signed certificate should only be trusted when the owner is rated as trusted. Such certificates were, as described, signed by the issuer of the certificate. Therefore, the issuer certificate has to be located in the **Trusted Issuer Store** of the application. When the issuer certificate cannot be found there , the certificate chain is declared incomplete and the certificate is not accepted by the counterpart. Yet, if the issuer certificate of the issuer of the application certificate is not a self-signed certificate, the certificate of its issuer has to be available in the **Trusted Issuer Store**.

User Identification

User Identification through Certificates

Next to the use of a certificate as an *identification card* for Client / Server applications, a certificate can also be used to identify a user. A Client application is always operated by a certain user by whom it

operates with the Server. Depending on the Server configuration a Server can request additional information about the identity of the Client's user from the Client. The user has the possibility to prove his identity through a certificate. How thoroughly a Server is examining the certificate on validity, authenticity and confidentiality depends on the Server. The Server provided by the Framework exclusively checks, if the Thumbprint information of the user identity can be found in its ACL (=Access Control List) for certificate-based user identities.

Aspects of Security

Productive use

The primary goal of the Framework is to make getting the grips of the OPC UA as easy as possible. This basic thought sadly also leads to the fact that without secondary configuration of the Server a completely save connection / communication between Client and Server does not occur. Yet, if the final Spike has been implemented and tested, second thought should be given to the *aspects of security*.

Even if the Server is *only* run within a local network one should consider the use of access control lists (**OpcServer.Security.UserNameAcl/CertificateAcl** property). Here user identities can be defined via a certificate or a username-password pair. A Certificate Identity increases security in signed data transmission, for example.

Especially in cases of the Server being accessible publicly, other Security-Policies with appropriately higher Security-Mode and a matching Security-Algorithm should be negotiated. The Security-Policy-Mode “None” used by default is in this matter literally the “Great Wide Open” into your Server (**OpcServer.Security.EndpointPolicies** Properties). Last but not least one should consider the access via an anonymous user identity (**OpcServer.Security.AnonymousAcl.IsEnabled** property). According to the OPC Foundation every Endpoint Policy used above its level should stress its “quality”, in which the rule applies that the higher the level, the “better” the endpoint this policy uses.

For simplified handling of certificates the Server accepts every certificate by default (**OpcServer.Security.AutoAcceptUntrustedCertificates** property), also those it should deny under productive conditions because only certificates known to the Server (located in the Trusted Peer Store) apply as truly trusted. Apart from that the validity of a certificate should always be verified, including the “expiration date” of the certificate, for example. Other properties of the certificate or looser rules for the validity and trustworthiness of a Client certificate can be furthermore carried out manually (**OpcServer.CertificateValidationFailed** event).

Table of Contents