OPC UA SDK
Client + Server Development

Use Cases

Tested? You want it?

License Model Prices Quotation Order Now

C# ▼ |VB|

Client API

Monitoring a Connection

The following types are mentioned here: OpcClient.

The state of the connection between the Client and Server is crucial for the communication between the two participants. Monitoring the status of the current session/connection can provide information whether a request can be successfully send. The following events are used to monitor the Client:

OpcClient.Connected and **OpcClient.Connecting**

- When is the OpcClient.Connected event triggered? If ...
  - the connection to the Server was established by calling OpcClient.Connect().
- How is the OpcClient.Connected event to be understood?
  - It is used to provide information that a connection attempt, which is reported at the beginning with the OpcClient.Connecting event, was successfully completed.

The events mentioned are only triggered by the manual call of OpcClient.Connect () and not after a connection break if the connection to the Server was automatically re-established.

OpcClient.Disconnected and **OpcClient.Disconnecting**

- When are these events triggered? If ...
  - the connection to the Server is terminated by calling OpcClient.Disconnect() or implicitly by calling OpcClient.Dispose().
  - the loss of the connection during the execution of a request (e.g. OpcClient.ReadNode, OpcClient.WriteNode, OpcClient.BrowseNode, ...) is determined. In this case, the OpcClient.Disconnected event is triggered without first triggering the OpcClient.Disconnecting event.
- How are the events to be understood?
  - They are used to provide information about the manual disconnection or the disconnection to the Server (Server-side, network-technical) that was determined during a request.

Loss of connection

The following types are mentioned here: OpcClient.

The configuration of the OpcClient class decides on the actions that take place as soon as the connection to the Server is lost. Regardless of the cause of the connection loss, the following events are critical:

OpcClient.StateChanged

- When is this event triggered? If ...
  - a change to the OpcClient.State property has been made.
  - the OpcClient.Connecting, OpcClient.Connected, OpcClient.Disconnecting or the OpcClient.Disconnected is triggered.
- How is the event to be understood?
  - It is used for the general “monitoring” of the status changes in the OpcClient and thus

provides a summary of the aforementioned events.
    - The event is also triggered before the events mentioned are triggered and can therefore also be used for general handling of changes to the connection status (OpcClient.State) when calling OpcClient.Connect(), OpcClient.Disconnect(), OpcClient.Dispose(), OpcClient.ReadNode(…), OpcClient.WriteNode(…), OpcClient.BrowseNode(…), etc..

OpcClient.BreakDetected

- When is this event triggered? If …
    - the connection to the Server was disconnected for whatever reason,
    - the ReconnectTimeout = 0 was set
    - and UseBreakDetection = true is set.
- How is this event to be understood?
    - It is generally used to detect the loss of an active connection to the Server.
    - A developer can use this event to make their own mechanisms including preparations to re-establish a connection to the Server.

OpcClient.Reconnecting and **OpcClient.Reconnected**

- When are these events triggered? If …
    - the connection to the Server was disconnected for whatever reason,
    - the ReconnectTimeout > 0 has been set
    - and UseBreakDetection = true is set.
- How are these events to be understood?
    - They are generally used to provide information about the loss of an active connection to the Server.
    - After the reconnecting event occurs, the Client automatically tries to reestablish the connection to the Server and, if possible, reuse the session that is currently in use.
    - It is periodically tried with the period = ReconnectTimeout to restore the connection to the Server. A session that has since been lost is automatically opened as a new one.
    - If a new connection to the Server is successfully established, the Reconnected event is triggered.
    - If UseBreakDetection = false is used, no automatic mechanisms for re-establishing the connection are applied, with the result that the developer of the application has to deal with such situations.

The events described are triggered based on a KeepAlive logic. A watchdog in the OPC Foundation stack used checks the status of the Server cyclically (every 5 seconds) and detects the loss of a connection after 5 seconds at the latest. If the loss of a connection is determined, the events described take action depending on the configuration of the OpcClient. The KeepAlive interval used can be changed via the KeepAlive property of the OpcClient class.

Processing Notifications

The following types are mentioned here: OpcClient, OpcSubscription, OpcMonitoredItem and OpcNotification.

Notifications are always received by the Client following an active subscription. The processing chain of a notification begins with the handler of the associated MonitoredItem. From there, the notification is forwarded to the responsible subscription and finally to the Client of the subscription. The notifications that can be treated here provide the information received from the Server about a data change or an alarm or

event that has occurred on the Server.

OpcClient.NotificationReceived

- When is this event triggered? If ...
  - at least one subscription has been set up by the Client on the Server,
    - at least one subscription has at least one MonitoredItem
    - and the filter conditions of the item were met on the Server side.
  - the Subscription Publishing Interval has expired,
    - no new notification has been received from the Server in the meantime
    - therefore the Client automatically sent a request to update the subscription
    - and the Server then replies with an empty response to the request.
- How is this event to be understood?
  - It is used for general processing of notifications.
  - Even without a specific MonitoredItem, such as the request to update.
  - In most cases, the handling of the event is not necessary, since the really relevant notifications are handled via the EventHandler of the MonitoredItems.

Filter Events

The following types are mentioned here: OpcClient, OpcSimpleAttributeOperand, OpcFilter, OpcFilterOperand and OpcAlarmCondition.

The following example subscribes to all events that are currently active but have not yet been confirmed:

```
var isActive = new OpcSimpleAttributeOperand(
        OpcEventTypes.AlarmCondition,
        "ActiveState", "Id");
var isAcked = new OpcSimpleAttributeOperand(
        OpcEventTypes.AcknowledgeableCondition,
        "AckedState", "Id");

var filter = OpcFilter.Using(client)
    .FromEvents(OpcEventTypes.AlarmCondition)
    .Where(OpcFilterOperand.OfType(OpcEventTypes.AlarmCondition)
        & isAcked == false
        & isActive == true)
    .Select();

var subscription = client.SubscribeEvent("ns=2;s=Machine", filter, (sender, e) => {
    if (e.Event is OpcAlarmCondition alarm) {
        Console.WriteLine(
                "{0}: {1}, IsActive = {2}, IsAcked = {3} ({4})",
                alarm.GetType().Name,
                alarm.Message,
                alarm.IsActive,
                alarm.IsAcked,
                alarm.Severity);
    }
});
```

Configuration using UWP

Under UWP (Universal Windows Platform), applications are strictly regulated, such as when developing a mobile application. The following “capabilities” must be set in the project so that the UWP app can act as a Client application via OPC UA:

- internetClient
- privateNetworkClientServer

The required application certificate cannot be automatically created by the OPC UA .NET SDK under UWP. Wherefore an application certificate must be created manually outside of the UWP application. This certificate can then be delivered as part of the `assets` of the application and loaded from it. An example of how a UWP project could be set up can be found here: OPC UA .NET Samples - UWP Client

Server API

NamespaceIndex Nr. 1

The following types are mentioned here: OpcServer and OpcNodeManager.

By default, all nodes are in the user-defined namespace with the number (= NamespaceIndex) two. The namespace with number one is described by the underlying stack of the OPC Foundation as “core namespace” and as “application namespace”. So that **nodes within the namespace with the index one** can be managed, the **user-defined OpcNodeManager must use the value of the OpcServer.ApplicationUri property as NamespaceUri**.

One session per User

The following types are mentioned here: OpcServer, OpcAccessControlEntry and OpcSession.

To disable subsequent user access attempts as long a user controlled through an access control entry (ACE) is connected, all available endpoints needs to be blocked for the user as long as a session is active for him. How this can work can bee seen here:

```
server.SessionActivated += (sender, e) => {
    var identityEntry = GetUserEntry(server.Security, e.Session);

    if (identityEntry != null)
        identityEntry.Disable(OpcEndpointIdentity.GetCurrent());
};

server.SessionClosing += (sender, e) => {
    var identityEntry = GetUserEntry(server.Security, e.Session);

    if (identityEntry != null)
        identityEntry.Enable(OpcEndpointIdentity.GetCurrent());
};


private static OpcAccessControlEntry GetUserEntry(
        OpcServerSecurity security,
        OpcSession session)
{
    var identity = session.UsedIdentity;

    // In case of anonymous.
    if (identity == null)
        return null;

    return (from userEntry in security.UserNameAcl.Entries
            let userPrincipal = userEntry.Principal
            let userIdentity = (OpcUserIdentity)userPrincipal.Identity
            where userIdentity.DisplayName == identity.DisplayName
            select userEntry).FirstOrDefault();
}
```

Table of Contents