OPC UA SDK
Client + Server Development

Anwendungsfälle

Getestet? Du willst es?

Lizenzmodell Preise Angebot Jetzt bestellen

C# ▼ |VB|

Client API

Überwachen einer Verbindung

Die folgenden Typen werden hier besprochen: OpcClient.

Der Zustand der Verbindung zwischen Client und Server ist ausschlaggebend für die Kommunikation zwischen den beiden Teilnehmern. Die Überwachung des Zustands der aktuellen Sitzung/Verbindung kann darüber Auskunft geben, ob eine Anfrage erfolgreich abgesetzt werden kann. Die folgenden Ereignisse ermöglichen hierbei die Überwachung des Clients:

OpcClient.Connected und **OpcClient.Connecting**

- Wann wird das Ereignis OpcClient.Connected ausgelöst? Wenn …
  - die Verbindung zum Server durch den Aufruf von OpcClient.Connect( ) hergestellt wurde.
- Wie ist das Ereignis OpcClient.Connected zu verstehen?
  - Es dient der Auskunft, dass ein Verbindungsversuch, über den zu Beginn mit dem Ereignis OpcClient.Connecting informiert wird, erfolgreich abgeschlossen wurde.

Die genannten Ereignisse werden nur durch den manuellen Aufruf von OpcClient.Connect() ausgelöst und nicht nach einen Verbindungsabbruch, wenn die Verbindung zum Server automatisch wieder hergestellt wurde.

OpcClient.Disconnected und **OpcClient.Disconnecting**

- Wann werden diese Ereignisse ausgelöst? Wenn …
  - die Verbindung zum Server durch den Aufruf von OpcClient.Disconnect( ) oder implizit durch den Aufruf von OpcClient.Dispose( ) getrennt wird.
  - der Verlust der Verbindung während der Ausführung einer Anfrage (z.B. OpcClient.ReadNode, OpcClient.WriteNode, OpcClient.BrowseNode, …) festgestellt wird. In diesem Fall wird das OpcClient.Disconnected Ereignis ohne vorheriges Auslösen des OpcClient.Disconnecting Ereignisses ausgelöst.
- Wie sind die Ereignisse zu verstehen?
  - Sie dienen der Auskunft über die manuelle Trennung oder während einer Anfrage festgestellten (Server-seitigen, Netzwerk-technischen) Trennung der Verbindung zum Server.

Verlust einer Verbindung

Die folgenden Typen werden hier besprochen: OpcClient.

Die Konfiguration der OpcClient-Klasse entscheidet über die Aktionen die eintreten, sobald die Verbindung zum Server verloren gegangen ist. Unabhängig von der Ursache des Verbindungsverlusts sind die folgenden Ereignisse ausschlaggebend:

OpcClient.StateChanged

- Wann wird dieses Ereignis ausgelöst? Wenn …
  - eine Änderung der OpcClient.State Eigenschaft vorgenommen wurde.
  - das OpcClient.Connecting, OpcClient.Connected, OpcClient.Disconnecting oder das

OpcClient.Disconnected ausgelöst wird.

- Wie ist das Ereignis zu verstehen?
  - Es dient der allgemeinen „Überwachung“ der Zustandsänderungen im OpcClient und stellt somit eine Zusammenfassung der zuvor genannten Ereignisse bereit.
  - Das Ereignis wird zudem vor dem Auslösen der genannten Ereignisse ausgelöst und kann deshalb auch zur allgemeinen Behandlung von Änderungen am Verbindungsstatus (OpcClient.State) beim Aufruf von OpcClient.Connect(), OpcClient.Disconnect(), OpcClient.Dispose(), OpcClient.ReadNode(…), OpcClient.WriteNode(…), OpcClient.BrowseNode(…), etc. verwendet werden.

OpcClient.BreakDetected

- Wann wird dieses Ereignis ausgelöst? Wenn …
  - die Verbindung zum Server aus welchen Grund auch immer getrennt wurde,
  - der ReconnectTimeout = 0 festgelegt wurde
  - und UseBreakDetection = true gesetzt ist.
- Wie ist dieses Ereignis zu verstehen?
  - Es dient im Allgemeinen der Erkennung des Verlustes einer aktiven Verbindung zum Server.
  - Ein Entwickler kann dieses Ereignis dazu verwenden, um eigene Mechanismen samt Vorbereitungen zum Wiederherstellen einer Verbindung zum Server zu treffen.

OpcClient.Reconnecting und **OpcClient.Reconnected**

- Wann werden diese Ereignisse ausgelöst? Wenn …
  - die Verbindung zum Server aus welchen Grund auch immer getrennt wurde,
  - der ReconnectTimeout > 0 festgelegt wurde
  - und UseBreakDetection = true gesetzt ist.
- Wie sind diese Ereignisse zu verstehen?
  - Sie dienen im Allgemeinen der Information über den Verlust einer aktiven Verbindung zum Server.
  - Nach Auftreten des Reconnecting Ereignisses versucht der Client selbstständig die Verbindung zum Server wiederherzustellen und ggf. auch die aktuell verwendete Sitzung wiederzuverwenden.
  - Es wird periodisch mit der Periodendauer = ReconnectTimeout versucht die Verbindung zum Server wiederherzustellen. Eine zwischenzeitlich verlorene Sitzung wird automatisch als neue eröffnet.
  - Wird erfolgreich eine neue Verbindung zum Server hergestellt, wird das Reconnected Ereignis ausgelöst.
  - Wird UseBreakDetection = false verwendet, werden keine Automatismen zum Wiederaufbau der Verbindung angewendet, was zur Folge hat, dass der Entwickler der Anwendung derartige Situationen behandeln muss.

Die beschriebenen Ereignisse werden auf der Basis einer KeepAlive-Logik ausgelöst. Ein im verwendeten Stack der OPC Foundation vorhandener Watchdog prüft zyklisch (alle 5 Sekunden) den Status des Servers und stellt dabei den Verlust einer Verbindung spätestens nach 5 Sekunden fest. Wird der Verlust einer Verbindung festgestellt, treten die beschriebenen Ereignisse in Abhängigkeit der Konfiguration des OpcClients in Aktion. Das verwendete KeepAlive-Interval kann über die KeepAlive-Eigenschaft der OpcClient-Klasse geändert werden.

Verarbeiten von Benachrichtigungen

Die folgenden Typen werden hier besprochen: OpcClient, OpcSubscription, OpcMonitoredItem und OpcNotification.

Benachrichtigungen (engl. Notifications) werden vom Client immer in Folge einer aktiven Subscription empfangen. Die Verarbeitungskette einer Notification beginnt beim Handler des zugehörigen MonitoredItem's. Von dort aus wird die Notification an die verantwortliche Subscription und schließlich zum Client der Subscription weitergereicht. Die hierbei behandelbaren Notifications liefern die vom Server erhaltenen Informationen über eine Datenänderung oder eines am Server aufgetretenen Alarm's beziehungsweise Event's.

OpcClient.NotificationReceived

- Wann wird dieses Ereignis ausgelöst? Wenn ...
  - mindestens eine Subscription vom Client beim Server eingerichtet wurde,
    - mindestens eine eingerichtete Subscription mindestens ein MonitoredItem besitzt
    - und die Filterbedingungen des Items Server-seitig erfüllt wurden.
  - das PublishingInterval der Subscription abgelaufen ist,
    - zwischenzeitlich vom Server keine neue Notification eingegangen ist
    - deshalb vom Client automatisch eine Aufforderung zur Aktualisierung der Subscription gesendet wurde
    - und der Server daraufhin mit einer leeren Antwort auf die Anforderung antwortet.
- Wie ist dieses Ereignis zu verstehen?
  - Es dient der allgemeinen Verarbeitung von Notifications.
  - Auch ohne spezifisches MonitoredItem, wie der Aufforderung zur Aktualisierung.
  - In den meisten Fällen ist die Behandlung des Ereignisses nicht notwendig, da die wirklich relevanten Notifications über die EventHandler der MonitoredItems behandelt werden.

Filtern von Ereignissen

Die folgenden Typen werden hier besprochen: OpcClient, OpcSimpleAttributeOperand, OpcFilter, OpcFilterOperand und OpcAlarmCondition.

Das folgende Beispiel abonniert alle Ereignisse, welche aktuell aktiv, jedoch noch nicht bestätigt worden sind:

```
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);
    }
});
```

Konfiguration unter UWP

Unter UWP (Universal Windows Platform) werden Anwendungen wie bei der Entwicklung einer Mobil-Anwendung streng reglementiert. Damit die UWP-App über OPC UA als Client-Anwendung agieren kann, müssen die folgenden „Capabilities“ im Projekt eingestellt werden:

- internetClient
- privateNetworkClientServer

Das zudem benötigte Anwendungszertifikat kann unter UWP nicht automatisch vom OPC UA .NET SDK erstellt werden. Weshalb manuell außerhalb der UWP-Anwendung ein Anwendungszertifikat erstellt werden muss. Dieses Zertifikat kann dann als Teil der `Assets` der Anwendung mit ausgeliefert und aus diesen geladen werden. Ein Beispiel, wie ein UWP-Projekt aufgebaut sein könnte ist hier zu finden: OPC UA .NET Samples - UWP Client

Server API

NamespaceIndex Nr. 1

Die folgenden Typen werden hier besprochen: OpcServer und OpcNodeManager.

Standardmässig befinden sich alle Nodes, im benutzerdefinierten Namespace mit der Nummer (= NamespaceIndex) zwei. Der Namespace mit der Nummer eins wird vom zugrundeliegenden Stack der OPC Foundation als „Core Namespace“ und als „Application Namespace“ beschrieben. Damit **Nodes innerhalb des Namespaces mit dem Index eins** verwaltet werden können, muss der **benutzerdefinierte OpcNodeManager den Wert der OpcServer.ApplicationUri-Eigenschaft als**

NamespaceUri verwenden.

Eine Sitzung je Benutzer

Die folgenden Typen werden hier besprochen: OpcServer, OpcAccessControlEntry und OpcSession.

Damit je verwalteten Access-Control-Entry (ACE), also je definierten Benutzer im OPC UA Server, sich ein Nutzer nur einmalig verbinden kann und erst wenn eine aktive Verbindung getrennt wurde wieder eine neue Verbindung hergestellt werden kann, müssen die verfügbaren Endpunkte für den Benutzer blockiert werden solange eine Sitzung des Nutzers aktiv ist. Wie das funktionieren kann ist hier zu sehen:

```
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();
}
```

Inhaltsverzeichnis