OPC UA SDK
Client + Server Development

Häufige Fragen

Getestet? Du willst es?

Lizenzmodell Preise Angebot Jetzt bestellen

C# ▼ |VB|

SDK

Debugger

Wieso verliert der Client im Debugger die Verbindung und was kann dagegen getan werden?

Problem: Nach kurzer Unterbrechung oder bei schrittweiser Ausführung der Anwendung verliert der Client die Verbindung und es kommt zur Ausnahme „BadServerNotConnected“ mit der Meldung: „The operation could not complete because the client is not connected to the server.“

- Kurzgefasst
- Detailliert

Die Unterschied ist semantischer Natur. Während OpcFolderNode eine Unterklasse von OpcObjectNode ist, wird ihr Einsatz wie folgt differenziert:

- **OpcObjectNode gruppiert Nodes**, welche auch **in einer physikalischen Beziehung** zueinander stehen
- **OpcFolderNode gruppiert Nodes**, welche eher **in einer logischen Beziehung** zueinander stehen

Technischer Hintergrund der Kommunikation zwischen Client und Server:

Es gibt in diesem Modell immer einen aktiven und einen passiven Teilnehmer. Vom Reverse-Connect-Verfahren abgesehen ist stets der Server der passive Teilnehmer und lauscht somit auf eigehende Verbindungen. Der Client hingegen ist der aktive Teilnehmer und verbindet sich auf einen vom Server bereitgestellten Endpunkt, um mit diesem zu kommunizieren. Geht zwischen den Teilnehmern (aus welchem Grund auch immer) die Verbindung verloren, muss vom Client zum Server logischerweise erneut eine Verbindung aufgebaut werden.

Hierbei ergeben sich jedoch folgende Fragen:

- Wie stellt ein Client fest, dass eine Verbindung verloren gegangen ist?
  - Ist diese physikalisch, also „wirklich“ getrennt?
  - Ist die Antwortzeit des Servers verzögert?
  - Ist das Netzwerk überlastet und Antworten kommen nur „schwer durch“?
- Wann stellt ein Client fest, dass die Verbindung verloren gegangen ist?
  - Ist die aktuelle Anfrage zu lange nicht beantwortet worden?
  - Ist die Antwort vom Server auf eine der letzten Anfragen „zu alt“?
  - Ist das Netzwerk und somit der Server nicht mehr erreichbar?

Diese Fragen können entsprechend Server-seitig entsprechend auch gestellt werden:

- Wie stellt ein Server fest, dass ein Client noch verbunden ist?
  - Ist die physikalische Verbindung noch „vorhanden“?
  - Ist die letzte Anfrage vom Client noch aktuell?
  - Ist das Netzwerk überlastet und Anfragen kommen nur „schwer“ durch?

- Wann stellt ein Server fest, dass eine bestimmte Client-Verbindung verloren gegangen ist?
  - Ist die letzte Anfrage schon zu lange her?
  - Ist die Anfrage vom Client auf einer der letzten Antworten „zu alt“?
  - Ist das Netzwerk und und somit der Client nicht mehr erreichbar?

Gegen einen Verbindungsverlust auf physikalischer Ebene kann kein Client oder Server etwas ausrichten. Anders sieht es auf logischer Ebene aus. Betrachtet man das OSI-Modell gibt es verschiedene Layer die zur Prüfung der „Gesundheit“ einer Verbindung verwendet werden können. Unabhängig auf welchem Layer man ansetzt, ist das Verfahren immer das gleiche: Ein Client muss Anfragen stellen und ein Server muss Anfragen beantworten, um zu 100% sicher zu sein, dass die Verbindung noch „gesund“ (genug) für die weitere Kommunikation ist.

Woher weiß nun ein Client bzw. Server, ab wann eine Verbindung als „tot“ bewertet werden kann?

1. Fall: Wenn ein Client / Server keine Anfrage / Antwort senden kann.
2. Fall: Wenn die letzte Anfrage / Antwort zu lange her und somit zu „alt“ ist.
3. Fall: Wenn keine Anfragen / Antworten gestellt / beantwortet werden.

Unabhängig welcher Fall geprüft werden soll, müssen Client als auch Server an diese Fälle bestimmte Bedingungen knüpfen. Hierfür dienen Informationen über die Zeit:

- Client-seitig bedeutet das:
  - Die Zeit, zu der eine Anfrage gesendet wurde.
  - Die Zeit, zu der eine Antwort erhalten wurde.
- Server-seitig bedeutet das:
  - Die Zeit, seit der zuletzt erhaltenen Anfrage.
  - Die Zeit, seit der zuletzt gesendeten Antwort.

Werden Anfragen und Antworten von den Teilnehmern mit dem aktuellen Zeitstempel (= Zeitpunkt an dem eine Nachricht abgesendet wird) versehen, kann der jeweils andere prüfen, ob noch eine Antwort bzw. weitere Anfrage erwartet werden kann. Ist der Zeitversatz zwischen der letzten Antwort / Anfrage zu groß, kann der jeweilige Teilnehmer von einem Verbindungsverlust ausgehen. Wie groß der dafür verwendete Zeitversatz sein darf legen Timeout-Einstellungen fest, die entweder Client- als auch Server-spezifisch oder auch zwischen Client und Server „ausgehandelt“ werden können.

ACHTUNG: Werden keine Timeout-Bedingungen für die Verbindung zwischen den Teilnehmern verwendet, wartet ein Client „ewig“ auf eine Antwort und ein Server reserviert „ewig“ einen Endpunkt für den Client. Während erster bis zur (unwahrscheinlichen) Antwort blockiert bleibt, gehen beim letzteren die Möglichen Endpunkte für weitere Clients aus und belegt somit wichtige Systemressourcen.

Unabhängig wie oder wer das Zeitfenster festlegt, muss garantiert werden, dass innerhalb des (insgesamt kleinsten) Zeitfensters von beiden Teilnehmern mindestens eine Nachricht ausgetauscht wurde. Die hierfür verwendete Logik wird als KeepAlive-Logik bezeichnet. Hierbei ist der aktive Teilnehmer verantwortlich dem passivem Teilnehmer zu signalisieren, dass die Verbindung zwischen den beiden weiterhin besteht. Hierzu definiert jedes Protokoll (das einen KeepAlive vorsieht), dass innerhalb eines bestimmten Intervalls (= Zeitfenster) entweder eine bestimmte „leichtgewichtige“ Anfrage oder überhaupt irgendeine Anfrage gestellt und mit einer zugehörigen Antwort beantwortet werden soll. Der passive Teilnehmer kann dann anhand dieser oder allgemein anhand der generell letzten Anfrage des anderen feststellen, ob der Teilnehmer „noch da“, somit „noch verbunden“ und somit die Verbindung zu diesem auch noch „gesund“ ist. Der aktive Teilnehmer kann wiederum durch die Antwort auf die Anfrage innerhalb des Zeitfensters sich ebenso sicher sein, dass die Verbindung „noch steht“.

PowerShell

Kann das SDK auch in der PowerShell verwendet werden?

Ja, ein Beispiel hierzu samt Bootstrapper existiert auf GitHub: OPC UA .NET Samples - PowerShell Client

NuGet Preview

Nach der Verwendung einer alten NuGet Preview (als Download für die eine NuGet-Source) kann nicht auf eine neue Version aktualisiert werden. Woran liegt das?

NuGet speichert alle installierten NuGet-Pakete entweder neben der Visual Studio Solution im Ordner „packages“ (im Falle von nicht SDK-Style Projekten) oder im lokalen NuGet-Cache (im Falle von SDK-Style Projekten). Damit auf die offizielle Version aktualisiert werden kann, müssen die Preview-Pakete aus der eigenen NuGet-Source und den lokalen NuGet-Cache gelöscht werden. Wo sich der lokale NuGet-Cache befindet hängt von ihrem Projekt ab.

- Wird das SDK über NuGet referenziert (mit packages.config-Datei):
  - Schließen Sie Visual Studio
  - Navigieren Sie im Explorer zum Projektverzeichnis
  - Löschen Sie das Verzeichnis „packages“ neben ihrer Solution
  - Öffnen Sie das Projekt und erstellen Sie es erneut
- Wird das SDK über NuGet referenziert (ohne packages.config-Datei):
  - Schließen Sie Visual Studio
  - Navigieren Sie zum lokalen NuGet-Cache: C:\Users\Benutzername\.nuget\packages
  - Löschen Sie das Verzeichnis „Opc.UaFx.Client“ bzw. „Opc.UaFx.Advanced“
  - Öffnen Sie das Projekt und erstellen Sie es erneut

Anschließend kann wie gewohnt über den NuGet-Client auf die aktuelle Version aktualisiert bzw. diese installiert werden.

.NET Exceptions

Beim Start meiner Anwendung erhalte ich beim Aufruf einer API des SDKs eine ''TypeInitializationException''. Woran kann das liegen?

Dieser Fehler kann auftreten, wenn das Visual Studio nach einem Update auf eine neuere Version des SDKs nicht alle verwendeten Binär-Dateien im Projektverzeichnis aktualisiert hat. Führen Sie deshalb die folgenden Schritte durch:

- Schließen Sie Visual Studio
- Navigieren Sie im Explorer zum Projektverzeichnis
- Löschen Sie alle „bin“ und „obj“ Verzeichnisse
- Öffnen Sie das Projekt und erstellen Sie es erneut

Sollte der Fehler bei der Ausführung weiterhin bestehen, dann sollten folgende Schritte das Problem lösen:

- Wird das SDK aus den ZIP Archiv referenziert:
  - wurden alle Dateien die Teil des Zielframework-Ordners im Projekt referenziert

    - wurden die Dateien aus dem korrekten Zielframework-Ordner referenziert
- Wird das SDK über NuGet referenziert (mit packages.config-Datei):
    - Schließen Sie Visual Studio
    - Navigieren Sie im Explorer zum Projektverzeichnis
    - Löschen Sie das Verzeichnis „packages“ neben ihrer Solution
    - Öffnen Sie das Projekt und erstellen Sie es erneut
- Wird das SDK über NuGet referenziert (ohne packages.config-Datei):
    - Schließen Sie Visual Studio
    - Navigieren Sie zum lokalen NuGet-Cache: C:\Users\Benutzername\.nuget\packages
    - Löschen Sie das Verzeichnis „Opc.UaFx.Client“ bzw. „Opc.UaFx.Advanced“
    - Öffnen Sie das Projekt und erstellen Sie es erneut
- Wird das SDK in einem Projekt mit app.config referenziert und eine der InnerExceptions ist eine FileLoadException:
    - app.config Datei öffnen und 'bindingRedirect'-Einträge prüfen
    - Alle Einträge zur in der Exception genannten Assembly aus der app.config entfernen
    - Erstellen Sie das Projekt erneut

Sollte keiner dieser Ansätze zur Lösung des Problems führen, senden Sie das Problem samt Details über die Exception beziehungsweise deren InnerException an support@traeger.de.

Nach ca. 30 Minuten erhalte ich eine ''LicenseException'' beim Aufruf einer Client API, wieso?

Das OPC UA .NET SDK kommt mit einer Testlizenz die je Anwendungsstart 30 Minuten uneingeschränkt zur Client- und Serverentwicklung verwendet werden kann. Sollte diese Einschränkung ihre Evaluationsmöglichkeiten einschränken, besteht die Möglichkeit eine alternative Testlizenz bei uns kostenlos zu beantragen.

Fragen Sie einfach unseren Support (via support@traeger.de) oder lassen Sie sich gleich direkt von uns beraten und offene Fragen durch unsere Entwickler klären!

Im laufenden Betrieb meines OPC UA Servers bekommen die Client Anwendungen nach ca. 30 Minuten eine Antwort mit dem Code ''BadLicenseExpired'', wieso?

Das OPC UA .NET SDK kommt mit einer Testlizenz die je Anwendungsstart 30 Minuten uneingeschränkt zur Client- und Serverentwicklung verwendet werden kann. Sollte diese Einschränkung ihre Evaluationsmöglichkeiten einschränken, besteht die Möglichkeit eine alternative Testlizenz bei uns kostenlos zu beantragen.

Fragen Sie einfach unseren Support (via support@traeger.de) oder lassen Sie sich gleich direkt von uns beraten und offene Fragen durch unsere Entwickler klären!

Client + Server

Ist es möglich in nur einer Anwendung den Client und den Server gleichzeitig zu betreiben?

Ja, das funktioniert ohne Probleme. Viele unserer Tests beruhen deshalb auch auf Codeausschnitten wie der folgende:

```
using (var server = new OpcServer()) {
    server.Start();

    using (var client = new OpcClient(server.Address)) {
        client.Connect();
        ...
    }
}
```

OPC UA + S7-Steuerungen

Warum erhalte ich nicht innerhalb der eingestellten Intervalle eine Benachrichtigung?

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

Ob ein Server die gewünschten Intervalle unterstützt beziehungsweise, ob diese Anwendung finden kann über diverse Eigenschaften geprüft werden:

- Das `PublishingInterval` kann über die `CurrentPublishingInterval`-Eigenschaft geprüft werden.
- Das `SamplingInterval` kann über die `Status.SamplingInterval`-Eigenschaft geprüft werden.

Das Hauptproblem ist häufig auch die Konfiguration des Servers. Je nach S7-Steuerung werden nur bestimmte Minimalwerte für die Einstellungen der Subscriptions/MonitoredItems unterstützt. Die folgenden Minimalwerte gelten laut Siemens:

Eigenschaft	**CPU**		
	1510, 1511, 1512, 1513	**1515, 1516, 1505S**	**1517, 1518, 1507S**
Maximale Anzahl Subscriptions mit mehr als 1000 überwachten Elementen pro Subscription über alle Sessions	10	10	10
Maximale Anzahl Subscriptions mit maximal 1000 überwachten Elementen pro Subscription über alle Sessions	20	20	20
Maximale Anzahl Subscriptions pro Session	20	20	20
Kleinstmögliches Abtastintervall	100 ms	100 ms	10 ms
Kleinstmögliches Sendeintervall	500 ms	200 ms	10 ms
Maximale Anzahl Sessions	32	48	64
Empfohlene maximale Anzahl überwachter Elemente über alle Subscriptions	1000	1000 (1505S), 2000	10000

Quelle: Welche Systemgrenzen enthält der OPC UA Server bei der S7-1500 mit der Firmware V2.8.x?

Welche Grenzwerte ein Server unterstützt kann über diverse Nodes im Pfad `/Objects/Server/ServerCapabilities` und darunter im Node `OperationLimits` geprüft werden. Eine Client-Anwendung hat auf diese Einstellungen keinen Einfluss.

Unterschiede

OpcObjectNode vs. OpcFolderNode

Was ist der Unterschied zwischen ''OpcObjectNode'' und ''OpcFolderNode'', wann sollte welcher Node verwendet werden?

Die folgenden Typen werden hier besprochen: OpcObjectNode und OpcFolderNode.

Die Unterschied ist semantischer Natur. Während OpcFolderNode eine Unterklasse von OpcObjectNode ist, wird ihr Einsatz wie folgt differenziert:

- **OpcObjectNode gruppiert Nodes**, welche auch **in einer physikalischen Beziehung** zueinander stehen
- **OpcFolderNode gruppiert Nodes**, welche eher **in einer logischen Beziehung** zueinander stehen

In der Dokumentation zum **OpcObjectNode** steht:
Definiert eine logische Einheit zur Darstellung komplexerer Informationen als OpcVariableNode. Objekte werden verwendet, um Systeme, Systemkomponenten, reale Objekte und Softwareobjekte darzustellen. Aus einer abstrakteren Sicht werden Objekte verwendet, um Variablen und andere Objekte im Adressraum zu gruppieren. Daher sollten Objekte verwendet werden, wenn einige gemeinsame Strukturen/Gruppen von Objekten und/oder Variablen beschrieben werden sollen. Einfache Objekte mit nur einem Wert (z.B. ein einfacher Wärmesensor) können ebenfalls als Variablen modelliert werden. Es sollten jedoch Erweiterungsmechanismen berücksichtigt werden (z.B. ein komplexer Wärmesensor-Subtyp könnte mehrere Werte haben) und ob dieses Objekt als Objekt in der GUI des Clients oder nur als Wert verfügbar gemacht werden soll. Wenn ein Modellierer Zweifel hat, welche Lösung das Objekt mit einer Variablen verwenden soll, sollte die Lösung mit dem Objekt bevorzugen.
In der Dokumentation zum **OpcFolderNode** steht:

Definiert einen Knoten, um den Adressraum in einer Hierarchie von Knoten zu organisieren. Sie stellen den Wurzelknoten eines Teilbaums dar und haben keine andere Semantik.
Aus dem Grund heraus, dass der OpcFolderNode eine Unterklasse des OpcObjectNode ist, gibt es keinen direkten strukturellen Unterschied zwischen den beiden Typen. Ein expliziter Unterschied ist der verwendete ReferenceType. Der OpcFolderNode verwendet den Referenztypen „Organizes“, während der OpcObjectNode den Referenztypen „HasComponent“ verwendet. Der Unterschied in den verwendeten Referenztypen spiegelt auch den Zweck der beiden bereits zuvor beschriebenen Knotentypen wieder. Vereinfacht kann somit definiert werden:

- **OpcObjectNode gruppiert Nodes**, welche auch **in einer physikalischen Beziehung** zueinander stehen
- **OpcFolderNode gruppiert Nodes**, welche eher **in einer logischen Beziehung** zueinander stehen

Nodes

OpcNodeId

Ist es möglich das Separator-Zeichen in NodeIds zu ändern?

Die folgenden Typen werden hier besprochen: OpcNodeId und OpcNominalNodeIdFactory.

Ja, dazu muss die Standard-NodeIdFactory wie folgt geändert werden:

```
OpcNodeId.Factory = new OpcNominalNodeIdFactory() {
    Separator = '.'
};
```

OpcMethodNode

Ist es möglich über einen OpcMethodNode mehrere Werte zurückzugeben?

Die folgenden Typen werden hier besprochen: OpcMethodNode und OpcArgumentAttribute.

Ja, durch die Verwendung von `return`-, `ref`- und/oder `out`-Werten. Zu beachten ist jedoch, dass `ref`-Werte als Input- und als Output-Argument verwendet werden. Die Definition eines solchen OpcMethodNode könnte wie folgt aussehen:

```
private delegate void CalculateDelegate(
        int a,
        int b,
        ref int c,
        out int additionResult,
        out int differenceResult);

private void Calculate(
        [OpcArgument("a", Description = "The left operand.")]
        int a,
        [OpcArgument("b", Description = "The right operand.")]
        int b,
        [OpcArgument("c", Description = "Factor")]
        ref int c,
        [OpcArgument("add", Description = "The result of addition.")]
        out int additionResult,
        [OpcArgument("diff", Description = "The result of difference.")]
        out int differenceResult)
{
    additionResult = (a + b) * c;
    differenceResult = (a - b) * c;
}

this.methodNode = new OpcMethodNode(
        this.folderNode,
        nameof(Calculate),
        new CalculateDelegate(this.Calculate));
```

OpcVariableNode

Gibt es eine Möglichkeit das Schreiben von Werten einer nicht-typisierten OpcVariableNode auf einen bestimmten Datentypen zu beschränken?

Die folgenden Typen werden hier besprochen: OpcVariableNode, OpcVariableValue und OpcWriteVariableValueContext.

Ja, indem über eine benutzerdefinierte Callback-Routine der zu schreibende Wert überprüft wird. Eine solche Callback-Routine könne dann wie folgt aussehen:

```
this.variableNode.WriteVariableValueCallback = this.WriteToNode;

// Deny write of a specific type of value.
private OpcVariableValue WriteToNode(
        OpcWriteVariableValueContext context,
        OpcVariableValue value)
{
    if (!(value.Value is short))
        value.Status.Update(OpcStatusCode.BadTypeMismatch);

    return value;
}
```

Ist es möglich den Wert, der für einen nicht-typisierten OpcVariableNode geschrieben werden soll, in einen bestimmten Datentypen zu konvertieren?

Die folgenden Typen werden hier besprochen: OpcVariableNode, OpcVariableValue und OpcWriteVariableValueContext.

Ja, mit Hilfe einer benutzerdefinierten Callback-Routine kann der Wert vor dem Schreiben in den gewünschten Zieldatentypen konvertiert werden. Das Vorgehen könnte in etwa so aussehen:

```
// Inline change type of value to the expected type.
private OpcVariableValue WriteToNode(
        OpcWriteVariableValueContext context,
        OpcVariableValue value)
{
    if (!(value.Value is short)) {
        value = new OpcVariableValue(
                Convert.ChangeType(value.Value, typeof(short)),
                value.Timestamp ?? DateTime.UtcNow,
                value.Status);
    }

    return value;
}
```

Optimierungen

Das SDK ist für zahlreiche Verhaltensweisen anderer OPC UA Client- und Server-Anwendungen optimiert und versucht diese selbstständig zu erkennen und sich entsprechend so zu verhalten, sodass das Ergebnis dem vom Benutzer erwarteten entspricht. Die im Folgenden explizit aufgeführten Maßnahmen und Themen werden häufig angefragt.

CVE-2021-26414

Wird die Lösung des Problems der 'Sicherheitsanfälligkeit im Windows DCOM-Server durch Umgehung von Sicherheitsfunktionen' adressiert?

Ja, ab Version 2.23.0.0 des OPC UA SDKs werden die von Microsoft empfohlenen Maßnahmen für **OPC Classic** (zur Kommunikation über DCOM bzw. **OPC DA**, **OPC HDA** und **OPC AE**) umgesetzt.

Hintergrund:

Das als **CVE-2021-26414** bekannte Problem adressiert Microsoft mit einen Leitfaden für DCOM-Client- und DCOM-Server-Anwendungen.

Als Bestandteil des Leitfadens beschreibt Microsoft die nach und nach ergriffenen Maßnahmen, um das Problem dauerhaft zu lösen. Nach Umsetzung der letzten Phase am 14. Juni 2022, kann es zu Kommunikations-Problemen zwischen DCOM-Client und DCOM-Server-Anwendungen kommen. Wovon voraussichtlich ein Teil der über **OPC Classic** kommunizierenden Anwendungen betroffen sein wird.

Zur weiteren Unterstützung, auch nach Abschluss der Maßnahmen zur Lösung des 'CVE-2021-26414'-Problems setzt das OPC UA SDK zur Kommunikation über **OPC Classic** die **von Microsoft empfohlenen Authentication Level** um. Diese Logik wurde als Teil der Version 2.23.0.0 des SDKs veröffentlicht.

Inhaltsverzeichnis