**Allgemeines:**

Bei jedem der nachfolgenden Tests habe ich den EmulationServer in der Version 0.9 verwendet. Sowohl CPS VA als auch beim EmulationServer habe ich das gleiche Informationsmodell verwendet, sprich der CPS VA liest die XML Datei „Robotics\_edited.xml“ ein (hier habe Fehler eingebaut, um die entsprechenden Prüffälle zu testen) und der EmulationServer liest die XML Datei „Robotics\_edited.xml“ ein.

**NodeId existiert nicht:**

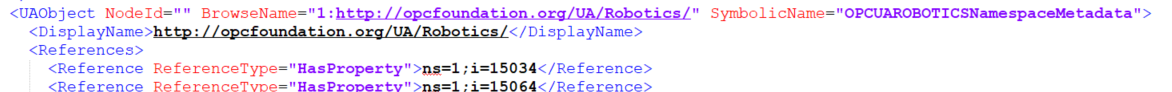
verwendete Informationsmodell: Robotics\_edited.xml

wie? NodeID einiger Knoten des XML Informationsmodells (Spezifikation) entfernen

**Test:**

Robotics\_edited.xml:

entfernte nodeId: ns=1;i=15011

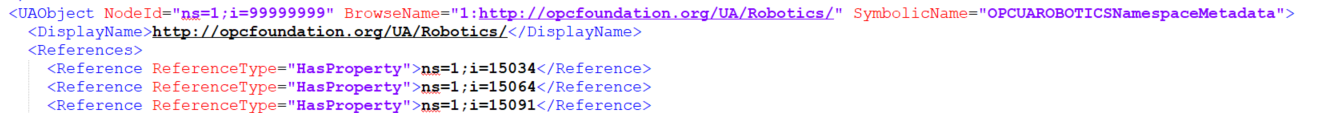


**Ergebnis:**

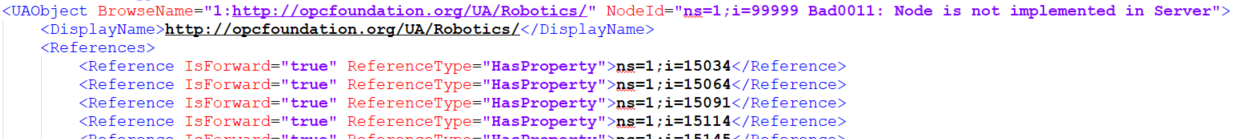
Knoten im Server wird anhand vom BrowseName gefunden.

Robotics\_edited.xml:

geänderte nodeId: ns=1;i=999999



**Ergebnis:**



Knoten wurde nicht gefunden. LogFile enthält Knoten mit NodeId und Fehlermeldung

**BrowseName existiert nicht:**

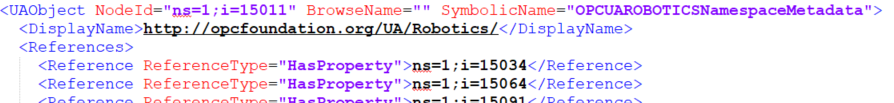
verwendete Informationsmodell: Robotics\_edited.xml

wie? BrowseName einiger Knoten des XML Informationsmodells (Spezifikation) entfernt

**Test:**

Robotics\_edited.xml

entfernte browseName: 1:http://opcfoundation.org/UA/Robotics/

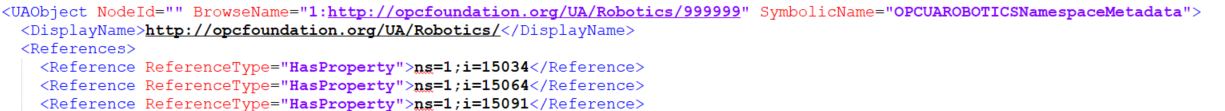


**Ergebnis:**

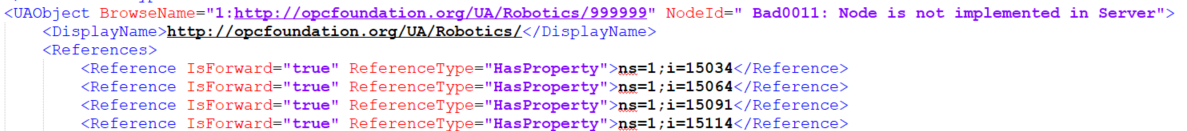
Knoten im Server wird anhand von NodeId gefunden

Robotics\_edited.xml

geänderter browseName: 1:http://opcfoundation.org/UA/Robotics/999999



**Ergebnis:**



Knoten konnte im Server nicht gefunden werden

**Pfad innerhalb der Struktur nicht gleich:**

Der CPS VA vergleich einen Knoten aus der XML mit dem entsprechenden Knoten, welcher der CPS VA anhand von NodeID oder BrowseName gefunden hat. Dementsprechend werden auch die Referenzen geprüft, welche die Struktur des Informationsmodells und der Pfade darstellen. Indirekt werden dadurch auch Pfade geprüft, jedoch lässt aktuell kein Pfad spezifizieren und testen.

**ParentNodeID ist ungleich:**

Beim Umbauen des CPS VA habe ich mich näher mit den Referenzen eines Knoten beschäftigt. Bisher habe ich angenommen das jeder Knoten genau eine Referenz mit dem Typ = Organizes, isForward=false und einer NodeID besitzt und das dieser Knoten der Elternknoten ist.

In OPC UA können aber Knoten eine Reihe von solchen Referenzen (Typ=Organizes, isForward=false + einer NodeID), demnach weiß ich aktuell nicht wie ich den ElternKnoten eines Knotens im Server identifizieren soll.

Das nachfolgende Bild soll dies etwas verdeutlichen. Jeder Pfeil stellt eine Referenz dar. Der Knoten rechts unten kann somit eine Reihe von Referenzen nach „oben“ besitzen. Jede dieser Referenz kann ein anderer Typ sein.

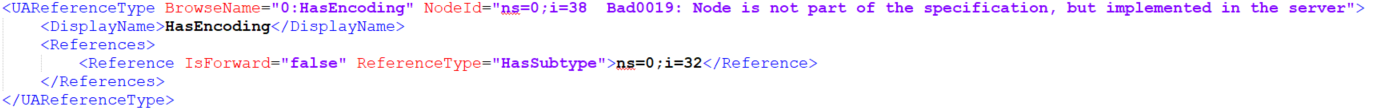


**Zusätzliche Knoten identifiziert:**

verwendete Informationsmodell: Robotics\_edited.xml

wie? das xml Informationsmodell stellt nur einen Teil der im Server implementierten Knoten dar, daher wird in der LogFile + Zeitstempel immer eine Reihe von Knoten auftauchen, die im Server implementiert sind, jedoch nicht in der Spezifikation sind.

**Ergebnis:**

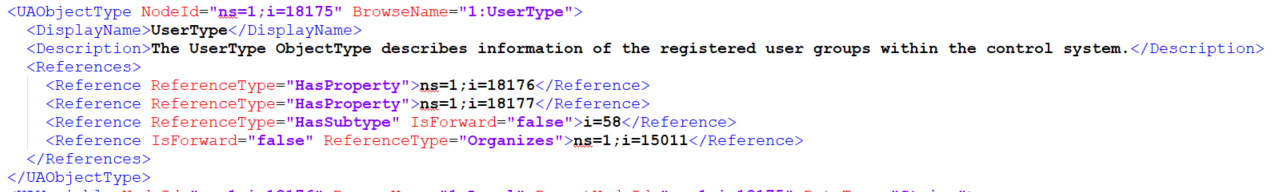


**Referenz nicht vorhanden:**

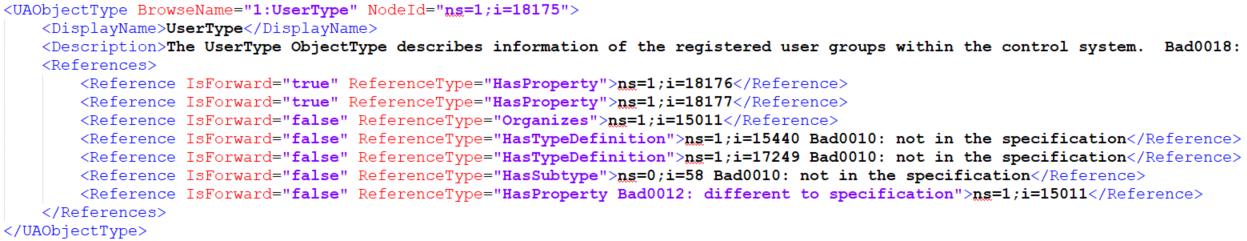
verwendete Informationsmodell: Robotics\_edited.xml

wie? Ein Knoten in der XML (Spezifikation) weist ebenfalls nur einen Teil der Referenzen auf. Der gleiche Knoten im Server hat weitere Referenzen, um z.B. den Typ des Knotens zu bestimmen. Nachfolgendes Bild zeigt einen Ausschnitt aus der LogFile (CPS VA testet Informationsmodell „Robotics\_edited.xml“ gegen EmulationServer, welcher auch mit „Robotics\_edited.xml“ läuft).

**Knoten aus der Spezifikation:**



**Ergebnis:**

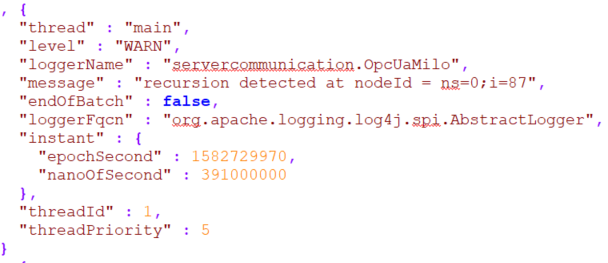


**Erkenntnis:**

Können Knoten in OPC UA mehrfach referenziert werden? Bspw. das obere Bild enthält zwei Referenzen mit der NodeID: ns=1;i=15011. Der Knoten mit ns=1;i=15011 wird dadurch zweimal referenziert. Allerdings unterscheidet der Typ der Referenz beide voneinander.

**Rekursive Referenzen:**

wie im Kapitel „Referenz nicht vorhanden“ etwas erläutert, können Knoten in OPC UA viele Referenzen in alle Richtungen besitzen. Eine Rekursion im Sinne einer Schleife ist somit erlaubt. Jedoch muss diese im CPS VA erkannt und abgefangen werden, damit der CPS VA nicht in eine Endlosschleife gerät und nicht terminiert. Um dies zu verhindern, speichert der CPS VA beim Traversieren des Server Informationsmodells jeden Knoten anhand von NodeID und BrowseName. Wenn der CPS VA bei einem bereits erkundeten Knoten ankommt, prüft dieser, ob dieser bereits erkundet wurde. Wenn ja, dann wird dieser Knoten nicht berücksichtigt. Zusätzlich wird jedes Mal wenn dieser Fall eintritt eine Fehlermeldung (mit der NodeID, wo die Rekursion aufgetreten ist) in die LogFile.json geschrieben. Das nachfolgende Bild ist ein Beispiel eines Eintrags in der Logfile.json, in dem eine Rekursion gemeldet wird.



**Attribute stimmen nicht überein:**

Durch den Umbau des CPS VA, um die Referenzen eines Knotens zu bekommen, konnte manche Attribute nicht mehr vom Server gelesen werden.

Durch den Umbau des CPS VA, um das Informationsmodells der XML einzulesen, stehen nun mehr Informationen bereit, die bisher nicht aus dem Server gelesen werden können.

Bspw.: AccessLevel, UserWriteMask, WriteMask, ArrayDimension, SymbolicName, EventNotifier, DisplayName, Extensions, Description, Documentation, ReleaseStatus, Category, AccessRestrictions, RolePermission, DataType, Translation, ValueRank, UserAccessLevel, MinimumSamplingInterval, InverseName etc.

Bisher weiß ich nicht wie ich diese Attribute eines Knotens vom Server lesen kann.

Alle Attribute, bei denen eine Informationslücke zwischen einem Knoten aus der XML und eines Knotens aus dem Server werden mit der Fehlermeldung: *„Bad0018: CPS VA cannot get this attribute from server“*

**Robustheit & Logging:**

Nach dem Umbau des CPS VA bin ich jede Klasse mit jeder Methode durchgegangen und habe mir angeschaut welche Parameter eine Methode erhält, ob diese null oder „“ o.ä. sein können und entsprechende vorher auf null oder „“ o.ä. geprüft. Gleichzeitig habe ich mir Gedanken, inwiefern diese Methode einen Einfluss auf den flüssigen bzw. sauberen Abschluss der Validierung des CPS VA hat und entsprechend mit IF-clause abgefangen.

Mein erster Ansatz des Loggings war in der Klasse LogFilemanagement den Logger zu initialisieren (das jede logger Nachicht in die LogFile.json geschrieben wird). Das hatte ich so implementiert, da ich damals Probleme hatte den Pfad zur Konfigurationsdatei des Loggers anpassbar zu gestalten. Ein anpassbarer Pfad war für die Implementierung in das TestTool von relayr notwendig. Da dies nicht mehr notwendig ist, habe ich die Konfiguration in eine Datei mit dem Namen *„Log4j2.properties“* ausgelagert, welche sich im Ordner „resources“ unter „src“ befindet. Dadurch konnte ich in jeder Java Klasse einen Logger initialisieren und Lognachrichten erstellen. Vorher hatte ich aus jeder Klasse einen statischen Aufruf in LogFileManagement mit einer Nachricht als String, um in den Logger in dieser Klasse zu verwenden.

**Problem:**

In der LogFile.json sollte zu jedem Knoten bei dem die Spezifikation zum Server abweicht eine Fehlermeldung entstehen.

Beim Traversieren des Informationsmodells des Server, wird bei einer Rekursion ebenfalls eine Fehlermeldung kreiert.

Dadurch ist die LogFile.json sehr lang. In NotePad++ kann es schon mal zu 7 000 000 Zeilen kommen bei einem Validierungsvorgang. Dadurch dauert es lange diese Datei zu öffnen, geschweige denn ist schlicht unmöglich sich in dieser Datei zurecht zu finden.