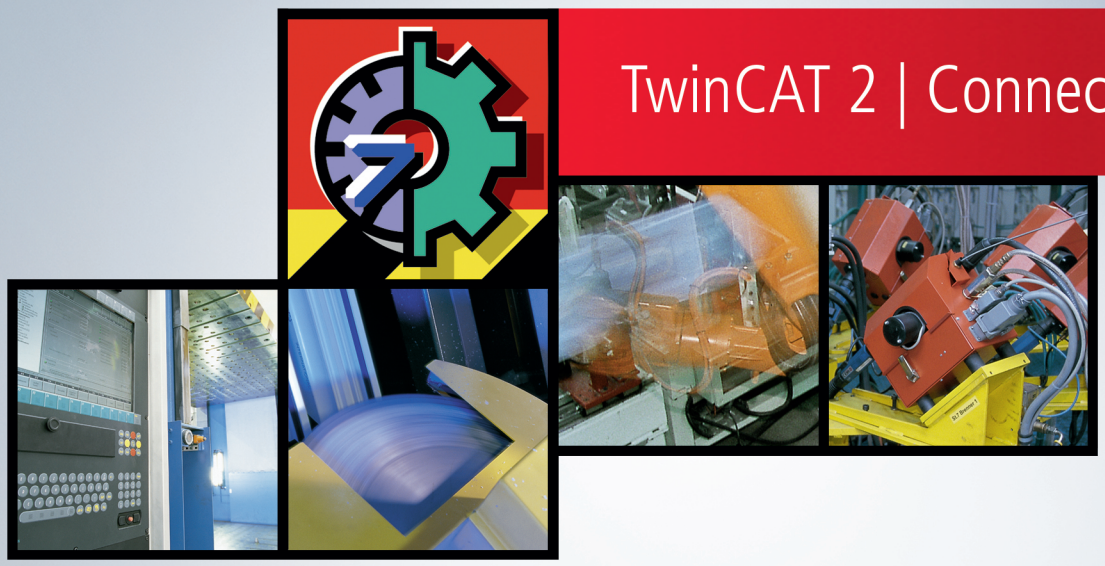
BECKHOFF New Automation Technology

Handbuch | DE

TX1000

TwinCAT 2 | ADS Java DLL



TwinCAT 2 | Connectivity

06.10.2022 | Version: 1.1

Inhaltsverzeichnis

1 Vorwort 5
1.1 Hinweise zur Dokumentation 5
1.2 Sicherheitshinweise 6
1.3 Hinweise zur Informationssicherheit 7
2 Einleitung 8
3 Einbinden in Eclipse Galileo 9
4 Einbinden in Netbeans 6.7.1 11
5 API 13
5.1 Funktionen 13
5.1.1 adsGetDllVersion 13
5.1.2 adsPortOpen 13
5.1.3 adsPortClose 13
5.1.4 adsGetLocalAddress 13
5.1.5 adsSyncWriteReq 14
5.1.6 adsSyncWriteReqArray 14
5.1.7 adsSyncReadReq 14
5.1.8 adsSyncReadReqEx 15
5.1.9 adsSyncReadWriteReq 15
5.1.10 adsSyncReadWriteReqEx 16
5.1.11 adsSyncReadDeviceInfoReq 16
5.1.12 adsSyncWriteControlReq 17
5.1.13 adsSyncReadStateReq 17
5.1.14 adsSyncAddDeviceNotificationReq 18
5.1.15 adsSyncDelDeviceNotificationReq 18
5.1.16 adsSyncSetTimeout 19
5.1.17 adsSyncGetTimeout 19
5.1.18 adsAmsRegisterRouterNotification 19
5.1.19 adsAmsUnRegisterRouterNotification 19
5.1.20 adsAmsPortEnabled 20
5.1.21 getAdsCallbackObject 20
5.2 Erweiterte Funktionen 20
5.2.1 adsPortOpenEx 20
5.2.2 adsPortCloseEx 21
5.2.3 adsGetLocalAddressEx 21
5.2.4 adsSyncWriteReqEx 21
5.2.5 adsSyncWriteReqExArray 22
5.2.6 adsSyncReadReqEx2 22
5.2.7 adsSyncReadWriteReqEx2 23
5.2.8 adsSyncReadDeviceInfoReqEx 23
5.2.9 adsSyncWriteControlReqEx 24
5.2.10 adsSyncReadStateReqEx 24
5.2.11 adsSyncAddDeviceNotificationReqEx 25
5.2.12 adsSyncDelDeviceNotificationReqEx 25

5.2.13 adsSyncSetTimeoutEx ... 26
5.2.14 adsSyncGetTimeoutEx ... 26
5.2.15 adsAmsPortEnabledEx ... 26

6 Beispiele: TcJavaToAds ... 28
6.1 Benutzen der AdsToJava.dll ... 28
6.2 Zugriff per Variablenname ... 32
6.3 Zugriff auf ein Array in der SPS ... 33
6.4 Übertragen von Strukturen an die SPS ... 34
6.5 Auslesen der SPS-Variablen Deklaration einer einzelnen Variablen ... 35
6.6 Merker synchron in die SPS schreiben ... 39
6.7 Merker synchron aus der SPS lesen ... 39
6.8 Handle einer SPS Variablen löschen ... 40
6.9 Ereignisgesteuertes Lesen ... 42
6.10 Auslesen von SMB-Werten aus dem TwinCAT I/O Treiber ... 44
6.11 Änderungen an der Symboltabelle ereignisgesteuert erkennen ... 47
6.12 ADS-Summenkommando: Holen und Freigeben von mehreren Handles ... 48
6.13 ADS-Summenkommando: Lesen und Schreiben von mehreren Variablen ... 52

7 ADS Return Codes ... 56

1 Vorwort

1.1 Hinweise zur Dokumentation

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs- und Automatisierungstechnik, das mit den geltenden nationalen Normen vertraut ist.
Zur Installation und Inbetriebnahme der Komponenten ist die Beachtung der Dokumentation und der nachfolgenden Hinweise und Erklärungen unbedingt notwendig.
Das Fachpersonal ist verpflichtet, für jede Installation und Inbetriebnahme die zu dem betreffenden Zeitpunkt veröffentliche Dokumentation zu verwenden.

Das Fachpersonal hat sicherzustellen, dass die Anwendung bzw. der Einsatz der beschriebenen Produkte alle Sicherheitsanforderungen, einschließlich sämtlicher anwendbaren Gesetze, Vorschriften, Bestimmungen und Normen erfüllt.

Disclaimer

Diese Dokumentation wurde sorgfältig erstellt. Die beschriebenen Produkte werden jedoch ständig weiter entwickelt.
Wir behalten uns das Recht vor, die Dokumentation jederzeit und ohne Ankündigung zu überarbeiten und zu ändern.
Aus den Angaben, Abbildungen und Beschreibungen in dieser Dokumentation können keine Ansprüche auf Änderung bereits gelieferter Produkte geltend gemacht werden.

Marken

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® und XPlanar® sind eingetragene und lizenzierte Marken der Beckhoff Automation GmbH.
Die Verwendung anderer in dieser Dokumentation enthaltenen Marken oder Kennzeichen durch Dritte kann zu einer Verletzung von Rechten der Inhaber der entsprechenden Bezeichnungen führen.

Patente

Die EtherCAT-Technologie ist patentrechtlich geschützt, insbesondere durch folgende Anmeldungen und Patente:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
mit den entsprechenden Anmeldungen und Eintragungen in verschiedenen anderen Ländern.

EtherCAT®

EtherCAT® ist eine eingetragene Marke und patentierte Technologie lizenziert durch die Beckhoff Automation GmbH, Deutschland

Copyright

1.2 Sicherheitshinweise

Sicherheitsbestimmungen

Beachten Sie die folgenden Sicherheitshinweise und Erklärungen!
Produktspezifische Sicherheitshinweise finden Sie auf den folgenden Seiten oder in den Bereichen Montage, Verdrahtung, Inbetriebnahme usw.

Haftungsausschluss

Die gesamten Komponenten werden je nach Anwendungsbestimmungen in bestimmten Hard- und Software-Konfigurationen ausgeliefert. Änderungen der Hard- oder Software-Konfiguration, die über die dokumentierten Möglichkeiten hinausgehen, sind unzulässig und bewirken den Haftungsausschluss der Beckhoff Automation GmbH & Co. KG.

Qualifikation des Personals

Diese Beschreibung wendet sich ausschließlich an ausgebildetes Fachpersonal der Steuerungs-, Automatisierungs- und Antriebstechnik, das mit den geltenden Normen vertraut ist.

Erklärung der Symbole

In der vorliegenden Dokumentation werden die folgenden Symbole mit einem nebenstehenden Sicherheitshinweis oder Hinweistext verwendet. Die Sicherheitshinweise sind aufmerksam zu lesen und unbedingt zu befolgen!

⚠ GEFAHR

Akute Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht unmittelbare Gefahr für Leben und Gesundheit von Personen!

⚠ WARNUNG

Verletzungsgefahr!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, besteht Gefahr für Leben und Gesundheit von Personen!

⚠ VORSICHT

Schädigung von Personen!

Wenn der Sicherheitshinweis neben diesem Symbol nicht beachtet wird, können Personen geschädigt werden!

HINWEIS

Schädigung von Umwelt oder Geräten

Wenn der Hinweis neben diesem Symbol nicht beachtet wird, können Umwelt oder Geräte geschädigt werden.

Tipp oder Fingerzeig

Dieses Symbol kennzeichnet Informationen, die zum besseren Verständnis beitragen.

1.3 Hinweise zur Informationssicherheit

Die Produkte der Beckhoff Automation GmbH & Co. KG (Beckhoff) sind, sofern sie online zu erreichen sind, mit Security-Funktionen ausgestattet, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen. Trotz der Security-Funktionen sind die Erstellung, Implementierung und ständige Aktualisierung eines ganzheitlichen Security-Konzepts für den Betrieb notwendig, um die jeweilige Anlage, das System, die Maschine und die Netzwerke gegen Cyber-Bedrohungen zu schützen. Die von Beckhoff verkauften Produkte bilden dabei nur einen Teil des gesamtheitlichen Security-Konzepts. Der Kunde ist dafür verantwortlich, dass unbefugte Zugriffe durch Dritte auf seine Anlagen, Systeme, Maschinen und Netzwerke verhindert werden. Letztere sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn entsprechende Schutzmaßnahmen eingerichtet wurden.

Zusätzlich sollten die Empfehlungen von Beckhoff zu entsprechenden Schutzmaßnahmen beachtet werden. Weiterführende Informationen über Informationssicherheit und Industrial Security finden Sie in unserem https://www.beckhoff.de/secguide.

Die Produkte und Lösungen von Beckhoff werden ständig weiterentwickelt. Dies betrifft auch die Security-Funktionen. Aufgrund der stetigen Weiterentwicklung empfiehlt Beckhoff ausdrücklich, die Produkte ständig auf dem aktuellen Stand zu halten und nach Bereitstellung von Updates diese auf die Produkte aufzuspielen. Die Verwendung veralteter oder nicht mehr unterstützter Produktversionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Hinweise zur Informationssicherheit zu Produkten von Beckhoff informiert zu sein, abonnieren Sie den RSS Feed unter https://www.beckhoff.de/secinfo.

2 Einleitung

Mit der AdsToJava.dll Bibliothek kann der eigenen Software die Kommunikation zu ADS-Geräten ermöglicht werden.

Zu beachten: Um in einem Projekt Zugriff auf die AdsToJava.dll zu haben, muss die TcJavaToAds.jar zu diesem Projekt hinzugefügt werden. Danach werden durch das Importieren des Pakets "de.beckhoff.jni.tcads" die entsprechenden Funktionen, Objekte und Konstanten verfügbar.

Die TcJavaToAds.jar befindet sich im Verzeichnis "TwinCAT\ADS Api\AdsToJava". Die AdsToJava.dll liegt im Unterverzeichnis "XP" (Eine zusätzliche WinCE-Version existiert ebenfalls, sie liegt im Unterordner "CE").

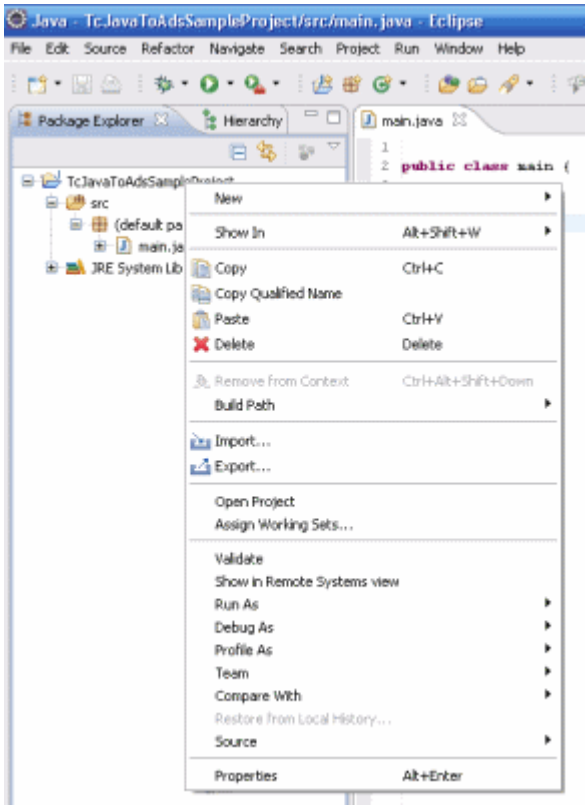
Beispiele

Hier finden Sie Artikel mit Beispielprogrammen [▶ 28] in denen die Benutzung der AdsToJava.dll in Kombination mit der TcJavaToAds.jar erklärt ist.

3 Einbinden in Eclipse Galileo

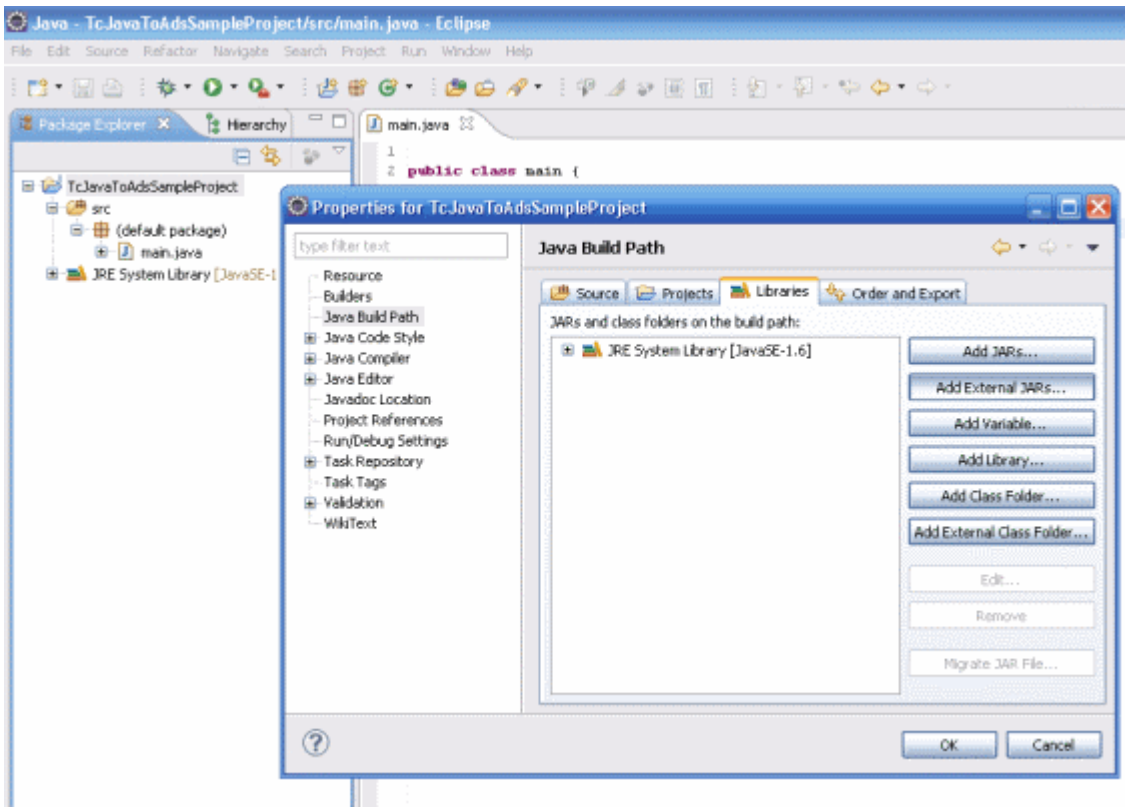
Um die AdsToJava.dll im aktuellen Eclipse-Projekt nutzen zu können, muss die TcJavaToAds.jar zu diesem hinzugefügt werden. Dieser Artikel zeigt, wie die Bibliothek hinzuzufügen ist und welche Importe aus dieser benötigt werden (Bitte beachten Sie, dass die Anweisungen auf Ihrem System leicht variiert werden müssen, falls die benutzte Version von dieser abweicht):

Zunächst muss Eclipse gestartet und ein neues Projekt angelegt werden. Danach navigieren Sie bitte zum "Package Explorer" und klicken Sie mit der rechten Maustaste auf ihr neues Projekt. In dem sich öffnenden Menü wählen Sie "Properties".



Im nächsten Schritt wählen Sie die Kategorie "Java Build Path" aus und klicken Sie auf die Schaltfläche "Add External JARs...". In dem nun erscheinenden Dateibrowser suchen Sie die Datei TcJavaToAds.jar.

Sie befindet sich im Verzeichnis "TwinCAT\ADS Api\AdsToJava".



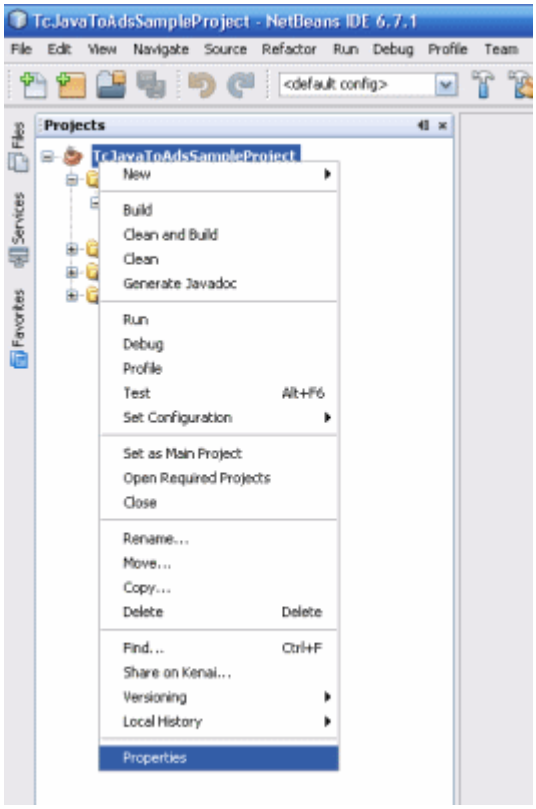
Zuletzt müssen die Pakete deren Komponenten Sie nutzen wollen importiert werden. Dazu finden Sie dokumentierte Beispielprogramme [▶ 28] im Information System.

```
import de.beckhoff.jni.AdsConstants;
import de.beckhoff.jni.Convert;
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.JNILong;
import de.beckhoff.jni.JNIBool;

import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.tcads.AdsCallbackObject;
import de.beckhoff.jni.tcads.AdsDevName;
import de.beckhoff.jni.tcads.AdsNotificationAttrib;
import de.beckhoff.jni.tcads.AdsNotificationHeader;
import de.beckhoff.jni.tcads.AdsState;
import de.beckhoff.jni.tcads.AdsSymbolEntry;
import de.beckhoff.jni.tcads.AdsVersion;
import de.beckhoff.jni.tcads.AmsAddr;
import de.beckhoff.jni.tcads.AmsNetId;
import de.beckhoff.jni.tcads.CallbackListenerAdsRouter;
import de.beckhoff.jni.tcads.CallbackListenerAdsState;
```

4 Einbinden in Netbeans 6.7.1

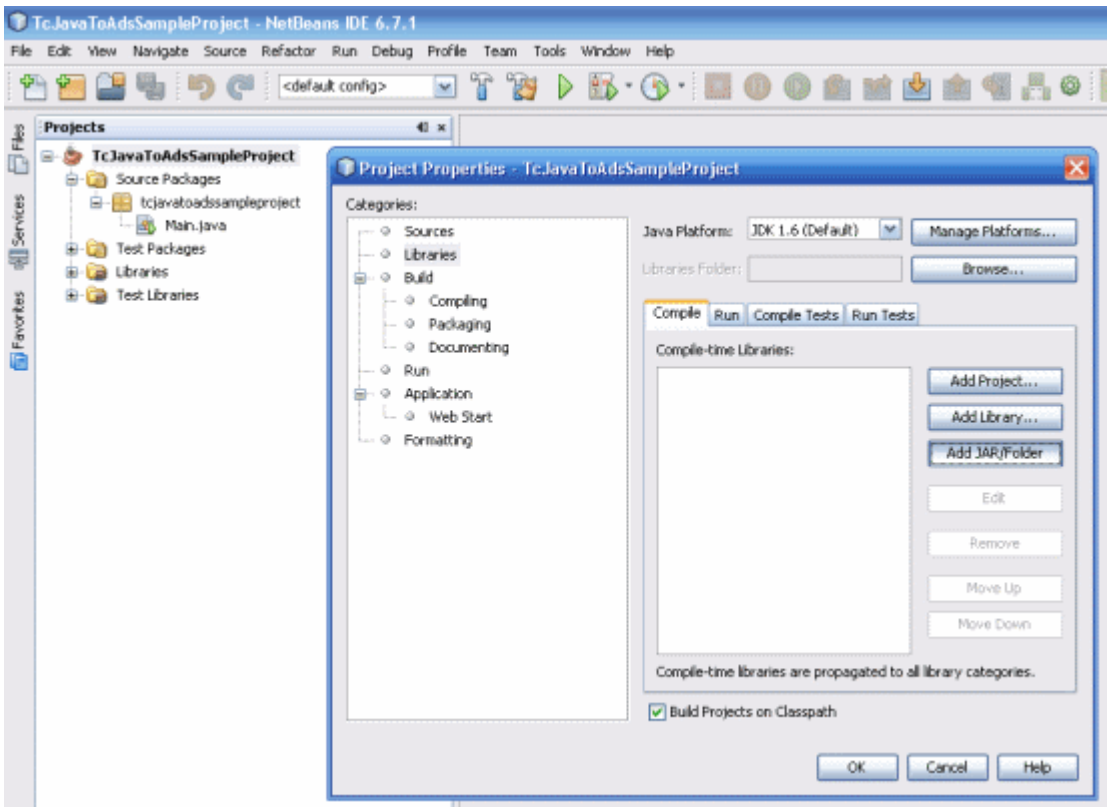
Um die AdsToJava.dll im aktuellen Netbeans-Projekt nutzen zu können, muss die TcJavaToAds.jar zu diesem hinzugefügt werden. Dieser Artikel zeigt, wie die Bibliothek hinzuzufügen ist und welche Importe aus dieser benötigt werden (Bitte beachten Sie, dass die Anweisungen auf Ihrem System leicht variiert werden müssen, falls die benutzte Version von dieser abweicht):

Zunächst muss Netbeans gestartet muss ein neues Projekt angelegt werden. Danach navigieren Sie bitte zu "Project" und klicken Sie mit der rechten Maustaste auf ihr neues Projekt. In dem sich öffnenden Menü wählen Sie "Properties".



Im nächsten Schritt wählen Sie die Kategorie "Libraries" aus und klicken Sie auf die Schaltfläche "Add JAR/ Folder". In dem nun erscheinenden Dateibrowser suchen Sie die Datei TcJavaToAds.jar.

Sie befindet sich im Verzeichnis "TwinCAT\ADS Api\AdsToJava".



Zuletzt müssen die Pakete deren Komponenten Sie nutzen wollen importiert werden. Dazu finden Sie dokumentierte Beispielprogramme [▶ 28] im Information System.

```
import de.beckhoff.jni.AdsConstants;
import de.beckhoff.jni.Convert;
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.JNILong;
import de.beckhoff.jni.JNIBool;

import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.tcads.AdsCallbackObject;
import de.beckhoff.jni.tcads.AdsDevName;
import de.beckhoff.jni.tcads.AdsNotificationAttrib;
import de.beckhoff.jni.tcads.AdsNotificationHeader;
import de.beckhoff.jni.tcads.AdsState;
import de.beckhoff.jni.tcads.AdsSymbolEntry;
import de.beckhoff.jni.tcads.AdsVersion;
import de.beckhoff.jni.tcads.AmsAddr;
import de.beckhoff.jni.tcads.AmsNetId;
import de.beckhoff.jni.tcads.CallbackListenerAdsRouter;
import de.beckhoff.jni.tcads.CallbackListenerAdsState;
```

5 API

5.1 Funktionen

5.1.1 adsGetDllVersion

Liefert die Versionnummer, Revisionnummer und die Buildnummer der ADS-DLL zurück.

```
AdsVersion adsGetDllVersion ();
```

Parameter

Return value

AdsVersion: Gibt eine AdsVersion Objekt, mit entsprechend gesetzter Versionsnummer, Revisionsnummer und Buildnummer zurück.

5.1.2 adsPortOpen

Stellt eine Verbindung (Kommunikationsport) zum TwinCAT Messagerouter her.

Wenn kein TwinCAT-MessageRouter vorhanden ist wird die Funktion AdsPortOpenEx fehlschlagen.

```
long adsPortOpen ();
```

Parameter

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.3 adsPortClose

Die Verbindung (Kommunikationsport) zum TwinCAT Messagerouter wird beendet.

```
long adsPortClose ();
```

Parameter

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.4 adsGetLocalAddress

Liefert die lokale NetId und die eigene Portnummer zurück.

```
long adsGetLocalAddress (AmsAddr lj_AmsAddr);
```

Parameter

- **AmsAddr lj_AmsAddr**: AmsAddr Objekt, welches nach dem Aufruf die lokale NetId und Portnummer enthält.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.5 adsSyncWriteReq

Schreibt Daten synchron in ein ADS-Gerät.

```
long adsSyncWriteReq(
  AmsAddr        lj_AmsAddrlonglj_indexGroup,
  long           lj_indexOffset,
  long           lj_length,
  JNIByteBuffer  lj_pData
);
```

Parameter

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **long** *lj_length*: Länge der Daten in Byte, die in den ADS-Server geschrieben werden.
- **JNIByteBuffer** *lj_pData*: JNIByteBuffer, dessen Daten in den ADS Server geschrieben werden.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.6 adsSyncWriteReqArray

Schreibt Daten synchron in ein ADS-Gerät.

```
long adsSyncWriteReqArray(
  AmsAddr  lj_AmsAddr,
  long     lj_indexGroup,
  long     lj_indexOffset,
  long     lj_length,
  byte[]   lj_pData
);
```

Parameter

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **long** *lj_length*: Länge der Daten in Byte, die in den ADS-Server geschrieben werden.
- **byte[]** *lj_pData*: ein Byte-Puffer, dessen Daten in den ADS Server geschrieben werden.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.7 adsSyncReadReq

Liest Daten synchron aus einem ADS-Server.

```
long adsSyncReadReq(
  AmsAddr        lj_AmsAddr,
  long           lj_indexGroup,
```

```
  long           lj_indexOffset,
  long           lj_length,
  JNIByteBuffer  lj_pData
);
```

Parameter

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_length*: Länge der Daten in Byte.
- **JNIByteBuffer** *lj_pData*: JNIByteBuffer, der die zu lesenden Daten aufnimmt.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.8 adsSyncReadReqEx

Liest Daten synchron aus einem ADS-Server.

Der zusätzliche JNILong-Buffer beinhaltet nach dem Aufruf die Zahl der gelesenen Bytes.

```
long adsSyncReadReqEx(
  AmsAddr        lj_AmsAddr,
  long           lj_indexGroup,
  long           lj_indexOffset,
  long           lj_length,
  JNIByteBuffer  lj_pData,
  JNILong        lj_pBytesRead
);
```

Parameter

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_length*: Länge der Daten in Byte.
- **JNIByteBuffer** *lj_pData*: JNIByteBuffer, der die zu lesenden Daten aufnimmt.
- **JNILong**: *lj_pBytesRead* JNILong-Buffer beinhaltet nach dem Aufruf die Zahl der gelesenen Bytes.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.9 adsSyncReadWriteReq

Schreibt Daten synchron in ein ADS-Server und bekommt von dem ADS-Gerät Daten wieder zurück.

```
long adsSyncReadWriteReq(
  AmsAddr        lj_AmsAddr,
  long           lj_indexGroup,
  long           lj_indexOffset,
  long           lj_lengthRead,
  JNIByteBuffer  lj_pDataRead,
  long           lj_lengthWrite,
  JNIByteBuffer  lj_pDataWrite
);
```

Parameter

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_lengthRead*: Länge der Daten in Byte, die das ADS-Gerät zurückliefert.
- **JNIByteBuffer**: *lj_pDataRead* Buffer mit Daten, die das ADS-Gerät zurückliefert.
- **long**: *lj_lengthWrite* Länge der Daten in Byte, die in das ADS-Gerät geschrieben werden.
- **JNIByteBuffer**: *lj_pDataWrite* Buffer mit Daten, die in das ADS-Gerät geschrieben werden.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.10 adsSyncReadWriteReqEx

Schreibt Daten synchron in einen ADS-Server und bekommt von dem ADS-Gerät Daten wieder zurück (erweiterte Funktion).

Der zusätzliche JNILong-Buffer beinhaltet nach dem Aufruf die Zahl der gelesenen Bytes.

```
long adsSyncReadWriteReqEx(
  AmsAddr        lj_AmsAddr,
  long           lj_indexGroup,
  long           lj_indexOffset,
  long           lj_lengthRead,
  JNIByteBuffer  lj_pDataRead,
  long           lj_lengthWrite,
  JNIByteBuffer  lj_pDataWrite,
  JNILong        lj_pBytesRead
);
```

Parameter

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_lengthRead*: Länge der Daten in Byte, die das ADS-Gerät zurückliefert.
- **JNIByteBuffer**: *lj_pDataRead* Puffer mit Daten, die das ADS-Gerät zurückliefert.
- **long**: *lj_lengthWrite* Länge der Daten in Byte, die in das ADS-Gerät geschrieben werden.
- **JNIByteBuffer**: *lj_pDataWrite* Puffer mit Daten, die in das ADS-Gerät geschrieben werden.
- **JNILong**: *lj_pBytesRead* JNILong-Buffer beinhaltet nach dem Aufruf die Zahl der gelesenen Bytes.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.11 adsSyncReadDeviceInfoReq

Liest die Bezeichnung und die Versionsnummer von einem ADS-Server.

```
long adsSyncReadDeviceInfoReq(
  AmsAddr     lj_AmsAddr,
  AdsDevName  lj_pDevName,
  AdsVersion  lj_pVersion
);
```

Parameter

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **AdsDevName** *lj_pDevName*: AdsDeviceName Objekt, welches nach dem Aufruf den ADS-Gerätenamen enthält.
- **AdsVersion** *lj_pVersion*: AdsDeviceName Objekt, welches nach dem Aufruf die verwendete Versionsnummer, Revisionsnummer und Buildnummer enthält.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.12 adsSyncWriteControlReq

Ändert den ADS-Status und den Geräte-Status von einem ADS-Server.

```
long adsSyncWriteControlReq(
  AmsAddr        lj_AmsAddr,
  int            lj_adsState,
  int            lj_deviceState,
  long           lj_length,
  JNIByteBuffer  lj_pData
);
```

Parameter

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long** *lj_adsState*: Neuer ADS-Status.
- **long** *lj_deviceState*: Neuer Geräte-Status.
- **long** *lj_length*: Länge der Daten in Byte, die zusätzlich in den ADS-Server geschrieben werden.
- **JNIByteBuffer**: *lj_pDataWrite* JNIByteBuffer mit Daten, die zusätzlich in den ADS-Server geschrieben werden.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

Comments

In addition to change the ADS status and the device status, it is also possible to send data to the ADS server in order to transfer further information. In the current ADS devices (PLC, NC, ...) this data has no further effect. Any ADS device can inform another ADS device of its current state. A distinction is drawn here between the status of the device itself (DeviceState) and the status of the ADS interface of the ADS device (AdsState). The states that the ADS interface can adopt are laid down in the ADS specification.

5.1.13 adsSyncReadStateReq

Liest den ADS-Status und den Geräte-Status von einem ADS-Server.

```
long adsSyncReadStateReq(
  AmsAddr   lj_AmsAddr,
  AdsState  lj_nAdsState,
  AdsState  lj_nDeviceState
);
```

Parameter

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **AdsState** *lj_nAdsState*: AdsState welcher nach dem Aufruf den aktuellen Ads-Status enthält

- **AdsState** *lj_nDeviceState*: AdsState welcher nach dem Aufruf den aktuellen Geräte-Status enthält

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.14 adsSyncAddDeviceNotificationReq

Innerhalb eines ADS-Servers (z.B. SPS) wird eine Notification definiert. Beim Eintreten bestimmter Ereignisse, wird eine Funktion (Callbackfunktion) im ADS-Client aufgerufen.

Ein AdsCallbackObject muss erstellt werden, um Informationen zu den gewünschten Ereignissen erhalten zu können. Diesem muss ein Listener zugewiesen werden, welcher das CallbackListenerAdsState-Interface implementiert. Der Listener wird aufgerufen, sobald das Ereignis eintritt.

```
long adsSyncAddDeviceNotificationReq(
  AmsAddr                lj_AmsAddr,
  long                   lj_indexGroup,
  long                   lj_indexOffset,
  AdsNotificationAttrib  lj_pNoteAttrib,
  long                   lj_hUser,
  JNILong                lj_pNotification
);
```

Parameter

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **AdsNotificationAttrib** *lj_pNoteAttrib*: AdsNotificationAttrib Objekt, welches zusaätzliche Informationen zur Ereignismeldung enthält.
- **long** *lj_hUser*: Benutzer-Handle
- **JNILong** *lj_pNotification*: JNILong-Buffer, welcher nach dem Aufruf die Notification-Nummer beinhaltet.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.15 adsSyncDelDeviceNotificationReq

Eine zuvor definierte Notification wird in einem ADS-Server gelöscht.

Dies hat keinen Effekt auf das AdsCallbackObjekt.

```
long adsSyncDelDeviceNotificationReq(
  AmsAddr  lj_AmsAddr,
  JNILong  lj_hNotification
);
```

Parameter

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **JNILong** *lj_hNotification*: JNILong Objekt, welches die zu löschende Notification-Nummer enthält.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.16 adsSyncSetTimeout

Verändert die Timeoutzeit für die ADS-Funktionen. Der Standardwert ist 5000ms.

```
long adsSyncSetTimeout (long lj_nMs);
```

Parameter

- **long** *lj_nMs*: Timeoutzeit in ms.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.17 adsSyncGetTimeout

Liefert die aktuell gesetzte Timeoutzeit für ADS-Aufrufe. Der Standardwert beträgt 5000 ms.

```
long adsSyncGetTimeout (JNILong lj_pMs);
```

Parameter

- **JNILong** *lj_pMs*: JNILong Objekt, welches nach dem Aufruf die aktuelle Timeoutzeit beinhaltet.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.18 adsAmsRegisterRouterNotification

Mit Hilfe der Funktion AdsAmsRegisterNotificationReq() kann eine Statusänderung des TwinCAT-Routers erkannt werden. Bei jeder Statusänderung wird die angegebende Callback-Funktion aufgerufen. Durch die Funktion AdsAmsUnRegisterNotification() wird die Statusüberwachung des Routers wieder beendet.

Ein AdsCallbackObject muss erstellt werden, um Informationen zu den gewünschten Ereignissen erhalten zu können. Diesem muss ein Listener zugewiesen werden, welcher das CallbackListenerAdsRouter-Interface implementiert. Der Listener wird aufgerufen, sobald das Ereignis eintritt.

```
long adsAmsRegisterRouterNotification ();
```

Parameter

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

Sehen Sie dazu auch

adsAmsUnRegisterRouterNotification [▶ 19]

5.1.19 adsAmsUnRegisterRouterNotification

Durch die Funktion AdsAmsUnRegisterNotification() wird die Statusüberwachung des TwinCAT-Routers beendet. Siehe auch AdsAmsRegisterNotificationReq() [▶ 19].

Dies hat keinen Effekt auf das AdsCallbackObjekt.

```
long adsAmsUnRegisterRouterNotification ();
```

Parameter

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.20 adsAmsPortEnabled

Liefert einen Wahrheitswert der angibt, ob der übergebene Port geöffnet ist.

```
long adsAmsPortEnabled(JNIBool lj_pEnabled);
```

Parameter

- **JNIBool**: *j_pEnabled* JNIBool-Buffer, welcher nach dem Aufruf den Wahrheitswert enthält.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.1.21 getAdsCallbackObject

Gibt das zugehörige AdsCallbackObjekt zurück.

```
AdsCallbackObject getAdsCallbackObject ();
```

Parameter

Return value

AdsCallbackObject: Das AdsCallbackObject Objekt

5.2 Erweiterte Funktionen

Mit den bisherigen Funktionen konnte nur ein ADS-Port pro Prozess erzeugt werden. Dies ist vor allem für Mulithreaded Anwendungen nicht ausreichend, da die einzelnen ADS Kommandos sich gegenseitig blockieren.

Mit den neuen Funktionen ist es nun möglich mehr als einen Port zu verwenden. Damit könnte z.B. pro Thread ein ADS-Port verwendet werden. Mit Hilfe der Funktion AdsPortOpenEx können neue Ports geöffnet werden. Die zurückgelieferte Port-Nummer wird dann als Parameter an die einzelnen Sync-Funktionen übergeben.

5.2.1 adsPortOpenEx

Stellt eine Verbindung (Kommunikationsport) zum TwinCAT Messagerouter her. Im Gegensatz zu AdsPortOpen wird jedesmal ein neuer ADS-Port geöffnet. Den als threadsicher markierten Methoden wird die von AdsPortOpenEx zurückgegebene Port-Nummer als Parameter übergeben.

Wenn kein TwinCAT-MessageRouter vorhanden ist wird die Funktion AdsPortOpenEx fehlschlagen.

```
long adsPortOpenEx ();
```

Parameter

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.2 adsPortCloseEx

Die Verbindung (Kommunikationsport) zum TwinCAT Messagerouter wird beendet. Der Port der geschlossen werden soll, muss vorher durch einen Aufruf von AdsPortOpenEx geöffnet worden sein.

```
long adsPortCloseEx (
  long lj_port
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.3 adsGetLocalAddressEx

Liefert die lokale NetId und die eigene Portnummer zurück (threadsicher).

```
long adsGetLocalAddressEx (
  long     lj_port,
  AmsAddr  lj_AmsAddr
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **AmsAddr** *lj_AmsAddr*: AmsAddr Objekt, welches nach dem Aufruf die lokale NetId und Portnummer enthält.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.4 adsSyncWriteReqEx

Schreibt Daten synchron in ein ADS-Gerät (threadsicher).

```
long adsSyncWriteReqEx (
  long           lj_portAmsAddrlj_AmsAddrlonglj_indexGroup,
  long           lj_indexOffset,
  long           lj_length,
  JNIByteBuffer  lj_pData
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.

- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **long** *lj_length*: Länge der Daten in Byte, die in den ADS-Server geschrieben werden.
- **JNIByteBuffer** *lj_pData*: JNIByteBuffer, dessen Daten in den ADS Server geschrieben werden.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.5 adsSyncWriteReqExArray

Schreibt Daten synchron in ein ADS-Gerät (threadsicher).

```
long adsSyncWriteReqExArray (
  longl   j_portAmsAddrlj_AmsAddrlonglj_indexGroup,
  longl   j_indexOffset,
  longl   j_length,
  byte[]  lj_pData
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **long** *lj_length*: Länge der Daten in Byte, die in den ADS-Server geschrieben werden.
- **byte[]** *lj_pData*: ein Byte-Array-Buffer, dessen Daten in den ADS Server geschrieben werden.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.6 adsSyncReadReqEx2

Liest Daten synchron aus einem ADS-Server (threadsicher).

Der zusätzliche JNILong-Buffer beinhaltet nach dem Aufruf die Zahl der gelesenen Bytes.

```
long adsSyncReadReqEx2 (
  long           lj_portAmsAddrlj_AmsAddr,
  long           lj_indexGroup,
  long           lj_indexOffset,
  long           lj_length,
  JNIByteBuffer  lj_pData,
  JNILong        lj_pBytesRead
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_length*: Länge der Daten in Byte.

- **JNIByteBuffer** *lj_pData*: JNIByteBuffer, der die zu lesenden Daten aufnimmt.
- **JNILong**: *lj_pBytesRead* JNILong-Buffer beinhaltet nach dem Aufruf die Zahl der gelesenen Bytes.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.7 adsSyncReadWriteReqEx2

Schreibt Daten synchron in ein ADS-Server und bekommt von dem ADS-Gerät Daten wieder zurück (threadsicher).

Der zusätzliche JNILong-Buffer beinhaltet nach dem Aufruf die Zahl der gelesenen Bytes.

```
long adsSyncReadWriteReqEx2 (
  long           lj_portAmsAddrlj_AmsAddr,
  long           lj_indexGroup,
  long           lj_indexOffset,
  long           lj_lengthRead,
  JNIByteBuffer  lj_pDataRead,
  long           lj_lengthWrite,
  JNIByteBuffer  lj_pDataWrite,
  JNILong        lj_pBytesRead);
```

Parameter

- **long**: *lj_port* Portnummer eines Ads-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_lengthRead*: Länge der Daten in Byte, die das ADS-Gerät zurückliefert.
- **JNIByteBuffer**: *lj_pDataRead* Puffer mit Daten, die das ADS-Gerät zurückliefert.
- **long**: *lj_lengthWrite* Länge der Daten in Byte, die in das ADS-Gerät geschrieben werden.
- **JNIByteBuffer**: *lj_pDataWrite* Puffer mit Daten, die in das ADS-Gerät geschrieben werden.
- **JNILong**: *lj_pBytesRead* JNILong-Buffer beinhaltet nach dem Aufruf die Zahl der gelesenen Bytes.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.8 adsSyncReadDeviceInfoReqEx

Liest die Bezeichnung und die Versionsnummer von einem ADS-Server (threadsafe).

```
long adsSyncReadDeviceInfoReqEx (
  long        lj_port,
  AmsAddr     lj_AmsAddr,
  AdsDevName  lj_pDevName,
  AdsVersion  lj_pVersion
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **AdsDevName** *lj_pDevName*: AdsDeviceName Objekt, welches nach dem Aufruf den ADS-Gerätenamen enthält.

- **AdsVersion** *lj_pVersion*: AdsDeviceName Objekt, welches nach dem Aufruf die verwendete Versionsnummer, Revisionsnummer und Buildnummer enthält.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.9 adsSyncWriteControlReqEx

Ändert den ADS-Status und den Geräte-Status von einem ADS-Server (threadsicher).

```
long adsSyncWriteControlReqEx (
  long           lj_port,
  AmsAddr        lj_AmsAddr,
  int            lj_adsState,
  int            lj_deviceState,
  long           lj_length,
  JNIByteBuffer  lj_pData
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long** *lj_adsState*: Neuer ADS-Status.
- **long** *lj_deviceState*: Neuer Geräte-Status.
- **long** *lj_length*: Länge der Daten in Byte, die in den ADS-Server geschrieben werden.
- **byte[]** *lj_pData*: ein JNIByteBuffer-Objekt, dessen Daten zusätzlich in den ADS Server geschrieben werden.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

Hinweise

Neben der Änderung des ADS-Status und des Gerätestatus ist es auch möglich, Daten an den ADS-Server zu senden, um weitere Informationen zu übertragen. In den aktuellen ADS-Geräten (PLC, NC, ...) haben diese Daten keine weiteren Auswirkungen. Jedes ADS-Gerät kann ein anderes ADS-Gerät über seinen aktuellen Zustand informieren. Dabei wird unterschieden zwischen dem Status des Gerätes selbst (DeviceState) und dem Status der ADS-Schnittstelle des ADS-Gerätes (AdsState). Die Zustände, die die ADS-Schnittstelle annehmen kann, sind in der ADS-Spezifikation festgelegt.

5.2.10 adsSyncReadStateReqEx

Liest den ADS-Status und den Geräte-Status von einem ADS-Server (threadsicher).

```
long adsSyncReadStateReqEx (
  long      lj_port,
  AmsAddr   lj_AmsAddr,
  AdsState  lj_nAdsState,
  AdsState  lj_nDeviceState
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.

- **AdsState** *lj_nAdsState*: AdsState welcher nach dem Aufruf den aktuellen Ads-Status enthält
- **AdsState** *lj_nDeviceState*: AdsState welcher nach dem Aufruf den aktuellen Geräte-Status enthält

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.11 adsSyncAddDeviceNotificationReqEx

Innerhalb eines ADS-Servers (z.B. SPS) wird eine Notification definiert. Beim Eintreten bestimmter Ereignisse, wird eine Funktion (Callbackfunktion) im ADS-Client aufgerufen (threadsicher).

Ein AdsCallbackObject muss erstellt werden, um Informationen zu den gewünschten Ereignissen erhalten zu können. Diesem muss ein Listener zugewiesen werden, welcher das CallbackListenerAdsState-Interface implementiert. Der Listener wird aufgerufen, sobald das Ereignis eintritt.

```
long adsSyncAddDeviceNotificationReqEx (
  long                  lj_port,
  AmsAddr               lj_AmsAddr,
  long                  lj_indexGroup,
  long                  lj_indexOffset,
  AdsNotificationAttrib lj_pNoteAttrib,
  long                  lj_hUser,
  JNILong               lj_pNotification
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **AdsNotificationAttrib** *lj_pNoteAttrib*: AdsNotificationAttrib Objekt, welches zusaätzliche Informationen zur Ereignismeldung enthält.
- **long** *lj_hUser*: Benutzer-Handle
- **JNILong** *lj_pNotification*: JNILong-Buffer, welcher nach dem Aufruf die Notification-Nummer beinhaltet.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.12 adsSyncDelDeviceNotificationReqEx

Eine zuvor definierte Notification wird in einem ADS-Server gelöscht (threadsicher).

Dies hat keinen Effekt auf das AdsCallbackObjekt.

```
long adsSyncDelDeviceNotificationReqEx (
  long     lj_port,
  AmsAddr  lj_AmsAddr,
  JNILong  lj_hNotification
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **AmsAddr** *lj_AmsAddr*: Objekt mit NetId und Portnummer vom ADS-Server.
- **JNILong** *lj_hNotification*: JNILong Objekt, welches die zu löschende Notification-Nummer enthält.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.13 adsSyncSetTimeoutEx

Verändert die Timeoutzeit für die ADS-Funktionen. Der Standardwert ist 5000ms (threadsicher).

```
long adsSyncSetTimeoutEx (
  long  lj_port,
  long  lj_nMs
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **long** *lj_nMs*: Timeoutzeit in ms.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.14 adsSyncGetTimeoutEx

Liefert die aktuell gesetzte Timeoutzeit für ADS-Aufrufe. Der Standardwert beträgt 5000 ms (threadsicher).

```
long adsSyncGetTimeoutEx (
  long     lj_port,
  JNILong  lj_pMs
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **JNILong** *lj_pMs*: JNILong Objekt, welches nach dem Aufruf die aktuelle Timeoutzeit beinhaltet.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

5.2.15 adsAmsPortEnabledEx

Liefert einen Wahrheitswert der angibt, ob der übergebene Port geöffnet ist (threadsicher).

```
long adsAmsPortEnabledEx (
  long     lj_port,
  JNIBool  lj_pEnabled
);
```

Parameter

- **long**: *lj_port* Portnummer eines ADS-Ports, der zuvor mit AdsPortOpenEx oder AdsPortOpen geöffnet worden ist.
- **JNIBool**: *j_pEnabled* JNIBool-Buffer, welcher nach dem Aufruf den Wahrheitswert enthält.

Rückgabewert

long: Gibt den Fehlerstatus der Funktion zurück.

6 Beispiele: TcJavaToAds

6.1 Benutzen der AdsToJava.dll

Callback-Funktionen

Falls Callback-Funktionen benutzt werden, muss sichergestellt sein, dass die AdsToJava.dll in der Version 1.1.0.2 oder größer installiert ist.

Aufgabe

SPS-Variablen mit einer JAava-Applikation lesen unter der VErwendung der Bibliothek AdsToJava.dll

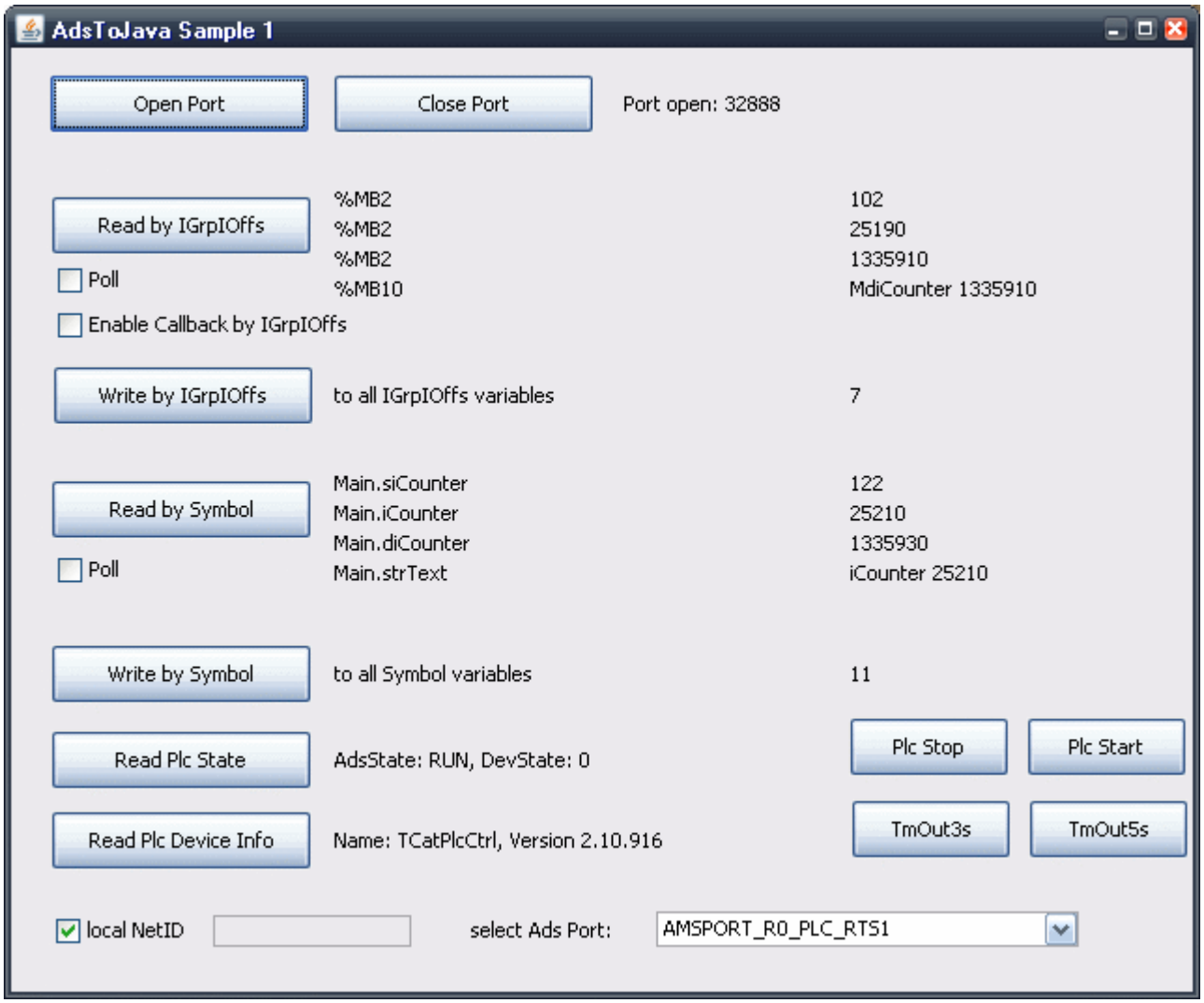
Beschreibung

In diesem Beispiel werden die grundlegenden Funktionalitäten der Java Version des ADS-Stacks vorgestellt.

Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample01.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

```
javac -classpath "lib/TcJavaToAds.jar;classes" -d . *.java
java -classpath "lib/TcJavaToAds.jar;classes" sample1/Application1
```

download sample: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475090187.zip



CallAdsFuncs.java

```
import de.beckhoff.jni.Convert;
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.JNILong;
import de.beckhoff.jni.tcads.AmsAddr;
import de.beckhoff.jni.tcads.AdsVersion;
import de.beckhoff.jni.tcads.AdsCallDllFunction;

public class CallAdsFuncs {
  public CallAdsFuncs() {
  }

  long port = 0;
  AmsAddr addr= new AmsAddr();
  long hUser = 0;
  JNILong pNotification = new JNILong();

  public long openPort(boolean localNetAddr, String txtNetString, int Prt) {
      if (port == 0) {
    AdsVersion lAdsVersion = AdsCallDllFunction.adsGetDllVersion();
    System.out.println("AdsVersion: " + new Integer(lAdsVersion.getVersion())
               + "." + new Integer(lAdsVersion.getRevision())
               + "." + lAdsVersion.getBuild());

    port = AdsCallDllFunction.adsPortOpen();
    //System.out.println("Port: " + port);

    if (localNetAddr == true) {
      addr.setNetIdString(txtNetString);
    }
    else {
      long nErr = AdsCallDllFunction.getLocalAddress(addr); //local netid
      if (nErr != 0) {
        System.out.println("getLocalAddress() failed with " + nErr);
        AdsCallDllFunction.adsPortClose();
        return 0;
      }
    }
    addr.mPort = Prt;
      }
    return port;
  }

  public long closePort(){
    if (port != 0){
      long nErr = AdsCallDllFunction.adsPortClose();
      //System.out.println("Ads port closed");

      port = 0;

      if (nErr != 0)
    System.out.println("adsPortClose() failed with " + nErr);
    }
    return port;
  }

  public long readByIGrpOffs(JNIByteBuffer databuff,long lj_idxGrp,long lj_idxOffs){
    long nErr = 0;

    //read by IndexGroup and IndexOffset
    if (port != 0)
      nErr = AdsCallDllFunction.adsSyncReadReq(addr, lj_idxGrp, lj_idxOffs,
    databuff.getUsedBytesCount(), databuff);
    else
      nErr = 1864; // error 1864 (0x748) ads-port not opened

    return nErr;
  }

  public long writeByIGrpOffs(JNIByteBuffer databuff,long lj_idxGrp,long lj_idxOffs){
    long nErr = 0;

    //write by IndexGroup and IndexOffset
    if (port != 0)
      nErr = AdsCallDllFunction.adsSyncWriteReq(addr, lj_idxGrp, lj_idxOffs,
    databuff.getUsedBytesCount(), databuff);
    else
      nErr = 1864; // error 1864 (0x748) ads-port not opened
```

```
  return nErr;
}

public long getHandleBySymbol(JNIByteBuffer hdlbuff, JNIByteBuffer symbuff){
  long nErr = 0;

  //get handle by symbol name (symbol like "MAIN.iCounter" name in symbuff)
  if (port != 0)
    nErr = AdsCallDllFunction.adsSyncReadWriteReq(addr, 0xF003, 0x0,
    hdlbuff.getUsedBytesCount(), hdlbuff, //buffer for getting handle
    symbuff.getUsedBytesCount(), symbuff); //buffer containg symbolpath
  else
    nErr = 1864; // error 1864 (0x748) ads-port not opened

  return nErr;
}

public long readByHandle(JNIByteBuffer databuff, long symHandle){
  long nErr = 0;

  if (port != 0)
  { // read variable by handle
    if (symHandle != 0)
  nErr = AdsCallDllFunction.adsSyncReadReq(addr, 0xF005, symHandle,
    databuff.getUsedBytesCount(), databuff);
    else
  nErr = 1809; // error 1809 (0x711) invalid symbol handle
  }
  else
    nErr = 1864; // error 1864 (0x748) ads-port not opened

  return nErr;
}

public long writeByHandle(JNIByteBuffer databuff, long symHandle){
  long nErr = 0;

  if (port != 0)
  { //write variable by handle
    if (symHandle != 0)
  nErr = AdsCallDllFunction.adsSyncWriteReq(addr, 0xF005, symHandle,
    databuff.getUsedBytesCount(), databuff);
    else
  nErr = 1809; // error 1809 (0x711) invalid symbol handle
  }
  else
    nErr = 1864; // error 1864 (0x748) ads-port not opened

  return nErr;
}

public long readBySymbol(JNIByteBuffer databuff, JNIByteBuffer symbuff){
  JNIByteBuffer hdlbuff = new JNIByteBuffer(4);
  long nErr = 0;
  long symHandle = 0;

  if (port != 0)
  {
    //get handle by symbol name (symbol like "MAIN.iCounter" name in symbuff)
    nErr = AdsCallDllFunction.adsSyncReadWriteReq(addr, 0xF003, 0x0,
    hdlbuff.getUsedBytesCount(), hdlbuff,  //buffer for getting handle
    symbuff.getUsedBytesCount(), symbuff); //buffer containg symbolpath

    if (nErr != 0)
  return nErr;

    //get handle
    byte[] byteArr = new byte[4];
    byteArr = hdlbuff.getByteArray();
    symHandle = Convert.ByteArrToInt(byteArr);

    // read variable by handle
    nErr = AdsCallDllFunction.adsSyncReadReq(addr, 0xF005, symHandle,
  databuff.getUsedBytesCount(), databuff);
  }
  else
    nErr = 1864; // error 1864 (0x748) ads-port not opened

  return nErr;
}
```

```
  public long writeBySymbol(JNIByteBuffer databuff, JNIByteBuffer symbuff){
    JNIByteBuffer hdlbuff = new JNIByteBuffer(4);
    long nErr = 0;
    long symHandle = 0;

    if (port != 0) {
      //get handle by symbol name (symbol like "MAIN.iCounter" name in symbuff)
      nErr = AdsCallDllFunction.adsSyncReadWriteReq(addr, 0xF003, 0x0,
      hdlbuff.getUsedBytesCount(), hdlbuff, //buffer for getting handle
      symbuff.getUsedBytesCount(), symbuff); //buffer containg symbolpath

      if (nErr != 0)
    return nErr;

     //get handle
      byte[] byteArr = new byte[4];
      byteArr = hdlbuff.getByteArray();
      symHandle = Convert.ByteArrToInt(byteArr);

      //write variable by handle
      nErr = AdsCallDllFunction.adsSyncWriteReq(addr, 0xF005, symHandle,
    databuff.getUsedBytesCount(), databuff);
    }
    else
      nErr = 1864; // error 1864 (0x748) ads-port not opened

    return nErr;
  }
}
```

PLC program

```
PROGRAM MAIN
VAR
    MdiCounter AT %MD2 : DINT := 20;
    MiCounter AT %MW2 : INT := 20;
    MsiCounter AT %MB2 : SINT := 20;
    MfCounter AT %MD6 : REAL := 1.1;
    MstrText AT %MD10 : STRING := 'qwertz';

    diCounter : DINT := 40;
    iCounter : INT := 40;
    siCounter : SINT := 40;
    fCounter : REAL := 9.3;
    strText : STRING := 'asdfgh';

    QdiCounter AT %QD8 : DINT := 60;
    QiCounter AT %QW8 : INT := 60;
    QsiCounter AT %QB8 : SINT := 60;
    QfCounter AT %QD12 : REAL := 17.4;
    QstrText AT %QD20 : STRING := 'yxcvbn';

    Timer : TON := (PT := T#100ms);
END_VAR


Timer(IN := TRUE);
IF Timer.Q THEN
Timer(IN := FALSE);

MdiCounter := MdiCounter + 1;
MfCounter := MfCounter + 1.1;

diCounter := diCounter + 1;
iCounter := iCounter + 1;
siCounter := siCounter + 1;
fCounter := fCounter + 1.1;

QdiCounter := QdiCounter + 1;
QfCounter := QfCounter + 1.1;
END_IF
```

6.2 Zugriff per Variablenname

Das folgende Programm greift auf eine SPS-Variable zu, die keine Adresse besitzt. Der Zugriff muss deshalb per Variablenname erfolgen.

Beispielprogramm 'Zugriff per Variablenname' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475091595.zip.

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample02.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

```
import de.beckhoff.jni.Convert;
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.tcads.AmsAddr;
import de.beckhoff.jni.tcads.AdsCallDllFunction;

public class Main
{
    public static void main(String[] args)
    {
    long err;
    AmsAddr addr = new AmsAddr();
    JNIByteBuffer handleBuff = new JNIByteBuffer(Integer.SIZE / Byte.SIZE);
    JNIByteBuffer symbolBuff = new JNIByteBuffer(
        Convert.StringToByteArr("MAIN.PLCVar",true));
    JNIByteBuffer dataBuff = new JNIByteBuffer(Integer.SIZE / Byte.SIZE);

    // Open communication
    AdsCallDllFunction.adsPortOpen();
    err = AdsCallDllFunction.getLocalAddress(addr);
    addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

    if (err != 0) {
        System.out.println("Error: Open communication: 0x"
            + Long.toHexString(err));
    } else {
        System.out.println("Success: Open communication!");
    }

    // Get handle by symbol name
    err = AdsCallDllFunction.adsSyncReadWriteReq(addr,
                        AdsCallDllFunction.ADSIGRP_SYM_HNDBYNAME,
                        0x0,
                        handleBuff.getUsedBytesCount(),
                        handleBuff,
                        symbolBuff.getUsedBytesCount(),
                        symbolBuff);
    if(err!=0) {
        System.out.println("Error: Get handle: 0x"
            + Long.toHexString(err));
    } else {
        System.out.println("Success: Get handle!");
    }

    // Handle: byte[] to int
    int hdlBuffToInt = Convert.ByteArrToInt(handleBuff.getByteArray());

    // Read value by handle
    err = AdsCallDllFunction.adsSyncReadReq(addr,
                        AdsCallDllFunction.ADSIGRP_SYM_VALBYHND,
                        hdlBuffToInt,
                        0x4,
                        dataBuff);
    if(err!=0)
    {
        System.out.println("Error: Read by handle: 0x"
                + Long.toHexString(err));
    }
    else
    {
        // Data: byte[] to int
        int intVal = Convert.ByteArrToInt(dataBuff.getByteArray());
        System.out.println("Success: PLCVar value: " + intVal);
    }

    // Release handle
    err = AdsCallDllFunction.adsSyncWriteReq(addr,
```

```
        AdsCallDllFunction.ADSIGRP_SYM_RELEASEHND,
        0,
        handleBuff.getUsedBytesCount(),
        handleBuff);

    if(err!=0) {
        System.out.println("Error: Release Handle: 0x"
            + Long.toHexString(err));
    } else {
        System.out.println("Success: Release Handle!");
    }

    // Close communication
    err = AdsCallDllFunction.adsPortClose();
    if(err!=0) {
        System.out.println("Error: Close Communication: 0x"
            + Long.toHexString(err));
    }

    try{
        System.in.read();
    }
    catch (Exception e){
        System.out.println("Error: Close program");
    }
    }
}
```

6.3 Zugriff auf ein Array in der SPS

Ein Array, welches sich in der SPS befindet, soll mit einen Lesebefehl ausgelesen werden. Die Variable wird hierbei per Variablenname angesprochen. Die Vorgehensweise weicht nur geringfügig von dem Auslesen diskreter Variablen ab:

Beispielprogramm 'Zugriff auf ein Array in der SPS' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475093003.zip

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample03.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

```
import de.beckhoff.jni.Convert;
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.tcads.AmsAddr;
import de.beckhoff.jni.tcads.AdsCallDllFunction;

public class Main
{
    public static void main(String[] args)
    {
    long err;
    AmsAddr addr = new AmsAddr();
    JNIByteBuffer dataBuff = new JNIByteBuffer(200);

    // Open communication
    AdsCallDllFunction.adsPortOpen();
    err = AdsCallDllFunction.getLocalAddress(addr);
    addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

    if (err != 0) {
        System.out.println("Error: Open communication: 0x"
            + Long.toHexString(err));
    } else {
        System.out.println("Success: Open communication!");
    }

    // Read value by IndexGroup and IndexOffset
    err = AdsCallDllFunction.adsSyncReadReq(addr,
                        0x4020,     // Index Group
                        0x0,    // Index Offset
                        200,
                        dataBuff);
    if(err!=0)
    {
        System.out.println("Error: Read by handle: 0x"
            + Long.toHexString(err));
```

```
    }
    else
    {
        for (int i = 0; i < dataBuff.getUsedBytesCount(); i=i+2)
        {
        // PLC datatype int consists of two bytes. Get them.
        byte lowByte = dataBuff.getByteArray()[i];
        byte highByte = dataBuff.getByteArray()[i+1];
        // Create new byte[]. Little endian!
        byte[] valBytes = { lowByte, highByte };
        // Integer value: byte[] to int
        int valInt = Convert.ByteArrToShort(valBytes);
        System.out.println("Value of PLCVar[" + i/2 + "]: " + valInt);
        }
    }

    // Close communication
    err = AdsCallDllFunction.adsPortClose();
    if(err!=0) {
        System.out.println("Error: Close Communication: 0x"
            + Long.toHexString(err));
    }

    try{
        System.in.read();
    }
    catch (Exception e){
        System.out.println("Error: Close program");
    }
    }
}
```

6.4 Übertragen von Strukturen an die SPS

Dieses Beispiel zeigt die Übertragung einer Struktur an die SPS per ADS. Die Struktur besteht aus Elementen von verschiedenen Datentypen:

Beispielprogramm 'Übertragen von Strukturen an die SPS' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475094411.zip

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample04.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

```
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.tcads.AmsAddr;
import de.beckhoff.jni.tcads.AdsCallDllFunction;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;

public class Main
{
    public static void main(String[] args)
    {
    long err;
    AmsAddr addr = new AmsAddr();

    TransferObject transfer = new TransferObject(); // See additional class
    JNIByteBuffer dataBuff = new JNIByteBuffer(19);

    // Open communication
    AdsCallDllFunction.adsPortOpen();
    err = AdsCallDllFunction.getLocalAddress(addr);
    addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

    if (err != 0) {
        System.out.println("Error: Open communication: 0x"
            + Long.toHexString(err));
    } else {
        System.out.println("Success: Open communication!");
    }

    // Use JNIByteBuffer as a backing array for ByteBuffer
    ByteBuffer bb = ByteBuffer.wrap(dataBuff.getByteArray());

    // Write elements to buffer. Litte Endian!
```

```
    bb.order(ByteOrder.LITTLE_ENDIAN);

    bb.putShort(transfer.getShortVal());
    bb.putInt(transfer.getIntVal());
    bb.put(transfer.getByteVal());
    bb.putDouble(transfer.getDoubleVal());
    bb.putFloat(transfer.getFloatVal());

    // Write struct to PLC
    err = AdsCallDllFunction.adsSyncWriteReq(addr,
        0x4020,     // Index Group
        0x0,    // Index Offset
        19,
        dataBuff);
    if(err!=0) {
        System.out.println("Error: Write request: 0x"
            + Long.toHexString(err));
    } else {
        System.out.println("Success: Write struct!");
    }

    // Close communication
    err = AdsCallDllFunction.adsPortClose();
    if(err!=0) {
        System.out.println("Error: Close Communication: 0x"
            + Long.toHexString(err));
    }
    }
}
```

Das zu übertragende Objekt:

```
public class TransferObject
{
    private short shortVal;
    private int intVal;
    private byte byteVal;
    private double doubleVal;
    private float floatVal;

    public TransferObject()
    {
    this.shortVal = Short.MAX_VALUE;
    this.intVal = Integer.MIN_VALUE;
    this.byteVal = 3;
    this.doubleVal = 4.1234;
    this.floatVal = 5.4321f;
    }

    public short getShortVal() {
    return shortVal;
    }
    public int getIntVal() {
    return intVal;
    }
    public byte getByteVal() {
    return byteVal;
    }
    public double getDoubleVal() {
    return doubleVal;
    }
    public float getFloatVal() {
    return floatVal;
    }
}
```

6.5 Auslesen der SPS-Variablen Deklaration einer einzelnen Variablen

Bei den Zugriff auf die Variablendeklaration werden folgende Informationen übertragen:

- Variablenname

- Datentyp
- Länge
- Adresse (IndexGroup / IndexOffset)
- Kommentar

Beispielprogramm 'Auslesen der SPS-Variablendeklaration einer einzelnen Variablen' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475095819.zip

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample05.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

Mit dem AdsSyncReadWriteReq() Aufruf wird die Information der Variablen ausgelesen. Der Name der Variablen wird der Funktion im Parameter pWriteData übergeben. Nach dem Aufruf steht die angeforderte Information in einem Buffer vom Typ JNIByteBuffer. Der Buffer wird in den Konstruktor einer Variablen vom Typ AdsSymbolEntry gereicht. In dieser Klasse sind die einzelnen Informationen der SPS-Variablen abgelegt. Die Felder entryLenght, typeLength und commentLength geben die Längen der hinter der Klasse befindlichen zugehörigen Strings an. Als nächstes wird der Datentyp der Variable ermittelt, indem im adsDatatypeString das zum Value passende Label gesucht wird. Falls der Datentyp dem Typ UDINT oder ARRAY OF UDINT entspricht wird zusätzlich der Wert dieser Variable ausgelesen:

```
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.tcads.AmsAddr;
import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.Convert;
import de.beckhoff.jni.tcads.AdsSymbolEntry;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;

public class Main
{
    // TODO: Check real values of each constant.
    private static final int ADST_VOID = 0;
    private static final int ADST_INT8 = 16;
    private static final int ADST_UINT8 = 17;
    private static final int ADST_INT16 = 2;
    private static final int ADST_UINT16 = 18;
    private static final int ADST_INT32 = 3;
    private static final int ADST_UINT32 = 19;
    private static final int ADST_INT64 = 20;
    private static final int ADST_UINT64 = 21;
    private static final int ADST_REAL32 = 4;
    private static final int ADST_REAL64 = 5;
    private static final int ADST_STRING = 30;
    private static final int ADST_WSTRING = 31;
    private static final int ADST_REAL80 = 32;
    private static final int ADST_BIT = 33;
    private static final int ADST_BIGTYPE = 65;
    private static final int ADST_MAXTYPES = 67;

    private static final ValueString[] adsDatatypeString = new ValueString[]{
    new ValueString(ADST_VOID, "ADST_VOID"),
    new ValueString(ADST_INT8, "ADST_INT8"),
    new ValueString(ADST_UINT8, "ADST_UINT8"),
    new ValueString(ADST_INT16, "ADST_INT16"),
    new ValueString(ADST_UINT16, "ADST_UINT16"),
    new ValueString(ADST_INT32, "ADST_INT32"),
    new ValueString(ADST_UINT32, "ADST_UINT32"),
    new ValueString(ADST_INT64, "ADST_INT64"),
    new ValueString(ADST_UINT64, "ADST_UINT64"),
    new ValueString(ADST_REAL32, "ADST_REAL32"),
    new ValueString(ADST_REAL64, "ADST_REAL64"),
    new ValueString(ADST_STRING, "ADST_STRING"),
    new ValueString(ADST_WSTRING, "ADST_WSTRING"),
    new ValueString(ADST_REAL80, "ADST_REAL80"),
    new ValueString(ADST_BIT, "ADST_BIT"),
    new ValueString(ADST_BIGTYPE, "ADST_BIGTYPE"),
    new ValueString(ADST_MAXTYPES, "ADST_MAXTYPES")
    };

    public static void main(String[] args)
```

```
{
long err;
AmsAddr addr = new AmsAddr();
JNIByteBuffer readBuff = new JNIByteBuffer(0xFFFF);
JNIByteBuffer writeBuff;
String userInput = null;
BufferedReader buffReader = null;

// Open communication
AdsCallDllFunction.adsPortOpen();
err = AdsCallDllFunction.getLocalAddress(addr);
addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

if (err != 0) {
    System.out.println("Error: Open communication: 0x"
        + Long.toHexString(err));
} else {
    System.out.println("Success: Open communication!");
}

while (true) {
    try {
    // User chooses which variable declaration to display
    System.out.print("Enter variable Name: ");

    buffReader = new BufferedReader(new InputStreamReader(System.in));
    userInput = buffReader.readLine();

    System.out.println(userInput);

    // Initialize writeBuff with user data
    writeBuff = new JNIByteBuffer(
        Convert.StringToByteArr(userInput, false));

    // Get variable declaration
    err = AdsCallDllFunction.adsSyncReadWriteReq(
                addr,
                AdsCallDllFunction.ADSIGRP_SYM_INFOBYNAMEEX,
                0,
                readBuff.getUsedBytesCount(),
                readBuff,
                writeBuff.getUsedBytesCount(),
                writeBuff);

    if(err!=0) {
        System.out.println("Error: AdsSyncReadReq: 0x"
            + Long.toHexString(err));
    } else {
        // Convert stream to AdsSymbolEntry
        AdsSymbolEntry adsSymbolEntry =
            new AdsSymbolEntry(readBuff.getByteArray());

        // Write information to stdout
        System.out.println("Name:\t\t"
                + adsSymbolEntry.getName());
        System.out.println("Index Group:\t"
                + adsSymbolEntry.getiGroup());
        System.out.println("Index Offset:\t"
                + adsSymbolEntry.getiOffs());
        System.out.println("Size:\t\t"
                + adsSymbolEntry.getSize());
        System.out.println("Type:\t\t"
                + adsSymbolEntry.getType());
        System.out.println("Comment:\t"
                + adsSymbolEntry.getComment());

        // Example: Separate treatment of data types
        switch(adsSymbolEntry.getDataType()) {
        case ADST_UINT32:
            System.out.println("Datatype:\tADST_UINT32");

            int elems = adsSymbolEntry.getSize()
                / (Integer.SIZE / Byte.SIZE);

            JNIByteBuffer uint32Buff = new JNIByteBuffer(
                        adsSymbolEntry.getSize());

            err = AdsCallDllFunction.adsSyncReadReq(addr,
                adsSymbolEntry.getiGroup(),
                adsSymbolEntry.getiOffs(),
```

```
                    adsSymbolEntry.getSize(),
                    uint32Buff);

                if(err!=0) {
                System.out.println("Error: AdsSyncReadReq: 0x"
                    + Long.toHexString(err));
                } else {
                ByteBuffer bb = ByteBuffer.allocate(0);
                bb = ByteBuffer.wrap(uint32Buff.getByteArray());
                bb.order(ByteOrder.LITTLE_ENDIAN);

                System.out.print("Value:\t\t");

                // In case of an array print each value.
                for (int i = 0; i < elems; i++)
                    System.out.print(bb.getInt() + "\t");

                System.out.println("\n");
                }
                break;

            default:
                // Iterate through ValueString[] and try to find a
                // datatype-match
                for (int i = 0; i < adsDatatypeString.length; i++) {
                if (adsDatatypeString[i].getValue() ==
                    adsSymbolEntry.getDataType()) {

                    System.out.println("Datatype: "
                        + "\t"
                        + adsDatatypeString[i].getLabel());
                }
                }
                break;
            }
        }

        // User chooses to repeat process or not
        System.out.print("Exit(y/n): ");
        buffReader=new BufferedReader(new InputStreamReader(System.in));
        userInput = buffReader.readLine();

        if("y".equals(userInput.toLowerCase()))
            break;

        } catch (Exception ex) {
        System.out.print(ex.getMessage());
        }
    }

    try {
        buffReader.close();
    } catch (Exception ex) {
        System.out.print(ex.getMessage());
    }

    // Close communication
    err = AdsCallDllFunction.adsPortClose();
    if(err!=0) {
        System.out.println("Error: Close Communication: 0x"
            + Long.toHexString(err));
    }
    }
}
```

```
public class ValueString {
    private int value;
    private String label;

    ValueString(int value, String label) {
    this.value = value;
    this.label = label;
    }

    public int getValue() {
    return value;
    }
```

```
    public String getLabel() {
    return label;
    }
}
```

6.6 Merker synchron in die SPS schreiben

Bei diesem Beispielprogramm wird der Wert, den der Bediener eingegeben hat, in das Merkerdoppelwort 0 geschrieben:

Beispielprogramm 'Merker synchron in die SPS schreiben' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475097227.zip

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample06.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

```
import de.beckhoff.jni.Convert;
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.tcads.AmsAddr;

public class Main {
    public static void main(String[] args) {
    try{
        long err;
        AmsAddr addr = new AmsAddr();

        // Open communication
        AdsCallDllFunction.adsPortOpen();
        err = AdsCallDllFunction.getLocalAddress(addr);
        addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

        if (err != 0) {
        System.out.println("Error: Open communication: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Open communication!");
        }

        // Specify IndexGroup, IndexOffset and write PLCVar
        err = AdsCallDllFunction.adsSyncWriteReq(addr,
                    0x4020,     // Index Group
                    0x0,    // Index Offset
                    Integer.SIZE / Byte.SIZE,
                    new JNIByteBuffer(Convert.IntToByteArr(1024)));
        if(err!=0) {
        System.out.println("Error: Write by adress: 0x"
            + Long.toHexString(err));
        }

        // Close communication
        err = AdsCallDllFunction.adsPortClose();
        if(err!=0) {
        System.out.println("Error: Close Communication: 0x"
            + Long.toHexString(err));
        }

        System.out.println("Press enter to continue..");
        System.in.read();
    }
    catch (Exception ex){
        System.out.println(ex.getMessage());
    }
    }
}
```

6.7 Merker synchron aus der SPS lesen

Bei diesem Beispielprogramm wird der Wert aus dem Merkerdoppelwort 0 der SPS ausgelesen und angezeigt:

Beispielprogramm 'Merker synchron aus der SPS lesen' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475098635.zip

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample07.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

```
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.tcads.AmsAddr;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;

public class Main {
    public static void main(String[] args) {
    try{
        long err;
        AmsAddr addr = new AmsAddr();
        JNIByteBuffer buff = new JNIByteBuffer(Integer.SIZE / Byte.SIZE);
        ByteBuffer bb = ByteBuffer.allocate(0);

        // Open communication
        AdsCallDllFunction.adsPortOpen();
        err = AdsCallDllFunction.getLocalAddress(addr);
        addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

        if (err != 0) {
        System.out.println("Error: Open communication: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Open communication!");
        }

        // Specify IndexGroup, IndexOffset and read PLCVar
        err = AdsCallDllFunction.adsSyncReadReq(addr,
                        0x4020,     // Index Group
                        0x0,    // Index Offset
                        buff.getUsedBytesCount(),
                        buff);
        if (err != 0) {
        System.out.println("Error: Read by adress: 0x"
            + Long.toHexString(err));
        } else {
        bb = ByteBuffer.wrap(buff.getByteArray());
        bb.order(ByteOrder.LITTLE_ENDIAN);

        System.out.println("" + bb.getInt());
        }

        // Close communication
        err = AdsCallDllFunction.adsPortClose();
        if(err!=0) {
        System.out.println("Error: Close Communication: 0x"
            + Long.toHexString(err));
        }

        System.out.println("Press enter to continue..");
        System.in.read();
    }
    catch (Exception ex){
        System.out.println(ex.getMessage());
    }
    }
}
```

6.8 Handle einer SPS Variablen löschen

In diesem Beispiel wird ein Handle einer SPS Variable geholt und im Anschluss gelöscht:

Beispielprogramm 'Handle einer SPS Variablen löschen' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475100043.zip

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample08.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

```
import de.beckhoff.jni.Convert;
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.tcads.AmsAddr;

public class Main {
    public static void main(String[] args) {
    long err;
    AmsAddr addr = new AmsAddr();
    JNIByteBuffer symBuff;
    JNIByteBuffer handleBuff = new JNIByteBuffer(Integer.SIZE / Byte.SIZE);

    try {
        // Open communication
        AdsCallDllFunction.adsPortOpen();
        err = AdsCallDllFunction.getLocalAddress(addr);
        addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

        if (err != 0) {
        System.out.println("Error: Open communication: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Open communication!");
        }

        // Convert Symbol name to byte buffer
        symBuff = new JNIByteBuffer(Convert.StringToByteArr(
            "MAIN.PLCVar",
            true));

        // Get handle via symbol name
        err = AdsCallDllFunction.adsSyncReadWriteReq(addr,
            AdsCallDllFunction.ADSIGRP_SYM_HNDBYNAME,
            0,
            handleBuff.getUsedBytesCount(),
            handleBuff,          //buffer for getting handle
            symBuff.getUsedBytesCount(),
            symBuff);            //buffer containg symbol name

        if(err!=0) {
        System.out.println("Error: Get Handle: 0x" + Long.toHexString(err));
        } else {
        System.out.println("Success: Get Handle!");
        }

        // Release handle
        err = AdsCallDllFunction.adsSyncWriteReq(addr,
            AdsCallDllFunction.ADSIGRP_SYM_RELEASEHND,
            0,
            handleBuff.getUsedBytesCount(),
            handleBuff);

        if(err!=0) {
        System.out.println("Error: Release Handle: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Handle removed!");
        }

        // Close communication
        err = AdsCallDllFunction.adsPortClose();
        if(err!=0) {
        System.out.println("Error: Close Communication: 0x"
            + Long.toHexString(err));
        }

        System.out.println("Press enter to continue..");
        System.in.read();

    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
    }
}
```

6.9 Ereignisgesteuertes Lesen

Sollen in einer Bedieneroberfläche kontinuierlich Werte aus der SPS oder NC angezeigt werden, so ist das benutzen von AdsSyncReadReq() sehr aufwendig, da diese Funktion zyklisch aufgerufen werden muss. Durch das Definieren sogenannter Notifications (Meldung) kann ein TwinCAT Server dazu veranlasst werden, Werte über ADS an ein anderes ADS-Gerät zu übertragen. Hierbei wird unterschieden ob der TwinCAT Server die Werte zyklisch oder nur bei Veränderung übertragen soll.

Mit der Funktion AdsSyncAddDeviceNotificationReq() wird eine Notification gestartet. Die Callback-Funktion wird anschließend selbständig von TwinCAT aufgerufen. Mit AdsSyncDelDeviceNotificationReq() wird die Notification wieder beendet. Da die Anzahl der Notifications begrenzt ist, sollten Sie in Ihrem Programm dafür sorgen, das nicht mehr benötigte Notifications gelöscht werden. Weiter Informationen finden Sie bei der Beschreibung zur Struktur AdsNotificationAttrib.

Beispielprogramm 'Ereignisgesteuertes Lesen' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475101451.zip

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample09.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

Das folgende Programm startet eine Notification auf das Merkerdoppeltwort 0 in der SPS. Bei jeder Änderung der SPS-Variablen, wird die Callback-Funktion aufgerufen. Als Parameter enthält die Callback-Funktion unter anderem eine Variable vom Typ AdsNotificationHeader(). In dieser Struktur sind alle notwendigen Informationen (Wert, Zeitstempel, ...) enthalten.

Zeitintensive Aktionen

Im Callback dürfen keine zeitintensiven Aktionen ausgeführt werden.

```
import de.beckhoff.jni.AdsConstants;
import de.beckhoff.jni.JNILong;
import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.tcads.AdsNotificationAttrib;
import de.beckhoff.jni.tcads.AdsCallbackObject;
import de.beckhoff.jni.tcads.AmsAddr;

public class Main {
    public static void main(String[] args) {
    try {
        long err;
        AmsAddr addr = new AmsAddr();
        JNILong notification = new JNILong();

        // Open communication
        AdsCallDllFunction.adsPortOpen();
        err = AdsCallDllFunction.getLocalAddress(addr);
        addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

        if (err != 0) {
        System.out.println("Error: Open communication: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Open communication!");
        }

        // Specify attributes of the notificationRequest
        AdsNotificationAttrib attr = new AdsNotificationAttrib();
        attr.setCbLength(Integer.SIZE / Byte.SIZE);
        attr.setNTransMode(AdsConstants.ADSTRANS_SERVERONCHA);
        attr.setDwChangeFilter(10000000);   // 1 sec
        attr.setNMaxDelay(20000000);    // 2 sec

        // Create and add listener
        AdsListener listener = new AdsListener();
        AdsCallbackObject callObject = new AdsCallbackObject();
        callObject.addListenerCallbackAdsState(listener);

        // Create notificationHandle
        err = AdsCallDllFunction.adsSyncAddDeviceNotificationReq(
        addr,
        0x4020,     // IndexGroup
        0x0,    // IndexOffset
```

```
        attr,       // The defined AdsNotificationAttrib object
        42,     // Choose arbitrary number
        notification);
        if(err!=0) {
        System.out.println("Error: Add notification: 0x"
            + Long.toHexString(err));
        }

        // Read as long as user does not press return
        System.out.println("Press enter to continue..");
        System.in.read();

        // Delete notificationHandle
        err = AdsCallDllFunction.adsSyncDelDeviceNotificationReq(
            addr,
            notification);
        if(err!=0) {
        System.out.println("Error: Remove notification: 0x"
            + Long.toHexString(err));
        }

        // Delete listener
        callObject.removeListenerCallbackAdsState(listener);

        //Close communication
        err = AdsCallDllFunction.adsPortClose();
        if(err!=0) {
        System.out.println("Error: Close Communication: 0x"
            + Long.toHexString(err));
        }
    } catch(Exception ex) {
        System.out.println(ex.getMessage());
    }
    }
}
```

Die Implementierung des CallbackListenerAdsState Interface:

```
import de.beckhoff.jni.Convert;
import de.beckhoff.jni.tcads.AdsNotificationHeader;
import de.beckhoff.jni.tcads.AmsAddr;
import de.beckhoff.jni.tcads.CallbackListenerAdsState;
import java.util.Date;

public class AdsListener implements CallbackListenerAdsState {
    private final static long SPAN = 11644473600000L;

    // Callback function
    public void onEvent(AmsAddr addr,
            AdsNotificationHeader notification,
            long user) {

    // The PLC timestamp is coded in Windows FILETIME.
    // Nano secs since 01.01.1601.
    long dateInMillis = notification.getNTimeStamp();

    // Date accepts millisecs since 01.01.1970.
    // Convert to millisecs and substract span.
    Date notificationDate = new Date(dateInMillis / 10000 - SPAN);

    System.out.println("Value:\t\t"
        + Convert.ByteArrToInt(notification.getData()));
    System.out.println("Notification:\t" + notification.getHNotification());
    System.out.println("Time:\t\t" + notificationDate.toString());
    System.out.println("User:\t\t" + user);
    System.out.println("ServerNetID:\t" + addr.getNetIdString() + "\n");
    }
}
```

6.10 Auslesen von SMB-Werten aus dem TwinCAT I/O Treiber

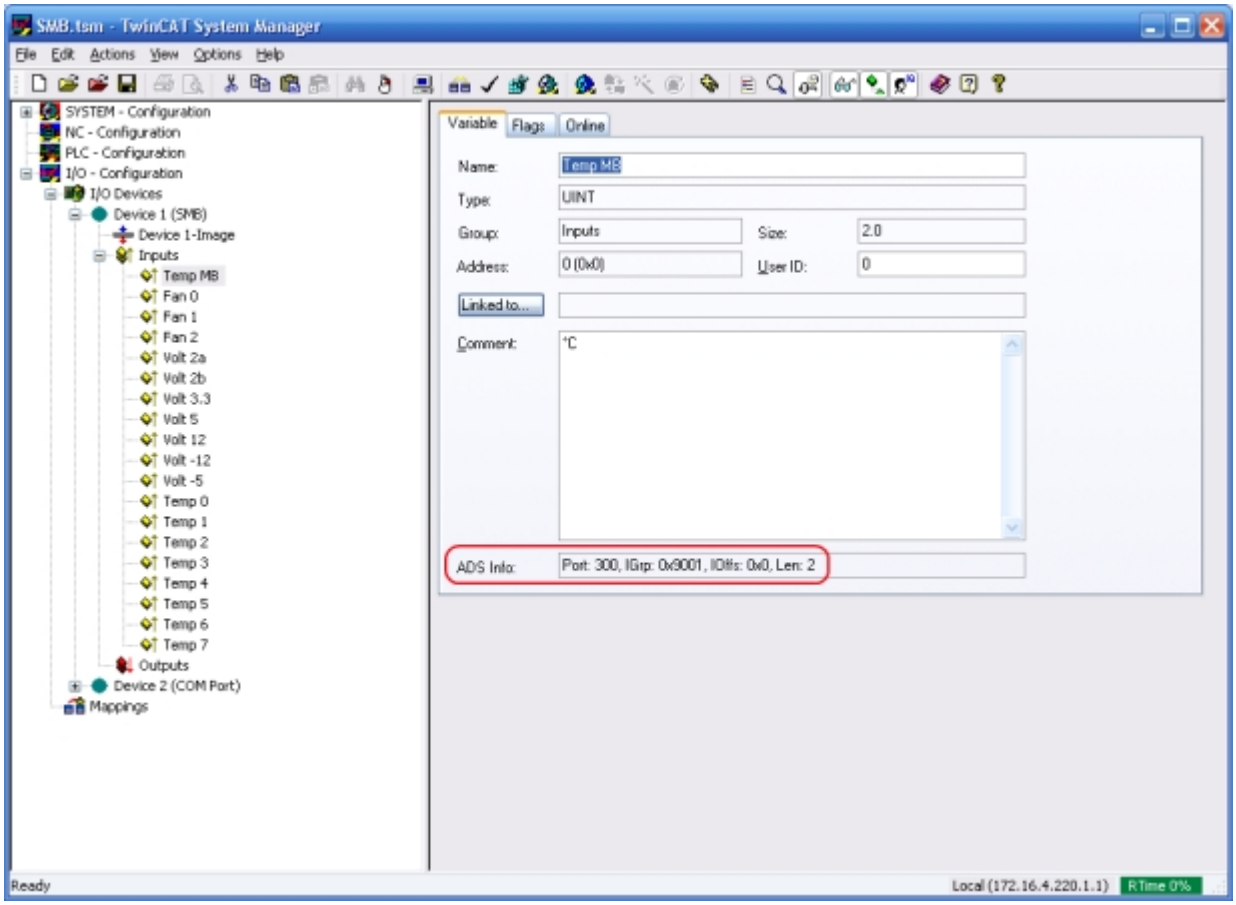
Falls statt einer TwinCAT PLC Laufzeitumgebung nur TwinCAT-CP zur Verfügung steht, kann das SMB-Gerät auch mit Hilfe des ADS-Clients ausgelesen werden. Die Visualisierung erlaubt ein ständiges Überwachen der gewünschten Temperatur- oder Lüfterwerte. Somit kann frühzeitig erkannt werden, ob sich die aktuelle Temperatur einem kritischen Wert annähert. Nachfolgend ein Beispiel einer möglichen Implementierung in Java (Für eine korrekte Funktionsweise muss ein SMB-Gerät in der aktuellen System Manager-Konfiguration eingetragen sein)

Beispielprogramm 'Auslesen von SMB-Werten aus dem TwinCAT I/O Treiber' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475102859.zip

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample10.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

```
import de.beckhoff.jni.Convert;
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.tcads.AdsState;
import de.beckhoff.jni.tcads.AmsAddr;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.nio.ByteBuffer;
import java.util.Timer;

public class Main {
    final static short SMB_ID = 32;

    public static void main(String[] args) {
    long err = 0;
    AmsAddr addr = new AmsAddr();

    Visualization visu = new Visualization();
    visu.addWindowListener(new WindowAdapter() {
        // When the form is closed the ADS port is closed as well
        @Override
        public void windowClosing(WindowEvent e) {
        AdsCallDllFunction.adsPortClose();
        }
    });

    try {
        // Open communication
        AdsCallDllFunction.adsPortOpen();
        err = AdsCallDllFunction.getLocalAddress(addr);
        addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

        if (err != 0) {
        System.out.println("Error: Open communication: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Open communication!");
        }

        //Gets the device count of devices appended to active configuration
        //if TwinCAT is in run mode
        AdsState adsState = new AdsState();
        AdsState deviceState = new AdsState();

        err = AdsCallDllFunction.adsSyncReadStateReq(addr,
            adsState,
            deviceState);
        if(err!=0) {
        visu.setInfoMsg("Error: Read device state: 0x"
            + Long.toHexString(err));
        }

        if (adsState.getState() != AdsState.ADSSTATE_RUN ) {
        visu.setInfoMsg("Bad TwinCAT state: "
            + "Please restart the application.");
        } else {
```

```
        //Get devices count
        JNIByteBuffer deviceCountBuff = new JNIByteBuffer(Integer.SIZE / Byte.SIZE);
        int devCount = 0;

        err = AdsCallDllFunction.adsSyncReadReq(addr,
            0x5000,
            0x2,
            deviceCountBuff.getUsedBytesCount(),
            deviceCountBuff);
        if(err!=0) {
            visu.setInfoMsg("Error: Read device count: 0x"
                + Long.toHexString(err));
        }

        devCount = Convert.ByteArrToInt(
            deviceCountBuff.getByteArray()) + 1;

        //Gets the device count of devices appended to active
        //configuration as zero-based index and gets the devices ID's of
        //all active devices for the remaining indices (inverted sort)
        JNIByteBuffer deviceBuff = new JNIByteBuffer(
            (Short.SIZE * devCount) / Byte.SIZE);
        ByteBuffer devices;

        err = AdsCallDllFunction.adsSyncReadReq(addr,
            0x5000,
            0x1,
            deviceBuff.getUsedBytesCount(),
            deviceBuff);
        if(err!=0) {
            visu.setInfoMsg("Error: Read devices: 0x"
                + Long.toHexString(err));
        }

        devices = ByteBuffer.wrap(deviceBuff.getByteArray());

        for (int i = 1; i <= devCount; i++) {
            JNIByteBuffer identNrBuff =
                new JNIByteBuffer(Short.SIZE / Byte.SIZE);
            short identNr = 0;

            //gets the device identification number
            //(Motherboard System Management Bus(SMB) => 32)
            err = AdsCallDllFunction.adsSyncReadReq(addr,
                0x5000 + devices.getShort(i),
                0x7,
                identNrBuff.getUsedBytesCount(),
                identNrBuff);
            if(err!=0) {
            visu.setInfoMsg("Error: Read ident numbers: 0x"
                + Long.toHexString(err));
            } else {
            identNr = Convert.ByteArrToShort(
                identNrBuff.getByteArray());

            //checks whether the device is of type SMB
            if (identNr == SMB_ID) {
                int indexGroup = 0x9000 + devices.getShort(i);

                // If a SMB-Device was found a TimerTask is started
                // to read the specified input from the device once
                // every second
                Timer tmr = new Timer();
                tmr.schedule(new SMBTask(indexGroup, addr, visu),
                    0,
                    1);

                break;
            }
            }
        }
        }

    } catch (Exception ex) {
        visu.setInfoMsg(ex.getMessage());
    }
    }
}
```



In dieser Ansicht kann der jeweilige IndexOffset der verschiedenen Eingänge abgelesen werden.

```
import de.beckhoff.jni.Convert;
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.tcads.AmsAddr;
import java.util.TimerTask;

class SMBTask extends TimerTask {
    private int indexGroup;
    private AmsAddr addr;
    private Visualization visu;

    public SMBTask(int indexGroup, AmsAddr addr, Visualization visu) {
    this.indexGroup = indexGroup;
    this.addr = addr;
    this.visu = visu;
    }

    @Override
    public void run() {
    long err = 0;
    JNIByteBuffer valueBuff = new JNIByteBuffer(Integer.SIZE / Byte.SIZE);

    //IOffset may be adjusted to monitor different inputs e.g. temperature
    err = AdsCallDllFunction.adsSyncReadReq(this.addr,
        this.indexGroup,
        0x4,
        valueBuff.getUsedBytesCount(),
        valueBuff);

    if(err!=0) {
        visu.setInfoMsg("Error: Retrieve value: 0x" + Long.toHexString(err));
    } else {
        visu.setCurValue(Convert.ByteArrToInt(valueBuff.getByteArray()));
    }
    }
}
```

6.11 Änderungen an der Symboltabelle ereignisgesteuert erkennen

ADS-Geräte, die Symbolnamen unterstützen (SPS, NC, ...), legen die Symbolnamen in eine interne Tabelle ab. Dabei wird jedem Symbol ein Handle zugeordnet. Das Symbolhandle wird benötigt, um auf die Variablen zugreifen zu können. Ändert sich die Symboltabelle, z.B. weil ein neues SPS-Programm in die Steuerung geschrieben wird, so müssen auch die Handles neu ermittelt werden. Wie Änderungen an der Symboltabelle erkannt werden können, zeigt das folgende Beispiel.

Beispielprogramm 'Änderungen an der Symboltabelle ereignisgesteuert erkennen' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475104267.zip

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample11.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

```
import de.beckhoff.jni.AdsConstants;
import de.beckhoff.jni.JNILong;
import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.tcads.AdsCallbackObject;
import de.beckhoff.jni.tcads.AdsNotificationAttrib;
import de.beckhoff.jni.tcads.AmsAddr;

public class Main {

    public static void main(String[] args) {
    long err = 0;
    AmsAddr addr = new AmsAddr();

    try {
        // Open communication
        AdsCallDllFunction.adsPortOpen();
        err = AdsCallDllFunction.getLocalAddress(addr);
        addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

        if (err != 0) {
        System.out.println("Error: Open communication: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Open communication!");
        }

        JNILong notification = new JNILong();

        AdsNotificationAttrib attr = new AdsNotificationAttrib();
        attr.setCbLength(1);
        attr.setNTransMode(AdsConstants.ADSTRANS_SERVERONCHA);
        attr.setNMaxDelay(5000000);
        attr.setNCycleTime(5000000);

        // Create and add listener
        AdsListener listener = new AdsListener();
        AdsCallbackObject callObject = new AdsCallbackObject();
        callObject.addListenerCallbackAdsState(listener);

        AdsCallDllFunction.adsSyncAddDeviceNotificationReq(addr,
            AdsCallDllFunction.ADSIGRP_SYM_VERSION,     // IndexGroup
            0,          // IndexOffset
            attr,       // The defined AdsNotificationAttrib object
            0,          // Choose arbitrary number
            notification);
        if (err != 0) {
        System.out.println("Error: Add notification: 0x"
            + Long.toHexString(err));
        }

        // Read as long as user does not press return
        System.out.println("Press enter to continue..\n");
        System.in.read();

        // Delete notificationHandle
        err = AdsCallDllFunction.adsSyncDelDeviceNotificationReq(
            addr,
```

```
            notification);

        if (err != 0) {
        System.out.println("Error: Remove notification: 0x"
            + Long.toHexString(err));
        }

        // Delete listener
        callObject.removeListenerCallbackAdsState(listener);

        //Close communication
        err = AdsCallDllFunction.adsPortClose();
        if (err != 0) {
        System.out.println("Error: Close Communication: 0x"
            + Long.toHexString(err));
        }

    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }
    }
}
```

Implementierung des CallbackListenerAdsState Interface:

```
import de.beckhoff.jni.tcads.AdsNotificationHeader;
import de.beckhoff.jni.tcads.AmsAddr;
import de.beckhoff.jni.tcads.CallbackListenerAdsState;

public class AdsListener implements CallbackListenerAdsState {
    // Callback function
    @Override
    public void onEvent(AmsAddr addr,
            AdsNotificationHeader notification,
            long user) {

    System.out.println("Symboltabelle hat sich geaendert!");
    }
}
```

6.12 ADS-Summenkommando: Holen und Freigeben von mehreren Handles

Dieses Beispiel zeigt, wie man unter Zuhilfenahme des ADS-Summenkommandos, viele Handles holen und wieder freigeben kann. Aufgebaut als AdsSyncReadWriteRequest, dient es als Behälter, in dem die Unterkommandos transportiert werden.

Systemvoraussetzungen:

- **TwinCAT v2.11 Build >= 1550**

Beispielprogramm 'ADS-Summenkommando: Holen und Freigeben von mehreren Handles' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475105675.zip

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample12.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

Handles holen

```
import java.nio.ByteBuffer;
import java.nio.ByteOrder;

import de.beckhoff.jni.Convert;
import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.tcads.AmsAddr;

public class Main {
```

```
    private static final String VAR_NAME1 = ".bVar01";
    private static final String VAR_NAME2 = ".bVar02";

    public static void main (String[] args) {
    long err = 0;

    try {
        AmsAddr addr = new AmsAddr();

        // Create request and response buffer
        JNIByteBuffer jniReqBuff;
        JNIByteBuffer jniResBuff = new JNIByteBuffer(new byte[24]);

        // Construct data objects
        RequestData req1 = new RequestData();
        req1.setIndexGroup(AdsCallDllFunction.ADSIGRP_SYM_HNDBYNAME);
        req1.setIndexOffset(0x0);
        req1.setReadLength(Integer.SIZE / Byte.SIZE);
        req1.setWriteLength(VAR_NAME1.length());

        RequestData req2 = new RequestData();
        req2.setIndexGroup(AdsCallDllFunction.ADSIGRP_SYM_HNDBYNAME);
        req2.setIndexOffset(0x0);
        req2.setReadLength(Integer.SIZE / Byte.SIZE);
        req2.setWriteLength(VAR_NAME2.length());


        // Concatenate byte[] representations of RequestData objects and
        // variable names (see picture 1)
        int reqBuffSize = RequestData.SIZE * 2 / Byte.SIZE
            + req1.getWriteLength() + req2.getWriteLength();

        ByteBuffer reqBuff = ByteBuffer.allocate(reqBuffSize);
        reqBuff.order(ByteOrder.LITTLE_ENDIAN);

        reqBuff.put(req1.toByteArray());
        reqBuff.put(req2.toByteArray());
        reqBuff.put(Convert.StringToByteArr(VAR_NAME1, false));
        reqBuff.put(Convert.StringToByteArr(VAR_NAME2, false));

        // Need to let a JNIByteBuffer wrap the byte[] to be able to send it
        jniReqBuff = new JNIByteBuffer(reqBuff.array());
```

Die Namen der Variablen deren Handles wir erhalten wollen, werden an das Ende des Bytearrays gehängt.

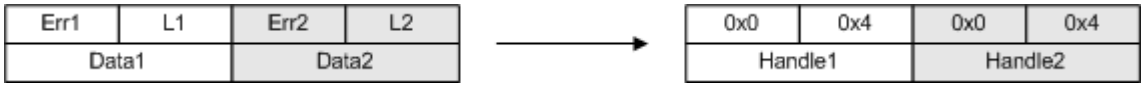
Für die Kommunikation wird ein Port geöffnet und die lokale Adresse übergeben. Kommt es zur Übertragung wird vorher der Port vom Laufzeitsystem 1 der Adresse zugewiesen.

Die Parameter für das Summenkommando bestehen aus IndexGroup (0xf082) - Aufruf des Summenkommandos, IndexOffset (0x2) - Anzahl der Unterkommandos, ReadLength (0x18) - Größenangabe der zu lesenden Daten, ReadData (pBuffRes) - Speicher, der gelesene Daten entgegen nimmt, WriteLength (cbReq) - Größenangabe der zu sendenen Daten und WriteLength (pBuffReq) - Speicher, der zu sendende Daten enthält.

```
        // Open communication
        AdsCallDllFunction.adsPortOpen();
        err = AdsCallDllFunction.getLocalAddress(addr);
        addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

        if (err != 0) {
        System.out.println("Error: Open communication: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Open communication!");
        }

        err = AdsCallDllFunction.adsSyncReadWriteReq(
                addr,
                0xf082,             // ADS list-read-write command
                0x2,                // number of ADS-sub commands
                jniResBuff.getUsedBytesCount(), // we expect an ADS error
                                //  return-code for each ADS-sub command
                jniResBuff,             // provide space for the response
```

```
                                // containing the return codes
                jniReqBuff.getUsedBytesCount(), // send 48 bytes(IG1,
                                // IO1, RLen1, WLen1, IG2, IO2, RLen2,
                                // WLen2, Data1, Data2)
                jniReqBuff);

        // Check return codes
        ByteBuffer resBuff = ByteBuffer.wrap(jniResBuff.getByteArray());
        resBuff.order(ByteOrder.LITTLE_ENDIAN);
        if (err != 0) {
        System.out.println("Error: Get handles: 0x"
                + Long.toHexString(err));
        } else {

        // Extract error codes from response (see picture 2)
        // Pattern is: err1, len1, .., errN, lenN, data1, .., dataN
        int req1Err = resBuff.getInt((Integer.SIZE / Byte.SIZE) * 0);
        int req2Err = resBuff.getInt((Integer.SIZE / Byte.SIZE) * 2);

        if (req1Err != 0) {
            System.out.println("Error: Get handle1: 0x"
                    + Integer.toHexString(req1Err));
        } else if (req2Err != 0) {
            System.out.println("Error: Get handle2: 0x"
                    + Integer.toHexString(req2Err));
        } else {
            System.out.println("Success: Get handles!");
        }
        }
```

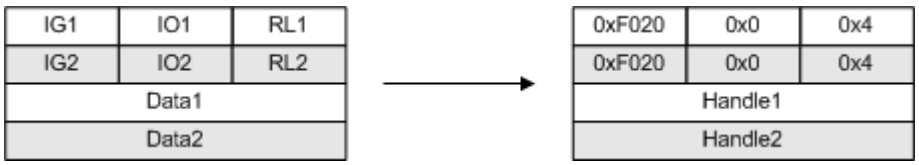
Nach dem Senden des Request, erwarten wir einen ADS Fehlercode und Länge für jeden Handle den wir versuchen zu bekommen. In diesem Fall wird die Länge immer 4 Bytes betragen.



Handles freigeben

```
        // Extract handles from response(see picture 2)
        int hnd1 = resBuff.getInt((Integer.SIZE / Byte.SIZE) * 4);
        int hnd2 = resBuff.getInt((Integer.SIZE / Byte.SIZE) * 5);


        //// Release handles
        // Construct buffers
        int relBuffSize =
            (ReleaseData.SIZE * 2 / Byte.SIZE) + (Integer.SIZE * 2 / Byte.SIZE);
        int relResBuffSize = Integer.SIZE * 2 / Byte.SIZE;

        ByteBuffer relBuff = ByteBuffer.allocate(relBuffSize);
        relBuff.order(ByteOrder.LITTLE_ENDIAN);
        JNIByteBuffer jniRelBuff = new JNIByteBuffer(relBuffSize);
        JNIByteBuffer jniRelResBuff = new JNIByteBuffer(relResBuffSize);

        // Construct data objects
        ReleaseData rel1 = new ReleaseData();
        rel1.setIndexGroup(AdsCallDllFunction.ADSIGRP_SYM_RELEASEHND);
        rel1.setIndexOffset(0);
        rel1.setLength(Integer.SIZE / Byte.SIZE);

        ReleaseData rel2 = new ReleaseData();
        rel2.setIndexGroup(AdsCallDllFunction.ADSIGRP_SYM_RELEASEHND);
        rel2.setIndexOffset(0);
        rel2.setLength(Integer.SIZE / Byte.SIZE);

        // Concatenate byte[] representations of ReleaseData objects and
        // handles (see picture 3)
        relBuff.put(rel1.toByteArray());
        relBuff.put(rel2.toByteArray());
        relBuff.putInt(hnd1);
        relBuff.putInt(hnd2);

        // Need to let a JNIByteBuffer wrap the byte[] to be able to send it
        jniRelBuff.setByteArray(relBuff.array(), true);
```

Die zu den Kommandos zugehörigen Handles werden an das Ende des Bytearrays gehängt.



Für die Freigabe der Handles wird die bestehende Verbindung benutzt.

Die Parameter für das Summenkommando bestehen aus IndexGroup (0xf081) - Aufruf des Summenkommandos, IndexOffset (0x2) - Anzahl der Unterkommandos, ReadLength (cbRelRes) - Größenangabe der zu lesenden Daten, ReadData (pBuffRelRes) - Speicher, der gelesene Daten entgegen nimmt, WriteLength (cbRel) - Größenangabe der zu sendenen Daten und WriteLength (pBuffRel) - Speicher, der zu sendene Daten enthält.

Zum Schluss müssen die Handles freigegeben und der Port geschlossen werden.

```
        // Release handles - Second task cleared
        err = AdsCallDllFunction.adsSyncReadWriteReq(
                addr,
                0xf081,         // ADS list-write command
                0x2,        // number of ADS-sub commands
                relResBuffSize,     // we expect an ADS-error-return-code for
                        // each ADS-sub command
                jniRelResBuff,      // provide space for the response containing
                        // the return codes
                relBuffSize,    // send 32 bytes (IG1, IO1, Len1, IG2, IO2,
                        // Len2, Data1, Data2)
                jniRelBuff);    // buffer with data


        // Check return codes
        if (err != 0) {
        System.out.println("Error: Release handles: 0x"
                + Long.toHexString(err));
        } else {
        ByteBuffer relResBuff =
            ByteBuffer.wrap(jniRelResBuff.getByteArray());
        relResBuff.order(ByteOrder.LITTLE_ENDIAN);

        // Extract error codes from response (see picture 4)
        // Pattern is: err1, .., errN
        int rel1Err = relResBuff.getInt((Integer.SIZE / Byte.SIZE) * 0);
        int rel2Err = relResBuff.getInt((Integer.SIZE / Byte.SIZE) * 1);

        if (rel1Err != 0) {
            System.out.println("Error: Release handle1: 0x"
                    + Integer.toHexString(rel1Err));
        } else if (rel2Err != 0) {
            System.out.println("Error: Release handle2: 0x"
                    + Integer.toHexString(rel2Err));
        } else {
            System.out.println("Success: Release handles!");
        }
        }

        System.out.println("\nPress enter to continue..");
        System.in.read();

    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        // Close communication
        err = AdsCallDllFunction.adsPortClose();

        if (err != 0) {
        System.out.println("Error: Close communication: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Close communication!");
        }
    }
    }
}
```

Als Antwort erhalten wir einen ADS Fehlercode für jedes Handle das wir versuchen freizugeben.

6.13 ADS-Summenkommando: Lesen und Schreiben von mehreren Variablen

Dieses Beispiel zeigt, wie man unter Zuhilfenahme des ADS-Summenkommandos, viele Variablen lesen oder schreiben kann. Aufgebaut als TcAdsClient.ReadWrite dient es als Behälter, in dem die Unterkommandos in einem ADS-Stream transportiert werden.

Systemvoraussetzungen:

- **TwinCAT v2.11 Build >= 1550**

Beispielprogramm 'ADS-Summenkommando: Lesen und Schreiben von mehreren Variablen' entpacken: https://infosys.beckhoff.com/content/1031/tcjavatoads/Resources/12475107083.zip

- Um das *.jar Sample ausführen zu können, muss in der Konsole im korrekten Verzeichnis der Befehl 'java -classpath "Sample13.jar;[Pfad zu TcJavaToAds.jar] Main' ausgeführt werden (Beispielpfad: "C:TwinCAT\Ads Api\AdsToJava*"). Dazu muss java in den Umgebungsvariablen eingetragen sein.

Symbolinformationen der SPS-Variablen holen

```
import java.nio.ByteBuffer;
import java.nio.ByteOrder;

import de.beckhoff.jni.JNIByteBuffer;
import de.beckhoff.jni.tcads.AdsCallDllFunction;
import de.beckhoff.jni.tcads.AmsAddr;

public class Main {

    private final static RequestData[] REQ_DATA = new RequestData[]{
    new RequestData("uintValue", 0x4020, 0x0, 0x2),
    new RequestData("boolValue", 0x4020, 0x8, 0x1),
    new RequestData("dintValue", 0x4020, 0x10, 0x4)
    };
    // The data that is going to be written in the third step
    private final static char CHAR_VAL = Character.MAX_VALUE;
    private final static byte BOOL_VAL = 1;
    private final static int INT_VAL = Integer.MIN_VALUE;
    private final static int DATA_LEN = 7;
    private final static int RESP_ERR_LEN = 4;

    public static void main(String[] args) {

    long err;
    AmsAddr addr = new AmsAddr();

    try {
        // Open communication
        AdsCallDllFunction.adsPortOpen();
        err = AdsCallDllFunction.getLocalAddress(addr);
        addr.setPort(AdsCallDllFunction.AMSPORT_R0_PLC_RTS1);

        if (err != 0) {
        System.out.println("Error: Open communication: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Open communication!");
        }
```

Wenn Variablen in der SPS beschrieben oder gelesen werden sollen, ohne dabei auf Handles zurückzugreifen, dann müssen IndexGroup und IndexOffset bekannt sein. Um diese Werte von allen relevanten Variablen zu erhalten, müssen üblicherweise die Symbolinformationen ausgelesen werden. In diesem Beispiel sind die Werte als konstant angenommen und in REQ_DATA definiert.

Werte lesen

Der RequestBuffer wird nun mit den Daten aus REQ_DATA nach untenstehendem Schema beschrieben. Dabei steht IG1 für die IndexGroup des ersten Subkommandos. IO2 für den IndexOffset des zweiten und L3 für die Länge der erwarteten Antwort des dritten Subkommandos.

IG1	IO1	L1
IG2	IO2	L2
IG3	IO3	L3

0x4040	0x2F	0x2
0x4040	0x5	0x4
0x4040	0x4	0x1

Als Antwort des Summenkommandos erhalten wir pro Subkommando einen Ads-Fehlercode (Err), sowie den angeforderten Wert (Data). Entsprechend beträgt die Länge des Buffers in diesem Fall: 3 * 4Byte (Error) + 7Byte (Die Summe aller Längen der erwarteten Antworten).

Err1	Err2	Err3
Data1	Data2	Data3

0x0	0x0	0x0
uintValue	dintValue	boolValue

Das Summenkommando selbst wird mit den Parametern IndexGroup: 0xF080 (Summenkommando-Lesen), IndexOffset: 0x3 (Anzahl Subkommandos), ResponseBuffer-Länge, ResponseBuffer, RequestBuffer-Länge, RequestBuffer aufgerufen.

```
        // Create request buffer (see picture 1)
        JNIByteBuffer jniRequestBuffer = null;
        ByteBuffer requestBuffer =
            ByteBuffer.allocate(RequestData.SIZE * REQ_DATA.length);
        requestBuffer.order(ByteOrder.LITTLE_ENDIAN);

        for (int i = 0; i < REQ_DATA.length; i++) {
        requestBuffer.put(REQ_DATA[i].toByteArray());
        }
        // Need to let a JNIByteBuffer wrap the byte[] to be able to send it
        jniRequestBuffer = new JNIByteBuffer(requestBuffer.array());

        // Create response buffer (see picture 2)
        JNIByteBuffer jniResponseBuffer = new JNIByteBuffer(
            REQ_DATA.length * RESP_ERR_LEN + DATA_LEN);

        // Read the values via sum command
        err = AdsCallDllFunction.adsSyncReadWriteReq(
            addr,
            0xF080,
            REQ_DATA.length,
            jniResponseBuffer.getUsedBytesCount(),
            jniResponseBuffer,
            jniRequestBuffer.getUsedBytesCount(),
            jniRequestBuffer);

        if (err != 0) {
        System.out.println("Error: Get values: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Get values!");

        // Evaluate the sub commands error codes and response values
        // (see picture)
        ByteBuffer responseBuffer =
            ByteBuffer.wrap(jniResponseBuffer.getByteArray());
        responseBuffer.order(ByteOrder.LITTLE_ENDIAN);

        // In each loop the errPosition is increased by the Size of an
        // error (int) and the valPosition by the size of the
        // corresponding data type

        int[] respErrs = new int[REQ_DATA.length];
        for (int i = 0; i < respErrs.length; i++) {
            respErrs[i] = responseBuffer.getInt();
        }

        for (int i = 0; i < REQ_DATA.length; i++) {
            String varName = REQ_DATA[i].getVarName();
            int val = 0;

            if (respErrs[i] == 0) {
            switch (REQ_DATA[i].getLength()) {
                case 1:
                val = (int) responseBuffer.get();
                break;
                case 2:
                val = responseBuffer.getChar();
                break;
                case 4:
                val = responseBuffer.getInt();
```

```
                break;
                default:
                break;
            }

            System.out.println(varName + ": " + val + "\t");
            } else {
            System.out.println(varName + ": Error: 0x"
                + Long.toHexString(err) + "\t");
            }
        }
        }
```

Werte schreiben

Um Daten schreiben zu können müssen diese Daten lediglich an den RequestBuffer aus dem oberen Teil angehängt werden (siehe Bild). Alle anderen Parameter bleiben unverändert. Lediglich eine Anpassung des Summenkommandos ist nötig (siehe unten).

Als Antwort des Summenkommandos erhalten wir pro Subkommando einen Ads-Fehlercode (Err), allerdings dieses Mal keine Daten, da wir einen Schreibvorgang ausgeführt haben. Entsprechend beträgt die Länge des Buffers in diesem Fall: 3 * 4Byte (Error).

Beim Summenkommando muss nun noch der Wert für die IndexGroup auf 0xF081 (Summenkommando-Schreiben) geändert werden.

```
        //// Write values
        // Construct buffers. Fill request buffer with the data from above
        // and append the data that is going to be written (see picture 3)
        byte[] requestBufferCopy = requestBuffer.array();
        requestBuffer = ByteBuffer.allocate(
            requestBuffer.capacity()
            + DATA_LEN);
        requestBuffer.order(ByteOrder.LITTLE_ENDIAN);

        requestBuffer.put(requestBufferCopy);
        requestBuffer.putChar(CHAR_VAL);
        requestBuffer.put(BOOL_VAL);
        requestBuffer.putInt(INT_VAL);

        // Need to let a JNIByteBuffer wrap the byte[] to be able to send it
        jniRequestBuffer.setByteArray(requestBuffer.array(), true);
        // Adjust the length of the response buffer (compare pictures 2 & 4)
        jniResponseBuffer.setUsedBytesCount(8 * REQ_DATA.length, true);

        // Write the values via sum command
        err = AdsCallDllFunction.adsSyncReadWriteReq(
            addr,
            0xF081,
            REQ_DATA.length,
            jniResponseBuffer.getUsedBytesCount(),
            jniResponseBuffer,
            jniRequestBuffer.getUsedBytesCount(),
            jniRequestBuffer);

        if (err != 0) {
        System.out.println("Error: Write values: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Write values!");

        // Evaluate the sub command error codes
        ByteBuffer responseBuffer =
            ByteBuffer.wrap(jniResponseBuffer.getByteArray());
        responseBuffer.order(ByteOrder.LITTLE_ENDIAN);

        for (int i = 0; i < REQ_DATA.length; i++) {
            String varName = REQ_DATA[i].getVarName();
```

```
            err = responseBuffer.getInt();

            if (err == 0) {
            System.out.println(varName + ": Success\t");
            } else {
            System.out.println(varName + ": Error: 0x"
                + Long.toHexString(err) + "\t");
            }
        }
        }

        System.out.println("\nPress enter to continue..");
        System.in.read();

    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        // Close communication
        err = AdsCallDllFunction.adsPortClose();

        if (err != 0) {
        System.out.println("Error: Close communication: 0x"
            + Long.toHexString(err));
        } else {
        System.out.println("Success: Close communication!");
        }
    }
    }
}
```

7 ADS Return Codes

Gruppierung der Fehlercodes:
Globale Fehlercodes: 0x0000 [▶ 56]... (0x9811_0000 ...)
Router Fehlercodes: 0x0500 [▶ 56]... (0x9811_0500 ...)
Allgemeine ADS Fehler: 0x0700 [▶ 57]... (0x9811_0700 ...)
RTime Fehlercodes: 0x1000 [▶ 58]... (0x9811_1000 ...)

Globale Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x0	0	0x98110000	ERR_NOERROR	Kein Fehler.
0x1	1	0x98110001	ERR_INTERNAL	Interner Fehler.
0x2	2	0x98110002	ERR_NORTIME	Keine Echtzeit.
0x3	3	0x98110003	ERR_ALLOCLOCKEDMEM	Zuweisung gesperrt - Speicherfehler.
0x4	4	0x98110004	ERR_INSERTMAILBOX	Postfach voll – Es konnte die ADS Nachricht nicht versendet werden. Reduzieren der Anzahl der ADS Nachrichten pro Zyklus bringt Abhilfe.
0x5	5	0x98110005	ERR_WRONGRECEIVEHMSG	Falsches HMSG.
0x6	6	0x98110006	ERR_TARGETPORTNOTFOUND	Ziel-Port nicht gefunden – ADS Server ist nicht gestartet oder erreichbar.
0x7	7	0x98110007	ERR_TARGETMACHINENOTFOUND	Zielrechner nicht gefunden – AMS Route wurde nicht gefunden.
0x8	8	0x98110008	ERR_UNKNOWNCMDID	Unbekannte Befehl-ID.
0x9	9	0x98110009	ERR_BADTASKID	Ungültige Task-ID.
0xA	10	0x9811000A	ERR_NOIO	Kein IO.
0xB	11	0x9811000B	ERR_UNKNOWNAMSCMD	Unbekannter AMS-Befehl.
0xC	12	0x9811000C	ERR_WIN32ERROR	Win32 Fehler.
0xD	13	0x9811000D	ERR_PORTNOTCONNECTED	Port nicht verbunden.
0xE	14	0x9811000E	ERR_INVALIDAMSLENGTH	Ungültige AMS-Länge.
0xF	15	0x9811000F	ERR_INVALIDAMSNETID	Ungültige AMS Net ID.
0x10	16	0x98110010	ERR_LOWINSTLEVEL	Installations-Level ist zu niedrig –TwinCAT 2 Lizenzfehler.
0x11	17	0x98110011	ERR_NODEBUGINTAVAILABLE	Kein Debugging verfügbar.
0x12	18	0x98110012	ERR_PORTDISABLED	Port deaktiviert – TwinCAT System Service nicht gestartet.
0x13	19	0x98110013	ERR_PORTALREADYCONNECTED	Port bereits verbunden.
0x14	20	0x98110014	ERR_AMSSYNC_W32ERROR	AMS Sync Win32 Fehler.
0x15	21	0x98110015	ERR_AMSSYNC_TIMEOUT	AMS Sync Timeout.
0x16	22	0x98110016	ERR_AMSSYNC_AMSERROR	AMS Sync Fehler.
0x17	23	0x98110017	ERR_AMSSYNC_NOINDEXINMAP	Keine Index-Map für AMS Sync vorhanden.
0x18	24	0x98110018	ERR_INVALIDAMSPORT	Ungültiger AMS-Port.
0x19	25	0x98110019	ERR_NOMEMORY	Kein Speicher.
0x1A	26	0x9811001A	ERR_TCPSEND	TCP Sendefehler.
0x1B	27	0x9811001B	ERR_HOSTUNREACHABLE	Host nicht erreichbar.
0x1C	28	0x9811001C	ERR_INVALIDAMSFRAGMENT	Ungültiges AMS Fragment.
0x1D	29	0x9811001D	ERR_TLSSEND	TLS Sendefehler – Secure ADS Verbindung fehlgeschlagen.
0x1E	30	0x9811001E	ERR_ACCESSDENIED	Zugriff Verweigert – Secure ADS Zugriff verweigert.

Router Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x500	1280	0x98110500	ROUTERERR_NOLOCKEDMEMORY	Lockierter Speicher kann nicht zugewiesen werden.
0x501	1281	0x98110501	ROUTERERR_RESIZEMEMORY	Die Größe des Routerspeichers konnte nicht geändert werden.
0x502	1282	0x98110502	ROUTERERR_MAILBOXFULL	Das Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.
0x503	1283	0x98110503	ROUTERERR_DEBUGBOXFULL	Das Debug Postfach hat die maximale Anzahl der möglichen Meldungen erreicht.

Hex	Dec	HRESULT	Name	Beschreibung
0x504	1284	0x98110504	ROUTERERR_UNKNOWNPORTTYPE	Der Porttyp ist unbekannt.
0x505	1285	0x98110505	ROUTERERR_NOTINITIALIZED	Router ist nicht initialisiert.
0x506	1286	0x98110506	ROUTERERR_PORTALREADYINUSE	Die Portnummer ist bereits vergeben.
0x507	1287	0x98110507	ROUTERERR_NOTREGISTERED	Der Port ist nicht registriert.
0x508	1288	0x98110508	ROUTERERR_NOMOREQUEUES	Die maximale Portanzahl ist erreicht.
0x509	1289	0x98110509	ROUTERERR_INVALIDPORT	Der Port ist ungültig.
0x50A	1290	0x9811050A	ROUTERERR_NOTACTIVATED	Der Router ist nicht aktiv.
0x50B	1291	0x9811050B	ROUTERERR_FRAGMENTBOXFULL	Das Postfach hat die maximale Anzahl für fragmentierte Nachrichten erreicht.
0x50C	1292	0x9811050C	ROUTERERR_FRAGMENTTIMEOUT	Fragment Timeout aufgetreten.
0x50D	1293	0x9811050D	ROUTERERR_TOBEREMOVED	Port wird entfernt.

Allgemeine ADS Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x700	1792	0x98110700	ADSERR_DEVICE_ERROR	Allgemeiner Gerätefehler.
0x701	1793	0x98110701	ADSERR_DEVICE_SRVNOTSUPP	Service wird vom Server nicht unterstützt.
0x702	1794	0x98110702	ADSERR_DEVICE_INVALIDGRP	Ungültige Index-Gruppe.
0x703	1795	0x98110703	ADSERR_DEVICE_INVALIDOFFSET	Ungültiger Index-Offset.
0x704	1796	0x98110704	ADSERR_DEVICE_INVALIDACCESS	Lesen oder Schreiben nicht gestattet.
0x705	1797	0x98110705	ADSERR_DEVICE_INVALIDSIZE	Parametergröße nicht korrekt.
0x706	1798	0x98110706	ADSERR_DEVICE_INVALIDDATA	Ungültige Daten-Werte.
0x707	1799	0x98110707	ADSERR_DEVICE_NOTREADY	Gerät nicht betriebsbereit.
0x708	1800	0x98110708	ADSERR_DEVICE_BUSY	Gerät beschäftigt.
0x709	1801	0x98110709	ADSERR_DEVICE_INVALIDCONTEXT	Ungültiger Kontext vom Betriebssystem - Kann durch Verwendung von ADS Bausteinen in unterschiedlichen Tasks auftreten. Abhilfe kann die Multitasking-Syncronisation in der SPS geben.
0x70A	1802	0x9811070A	ADSERR_DEVICE_NOMEMORY	Nicht genügend Speicher.
0x70B	1803	0x9811070B	ADSERR_DEVICE_INVALIDPARM	Ungültige Parameter-Werte.
0x70C	1804	0x9811070C	ADSERR_DEVICE_NOTFOUND	Nicht gefunden (Dateien,...).
0x70D	1805	0x9811070D	ADSERR_DEVICE_SYNTAX	Syntax-Fehler in Datei oder Befehl.
0x70E	1806	0x9811070E	ADSERR_DEVICE_INCOMPATIBLE	Objekte stimmen nicht überein.
0x70F	1807	0x9811070F	ADSERR_DEVICE_EXISTS	Objekt ist bereits vorhanden.
0x710	1808	0x98110710	ADSERR_DEVICE_SYMBOLNOTFOUND	Symbol nicht gefunden.
0x711	1809	0x98110711	ADSERR_DEVICE_SYMBOLVERSIONINVALID	Symbol-Version ungültig – Kann durch einen Online-Change auftreten. Erzeuge einen neuen Handle.
0x712	1810	0x98110712	ADSERR_DEVICE_INVALIDSTATE	Gerät (Server) ist im ungültigen Zustand.
0x713	1811	0x98110713	ADSERR_DEVICE_TRANSMODENOTSUPP	AdsTransMode nicht unterstützt.
0x714	1812	0x98110714	ADSERR_DEVICE_NOTIFYHNDINVALID	Notification Handle ist ungültig.
0x715	1813	0x98110715	ADSERR_DEVICE_CLIENTUNKNOWN	Notification-Client nicht registriert.
0x716	1814	0x98110716	ADSERR_DEVICE_NOMOREHDLS	Keine weiteren Handles verfügbar.
0x717	1815	0x98110717	ADSERR_DEVICE_INVALIDWATCHSIZE	Größe der Notification zu groß.
0x718	1816	0x98110718	ADSERR_DEVICE_NOTINIT	Gerät nicht initialisiert.
0x719	1817	0x98110719	ADSERR_DEVICE_TIMEOUT	Gerät hat einen Timeout.
0x71A	1818	0x9811071A	ADSERR_DEVICE_NOINTERFACE	Interface Abfrage fehlgeschlagen.
0x71B	1819	0x9811071B	ADSERR_DEVICE_INVALIDINTERFACE	Falsches Interface angefordert.
0x71C	1820	0x9811071C	ADSERR_DEVICE_INVALIDCLSID	Class-ID ist ungültig.
0x71D	1821	0x9811071D	ADSERR_DEVICE_INVALIDOBJID	Object-ID ist ungültig.
0x71E	1822	0x9811071E	ADSERR_DEVICE_PENDING	Anforderung steht aus.
0x71F	1823	0x9811071F	ADSERR_DEVICE_ABORTED	Anforderung wird abgebrochen.
0x720	1824	0x98110720	ADSERR_DEVICE_WARNING	Signal-Warnung.
0x721	1825	0x98110721	ADSERR_DEVICE_INVALIDARRAYIDX	Ungültiger Array-Index.
0x722	1826	0x98110722	ADSERR_DEVICE_SYMBOLNOTACTIVE	Symbol nicht aktiv.
0x723	1827	0x98110723	ADSERR_DEVICE_ACCESSDENIED	Zugriff verweigert.
0x724	1828	0x98110724	ADSERR_DEVICE_LICENSENOTFOUND	Fehlende Lizenz.
0x725	1829	0x98110725	ADSERR_DEVICE_LICENSEEXPIRED	Lizenz abgelaufen.
0x726	1830	0x98110726	ADSERR_DEVICE_LICENSEEXCEEDED	Lizenz überschritten.
0x727	1831	0x98110727	ADSERR_DEVICE_LICENSEINVALID	Lizenz ungültig.

Hex	Dec	HRESULT	Name	Beschreibung
0x728	1832	0x98110728	ADSERR_DEVICE_LICENSESYSTEMID	Lizenzproblem: System-ID ist ungültig.
0x729	1833	0x98110729	ADSERR_DEVICE_LICENSENOTIMELIMIT	Lizenz nicht zeitlich begrenzt.
0x72A	1834	0x9811072A	ADSERR_DEVICE_LICENSEFUTUREISSUE	Lizenzproblem: Zeitpunkt in der Zukunft.
0x72B	1835	0x9811072B	ADSERR_DEVICE_LICENSETIMETOLONG	Lizenz-Zeitraum zu lang.
0x72C	1836	0x9811072C	ADSERR_DEVICE_EXCEPTION	Exception beim Systemstart.
0x72D	1837	0x9811072D	ADSERR_DEVICE_LICENSEDUPLICATED	Lizenz-Datei zweimal gelesen.
0x72E	1838	0x9811072E	ADSERR_DEVICE_SIGNATUREINVALID	Ungültige Signatur.
0x72F	1839	0x9811072F	ADSERR_DEVICE_CERTIFICATEINVALID	Zertifikat ungültig.
0x730	1840	0x98110730	ADSERR_DEVICE_LICENSEOEMNOTFOUND	Public Key vom OEM nicht bekannt.
0x731	1841	0x98110731	ADSERR_DEVICE_LICENSERESTRICTED	Lizenz nicht gültig für diese System.ID.
0x732	1842	0x98110732	ADSERR_DEVICE_LICENSEDEMODENIED	Demo-Lizenz untersagt.
0x733	1843	0x98110733	ADSERR_DEVICE_INVALIDFNCID	Funktions-ID ungültig.
0x734	1844	0x98110734	ADSERR_DEVICE_OUTOFRANGE	Außerhalb des gültigen Bereiches.
0x735	1845	0x98110735	ADSERR_DEVICE_INVALIDALIGNMENT	Ungültiges Alignment.
0x736	1846	0x98110736	ADSERR_DEVICE_LICENSEPLATFORM	Ungültiger Plattform Level.
0x737	1847	0x98110737	ADSERR_DEVICE_FORWARD_PL	Kontext – Weiterleitung zum Passiv-Level.
0x738	1848	0x98110738	ADSERR_DEVICE_FORWARD_DL	Kontext – Weiterleitung zum Dispatch-Level.
0x739	1849	0x98110739	ADSERR_DEVICE_FORWARD_RT	Kontext – Weiterleitung zur Echtzeit.
0x740	1856	0x98110740	ADSERR_CLIENT_ERROR	Clientfehler.
0x741	1857	0x98110741	ADSERR_CLIENT_INVALIDPARM	Dienst enthält einen ungültigen Parameter.
0x742	1858	0x98110742	ADSERR_CLIENT_LISTEMPTY	Polling-Liste ist leer.
0x743	1859	0x98110743	ADSERR_CLIENT_VARUSED	Var-Verbindung bereits im Einsatz.
0x744	1860	0x98110744	ADSERR_CLIENT_DUPLINVOKEID	Die aufgerufene ID ist bereits in Benutzung.
0x745	1861	0x98110745	ADSERR_CLIENT_SYNCTIMEOUT	Timeout ist aufgetreten – Die Gegenstelle antwortet nicht im vorgegebenen ADS Timeout. Die Routeneinstellung der Gegenstelle kann falsch konfiguriert sein.
0x746	1862	0x98110746	ADSERR_CLIENT_W32ERROR	Fehler im Win32 Subsystem.
0x747	1863	0x98110747	ADSERR_CLIENT_TIMEOUTINVALID	Ungültiger Client Timeout-Wert.
0x748	1864	0x98110748	ADSERR_CLIENT_PORTNOTOPEN	Port nicht geöffnet.
0x749	1865	0x98110749	ADSERR_CLIENT_NOAMSADDR	Keine AMS Adresse.
0x750	1872	0x98110750	ADSERR_CLIENT_SYNCINTERNAL	Interner Fehler in Ads-Sync.
0x751	1873	0x98110751	ADSERR_CLIENT_ADDHASH	Überlauf der Hash-Tabelle.
0x752	1874	0x98110752	ADSERR_CLIENT_REMOVEHASH	Schlüssel in der Tabelle nicht gefunden.
0x753	1875	0x98110753	ADSERR_CLIENT_NOMORESYM	Keine Symbole im Cache.
0x754	1876	0x98110754	ADSERR_CLIENT_SYNCRESINVALID	Ungültige Antwort erhalten.
0x755	1877	0x98110755	ADSERR_CLIENT_SYNCPORTLOCKED	Sync Port ist verriegelt.

RTime Fehlercodes

Hex	Dec	HRESULT	Name	Beschreibung
0x1000	4096	0x98111000	RTERR_INTERNAL	Interner Fehler im Echtzeit-System.
0x1001	4097	0x98111001	RTERR_BADTIMERPERIODS	Timer-Wert nicht gültig.
0x1002	4098	0x98111002	RTERR_INVALIDTASKPTR	Task-Pointer hat den ungültigen Wert 0 (null).
0x1003	4099	0x98111003	RTERR_INVALIDSTACKPTR	Stack-Pointer hat den ungültigen Wert 0 (null).
0x1004	4100	0x98111004	RTERR_PRIOEXISTS	Die Request Task Priority ist bereits vergeben.
0x1005	4101	0x98111005	RTERR_NOMORETCB	Kein freier TCB (Task Control Block) verfügbar. Maximale Anzahl von TCBs beträgt 64.
0x1006	4102	0x98111006	RTERR_NOMORESEMAS	Keine freien Semaphoren zur Verfügung. Maximale Anzahl der Semaphoren beträgt 64.
0x1007	4103	0x98111007	RTERR_NOMOREQUEUES	Kein freier Platz in der Warteschlange zur Verfügung. Maximale Anzahl der Plätze in der Warteschlange beträgt 64.
0x100D	4109	0x9811100D	RTERR_EXTIRQALREADYDEF	Ein externer Synchronisations-Interrupt wird bereits angewandt.
0x100E	4110	0x9811100E	RTERR_EXTIRQNOTDEF	Kein externer Sync-Interrupt angewandt.
0x100F	4111	0x9811100F	RTERR_EXTIRQINSTALLFAILED	Anwendung des externen Synchronisierungs-Interrupts ist fehlgeschlagen.
0x1010	4112	0x98111010	RTERR_IRQLNOTLESSOREQUAL	Aufruf einer Service-Funktion im falschen Kontext
0x1017	4119	0x98111017	RTERR_VMXNOTSUPPORTED	Intel VT-x Erweiterung wird nicht unterstützt.
0x1018	4120	0x98111018	RTERR_VMXDISABLED	Intel VT-x Erweiterung ist nicht aktiviert im BIOS.

Hex	Dec	HRESULT	Name	Beschreibung
0x1019	4121	0x98111019	RTERR_VMXCONTROLSMISSING	Fehlende Funktion in Intel VT-x Erweiterung.
0x101A	4122	0x9811101A	RTERR_VMXENABLEFAILS	Aktivieren von Intel VT-x schlägt fehl.

Spezifische positive HRESULT Return Codes:

HRESULT	Name	Beschreibung
0x0000_0000	S_OK	Kein Fehler.
0x0000_0001	S_FALSE	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch ein negatives oder unvollständiges Ergebnis erzielt wurde.
0x0000_0203	S_PENDING	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch noch kein Ergebnis vorliegt.
0x0000_0256	S_WATCHDOG_TIMEOUT	Kein Fehler. Bsp.: erfolgreiche Abarbeitung, bei der jedoch eine Zeitüberschreitung eintrat.

TCP Winsock-Fehlercodes

Hex	Dec	Name	Beschreibung
0x274C	10060	WSAETIMEDOUT	Verbindungs Timeout aufgetreten - Fehler beim Herstellen der Verbindung, da die Gegenstelle nach einer bestimmten Zeitspanne nicht ordnungsgemäß reagiert hat, oder die hergestellte Verbindung konnte nicht aufrecht erhalten werden, da der verbundene Host nicht reagiert hat.
0x274D	10061	WSAECONNREFUSED	Verbindung abgelehnt - Es konnte keine Verbindung hergestellt werden, da der Zielcomputer dies explizit abgelehnt hat. Dieser Fehler resultiert normalerweise aus dem Versuch, eine Verbindung mit einem Dienst herzustellen, der auf dem fremden Host inaktiv ist—das heißt, einem Dienst, für den keine Serveranwendung ausgeführt wird.
0x2751	10065	WSAEHOSTUNREACH	Keine Route zum Host - Ein Socketvorgang bezog sich auf einen nicht verfügbaren Host.
Weitere Winsock-Fehlercodes: Win32-Fehlercodes			

Mehr Informationen:
www.beckhoff.de

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Deutschland
Telefon: +49 5246 9630
info@beckhoff.de
www.beckhoff.de