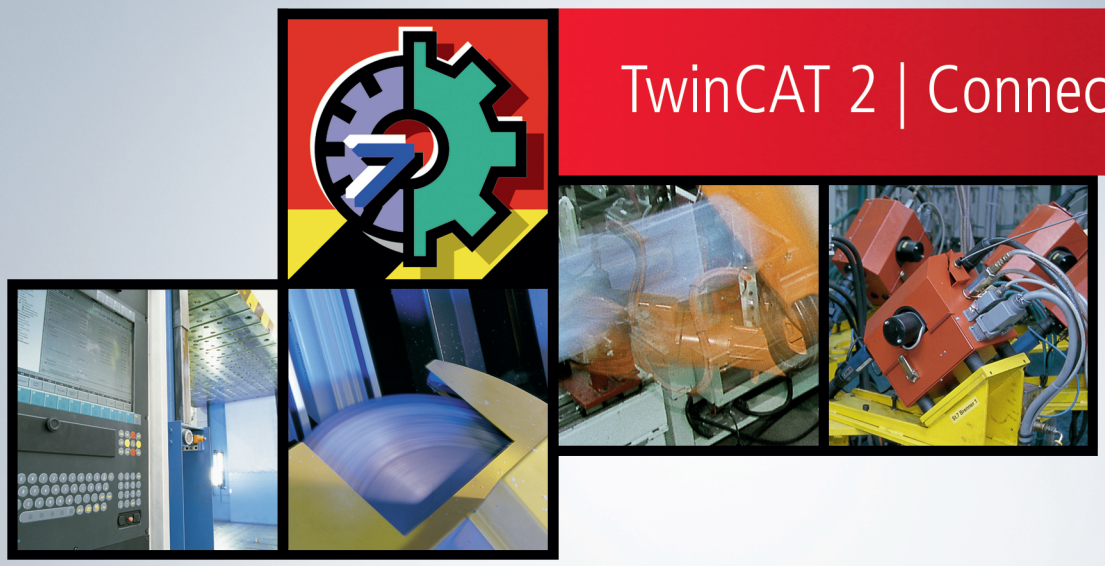
BECKHOFF New Automation Technology

Manual | EN

# TX1000

## TwinCAT 2 | ADS Java DLL



TwinCAT 2 | Connectivity

2022-10-06 | Version: 1.1

# Table of contents

**1 Foreword ..... 5**
1.1 Notes on the documentation ..... 5
1.2 Safety instructions ..... 6
1.3 Notes on information security ..... 7
**2 Introduction ..... 8**
**3 Implementing in Eclipse Galileo ..... 9**
**4 Implementing in Netbeans 6.7.1 ..... 11**
**5 API ..... 13**
5.1 Functions ..... 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 Extended functionalities ..... 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 Samples: TcJavaToAds ... 28**
6.1 Using AdsToJava.dll ... 28
6.2 Access by variable name ... 32
6.3 Accessing an array in the PLC ... 33
6.4 Transmitting structures to the PLC ... 34
6.5 Reading out the PLC variable declaration of a single variable ... 35
6.6 Write flag synchronously into the PLC ... 39
6.7 Read flag synchronously from the PLC ... 39
6.8 Delete handle of a PLC variable ... 40
6.9 Event driven reading ... 42
6.10 Reading SMB values from the TwinCAT I/O driver ... 44
6.11 Event-Driven Detection of Changes to the Symbol Table ... 47
6.12 ADS sum command: fetch and release multiple handles ... 48
6.13 ADS sum command: read and write multiple variables ... 52

**7 ADS Return Codes ... 56**

# 1 Foreword

## 1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

### Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

### Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

### Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.


EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

### Copyright

© Beckhoff Automation GmbH & Co. KG, Germany.
The reproduction, distribution and utilization of this document as well as the communication of its contents to others without express authorization are prohibited.
Offenders will be held liable for the payment of damages. All rights reserved in the event of the grant of a patent, utility model or design.

## 1.2 Safety instructions

### Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

### Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

### Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

### Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

**⚠ DANGER**

**Serious risk of injury!**

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

**⚠ WARNING**

**Risk of injury!**

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

**⚠ CAUTION**

**Personal injuries!**

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

***NOTE***

**Damage to the environment or devices**

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

**Tip or pointer**

This symbol indicates information that contributes to better understanding.

## 1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

# 2 Introduction

The AdsToJava.dll library can be used to enable your own software to communicate with ADS devices.

**To note:** to have access to the AdsToJava.dll in a project, the TcJavaToAds.jar must be added to that project. After that, by importing the package "de.beckhoff.jni.tcads", the corresponding functions, objects and constants become available.

The TcJavaToAds.jar is located in the "TwinCAT\ADS Api\AdsToJava" directory. The AdsToJava.dll is located in the subdirectory "XP" (An additional WinCE version also exists, it is located in the subfolder "CE").

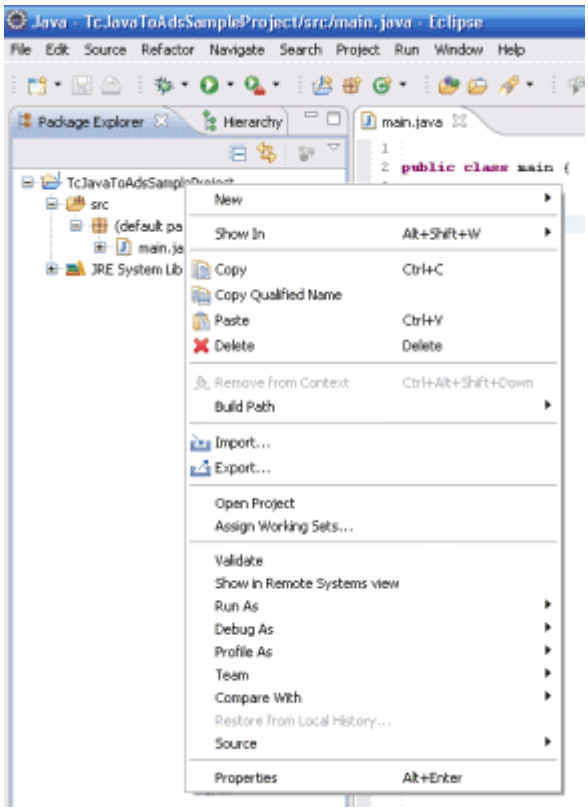
Samples

Here you can find article with sample programs [▶ 28] explaining how to use AdsToJava.dll in combination with TcJavaToAds.jar.

# 3 Implementing in Eclipse Galileo

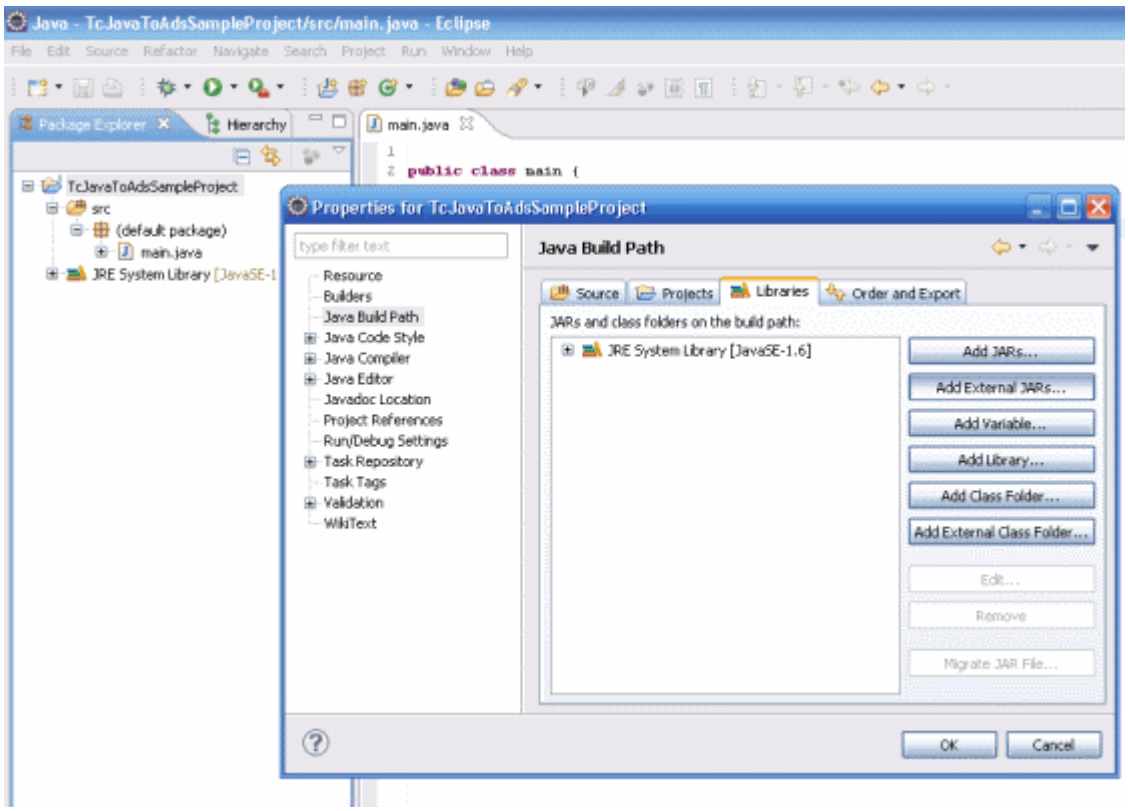
To use the AdsToJava.dll in the current Eclipse project, the TcJavaToAds.jar must be added to it. This article shows how to add the library and what imports are needed from it (please note that the instructions may need to be slightly varied on your system if the version you are using is different from this one):

First, Eclipse must be started and a new project created. After that please navigate to the "Package Explorer" and right click on your new project. In the menu that opens, select "Properties".




In the next step, select the "Java Build Path" category and click the "Add External JARs..." button. In the file browser that appears, locate the TcJavaToAds.jar file.

It is located in the "TwinCAT\ADS Api\AdsToJava" directory.



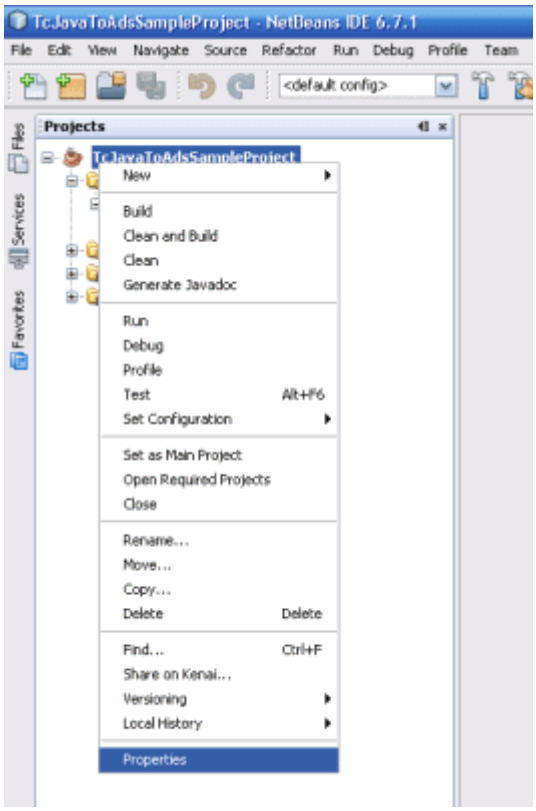

Finally, the packages whose components you want to use must be imported. For this purpose you can find documented sample programs [▶ 28] in the Information System.

```
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 Implementing in Netbeans 6.7.1

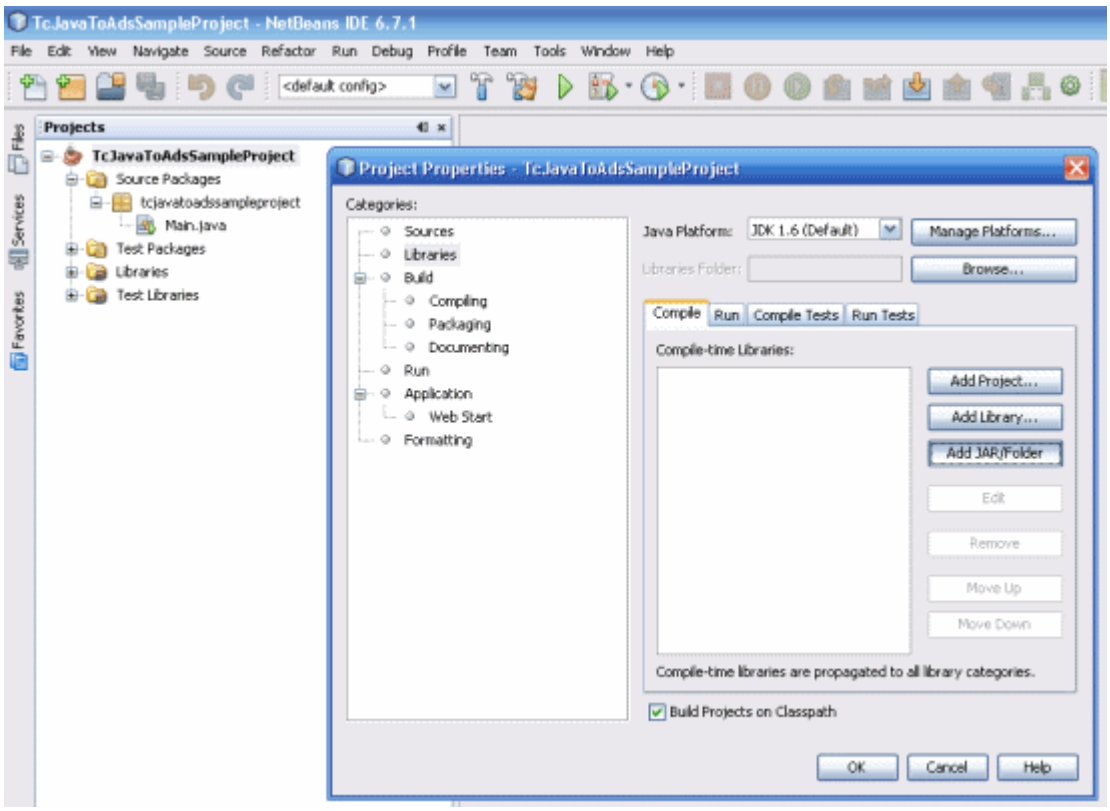
In order to use AdsToJava.dll in the current Netbeans project, TcJavaToAds.jar must be added to it. This article shows how to add the library and what imports are needed from it (please note that the instructions may need to be slightly varied on your system if the version you are using is different from this one):

First, Netbeans must be started, a new project must be created. After that please navigate to "Project" and right click on your new project. In the menu that opens, select "Properties".




In the next step, select the "Libraries" category and click the "Add JAR/Folder" button. In the file browser that appears, locate the TcJavaToAds.jar file.

It is located in the "TwinCAT\ADS Api\AdsToJava" directory.




Finally, the packages whose components you want to use must be imported. For this purpose you can find documented sample programs [▶ 28] in the 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 Functions

### 5.1.1 adsGetDllVersion

Returns the version number, revision number and build number of the ADS-DLL.

```
AdsVersion adsGetDllVersion ();
```

**Parameter**

**Return value**

*AdsVersion*: returns an AdsVersion object, with version number, revision number and build number set accordingly.

### 5.1.2 adsPortOpen

Establishes a connection (communication port) to the TwinCAT message router.

If no TwinCAT MessageRouter is present, the AdsPortOpenEx function will fail.

```
long adsPortOpen ();
```

**Parameter**

**Return value**

*long*: returns the error state of the function.

### 5.1.3 adsPortClose

The connection (communication port) to the TwinCAT message router is closed.

```
long adsPortClose ();
```

**Parameter**

**Return value**

*long*: returns the error state of the function.

### 5.1.4 adsGetLocalAddress

Returns the local NetId and the own port number.

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

**Parameter**

- **AmsAddr lj_AmsAddr**: AmsAddr object which contains the local NetId and port number after the call.

**Return value**

*long*: returns the error state of the function.

## 5.1.5 adsSyncWriteReq

Writes data synchronously to an ADS device.

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

**Parameter**

- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **long** *lj_length*: length of the data in bytes that are written to the ADS server.
- **JNIByteBuffer** *lj_pData*: JNIByteBuffer whose data is written to the ADS server.

**Return value**

*long*: returns the error state of the function.

## 5.1.6 adsSyncWriteReqArray

Writes data synchronously to an ADS device.

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

**Parameter**

- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **long** *lj_length*: length of the data in bytes that are written to the ADS server.
- **byte[]** *lj_pData*: a byte buffer whose data is written to the ADS server.

**Return value**

*long*: returns the error state of the function.

## 5.1.7 adsSyncReadReq

Reads data synchronously from an ADS server.

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

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

**Parameter**

- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_length*: length of the data in bytes.
- **JNIByteBuffer** *lj_pData*: JNIByteBuffer that stores the data to be read.

**Return value**

*long*: returns the error state of the function.

## 5.1.8 adsSyncReadReqEx

Reads data synchronously from an ADS server.

The additional JNILong buffer contains the number of bytes read after the call.

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

**Parameter**

- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_length*: length of the data in bytes.
- **JNIByteBuffer** *lj_pData*: JNIByteBuffer that stores the data to be read.
- **JNILong**: *lj_pBytesRead* JNILong buffer contains the number of bytes read after the call.

**Return value**

*long*: returns the error state of the function.

## 5.1.9 adsSyncReadWriteReq

Writes data synchronously into an ADS server and receives data back from the ADS device.

```
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*: object with NetId and port number from ADS server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_lengthRead*: length of the data in bytes that the ADS device returns.
- **JNIByteBuffer**: *lj_pDataRead* buffer with data returned by the ADS device.
- **long**: *lj_lengthWrite* length of data written to the ADS device in bytes.
- **JNIByteBuffer**: *lj_pDataWrite* buffer with data written to the ADS device.

**Return value**

*long*: returns the error state of the function.

## 5.1.10 adsSyncReadWriteReqEx

Writes data synchronously into an ADS server and receives data back from the ADS device (extended functionalities).

The additional JNILong buffer contains the number of bytes read after the call.

```
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*: object with NetId and port number from ADS server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_lengthRead*: length of the data in bytes that the ADS device returns.
- **JNIByteBuffer**: *lj_pDataRead* buffer with data returned by the ADS device.
- **long**: *lj_lengthWrite* length of data written to the ADS device in bytes.
- **JNIByteBuffer**: *lj_pDataWrite* buffer with data written to the ADS device.
- **JNILong**: *lj_pBytesRead* JNILong buffer contains the number of bytes read after the call.

**Return value**

*long*: returns the error state of the function.

## 5.1.11 adsSyncReadDeviceInfoReq

Reads the identification and version number of an ADS server.

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

**Parameter**

- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **AdsDevName** *lj_pDevName*: AdsDeviceName object containing the ADS device name after the call.
- **AdsVersion** *lj_pVersion*: AdsDeviceName object which contains the used version number, revision number and build number after the call.

**Return value**

*long*: returns the error state of the function.

## 5.1.12 adsSyncWriteControlReq

Changes the ADS state and the device state of an ADS server.

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

**Parameter**

- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **long** *lj_adsState*: new ADS state.
- **long** *lj_deviceState*: new device state.
- **long** *lj_length*: length of the data in bytes that are additionally written to the ADS server.
- **JNIByteBuffer**: *lj_pDataWrite* JNIByteBuffer with data that are additionally written to the ADS server.

**Return value**

*long*: returns the error state of the function.

**Comments**

In addition to change the ADS state and the device state, 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 state of the device itself (DeviceState) and the state 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

Reads the ADS state and the device state from an ADS server.

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

**Parameter**

- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **AdsState** *lj_nAdsState*: AdsState which contains the current ADS state after the call
- **AdsState** *lj_nDeviceState*: AdsState which contains the current device state after the call

**Return value**

*long*: returns the error state of the function.

## 5.1.14 adsSyncAddDeviceNotificationReq

A notification is defined within an ADS server (e.g. PLC). When a certain event occurs a function (the callback function) is invoked in the ADS client.

An AdsCallbackObject must be created to be able to receive information about the desired events. This must be assigned to a listener that implements the CallbackListenerAdsState interface. The listener is called as soon as the event occurs.

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

**Parameter**

- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **AdsNotificationAttrib** *lj_pNoteAttrib*: AdsNotificationAttrib object containing additional information about the event message.
- **long** *lj_hUser*: user handle
- **JNILong** *lj_pNotification*: JNILong buffer which contains the notification number after the call.

**Return value**

*long*: returns the error state of the function.

## 5.1.15 adsSyncDelDeviceNotificationReq

A notification defined previously is deleted from an ADS server.

This has no effect on the AdsCallback object.

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

**Parameter**

- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **JNILong** *lj_hNotification*: JNILong object containing the notification number to be deleted.

**Return value**

*long*: returns the error state of the function.

## 5.1.16 adsSyncSetTimeout

Alters the timeout for the ADS functions. The default value is 5000 ms.

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

### Parameter

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

### Return value

*long*: returns the error state of the function.

## 5.1.17 adsSyncGetTimeout

Returns the currently set timeout for ADS calls. The default value is 5000 ms.

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

### Parameter

- **JNILong** *lj_pMs*: JNILong object which contains the current timeout after the call.

### Return value

*long*: returns the error state of the function.

## 5.1.18 adsAmsRegisterRouterNotification

The AdsAmsRegisterNotificationReq()function can be used to detect a change in the status of the TwinCAT router. The given callback function is invoked each time the state changes. The AdsAmsUnRegisterNotification() function ends the status monitoring of the router again.

An AdsCallbackObject must be created to be able to receive information about the desired events. This must be assigned to a listener that implements the CallbackListenerAdsRouter interface. The listener is called as soon as the event occurs.

```
long adsAmsRegisterRouterNotification ();
```

### Parameter

### Return value

*long*: returns the error state of the function.

### Also see about this

adsAmsUnRegisterRouterNotification [▶ 19]

## 5.1.19 adsAmsUnRegisterRouterNotification

The AdsAmsUnRegisterNotification() function terminates the status monitoring of the TwinCAT router. See also AdsAmsRegisterNotificationReq() [▶ 19].

This has no effect on the AdsCallback object.

```
long adsAmsUnRegisterRouterNotification ();
```

**Parameter**

**Return value**

*long*: returns the error state of the function.

### 5.1.20 adsAmsPortEnabled

Returns a truth value indicating whether the passed port is open.

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

**Parameter**

- **JNIBool**: *j_pEnabled* JNIBool buffer which contains the truth value after the call.

**Return value**

*long*: returns the error state of the function.

### 5.1.21 getAdsCallbackObject

Returns the associated AdsCallback object.

```
AdsCallbackObject getAdsCallbackObject ();
```

**Parameter**

**Return value**

*AdsCallbackObject*: the AdsCallbackObject object

## 5.2 Extended functionalities

With the existing functions only one ADS port could be created for each process. Particularly for multithreaded applications this is not sufficient, since the individual ADS commands would block each other.

With the new functions it is now possible to use more than one port. This would enable a separate ADS port to be used for each thread, for example. New ports can be opened via the AdsPortOpenEx function. The returned port number is then transferred as parameter to the individual sync functions.

### 5.2.1 adsPortOpenEx

Establishes a connection (communication port) to the TwinCAT message router. Unlike with AdsPortOpen, a new ADS port is opened each time. The methods marked as thread-safe are passed the port number returned by AdsPortOpenEx as a parameter.

If no TwinCAT MessageRouter is present, the AdsPortOpenEx function will fail.

```
long adsPortOpenEx ();
```

**Parameter**

**Return value**

*long*: returns the error state of the function.

## 5.2.2 adsPortCloseEx

The connection (communication port) to the TwinCAT message router is closed. The port to be closed must previously have been opened via an AdsPortOpenEx call.

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

**Parameter**

- **long**: *lj_port* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.

**Return value**

*long*: returns the error state of the function.

## 5.2.3 adsGetLocalAddressEx

Returns the local NetId and the own port number (thread-safe).

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

**Parameter**

- **long**: *lj_port* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **AmsAddr** *lj_AmsAddr*: AmsAddr object which contains the local NetId and port number after the call.

**Return value**

*long*: returns the error state of the function.

## 5.2.4 adsSyncWriteReqEx

Writes data synchronously to an ADS device (thread-safe).

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

**Parameter**

- **long**: *lj_port* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.

- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **long** *lj_length*: length of the data in bytes that are written to the ADS server.
- **JNIByteBuffer** *lj_pData*: JNIByteBuffer whose data is written to the ADS server.

#### Return value

*long*: returns the error state of the function.

## 5.2.5 adsSyncWriteReqExArray

Writes data synchronously to an ADS device (thread-safe).

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

#### Parameter

- **long**: *lj_port* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **long** *lj_length*: length of the data in bytes that are written to the ADS server.
- **byte[]** *lj_pData*: a byte array buffer whose data is written to the ADS server.

#### Return value

*long*: returns the error state of the function.

## 5.2.6 adsSyncReadReqEx2

Reads data synchronously from an ADS device (thread-safe).

The additional JNILong buffer contains the number of bytes read after the call.

```
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* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_length*: length of the data in bytes.
- **JNIByteBuffer** *lj_pData*: JNIByteBuffer that stores the data to be read.

- **JNILong**: *lj_pBytesRead* JNILong buffer contains the number of bytes read after the call.

**Return value**

*long*: returns the error state of the function.

## 5.2.7 adsSyncReadWriteReqEx2

Writes data synchronously into an ADS server and receives data back from the ADS device (thread-safe).

The additional JNILong buffer contains the number of bytes read after the call.

```
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* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **long**: *lj_indexGroup* Index Group.
- **long**: *lj_indexOffset* Index Offset.
- **long** *lj_lengthRead*: length of the data in bytes that the ADS device returns.
- **JNIByteBuffer**: *lj_pDataRead* buffer with data returned by the ADS device.
- **long**: *lj_lengthWrite* length of data written to the ADS device in bytes.
- **JNIByteBuffer**: *lj_pDataWrite* buffer with data written to the ADS device.
- **JNILong**: *lj_pBytesRead* JNILong buffer contains the number of bytes read after the call.

**Return value**

*long*: returns the error state of the function.

## 5.2.8 adsSyncReadDeviceInfoReqEx

Reads the identification and version number of an ADS server (thread-safe).

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

**Parameter**

- **long**: *lj_port* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **AdsDevName** *lj_pDevName*: AdsDeviceName object containing the ADS device name after the call.
- **AdsVersion** *lj_pVersion*: AdsDeviceName object which contains the used version number, revision number and build number after the call.

### Return value

*long*: returns the error state of the function.

## 5.2.9 adsSyncWriteControlReqEx

Changes the ADS state and the device state of an ADS server (thread-safe).

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

### Parameter

- **long**: *lj_port* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **long** *lj_adsState*: new ADS state.
- **long** *lj_deviceState*: new device state.
- **long** *lj_length*: length of the data in bytes that are written to the ADS server.
- **byte[]** *lj_pData*: a JNIByteBuffer object whose data is additionally written to the ADS server.

### Return value

*long*: returns the error state of the function.

### Notes

In addition to changing the ADS state and the device state, it is also possible to send data to the ADS server to transfer further information. In the current ADS devices (PLC, NC, ...) this data has no further effect. Each ADS device can inform another ADS device about its current state. A distinction is made between the state of the device itself (DeviceState) and the state of the ADS interface of the ADS device (AdsState). The states that the ADS interface can assume are defined in the ADS specification.

## 5.2.10 adsSyncReadStateReqEx

Reads the ADS state and the device state of an ADS server (thread-safe).

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

### Parameter

- **long**: *lj_port* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **AdsState** *lj_nAdsState*: AdsState which contains the current ADS state after the call
- **AdsState** *lj_nDeviceState*: AdsState which contains the current device state after the call

**Return value**

*long*: returns the error state of the function.

## 5.2.11 adsSyncAddDeviceNotificationReqEx

A notification is defined within an ADS server (e.g. PLC). When a certain event occurs a function (the callback function) is invoked in the ADS client (tread-safe).

An AdsCallbackObject must be created to be able to receive information about the desired events. This must be assigned to a listener that implements the CallbackListenerAdsState interface. The listener is called as soon as the event occurs.

```
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* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **long** *lj_indexGroup*: Index Group
- **long** *lj_indexOffset*: Index Offset
- **AdsNotificationAttrib** *lj_pNoteAttrib*: AdsNotificationAttrib object containing additional information about the event message.
- **long** *lj_hUser*: user handle
- **JNILong** *lj_pNotification*: JNILong buffer which contains the notification number after the call.

**Return value**

*long*: returns the error state of the function.

## 5.2.12 adsSyncDelDeviceNotificationReqEx

A notification defined previously is deleted from an ADS server (thread-safe).

This has no effect on the AdsCallback object.

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

**Parameter**

- **long**: *lj_port* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **AmsAddr** *lj_AmsAddr*: object with NetId and port number from ADS server.
- **JNILong** *lj_hNotification*: JNILong object containing the notification number to be deleted.

**Return value**

*long*: returns the error state of the function.

## 5.2.13 adsSyncSetTimeoutEx

Alters the timeout for the ADS functions. The default value is 5000 ms (thread-safe).

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

**Parameter**

- **long**: *lj_port* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **long** *lj_nMs*: timeout in ms.

**Return value**

*long*: returns the error state of the function.

## 5.2.14 adsSyncGetTimeoutEx

Returns the currently set timeout for ADS calls. The default value is 5000 ms (thread-safe).

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

**Parameter**

- **long**: *lj_port* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **JNILong** *lj_pMs*: JNILong object which contains the current timeout after the call.

**Return value**

*long*: returns the error state of the function.

## 5.2.15 adsAmsPortEnabledEx

Returns a truth value indicating whether the passed port is open (thread-safe).

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

**Parameter**

- **long**: *lj_port* port number of an ADS port that had previously been opened with AdsPortOpenEx or AdsPortOpen.
- **JNIBool**: *j_pEnabled* JNIBool buffer which contains the truth value after the call.

**Return value**

*long*: returns the error state of the function.

# 6 Samples: TcJavaToAds

## 6.1 Using AdsToJava.dll

**Callback functions**

If callback functions are used, make sure that AdsToJava.dll version 1.1.0.2 or higher is installed.

### Task

Read PLC variables with a JAava application using the AdsToJava.dll library

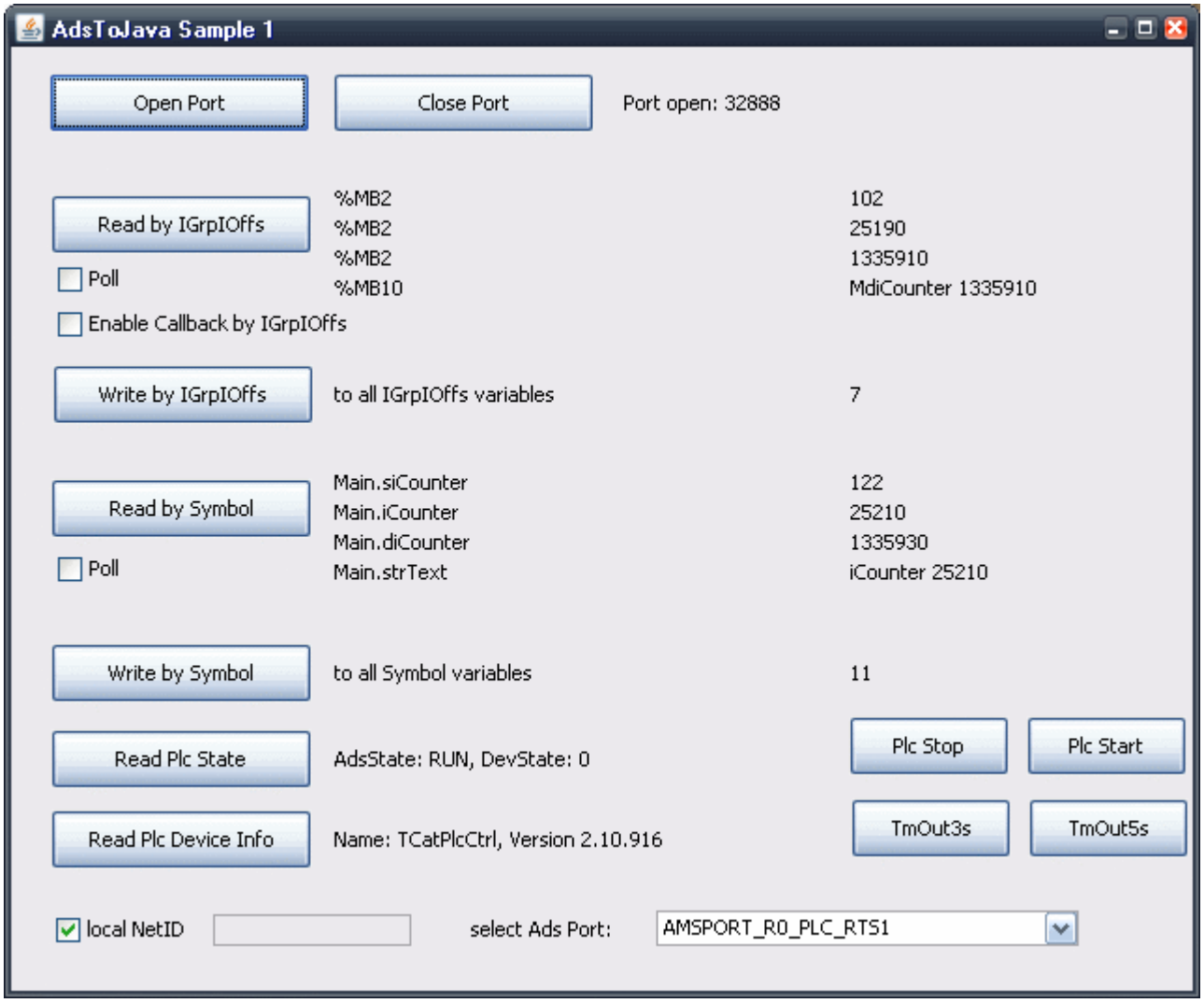
### Description

In this sample, the basic functionalities of the Java version of the ADS stack are presented.

To execute the *.jar sample, the command 'java -classpath "Sample01.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

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

download sample: https://infosys.beckhoff.com/content/1033/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 Access by variable name

The following program accesses a PLC variable that does not have an address. Access must therefore be made by the variable name.

Unpack the sample program 'Access by Variable Name': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475091595/.zip.

- To execute the *.jar sample, the command 'java -classpath "Sample02.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

```
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 Accessing an array in the PLC

An array, located in the PLC, is to be read by means of a read command. The variable is addressed here by its variable name. The procedure is only slightly different from that for reading discrete variables:

Unpack the sample program 'Access an array in the PLC': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475093003/.zip

- To execute the *.jar sample, the command 'java -classpath "Sample03.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

```
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 Transmitting structures to the PLC

This sample shows the transfer of a structure to the PLC via ADS. The structure consists of elements of different data types:

Unpack the sample program 'Transmitting structures to the PLC': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475094411/.zip

- To execute the *.jar sample, the command 'java -classpath "Sample04.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

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

The object to be transferred:

```
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 Reading out the PLC variable declaration of a single variable

The following information is transferred when accessing the variable declaration:

- Variable name
- Data type

- Length
- Address (IndexGroup / IndexOffset)
- Comment

Unpack the sample program 'Reading the PLC variable declaration of an individual variable': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475095819/.zip

- To execute the *.jar sample, the command 'java -classpath "Sample05.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

The AdsSyncReadWriteReq() call is used to read the variable information. The variable name is transferred to the function via parameter pWriteData. After the call the requested information is in a buffer of type JNIByteBuffer. The buffer is passed into the constructor of a variable of type AdsSymbolEntry. The individual information items in the PLC variables are stored in this class. The entryLenght, typeLength and commentLength fields specify the lengths of the associated strings located after the class. Next, the data type of the variable is determined by searching the adsDatatypeString for the label that matches the value. If the data type is UDINT or ARRAY OF UDINT, the value of this variable is also read:

```
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 Write flag synchronously into the PLC

In this sample program, the value that the user has entered is written into flag double word 0:

Unpack the sample program 'Write flag synchronously into PLC': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475097227/.zip

- To execute the *.jar sample, the command 'java -classpath "Sample06.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

```
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 Read flag synchronously from the PLC

In this sample program the value in flag double word 0 in the PLC is read and displayed:

Unpack the sample program 'Read flag synchronously from PLC': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475098635/.zip

- To execute the *.jar sample, the command 'java -classpath "Sample07.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

```
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 Delete handle of a PLC variable

In this sample a handle of a PLC variable is fetched and deleted afterwards:

Unpack the sample program 'Delete handle of a PLC variable': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475100043/.zip

- To execute the *.jar sample, the command 'java -classpath "Sample08.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

```
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 Event driven reading

If values from a PLC or NC are to be displayed continuously on a user interface, then it is very inefficient to use AdsSyncReadReq(), since this function must be called cyclically. By defining so-called notifications, a TwinCAT server can be caused to transfer values via ADS to another ADS device. A distinction is drawn between whether the TwinCAT server is to transmit the values cyclically, or only when the values change.

A notification is begun with the AdsSyncAddDeviceNotificationReq() function. After this, the callback function is automatically invoked by TwinCAT. AdsSyncDelDeviceNotificationReq() is used to halt the notification again. Since the number of notifications is limited, you should ensure the notifications no longer required by your program are deleted. You will find further information under the description of the AdsNotificationAttrib structure.

Unpack the sample program 'Event driven reading': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475101451/.zip

- To execute the *.jar sample, the command 'java -classpath "Sample09.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

The following program starts a notification on flag double word 0 in the PLC. Each time the PLC variable changes, the callback function is invoked. The callback function receives a variable of type AdsNotificationHeader() as one of its parameters. This structure contains all the necessary information (value, timestamp, ...).

**Time intensive actions**

No time-intensive actions may be executed in the callback.

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

The implementation of the 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 Reading SMB values from the TwinCAT I/O driver

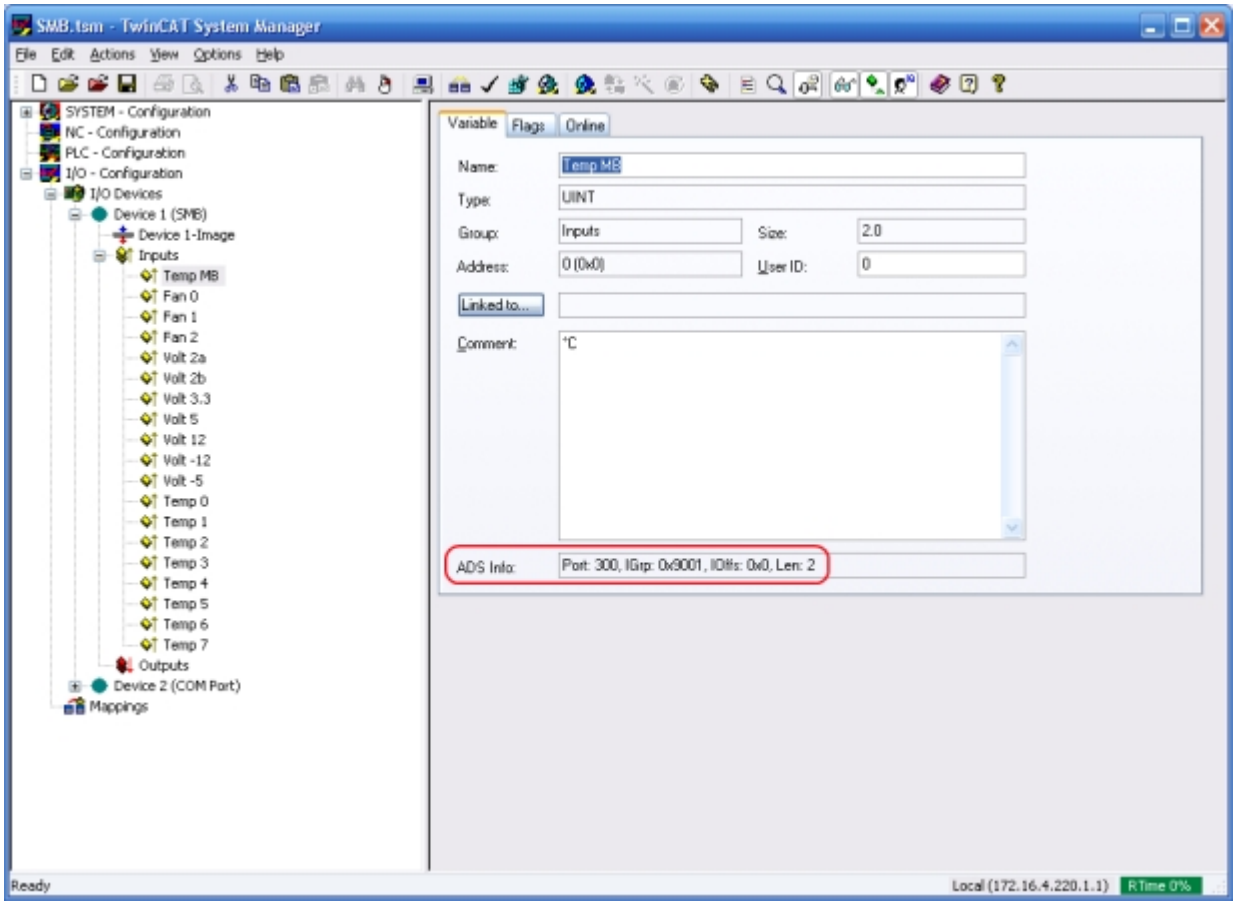
If only TwinCAT-CP is available instead of a TwinCAT PLC runtime environment, the SMB device can also be read out using the ADS client. The visualization allows constant monitoring of the desired temperature or fan values. Thus, it can be detected at an early stage whether the current temperature is approaching a critical value. The following is a sample of a possible implementation in Java (for correct operation, an SMB device must be entered in the current System Manager configuration)

Unpack the sample program 'Reading SMB values from the TwinCAT I/O driver': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475102859/.zip

- To execute the *.jar sample, the command 'java -classpath "Sample10.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

```
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 this view the respective IndexOffset of the different inputs can be read.

```
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 Event-Driven Detection of Changes to the Symbol Table

ADS devices that support symbol names (PLC, NC, ...) store those names in an internal table. A handle is assigned here to each symbol. The symbol handle is necessary in order to be able to access the variables. If the symbol table changes because, for instance, a new PLC program is written into the controller, the handles must be ascertained once again. The following sample illustrates how changes to the symbol table can be detected.

Unpack the sample program 'Event-driven detection of changes to the symbol table': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475104267/.zip

- To execute the *.jar sample, the command 'java -classpath "Sample11.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

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

Implementation of the 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 sum command: fetch and release multiple handles

This sample shows how to fetch and release many handles using the ADS sum command. Set up as AdsSyncReadWriteRequest, it serves as a container in which the subcommands are transported.

### System requirements:

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

Unpack the sample program 'ADS sum command: fetch and release multiple handles': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475105675/.zip

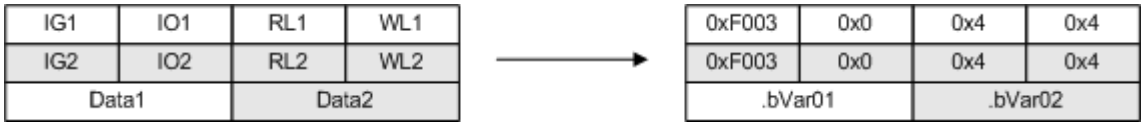
- To execute the *.jar sample, the command 'java -classpath "Sample12.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

### Fetch handles

```
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());
```

The names of the variables whose handles we want to get are appended to the end of the byte array.



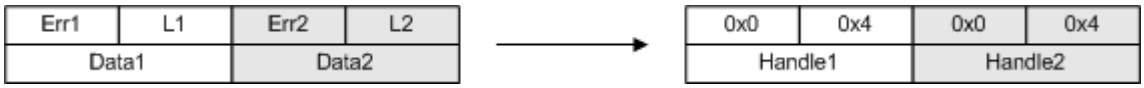

For communication, a port is opened and the local address is passed. If a transmission takes place, the port is assigned to the address by runtime system 1 beforehand.

The parameters for the sum command consist of IndexGroup (0xf082) - call of the sum command, IndexOffset (0x2) - number of subcommands, ReadLength (0x18) - size specification of the data to be read, ReadData (pBuffRes) - memory that accepts read data, WriteLength (cbReq) - size specification of the data to be sent and WriteLength (pBuffReq) - memory that contains data to be sent.

```
        // 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!");
        }
        }
```

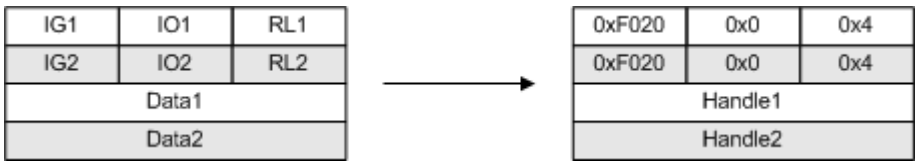
After sending the request, we expect an ADS error code and length for each handle we try to fetch. In this case the length will always be 4 bytes.




### Release handles

```
        // 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);
```

The handles associated with the commands are appended to the end of the byte array.




The existing connection is used to release the handles.

The parameters for the sum command consist of IndexGroup (0xf081) - call of the sum command, IndexOffset (0x2) - number of subcommands, ReadLength (cbRelRes) - size specification of the data to be read, ReadData (pBuffRelRes) - memory that accepts read data, WriteLength (cbRel) - size specification of the data to be sent and WriteLength (pBuffRel) - memory that contains data to be sent.

Finally, the handles must be released and the port closed.

```
        // 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!");
        }
    }
    }
}
```

In response we get an ADS error code for each handle we try to release.

# 6.13 ADS sum command: read and write multiple variables

This sample shows how to read or write many variables using the ADS sum command. Set up as TcAdsClient.ReadWrite it serves as a container in which the subcommands are transported in an ADS stream.

**System requirements:**

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

Unpack the sample program 'ADS sum command: read and write multiple variables': https://infosys.beckhoff.com/content/1033/tcjavatoads/Resources/12475107083/.zip

- To execute the *.jar sample, the command 'java -classpath "Sample13.jar;[path to TcJavaToAds.jar] Main' must be executed in the console in the correct directory (example path: "C:TwinCAT\Ads Api\AdsToJava*"). For this, java must be entered in the environment variables.

### Fetch symbol information of the PLC variables

```
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!");
        }
```

If variables are to be written or read in the PLC without resorting to handles, then IndexGroup and IndexOffset must be known. To get these values from all relevant variables, usually the symbol information has to be read out. In this sample, the values are assumed to be constant and defined in REQ_DATA.

### Read values

The RequestBuffer is now written with the data from REQ_DATA according to the scheme below. Here IG1 stands for the IndexGroup of the first subcommand. IO2 for the IndexOffset of the second subcommand and L3 for the length of the expected response of the third subcommand.

| IG1 | IO1 | L1 |
|---|---|---|
| IG2 | IO2 | L2 |
| IG3 | IO3 | L3 |

→

| 0x4040 | 0x2F | 0x2 |
|---|---|---|
| 0x4040 | 0x5 | 0x4 |
| 0x4040 | 0x4 | 0x1 |

As a response of the sum command we receive an Ads error code (Err) per subcommand, as well as the requested value (Data). Accordingly the length of the buffer in this case is: 3 * 4 bytes (Error) + 7 bytes (The sum of all lengths of the expected responses).

| Err1 | Err2 | Err3 |
|---|---|---|
| Data1 | Data2 | Data3 |

→

| 0x0 | 0x0 | 0x0 |
|---|---|---|
| uintValue | dintValue | boolValue |

The sum command itself is called with the parameters IndexGroup: 0xF080 (sum command read), IndexOffset: 0x3 (number of subcommands), ResponseBuffer length, ResponseBuffer, RequestBuffer length, RequestBuffer.

```
        // 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");
            }
        }
        }
```

### Write values

To be able to write data, this data must only be appended to the RequestBuffer from the upper part (see picture). All other parameters remain unchanged. Only an adjustment of the sum command is necessary (see below).


As a response of the sum command we get an Ads error code (Err) per subcommand, but this time no data, because we executed a write operation. Accordingly the length of the buffer in this case is: 3 * 4 bytes (Error).


For the sum command, the value for the IndexGroup must now be changed to 0xF081 (sum command write).

```
        //// 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

Grouping of error codes:
Global error codes: ADS Return Codes [▶ 56]... (0x9811_0000 ...)
Router error codes: ADS Return Codes [▶ 56]... (0x9811_0500 ...)
General ADS errors: ADS Return Codes [▶ 57]... (0x9811_0700 ...)
RTime error codes: ADS Return Codes [▶ 58]... (0x9811_1000 ...)

### Global error codes

| Hex | Dec | HRESULT | Name | Description |
|---|---|---|---|---|
| 0x0 | 0 | 0x98110000 | ERR_NOERROR | No error. |
| 0x1 | 1 | 0x98110001 | ERR_INTERNAL | Internal error. |
| 0x2 | 2 | 0x98110002 | ERR_NORTIME | No real time. |
| 0x3 | 3 | 0x98110003 | ERR_ALLOCLOCKEDMEM | Allocation locked – memory error. |
| 0x4 | 4 | 0x98110004 | ERR_INSERTMAILBOX | Mailbox full – the ADS message could not be sent. Reducing the number of ADS messages per cycle will help. |
| 0x5 | 5 | 0x98110005 | ERR_WRONGRECEIVEHMSG | Wrong HMSG. |
| 0x6 | 6 | 0x98110006 | ERR_TARGETPORTNOTFOUND | Target port not found – ADS server is not started or is not reachable. |
| 0x7 | 7 | 0x98110007 | ERR_TARGETMACHINENOTFOUND | Target computer not found – AMS route was not found. |
| 0x8 | 8 | 0x98110008 | ERR_UNKNOWNCMDID | Unknown command ID. |
| 0x9 | 9 | 0x98110009 | ERR_BADTASKID | Invalid task ID. |
| 0xA | 10 | 0x9811000A | ERR_NOIO | No IO. |
| 0xB | 11 | 0x9811000B | ERR_UNKNOWNAMSCMD | Unknown AMS command. |
| 0xC | 12 | 0x9811000C | ERR_WIN32ERROR | Win32 error. |
| 0xD | 13 | 0x9811000D | ERR_PORTNOTCONNECTED | Port not connected. |
| 0xE | 14 | 0x9811000E | ERR_INVALIDAMSLENGTH | Invalid AMS length. |
| 0xF | 15 | 0x9811000F | ERR_INVALIDAMSNETID | Invalid AMS Net ID. |
| 0x10 | 16 | 0x98110010 | ERR_LOWINSTLEVEL | Installation level is too low –TwinCAT 2 license error. |
| 0x11 | 17 | 0x98110011 | ERR_NODEBUGINTAVAILABLE | No debugging available. |
| 0x12 | 18 | 0x98110012 | ERR_PORTDISABLED | Port disabled – TwinCAT system service not started. |
| 0x13 | 19 | 0x98110013 | ERR_PORTALREADYCONNECTED | Port already connected. |
| 0x14 | 20 | 0x98110014 | ERR_AMSSYNC_W32ERROR | AMS Sync Win32 error. |
| 0x15 | 21 | 0x98110015 | ERR_AMSSYNC_TIMEOUT | AMS Sync Timeout. |
| 0x16 | 22 | 0x98110016 | ERR_AMSSYNC_AMSERROR | AMS Sync error. |
| 0x17 | 23 | 0x98110017 | ERR_AMSSYNC_NOINDEXINMAP | No index map for AMS Sync available. |
| 0x18 | 24 | 0x98110018 | ERR_INVALIDAMSPORT | Invalid AMS port. |
| 0x19 | 25 | 0x98110019 | ERR_NOMEMORY | No memory. |
| 0x1A | 26 | 0x9811001A | ERR_TCPSEND | TCP send error. |
| 0x1B | 27 | 0x9811001B | ERR_HOSTUNREACHABLE | Host unreachable. |
| 0x1C | 28 | 0x9811001C | ERR_INVALIDAMSFRAGMENT | Invalid AMS fragment. |
| 0x1D | 29 | 0x9811001D | ERR_TLSSEND | TLS send error – secure ADS connection failed. |
| 0x1E | 30 | 0x9811001E | ERR_ACCESSDENIED | Access denied – secure ADS access denied. |

### Router error codes

| Hex | Dec | HRESULT | Name | Description |
|---|---|---|---|---|
| 0x500 | 1280 | 0x98110500 | ROUTERERR_NOLOCKEDMEMORY | Locked memory cannot be allocated. |
| 0x501 | 1281 | 0x98110501 | ROUTERERR_RESIZEMEMORY | The router memory size could not be changed. |
| 0x502 | 1282 | 0x98110502 | ROUTERERR_MAILBOXFULL | The mailbox has reached the maximum number of possible messages. |
| 0x503 | 1283 | 0x98110503 | ROUTERERR_DEBUGBOXFULL | The Debug mailbox has reached the maximum number of possible messages. |
| 0x504 | 1284 | 0x98110504 | ROUTERERR_UNKNOWNPORTTYPE | The port type is unknown. |
| 0x505 | 1285 | 0x98110505 | ROUTERERR_NOTINITIALIZED | The router is not initialized. |
| 0x506 | 1286 | 0x98110506 | ROUTERERR_PORTALREADYINUSE | The port number is already assigned. |

| Hex | Dec | HRESULT | Name | Description |
|---|---|---|---|---|
| 0x507 | 1287 | 0x98110507 | ROUTERERR_NOTREGISTERED | The port is not registered. |
| 0x508 | 1288 | 0x98110508 | ROUTERERR_NOMOREQUEUES | The maximum number of ports has been reached. |
| 0x509 | 1289 | 0x98110509 | ROUTERERR_INVALIDPORT | The port is invalid. |
| 0x50A | 1290 | 0x9811050A | ROUTERERR_NOTACTIVATED | The router is not active. |
| 0x50B | 1291 | 0x9811050B | ROUTERERR_FRAGMENTBOXFULL | The mailbox has reached the maximum number for fragmented messages. |
| 0x50C | 1292 | 0x9811050C | ROUTERERR_FRAGMENTTIMEOUT | A fragment timeout has occurred. |
| 0x50D | 1293 | 0x9811050D | ROUTERERR_TOBEREMOVED | The port is removed. |

### General ADS error codes

| Hex | Dec | HRESULT | Name | Description |
|---|---|---|---|---|
| 0x700 | 1792 | 0x98110700 | ADSERR_DEVICE_ERROR | General device error. |
| 0x701 | 1793 | 0x98110701 | ADSERR_DEVICE_SRVNOTSUPP | Service is not supported by the server. |
| 0x702 | 1794 | 0x98110702 | ADSERR_DEVICE_INVALIDGRP | Invalid index group. |
| 0x703 | 1795 | 0x98110703 | ADSERR_DEVICE_INVALIDOFFSET | Invalid index offset. |
| 0x704 | 1796 | 0x98110704 | ADSERR_DEVICE_INVALIDACCESS | Reading or writing not permitted. |
| 0x705 | 1797 | 0x98110705 | ADSERR_DEVICE_INVALIDSIZE | Parameter size not correct. |
| 0x706 | 1798 | 0x98110706 | ADSERR_DEVICE_INVALIDDATA | Invalid data values. |
| 0x707 | 1799 | 0x98110707 | ADSERR_DEVICE_NOTREADY | Device is not ready to operate. |
| 0x708 | 1800 | 0x98110708 | ADSERR_DEVICE_BUSY | Device is busy. |
| 0x709 | 1801 | 0x98110709 | ADSERR_DEVICE_INVALIDCONTEXT | Invalid operating system context. This can result from use of ADS blocks in different tasks. It may be possible to resolve this through multitasking synchronization in the PLC. |
| 0x70A | 1802 | 0x9811070A | ADSERR_DEVICE_NOMEMORY | Insufficient memory. |
| 0x70B | 1803 | 0x9811070B | ADSERR_DEVICE_INVALIDPARM | Invalid parameter values. |
| 0x70C | 1804 | 0x9811070C | ADSERR_DEVICE_NOTFOUND | Not found (files, ...). |
| 0x70D | 1805 | 0x9811070D | ADSERR_DEVICE_SYNTAX | Syntax error in file or command. |
| 0x70E | 1806 | 0x9811070E | ADSERR_DEVICE_INCOMPATIBLE | Objects do not match. |
| 0x70F | 1807 | 0x9811070F | ADSERR_DEVICE_EXISTS | Object already exists. |
| 0x710 | 1808 | 0x98110710 | ADSERR_DEVICE_SYMBOLNOTFOUND | Symbol not found. |
| 0x711 | 1809 | 0x98110711 | ADSERR_DEVICE_SYMBOLVERSIONINVALID | Invalid symbol version. This can occur due to an online change. Create a new handle. |
| 0x712 | 1810 | 0x98110712 | ADSERR_DEVICE_INVALIDSTATE | Device (server) is in invalid state. |
| 0x713 | 1811 | 0x98110713 | ADSERR_DEVICE_TRANSMODENOTSUPP | AdsTransMode not supported. |
| 0x714 | 1812 | 0x98110714 | ADSERR_DEVICE_NOTIFYHNDINVALID | Notification handle is invalid. |
| 0x715 | 1813 | 0x98110715 | ADSERR_DEVICE_CLIENTUNKNOWN | Notification client not registered. |
| 0x716 | 1814 | 0x98110716 | ADSERR_DEVICE_NOMOREHDLS | No further handle available. |
| 0x717 | 1815 | 0x98110717 | ADSERR_DEVICE_INVALIDWATCHSIZE | Notification size too large. |
| 0x718 | 1816 | 0x98110718 | ADSERR_DEVICE_NOTINIT | Device not initialized. |
| 0x719 | 1817 | 0x98110719 | ADSERR_DEVICE_TIMEOUT | Device has a timeout. |
| 0x71A | 1818 | 0x9811071A | ADSERR_DEVICE_NOINTERFACE | Interface query failed. |
| 0x71B | 1819 | 0x9811071B | ADSERR_DEVICE_INVALIDINTERFACE | Wrong interface requested. |
| 0x71C | 1820 | 0x9811071C | ADSERR_DEVICE_INVALIDCLSID | Class ID is invalid. |
| 0x71D | 1821 | 0x9811071D | ADSERR_DEVICE_INVALIDOBJID | Object ID is invalid. |
| 0x71E | 1822 | 0x9811071E | ADSERR_DEVICE_PENDING | Request pending. |
| 0x71F | 1823 | 0x9811071F | ADSERR_DEVICE_ABORTED | Request is aborted. |
| 0x720 | 1824 | 0x98110720 | ADSERR_DEVICE_WARNING | Signal warning. |
| 0x721 | 1825 | 0x98110721 | ADSERR_DEVICE_INVALIDARRAYIDX | Invalid array index. |
| 0x722 | 1826 | 0x98110722 | ADSERR_DEVICE_SYMBOLNOTACTIVE | Symbol not active. |
| 0x723 | 1827 | 0x98110723 | ADSERR_DEVICE_ACCESSDENIED | Access denied. |
| 0x724 | 1828 | 0x98110724 | ADSERR_DEVICE_LICENSENOTFOUND | Missing license. |
| 0x725 | 1829 | 0x98110725 | ADSERR_DEVICE_LICENSEEXPIRED | License expired. |
| 0x726 | 1830 | 0x98110726 | ADSERR_DEVICE_LICENSEEXCEEDED | License exceeded. |
| 0x727 | 1831 | 0x98110727 | ADSERR_DEVICE_LICENSEINVALID | Invalid license. |
| 0x728 | 1832 | 0x98110728 | ADSERR_DEVICE_LICENSESYSTEMID | License problem: System ID is invalid. |
| 0x729 | 1833 | 0x98110729 | ADSERR_DEVICE_LICENSENOTIMELIMIT | License not limited in time. |
| 0x72A | 1834 | 0x9811072A | ADSERR_DEVICE_LICENSEFUTUREISSUE | Licensing problem: time in the future. |
| 0x72B | 1835 | 0x9811072B | ADSERR_DEVICE_LICENSETIMETOLONG | License period too long. |

| Hex | Dec | HRESULT | Name | Description |
|---|---|---|---|---|
| 0x72C | 1836 | 0x9811072C | ADSERR_DEVICE_EXCEPTION | Exception at system startup. |
| 0x72D | 1837 | 0x9811072D | ADSERR_DEVICE_LICENSEDUPLICATED | License file read twice. |
| 0x72E | 1838 | 0x9811072E | ADSERR_DEVICE_SIGNATUREINVALID | Invalid signature. |
| 0x72F | 1839 | 0x9811072F | ADSERR_DEVICE_CERTIFICATEINVALID | Invalid certificate. |
| 0x730 | 1840 | 0x98110730 | ADSERR_DEVICE_LICENSEOEMNOTFOUND | Public key not known from OEM. |
| 0x731 | 1841 | 0x98110731 | ADSERR_DEVICE_LICENSERESTRICTED | License not valid for this system ID. |
| 0x732 | 1842 | 0x98110732 | ADSERR_DEVICE_LICENSEDEMODENIED | Demo license prohibited. |
| 0x733 | 1843 | 0x98110733 | ADSERR_DEVICE_INVALIDFNCID | Invalid function ID. |
| 0x734 | 1844 | 0x98110734 | ADSERR_DEVICE_OUTOFRANGE | Outside the valid range. |
| 0x735 | 1845 | 0x98110735 | ADSERR_DEVICE_INVALIDALIGNMENT | Invalid alignment. |
| 0x736 | 1846 | 0x98110736 | ADSERR_DEVICE_LICENSEPLATFORM | Invalid platform level. |
| 0x737 | 1847 | 0x98110737 | ADSERR_DEVICE_FORWARD_PL | Context – forward to passive level. |
| 0x738 | 1848 | 0x98110738 | ADSERR_DEVICE_FORWARD_DL | Context – forward to dispatch level. |
| 0x739 | 1849 | 0x98110739 | ADSERR_DEVICE_FORWARD_RT | Context – forward to real time. |
| 0x740 | 1856 | 0x98110740 | ADSERR_CLIENT_ERROR | Client error. |
| 0x741 | 1857 | 0x98110741 | ADSERR_CLIENT_INVALIDPARM | Service contains an invalid parameter. |
| 0x742 | 1858 | 0x98110742 | ADSERR_CLIENT_LISTEMPTY | Polling list is empty. |
| 0x743 | 1859 | 0x98110743 | ADSERR_CLIENT_VARUSED | Var connection already in use. |
| 0x744 | 1860 | 0x98110744 | ADSERR_CLIENT_DUPLINVOKEID | The called ID is already in use. |
| 0x745 | 1861 | 0x98110745 | ADSERR_CLIENT_SYNCTIMEOUT | Timeout has occurred – the remote terminal is not responding in the specified ADS timeout. The route setting of the remote terminal may be configured incorrectly. |
| 0x746 | 1862 | 0x98110746 | ADSERR_CLIENT_W32ERROR | Error in Win32 subsystem. |
| 0x747 | 1863 | 0x98110747 | ADSERR_CLIENT_TIMEOUTINVALID | Invalid client timeout value. |
| 0x748 | 1864 | 0x98110748 | ADSERR_CLIENT_PORTNOTOPEN | Port not open. |
| 0x749 | 1865 | 0x98110749 | ADSERR_CLIENT_NOAMSADDR | No AMS address. |
| 0x750 | 1872 | 0x98110750 | ADSERR_CLIENT_SYNCINTERNAL | Internal error in Ads sync. |
| 0x751 | 1873 | 0x98110751 | ADSERR_CLIENT_ADDHASH | Hash table overflow. |
| 0x752 | 1874 | 0x98110752 | ADSERR_CLIENT_REMOVEHASH | Key not found in the table. |
| 0x753 | 1875 | 0x98110753 | ADSERR_CLIENT_NOMORESYM | No symbols in the cache. |
| 0x754 | 1876 | 0x98110754 | ADSERR_CLIENT_SYNCRESINVALID | Invalid response received. |
| 0x755 | 1877 | 0x98110755 | ADSERR_CLIENT_SYNCPORTLOCKED | Sync Port is locked. |

### RTime error codes

| Hex | Dec | HRESULT | Name | Description |
|---|---|---|---|---|
| 0x1000 | 4096 | 0x98111000 | RTERR_INTERNAL | Internal error in the real-time system. |
| 0x1001 | 4097 | 0x98111001 | RTERR_BADTIMERPERIODS | Timer value is not valid. |
| 0x1002 | 4098 | 0x98111002 | RTERR_INVALIDTASKPTR | Task pointer has the invalid value 0 (zero). |
| 0x1003 | 4099 | 0x98111003 | RTERR_INVALIDSTACKPTR | Stack pointer has the invalid value 0 (zero). |
| 0x1004 | 4100 | 0x98111004 | RTERR_PRIOEXISTS | The request task priority is already assigned. |
| 0x1005 | 4101 | 0x98111005 | RTERR_NOMORETCB | No free TCB (Task Control Block) available. The maximum number of TCBs is 64. |
| 0x1006 | 4102 | 0x98111006 | RTERR_NOMORESEMAS | No free semaphores available. The maximum number of semaphores is 64. |
| 0x1007 | 4103 | 0x98111007 | RTERR_NOMOREQUEUES | No free space available in the queue. The maximum number of positions in the queue is 64. |
| 0x100D | 4109 | 0x9811100D | RTERR_EXTIRQALREADYDEF | An external synchronization interrupt is already applied. |
| 0x100E | 4110 | 0x9811100E | RTERR_EXTIRQNOTDEF | No external sync interrupt applied. |
| 0x100F | 4111 | 0x9811100F | RTERR_EXTIRQINSTALLFAILED | Application of the external synchronization interrupt has failed. |
| 0x1010 | 4112 | 0x98111010 | RTERR_IRQLNOTLESSOREQUAL | Call of a service function in the wrong context |
| 0x1017 | 4119 | 0x98111017 | RTERR_VMXNOTSUPPORTED | Intel VT-x extension is not supported. |
| 0x1018 | 4120 | 0x98111018 | RTERR_VMXDISABLED | Intel VT-x extension is not enabled in the BIOS. |
| 0x1019 | 4121 | 0x98111019 | RTERR_VMXCONTROLSMISSING | Missing function in Intel VT-x extension. |
| 0x101A | 4122 | 0x9811101A | RTERR_VMXENABLEFAILS | Activation of Intel VT-x fails. |

### Specific positive HRESULT Return Codes:

| HRESULT | Name | Description |
|---|---|---|
| 0x0000_0000 | S_OK | No error. |
| 0x0000_0001 | S_FALSE | No error.<br>Example: successful processing, but with a negative or incomplete result. |
| 0x0000_0203 | S_PENDING | No error.<br>Example: successful processing, but no result is available yet. |
| 0x0000_0256 | S_WATCHDOG_TIMEOUT | No error.<br>Example: successful processing, but a timeout occurred. |

### TCP Winsock error codes

| Hex | Dec | Name | Description |
|---|---|---|---|
| 0x274C | 10060 | WSAETIMEDOUT | A connection timeout has occurred - error while establishing the connection, because the remote terminal did not respond properly after a certain period of time, or the established connection could not be maintained because the connected host did not respond. |
| 0x274D | 10061 | WSAECONNREFUSED | Connection refused - no connection could be established because the target computer has explicitly rejected it. This error usually results from an attempt to connect to a service that is inactive on the external host, that is, a service for which no server application is running. |
| 0x2751 | 10065 | WSAEHOSTUNREACH | No route to host - a socket operation referred to an unavailable host. |
| More Winsock error codes: Win32 error codes | | | |

More Information:
**www.beckhoff.com**

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com