AusweisApp SDK

Release 2.2.0

Governikus GmbH & Co. KG

Jul 01, 2024

Contents

1 Introduction

This documentation will explain how to initialize and start up the AusweisApp as an additional service. It distinguishes between a connection to the application and the communication between your application and AusweisApp.

The section *Connection* (page 4) will show you what you need to do to set up a connection to AusweisApp. Once you have established a connection you can send and receive JSON documents in a bi-directional manner. There are different commands and messages. These are listed and described in the section *Protocol* (page 22). The protocol is split up in *Commands* (page 22) and *Messages* (page 31). Commands will be sent by your application to control AusweisApp. Messages contain additional information to your command or will be sent as an event.

Also this documentation provides some example workflows to show a possible communication.

See also:

For Android and iOS there is also the AusweisApp SDK Wrapper[1] which is a software library that offers a high-level interface to the AusweisApp SDK.

Important: The AusweisApp does **not** provide any personal data to your client application directly as AusweisApp does not have access to this data for security reasons. AusweisApp facilitates a secure connection between the eID server and the ID card, enabling the eID server to get those data from the card.

This way your backend receives high level trust data. Since your client application runs in a user's environment, you could not be sure about the integrity of the data if your client application were to receive high sensitive data from the AusweisApp directly as your backend does not have any possibility to verify the source of the data.

Also this approach, recommended for compliance reasons by the Federal Office for Information Security, spares your client application the necessity of encypting these high sensitive data.

In case your client application requires data input from the ID card, you need to get this from the backend system (e.g. the eID server) after a successful authentication.

See also:

TR-03124[2], part 1: Specifications

---

2 Changelog

2.1 Version 2.2.0

- Introduced the following additions in *API_LEVEL* (page 33) **3**:
  - The command *CONTINUE* (page 27).
  - The message *PAUSE* (page 45).
  - Extended parameter **card** of the message *READER* (page 42) to signal an unknown card.

[1] https://www.ausweisapp.bund.de/sdkwrapper/

[2] https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03124/TR-03124-1.pdf

- Added parameter **content** to the *Filesystem* (page 62) of the Simulator.
- Declared parameter **private** of the Simulator's *Filesystem* (page 62) deprecated.

2.2 Version 1.26.4

- Added parameter **keys** to the command *SET_CARD* (page 26).
- Added parameter **reason** to the message *CHANGE_PIN* (page 36).

2.3 Version 1.26.3

- Added variable "AUSWEISAPP2_AUTOMATIC_DEVELOPERMODE" to the mode *Automatic* (page 19).
- Added parameter **reason** to message *AUTH* (page 34).

2.4 Version 1.24.0

- Introduced the following changes in *API_LEVEL* (page 33) **2**:
  - The commands *GET_STATUS* (page 22) and *SET_CARD* (page 26).
  - Removed the parameter **handleInterrupt** of the commands *RUN_AUTH* (page 23) and *RUN_CHANGE_PIN* (page 24).
  - Added the parameter **status** to the commands *RUN_AUTH* (page 23) and *RUN_CHANGE_PIN* (page 24).
  - Renamed parameter **reader** to **readers** in message *READER* (page 42)
  - Added message *STATUS* (page 44).
- Added mode *Automatic* (page 19) for Desktop Systems.
- Added parameter **insertable** to message *READER* (page 42).

2.5 Version 1.22.3

- Added parameter **developerMode** to the command *RUN_AUTH* (page 23).

2.6 Version 1.22.1

- Added parameter **handleInterrupt** and **messages** to the commands *RUN_AUTH* (page 23) and *RUN_CHANGE_PIN* (page 24) for *API_LEVEL* (page 33) **1**.
- Added the command *INTERRUPT* (page 28).

2.7 Version 1.22.0

- Added the commands *RUN_CHANGE_PIN* (page 24) and *SET_NEW_PIN* (page 29).
- Extended the message *ACCESS_RIGHTS* (page 31) for “CAN allowed right” and “PIN management right”.
- Added messages *CHANGE_PIN* (page 36) and *ENTER_NEW_PIN* (page 39).

2.8 Version 1.20.0

- Extended *ACCESS_RIGHTS* (page 31) for write access.

2.9 Version 1.16.0

- Changed parameter **value** for the commands *SET_PIN* (page 29), *SET_CAN* (page 30), and *SET_PUK* (page 30) for readers with a keypad.
- Added parameter **keypad** to message *READER* (page 42).

2.10 Version 1.14.2

- The message *SET_CAN* (page 30) may now be used with the “CAN allowed right”.

3 Android

This chapter deals with the Android specific properties of the AusweisApp SDK. The AusweisApp core is encapsulated into an **Android service** which is running in the background without a user interface. This service is interfaced via an Android specific interprocess communication (IPC) mechanism. The basics of this mechanism - the **Android Interface Definition Language** (AIDL) - are introduced in the following section. Subsequent sections deal with the SDK interface itself and explain which steps are necessary in order to communicate with the AusweisApp SDK.

The AusweisApp is available as an AAR package that can automatically be fetched by Android's default build system **gradle**.

See also:

For Android there is also the AusweisApp SDK Wrapper[3] which is a software library that offers a high-level interface to the AusweisApp SDK.

Important: The AAR package is available in maven central for free. If you need enterprise support feel free to contact us.

Important: Please note that only Android devices that feature Extended Length NFC communication are supported by the SDK. This can be checked via the Android API methods **getMaxTransceiveLength**() and **isExtendedLengthApduSupported**().

https://developer.android.com/reference/android/nfc/tech/NfcA
https://developer.android.com/reference/android/nfc/tech/IsoDep

---

3.1 SDK

The AusweisApp SDK is distributed as an AAR package that contains native **arm64-v8a** libraries only. The AAR package is available in the default repository of Android. The following listing shows the required **mavenCentral** in **build.gradle**.

```
buildscript {
    repositories {
        mavenCentral()
    }
}
```

The AusweisApp SDK will be fetched automatically as a dependency by your **app/build.gradle** file. It is recommended to always use the latest version (2.2.0) of AusweisApp.

```
dependencies {
    implementation 'com.governikus:ausweisapp:x.y.z'
}
```

[3] https://www.ausweisapp.bund.de/sdkwrapper/

Note: All artifacts are signed with the following key (available on all public key servers): 0x699BF3055B0A49224EFDE7C72D7479A531451088

See also:

The AAR package provides an **AndroidManifest.xml** to register required permissions and the background service. If your own AndroidManifest has conflicts with our provided file you can add some attributes to resolve those conflicts.

https://developer.android.com/studio/build/manifest-merge.html

App Bundle

The AusweisApp SDK uses native libraries which need to be extracted when used in an App Bundle, otherwise the SDK will not work correctly.

Add the following statement to your app's build.gradle file:

```
android { packagingOptions { jniLibs { useLegacyPackaging = true } } }
```

Logging

The AusweisApp SDK uses default logging of Android and has its own log file. It is **recommended** to collect that log file if an error occurs in your application to receive better support.

The log file is in your application path:

```
/data/data/your.application.name/files/AusweisApp.XXXXXX.log
```

The *XXXXXX* characters will be replaced by an automatically generated portion of the filename to avoid conflicts with previous instances.

A new log file will be created for each new instance of the AusweisApp and will be deleted after a correct shutdown. In case of old or multiple log files, it is highly probable that the previous instance crashed.

The AusweisApp deletes any log files that are older than 14 days.

Initialization of the Android Application

The AusweisApp SDK creates a fork of the Android "main" Application if started. Due to this, the Application is instantiated a second time. Thus, it must ensure that any initialization (e.g. Firebase connections) is only carried out once. To do so the following snippet may prove useful:

```
public class MyAwesomeApp extends Application
{
    private static final String AA2_PROCESS = "ausweisapp2_service";

    @Override
    public void onCreate()
    {
```

(continues on next page)

(continued from previous page)

```
        super.onCreate();
        if (isAA2Process())
            return;

        // Perform one-time initialization of YOUR app, e.g. Firebase␣
↪connection
    }

    private boolean isAA2Process()
    {
        return Application.getProcessName().endsWith(AA2_PROCESS);
    }
}
```

Import the AIDL files

Android provides an interprocess communication (IPC) mechanism which is based on messages consisting of primitive types. In order to abstract from this very basic mechanism, there is the Android Interface Definition Language (AIDL). It allows the definition of Java like interfaces. The Android SDK generates the necessary interface implementations from supplied AIDL files in order to perform IPC, as if this function had been called directly in the current process.

In order to interact with the AusweisApp SDK there are two AIDL interfaces. The first one is given to the client application by the SDK and allows the client to establish a session with the SDK, to send JSON commands to the SDK and to pass discovered NFC tags to the SDK.

The second AIDL interface is given to the SDK by the client application. It enables the client to receive the initial session parameters as well as JSON messages from the SDK. Furthermore it has a function which is called when an existing connection with the SDK is dropped by the SDK. Both interfaces are listed below and you need to import them into your build environment.

See also:

https://developer.android.com/guide/components/aidl.html

Interface

```
package com.governikus.ausweisapp2;

import com.governikus.ausweisapp2.IAusweisApp2SdkCallback;
import android.nfc.Tag;

interface IAusweisApp2Sdk
{
    boolean connectSdk(IAusweisApp2SdkCallback pCallback);
    boolean send(String pSessionId, String pMessageFromClient);
    boolean updateNfcTag(String pSessionId, in Tag pTag);
}
```

Callback

```
package com.governikus.ausweisapp2;

interface IAusweisApp2SdkCallback
{
    void sessionIdGenerated(String pSessionId, boolean pIsSecureSessionId);
    void receive(String pJson);
    void sdkDisconnected();
}
```

3.2 Background service

The AusweisApp SDK is an embedded background service in your own application.

Binding to the service

In order to start the AusweisApp SDK it is necessary to bind to the Android service supplied by the SDK. This binding fulfils two purposes:

- First it starts the SDK.
- Second it enables the client to establish an IPC connection as mentioned above.

Due to the nature of an Android service, there can be only one instance of the SDK running. If multiple clients in your application bind to the service, they are interacting with the same instance of the service. The service is terminated once all previously bound clients are unbound.

To differentiate between different connected clients, virtual sessions are used once the binding is completed. These sessions are discussed in a separate section, section *Create session to AusweisApp* (page 9).

Create connection

First of all, in order to bind to the service, one needs to instantiate an Android ServiceConnection. Subsequently, the object is passed to the Android API and the contained methods are invoked by Android on service connection and disconnection.

```
import android.content.ServiceConnection;

// [...]

ServiceConnection mConnection = new ServiceConnection()
{
  @Override
  public void onServiceConnected(ComponentName className, IBinder service)
  {
      // ... details below
  }
```

(continues on next page)

(continued from previous page)

```
  @Override
  public void onServiceDisconnected(ComponentName className)
  {
      // ... details below
  }
}
```

Bind service to raw connection

In order to perform the actual binding a directed Intent, which identifies the AusweisApp SDK, is created. This Intent is sent to the Android API along with the ServiceConnection created above. This API call either starts up the SDK if it is the first client, or connects to the running SDK instance if there is already another client bound.

You need to pass your own package name as the AusweisApp SDK is a background service of your application.

```
import android.app.Activity;
import android.content.Context;
import android.content.Intent;

// [...]

String pkg = getApplicationContext().getPackageName();
String name = "com.governikus.ausweisapp2.START_SERVICE";
Intent serviceIntent = new Intent(name);
serviceIntent.setPackage(pkg);
bindService(serviceIntent, mConnection, Context.BIND_AUTO_CREATE);
```

See also:

https://developer.android.com/guide/components/bound-services.html

https://developer.android.com/reference/android/app/Activity.html

Initializing the AIDL connection

Once the Android service of the AusweisApp SDK is successfully started and bound to by the client, the Android system calls the onServiceConnected method of the ServiceConnection created and supplied above. This method receives an instance of the IBinder Android service interface.

The IBinder is then used by the client application to initialize the auto generated AIDL stub in order to use the AIDL IPC mechanism. The used stub is supposed to be auto generated by the Android SDK if you have properly configured your build environment.

The stub initialization returns an instance of **IAusweisApp2Sdk** which is used to interact with the SDK. The example below stores this instance in the member variable mSdk.

```
import android.content.ComponentName;
import android.content.ServiceConnection;
```

(continues on next page)

(continued from previous page)

```
import android.os.IBinder;

import com.governikus.ausweisapp2.IAusweisApp2Sdk;

// [...]

IAusweisApp2Sdk mSdk;

ServiceConnection mConnection = new ServiceConnection(){
  @Override
  public void onServiceConnected(ComponentName className, IBinder service)
  {
      try {
          mSdk = IAusweisApp2Sdk.Stub.asInterface(service);
      } catch (ClassCastException|RemoteException e) {
          // ...
      }
  }

  @Override
  public void onServiceDisconnected(ComponentName className)
  {
      mSdk = null;
  }
}
```

See also:

Import the AIDL files (page 6)

Create session to AusweisApp

Once your client is bound to the AusweisApp SDK service and you have initialized the AIDL IPC mechanism, you are ready to use the actual SDK API.

Since the Android system does not allow to limit the number of clients which can connect to a service, the SDK API uses custom **sessions** to manage the connected clients. There is a maximum of one established session at a time.

In order to open a session with the SDK you need to pass an instance of **IAusweisApp2SdkCallback** to the **connectSdk** function of your previously acquired instance of **IAusweisApp2Sdk**. If your callback is accepted, the function returns true. Otherwise there is a problem with your supplied callback. Sessions will be disconnected once the IBinder instance of the connected client is invalidated, another communication error occurs or another Client connects. Please see *Disconnect from SDK* (page 11) for instructions to gracefully disconnect from the SDK.

As mentioned above: If there already is a connected client and a second client attempts to connect, the first client is disconnected and the second client is granted exclusive access to the SDK. The first client is informed via its callback by **sdkDisconnected**. The second client is presented a fresh environment and it has no access to any data of the first client.

If you have successfully established a session, the **sessionIdGenerated** function of your callback is invoked. With this invocation you receive two arguments. **pIsSecureSessionId** is true if the SDK was able to gather enough entropy in order to generate a secure random session ID. If it is false, there is no session ID passed. There is nothing you can do about such an error. It results from a problem with the random number generator, which in turn is very likely the result of an ongoing local attack. The device should be considered manipulated and the user should be informed.

On success **pSessionId** holds the actual session ID generated by the SDK. This ID is used to identify your session and you need to pass it to all future SDK function invocations of this session.

The listing below shows an example for an instantiation of IAusweisApp2SdkCallback and establishing a session.

```
import com.governikus.ausweisapp2.IAusweisApp2Sdk;
import com.governikus.ausweisapp2.IAusweisApp2SdkCallback;

// [...]

LocalCallback mCallback = new LocalCallback();
class LocalCallback extends IAusweisApp2SdkCallback.Stub
{
  public String mSessionID = null;

  @Override
  public void sessionIdGenerated(
    String pSessionId, boolean pIsSecureSessionId) throws RemoteException
  {
      mSessionID = pSessionId;
  }

  @Override
  public void receive(String pJson) throws RemoteException
  {
      // handle message from SDK
  }
}

// [...]

try
{
  if (!mSdk.connectSdk(mCallback))
  {
      // already connected? Handle error...
  }
}
catch (RemoteException e)
{
    // handle exception
}
```

See also:

Initializing the AIDL connection (page 8) *Disconnect from SDK* (page 11)

Send command

In order to send a JSON command to the AusweisApp SDK, you need to invoke the **send** function of your instance of **IAusweisApp2Sdk**. For this command to be processed by the SDK you need to supply the session ID which you have previously received. The listing below shows an example.

```
String cmd = "{\"cmd\": \"GET_INFO\"}";
try
{
  if (!mSdk.send(mCallback.mSessionID, cmd))
  {
      // disconnected? Handle error...
  }
}
catch (RemoteException e)
{
    // handle exception
}
```

Receive message

Messages from the AusweisApp SDK are passed to you via the same instance of **IAusweisApp2SdkCallback** in which you have received the session ID. The **receive** method is called each time the SDK sends a message.

See also:

Create session to AusweisApp (page 9)

Disconnect from SDK

In order to disconnect from the AusweisApp SDK you need to invalidate your instance of **IBinder**. There are two possibilities to do this. The first one is to unbind from the SDK Android service to undo your binding, like shown in the code listing below. The second one is to return false in the **pingBinder** function of your IBinder instance.

```
unbindService(mConnection);
```

See also:

Binding to the service (page 7)

https://developer.android.com/reference/android/os/IBinder.html

Passing NFC tags to the SDK

NFC tags can only be detected by applications which have a foreground window on the Android platform. A common workaround for this problem is to equip background services with a transparent window which is shown to dispatch NFC tags.

However, if there are multiple applications installed, which are capable of dispatching NFC tags, the Android system will display an **App Chooser** for each discovered tag enabling the user to select the appropriate application to handle the NFC tag. To have such a chooser display the name and image of the client application instead of the SDK, the client application is required to dispatch discovered NFC tags and to pass them to the SDK.

Furthermore, this interface design enables the client application to do **foreground dispatching** of NFC tags. If the active application registers itself for foreground dispatching, it receives discovered NFC tags directly without Android displaying an App Chooser.

Intent-Filter in AndroidManifest.xml

In order to be informed about attached NFC tags by Android, the client application is required to register an intent filter. The appropriate filter is shown in the listing below.

```
<intent-filter>
  <action android:name="android.nfc.action.TECH_DISCOVERED" />
</intent-filter>
<meta-data android:name="android.nfc.action.TECH_DISCOVERED" android:resource=
↪"@xml/nfc_tech_filter" />
```

See also:

https://developer.android.com/guide/components/intents-filters.html

NFC Technology Filter

Since there are many different kinds of NFC tags, Android requires the application to register a technology filter for the kind of tags the application wants to receive. The proper filter for the German eID card is shown in the listing below.

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
  <tech-list>
    <tech>android.nfc.tech.IsoDep</tech>
  </tech-list>
</resources>
```

Implementation

As it is common on the Android platform, information is sent to applications encapsulated in instances of the **Intent** class. In order to process newly discovered NFC tags, Intents which are given to the application need to be checked for the parcelable NFC extra as shown in the code listing below. Subsequently the client is required to send them to the AusweisApp SDK by calling the **updateNfcTag** method of the previously acquired **IAusweisApp2Sdk** instance. The listing below shows an example for the described process.

```
import android.content.Intent;
import android.nfc.NfcAdapter;
import android.nfc.Tag;

import com.governikus.ausweisapp2.IAusweisApp2Sdk;
import com.governikus.ausweisapp2.IAusweisApp2SdkCallback;

// [...]

void handleIntent(Intent intent)
{
  final Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
  if (tag != null)
  {
    try {
      mSdk.updateNfcTag(mCallback.mSessionID, tag);
    } catch (RemoteException e) {
      // ...
    }
  }
}
```

Dispatching NFC tags in foreground

As already mentioned under *Passing NFC tags to the SDK* (page 12), an App Chooser is displayed for discovered NFC tags by Android if multiple applications which are able to dispatch NFC tags are installed. An application can suppress this App Chooser if it registers itself for **foreground dispatching** at runtime. This way NFC tags are handled directly by the application without a chooser being displayed. Subsequently the client is required to send them to the AusweisApp SDK by calling the **updateNfcTag** method of the previously acquired **IAusweisApp2Sdk** instance. The required steps to handle NFC tags directly are shown in the code listing below by way of example.

```
import android.app.Activity;
import android.nfc.NfcAdapter;
import android.nfc.tech.IsoDep;
import java.util.Arrays;

import com.governikus.ausweisapp2.IAusweisApp2Sdk;

class ForegroundDispatcher
{
```

(continues on next page)

(continued from previous page)

```
  private final Activity mActivity;
  private final NfcAdapter mAdapter;
  private final int mFlags;
  private final NfcAdapter.ReaderCallback mReaderCallback;

  ForegroundDispatcher(Activity pActivity, final IAusweisApp2Sdk pSdk, final␣
↪String pSdkSessionID)
  {
    mActivity = pActivity;
    mAdapter = NfcAdapter.getDefaultAdapter(mActivity);
    mFlags = NfcAdapter.FLAG_READER_NFC_A | NfcAdapter.FLAG_READER_NFC_B |␣
↪NfcAdapter.FLAG_READER_SKIP_NDEF_CHECK;
    mReaderCallback = new NfcAdapter.ReaderCallback()
    {
      public void onTagDiscovered(Tag pTag)
      {
        if (Arrays.asList(pTag.getTechList()).contains(IsoDep.class.
↪getName()))
        {
          pSdk.updateNfcTag(pSdkSessionID, pTag);
        }
      }
    };
  }

  void enable()
  {
    if (mAdapter != null)
    {
      mAdapter.enableReaderMode(mActivity, mReaderCallback, mFlags, null);
    }
  }

  void disable()
  {
    if (mAdapter != null)
    {
      mAdapter.disableReaderMode(mActivity);
    }
  }
}

// [...]

ForegroundDispatcher mDispatcher = new ForegroundDispatcher(this);
```

The example implementation from above needs to be invoked when the application is brought to foreground and when it looses focus. An example for appropriate places are the **onResume** and the **onPause** methods of Activities as shown in the code listing below.

```
@Override
public void onResume()
{
  super.onResume();
  mDispatcher.enable();
}

@Override
public void onPause()
{
  super.onPause();
  mDispatcher.disable();
}
```

See also:

https://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle

4 iOS

This chapter deals with the iOS specific properties of the AusweisApp SDK. The AusweisApp core is encapsulated into an **XCFramework** which needs to be linked into your application.

Subsequent sections deal with the SDK interface itself and explain which steps are necessary in order to communicate with the AusweisApp SDK.

See also:

For iOS there is also the AusweisApp SDK Wrapper[4] which is a software library that offers a high-level interface to the AusweisApp SDK.

Important: Apple released the necessary NFC API with iOS 13.0! Be aware that it is not possible to support older versions.

4.1 Use XCFramework

The interface `AusweisApp2.h` of the SDK for iOS is provided as **C-Header** that you need to import/include into your application. It grants access to start and shutdown a separate background thread with the integrated AusweisApp core.

After you established a connection to the AusweisApp SDK your application can send *Commands* (page 22) and receive *Messages* (page 31).

[4] https://www.ausweisapp.bund.de/sdkwrapper/

Clone

The XCFramework is provided in our github repository. It is recommended to use at least Xcode 12 and the dependency handling of at least Swift 5.3 (Swift Package Manager).

https://github.com/Governikus/AusweisApp2-SDK-iOS

You can clone and add that repository with Xcode and import it into your project under the following menu.

File → Swift Packages → Add Package Dependency

Import

After you added the repository to your Xcode project you can import the module via `import AusweisApp2` in Swift classes or `@import AusweisApp2;` in Objective-C classes and call the functions of the `AusweisApp2.h` header.

```
typedef void (* AusweisApp2Callback)(const char* pMsg);
bool ausweisapp2_init(AusweisApp2Callback pCallback, const char* pCmdline);
void ausweisapp2_shutdown(void);
bool ausweisapp2_is_running(void);
void ausweisapp2_send(const char* pCmd);
```

Changed in version 1.24.0: Added optional parameter `pCmdline` to function `ausweisapp2_init`.

First, you need to define a callback function that will be called by the AusweisApp to request or provide additional information. If your application initializes the SDK you must pass that callback to `ausweisapp2_init`. That function will return `false` if the callback is `NULL` or the SDK is already running. The Parameter `pCmdline` is optional and can be `NULL`. This allows your application to provide additional commandline arguments like `--no-loghandler`.

After you called that function the AusweisApp SDK will start up. If the initialization is finished the SDK calls your callback function once with `NULL` as parameter to indicate that it is ready to accept *Commands* (page 22). Do not call `ausweisapp2_send` until your callback received that message, otherwise that command will be ignored.

Once the SDK is ready to go you can send *Commands* (page 22) by `ausweisapp2_send`. Your callback will receive the *Messages* (page 31).

If you call `ausweisapp2_shutdown` the AusweisApp SDK will be terminated. This function joins the thread of the AusweisApp and blocks until the AusweisApp is finished. You should not call this function in your callback as it is called by the AusweisApp thread. In that case `ausweisapp2_shutdown` cannot be a blocking call to avoid a deadlock. If you call this function while a workflow is running the workflow will be canceled automatically before the shutdown.

Important: Your callback will be called by the separate AusweisApp thread. Do **not** make long running or blocking calls! It is recommended to use an async dispatcher.

Also, you should not call `ausweisapp2_send` or `ausweisapp2_shutdown` within your callback function.

4.2 Info.plist

You need to enable the card identifier in your applications `Info.plist` like this, otherwise iOS will not recognize any identity cards. Also, it is necessary to provide a message why your application needs access to the NFC hardware.

```
<key>com.apple.developer.nfc.readersession.iso7816.select-identifiers</key>
<array>
  <string>E80704007F00070302</string>
</array>

<key>NFCReaderUsageDescription</key>
<string>AusweisApp needs NFC to access the ID card.</string>
```

See also:

- https://developer.apple.com/documentation/bundleresources/information_property_list/select-identifiers
- https://developer.apple.com/documentation/bundleresources/information_property_list/nfcreaderusagedescription

4.3 Entitlements

Your application needs to provide an entitlement file to request the format of reader sessions.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/
↪DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>com.apple.developer.nfc.readersession.formats</key>
    <array>
      <string>TAG</string>
    </array>
  </dict>
</plist>
```

See also:

https://developer.apple.com/documentation/bundleresources/entitlements/com_apple_developer_nfc_readersession_formats

4.4 Logging

The AusweisApp uses default logging of iOS and has its own log file. It is **recommended** to collect that log file if an error occurs in your application to receive better support.

The log file is in your application path:

```
NSTemporaryDirectory() + /AusweisApp.XXXXXX.log
```

The *XXXXXX* characters will be replaced by an automatically generated portion of the filename to avoid conflicts with previous instances.

A new log file will be created for each new instance of the AusweisApp and will be deleted after a correct shutdown. In case of old or multiple log files, it is highly probable that the previous instance crashed.

The AusweisApp deletes any log files that are older than 14 days.

5 Desktop

This chapter deals with the desktop specific properties of the AusweisApp SDK. The AusweisApp core is reachable over a **WebSocket** which is running by default since AusweisApp 1.16.0. Subsequent sections deal with the SDK interface itself and explain which steps are necessary in order to communicate with the AusweisApp SDK.

5.1 WebSocket

The AusweisApp uses the same default port as defined in TR-03124-1. Your application can connect to `ws://localhost:24727/eID-Kernel` to establish a bidirectional connection.

You can check the version of AusweisApp by the `Server` header of the HTTP response or by an additional query to get the *Status* (page 20).

If the WebSocket handshake was successful your application can send *Commands* (page 22) and receive *Messages* (page 31). The AusweisApp will send an HTTP error 503 “Service Unavailable” if the WebSocket is disabled.

See also:

https://tools.ietf.org/html/rfc6455

User installed

Your application can connect to a user installed AusweisApp. If the user already has an active workflow your request will be refused by an HTTP error `409 Conflict`. Also it is not possible to connect multiple times to the WebSocket as only one connection is allowed and will be refused by an HTTP error `429 Too Many Requests`. Once an application is connected to the WebSocket the graphical user interface of the AusweisApp will be blocked and shows a hint that another application uses the AusweisApp.

Important: Please provide a `User-Agent` in your HTTP upgrade request! The AusweisApp will show the content to the user as a hint which application uses the AusweisApp.

Integrated

You can start an already installed AusweisApp for your application. If your application spawns a separate process you should provide the commandline parameter `--port 0` to avoid conflicts with a user started AusweisApp and other processes that uses a specified port.

The AusweisApp will create a text file in the system temporary directory to provide the selected port. The port filename contains the PID of the running process to allow multiple instances at the same time.

Example: **$TMPDIR/AusweisApp.12345.port**

Your application can avoid the graphical interface of AusweisApp by providing the commandline parameter `--ui websocket`.

Important: If your application changes the used port the "smartphone as card reader" is not possible.

5.2 Automatic

New in version 1.24.0: Mode "automatic" added.

New in version 1.26.3: Variable "AUSWEISAPP2_AUTOMATIC_DEVELOPERMODE" added.

You can enable an automatic authentication with the automatic plugin by providing the commandline parameter `--ui automatic`.

It uses the *Simulator* (page 62) by default with default *Filesystem* (page 62) if there is no available card. Otherwise it uses the already inserted card. If you use an inserted card (e.g. via PCSC) you can provide additional environment variables for PIN, CAN and PUK on start up.

- `AUSWEISAPP2_AUTOMATIC_PIN`
- `AUSWEISAPP2_AUTOMATIC_CAN`
- `AUSWEISAPP2_AUTOMATIC_PUK`

The default value for the PIN is **123456**. If a value is not defined or the card refuses a PIN, CAN or PUK the AusweisApp will cancel the whole workflow. Also the workflow will be canceled if the card reader is not a basic reader as it is not possible to automatically enter the values.

The **developerMode** (like in *RUN_AUTH* (page 23)) can be enabled with the environment variable `AUSWEISAPP2_AUTOMATIC_DEVELOPERMODE`. This will be evaluated if the automatic plugin takes control over the workflow.

Note: It is possible to pass multiple plugins to the AusweisApp, e.g.: `--ui websocket --ui automatic`.

See also:

The *Container* (page 20) SDK is designed for scripted and automatic workflows and enables the automatic mode by default.

5.3 Status

TR-03124-1 defined a query for status information. This is useful to fetch current version of installed AusweisApp to check if the version supports the WebSocket-API.

You can get this by a HTTP GET query to `http://localhost:24727/eID-Client?Status`. If you prefer the JSON syntax you can add it to the parameter `?Status=json` to get the following information.

```
{
  "Name": "AusweisApp2",
  "Implementation-Title": "AusweisApp2",
  "Implementation-Vendor": "Governikus GmbH & Co. KG",
  "Implementation-Version": "2.0.0",
  "Specification-Title": "TR-03124-1",
  "Specification-Vendor": "Federal Office for Information Security",
  "Specification-Version": "1.4"
}
```

See also:

The AusweisApp SDK provides a *GET_INFO* (page 22) command and an *INFO* (page 40) message to fetch the same information to check the compatibility of used AusweisApp.

5.4 Reader

The AusweisApp SDK uses PC/SC and paired Smartphones as card reader. If the user wants to use the "smartphone as card reader" feature it is necessary to pair the devices by the graphical interface of AusweisApp. The AusweisApp SDK provides no API to pair those devices.

6 Container

This chapter deals with the container specific properties of the AusweisApp SDK. The AusweisApp core is reachable over a *WebSocket* (page 18) like the *Desktop* (page 18) variant and also enables the *Automatic* (page 19) mode by default. Subsequent sections deal with the container itself and explain which steps are necessary in order to communicate with the AusweisApp SDK in the container.

Note: The container SDK does not provide any graphical user interface.

6.1 Start

In order to start up the container SDK you can use the usual docker commands. It is recommended to use `--rm` as it not necessary to save the configuration. If you need to restore the configuration you can add a VOLUME[5] for `/home/ausweisapp/.config`.

```
docker run --rm -p 127.0.0.1:24727:24727 governikus/ausweisapp2
```

[5] https://docs.docker.com/storage/volumes/

Proxy

If you need a proxy in your network you can pass the necessary environment variables by `docker run -e https_proxy=IP:PORT -e http_proxy=IP:PORT`.

6.2 Behaviour

The AusweisApp uses the *Automatic* (page 19) mode with the default *Simulator* (page 62) if there is no connected *WebSocket* (page 18) client. That mode will be disabled if a client is connected and enabled again after disconnection. If the client disconnects during an active workflow the workflow will be canceled.

Note: The usual eID activation via localhost is still possible with both options.

---

7 Commands

Your application (client) can send some commands (**cmd**) to control the AusweisApp. The AusweisApp (server) will send some proper *Messages* (page 31) during the whole workflow or as an answer to your command.

7.1 GET_INFO

Returns information about the current installation of AusweisApp.

The AusweisApp will send an *INFO* (page 40) message as an answer.

```
{"cmd": "GET_INFO"}
```

7.2 GET_STATUS

Returns information about the current workflow and state of AusweisApp.

The AusweisApp will send a *STATUS* (page 44) message as an answer.

New in version 1.24.0: Support of GET_STATUS command in *API_LEVEL* (page 33) **2**.

```
{"cmd": "GET_STATUS"}
```

7.3 GET_API_LEVEL

Returns information about the available and current API level.

The AusweisApp will send an *API_LEVEL* (page 33) message as an answer.

```
{"cmd": "GET_API_LEVEL"}
```

7.4 SET_API_LEVEL

Set supported API level of your application.

If you initially develop your application against the AusweisApp SDK you should check with *GET_API_LEVEL* (page 22) the highest supported level and set this value with this command if you connect to the SDK. This will set the SDK to act with the defined level even if a newer level is available.

The AusweisApp will send an *API_LEVEL* (page 33) message as an answer.

- **level**: Supported API level of your app.

```
{
  "cmd": "SET_API_LEVEL",
  "level": 1
}
```

7.5 GET_READER

Returns information about the requested reader.

If you explicitly want to ask for information of a known reader name you can request it with this command.

The AusweisApp will send a *READER* (page 42) message as an answer.

- **name**: Name of the reader.

```
{
  "cmd": "GET_READER",
  "name": "NAME OF THE READER"
}
```

7.6 GET_READER_LIST

Returns information about all connected readers.

If you explicitly want to ask for information of all connected readers you can request it with this command.

The AusweisApp will send a *READER_LIST* (page 43) message as an answer.

```
{"cmd": "GET_READER_LIST"}
```

7.7 RUN_AUTH

Starts an authentication.

The AusweisApp will send an *AUTH* (page 34) message when the authentication is started.

The system dialog on iOS can be customized by **messages**. This dialog won't be stopped by default after an *ENTER_PIN* (page 38), *ENTER_CAN* (page 37) and *ENTER_PUK* (page 39). Command *INTERRUPT* (page 28) allows to stop the dialog manually, if needed.

Changed in version 1.24.0: Parameter **handleInterrupt** removed with *API_LEVEL* (page 33) v2 and defaults to `false`.

New in version 1.24.0: Parameter **status** added.

New in version 1.22.3: Parameter **developerMode** added.

New in version 1.22.1: Parameter **handleInterrupt** and **messages** added.

Deprecated since version 1.22.1: Parameter **handleInterrupt** has been removed with *API_LEVEL* (page 33) v2 and defaults to `false`.

- **tcTokenURL**: URL to the TcToken. This is equal to the desktop style activation URL. *(http://127.0.0.1:24727/eID-Client?tcTokenURL=)*
- **developerMode**: True to enable "Developer Mode" for test cards and disable some security checks according to BSI TR-03124-1, otherwise false. (optional, default: false)
- **handleInterrupt**: True to automatically handle system dialog on iOS to enter a password, otherwise false. *API_LEVEL* (page 33) v1 only. (optional, default: false)

- **status**: True to enable automatic *STATUS* (page 44) messages, otherwise false. *API_LEVEL* (page 33) v2 only. (optional, default: true)
- **messages**: Messages for the system dialog on iOS. (optional, default: empty)
  - **sessionStarted**: Shown if scanning is started.
  - **sessionFailed**: Shown if communication was stopped with an error.
  - **sessionSucceeded**: Shown if communication was stopped successfully.
  - **sessionInProgress**: Shown if communication is in progress. This message will be appended with current percentage level.

```
{
  "cmd": "RUN_AUTH",
  "tcTokenURL": "https://test.governikus-eid.de/AusweisAuskunft/
↪WebServiceRequesterServlet",
  "developerMode": false,
  "handleInterrupt": false,
  "status": true,
  "messages":
      {
       "sessionStarted": "Please place your ID card on the top of the device
↪'s back side.",
       "sessionFailed": "Scanning process failed.",
       "sessionSucceeded": "Scanning process has been finished successfully.",
       "sessionInProgress": "Scanning process is in progress."
      }
}
```

Note: This command is allowed only if the AusweisApp has no running authentication or other workflow. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.8 RUN_CHANGE_PIN

Starts a change PIN workflow.

The AusweisApp will send a *CHANGE_PIN* (page 36) message when the workflow is started.

The system dialog on iOS can be customized by **messages**. This dialog won't be stopped by default after an *ENTER_PIN* (page 38), *ENTER_CAN* (page 37), *ENTER_NEW_PIN* (page 39) and *ENTER_PUK* (page 39). Command *INTERRUPT* (page 28) allows to stop the dialog manually, if needed.

Changed in version 1.24.0: Parameter **handleInterrupt** removed with *API_LEVEL* (page 33) v2 and defaults to `false`.

New in version 1.24.0: Parameter **status** added.

New in version 1.22.1: Parameter **handleInterrupt** and **messages** added.

Deprecated since version 1.22.1: Parameter **handleInterrupt** has been removed with *API_LEVEL* (page 33) v2 and defaults to `false`.

New in version 1.22.0: Support of RUN_CHANGE_PIN command.

- **handleInterrupt**: True to automatically handle system dialog on iOS, otherwise false. *API_LEVEL* (page 33) v1 only. (optional, default: false)
- **status**: True to enable automatic *STATUS* (page 44) messages, otherwise false. *API_LEVEL* (page 33) v2 only. (optional, default: true)
- **messages**: Messages for the system dialog on iOS. (optional, default: empty)
  - **sessionStarted**: Shown if scanning is started.
  - **sessionFailed**: Shown if communication was stopped with an error.
  - **sessionSucceeded**: Shown if communication was stopped successfully.
  - **sessionInProgress**: Shown if communication is in progress. This message will be appended with current percentage level.

```
{
  "cmd": "RUN_CHANGE_PIN",
  "handleInterrupt": false,
  "status": true,
  "messages":
      {
       "sessionStarted": "Please place your ID card on the top of the device
↪'s back side.",
       "sessionFailed": "Scanning process failed.",
       "sessionSucceeded": "Scanning process has been finished successfully.",
       "sessionInProgress": "Scanning process is in progress."
      }
}
```

Note: This command is allowed only if the AusweisApp has no running authentication or other workflow. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.9 GET_ACCESS_RIGHTS

Returns information about the requested access rights.

The AusweisApp will send an *ACCESS_RIGHTS* (page 31) message as an answer.

```
{"cmd": "GET_ACCESS_RIGHTS"}
```

Note: This command is allowed only if the AusweisApp sends an initial *ACCESS_RIGHTS* (page 31) message. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.10 SET_ACCESS_RIGHTS

Set effective access rights.

By default the **effective** access rights are **optional** + **required**. If you want to enable or disable some **optional** access rights you can send this command to modify the **effective** access rights.

The AusweisApp will send an *ACCESS_RIGHTS* (page 31) message as an answer.

- **chat**: List of enabled **optional** access rights. If you send an empty **[]** all **optional** access rights are disabled.

```
{
  "cmd": "SET_ACCESS_RIGHTS",
  "chat": []
}
```

```
{
  "cmd": "SET_ACCESS_RIGHTS",
  "chat": ["FamilyName"]
}
```

Note: This command is allowed only if the AusweisApp sends an initial *ACCESS_RIGHTS* (page 31) message. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

See also:

List of possible access rights are listed in *ACCESS_RIGHTS* (page 31).

7.11 SET_CARD

Insert “virtual” card.

Since *API_LEVEL* (page 33) **2** it is possible to provide a “virtual” card. The information whether this is possible will be indicated in a *READER* (page 42) message.

New in version 1.24.0: This command was introduced in *API_LEVEL* (page 33) **2**.

New in version 1.26.4: Parameter **keys** added.

- **name**: Name of the *READER* (page 42).
- **simulator**: Specific data for *Simulator* (page 62). (optional)
  - **files**: Content of card *Filesystem* (page 62).
  - **keys**: Keys of card *Filesystem* (page 62).

```
{
  "cmd": "SET_CARD",
  "name": "reader name",
  "simulator":
  {
     "files": [],
```

(continues on next page)

(continued from previous page)

```
    "keys": []
  }
}
```

Note: This command is allowed only if the AusweisApp sends an initial *INSERT_CARD* (page 41) message. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.12 CONTINUE

Continues the workflow after a *PAUSE* (page 45) was sent.

The AusweisApp will send a *PAUSE* (page 45) message with an appropriate *cause* (page 45) for the waiting condition. After the issue was fixed you have to send CONTINUE to go on with the workflow.

New in version 2.2.0: The command *CONTINUE* (page 27) was introduced in *API_LEVEL* (page 33) **3**.

```
{
  "cmd": "CONTINUE"
}
```

Note: This command is allowed only if the AusweisApp sent a *PAUSE* (page 45). Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.13 GET_CERTIFICATE

Returns the certificate of current authentication.

The AusweisApp will send a *CERTIFICATE* (page 35) message as an answer.

```
{"cmd": "GET_CERTIFICATE"}
```

Note: This command is allowed only if the AusweisApp sends an initial *ACCESS_RIGHTS* (page 31) message. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.14 CANCEL

Cancel the whole workflow.

If your application sends this command the AusweisApp will cancel the workflow. You can send this command in any state of a running workflow to abort it.

```
{"cmd": "CANCEL"}
```

Note: This command is allowed only if the AusweisApp started an authentication or the *CHANGE_PIN* (page 36) workflow. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.15 ACCEPT

Accept the current state.

If the AusweisApp will send the message *ACCESS_RIGHTS* (page 31) the user needs to **accept** or **cancel**. So the workflow is paused until your application sends this command to accept the requested information.

If the user does not accept the requested information your application needs to send the command *CANCEL* (page 27) to abort the whole workflow.

This command will be used later for additional requested information if the AusweisApp needs to pause the workflow. In *API_LEVEL* (page 33) v1 only *ACCESS_RIGHTS* (page 31) needs to be accepted.

```
{"cmd": "ACCEPT"}
```

Note: This command is allowed only if the AusweisApp sends an initial *ACCESS_RIGHTS* (page 31) message. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.16 INTERRUPT

Interrupts current system dialog on iOS.

If your application provides **false** to parameter **handleInterrupt** in *RUN_AUTH* (page 23) or *RUN_CHANGE_PIN* (page 24) and you need to request more information from the user, you need to interrupt the system dialog manually.

This command will be used later for additional information if your application needs to interrupt the workflow. Currently only the iOS system dialog can be interrupted.

New in version 1.22.1: Support of INTERRUPT command.

```
{"cmd": "INTERRUPT"}
```

Note: This command is allowed only if the AusweisApp sends a *ENTER_PIN* (page 38), *ENTER_CAN* (page 37), *ENTER_NEW_PIN* (page 39) or *ENTER_PUK* (page 39) message. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.17 SET_PIN

Set PIN of inserted card.

If the AusweisApp sends message *ENTER_PIN* (page 38) you need to send this command to unblock the card with the PIN.

The AusweisApp will send an *ENTER_PIN* (page 38) message on error or message *ENTER_CAN* (page 37) if the retryCounter of the card is decreased to **1**. For detailed information see message *ENTER_PIN* (page 38).

If the PIN was correct, the workflow will continue.

If the last attempt to enter the PIN failed, AusweisApp will send the message *ENTER_PUK* (page 39) as the retryCounter is decreased to **0**.

Changed in version 1.16.0: The parameter "value" must be omitted if the used *READER* (page 42) has a keypad.

- **value**: The Personal Identification Number (PIN) of the card. This must be 6 digits in an *AUTH* (page 34) workflow. If a *CHANGE_PIN* (page 36) workflow is in progress the value must be 5 or 6 digits because of a possible transport PIN. If the *READER* (page 42) has a keypad this parameter must be omitted.

```
{
  "cmd": "SET_PIN",
  "value": "123456"
}
```

Note: This command is allowed only if the AusweisApp sends an initial *ENTER_PIN* (page 38) message. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.18 SET_NEW_PIN

Set new PIN of inserted card.

If the AusweisApp sends message *ENTER_NEW_PIN* (page 39) you need to send this command to set the new PIN of the card.

New in version 1.22.0: Support of SET_NEW_PIN command.

- **value**: The new personal identification number (PIN) of the card. This must be 6 digits if the *READER* (page 42) has no keypad, otherwise this parameter must be omitted.

```
{
  "cmd": "SET_NEW_PIN",
  "value": "123456"
}
```

Note: This command is allowed only if the AusweisApp sends an initial *ENTER_NEW_PIN* (page 39) message. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.19 SET_CAN

Set CAN of inserted card.

If the AusweisApp sends message *ENTER_CAN* (page 37) you need to send this command to unblock the last retry of *SET_PIN* (page 29).

The AusweisApp will send an *ENTER_CAN* (page 37) message on error. Otherwise the workflow will continue with *ENTER_PIN* (page 38).

Changed in version 1.16.0: The parameter "value" must be omitted if the used *READER* (page 42) has a keypad.

- **value**: The Card Access Number (CAN) of the card. This must be 6 digits if the *READER* (page 42) has no keypad, otherwise this parameter must be omitted.

```
{
  "cmd": "SET_CAN",
  "value": "123456"
}
```

Note: This command is allowed only if the AusweisApp sends an initial *ENTER_CAN* (page 37) message. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

7.20 SET_PUK

Set PUK of inserted card.

If the AusweisApp sends message *ENTER_PUK* (page 39) you need to send this command to unblock *SET_PIN* (page 29).

The AusweisApp will send an *ENTER_PUK* (page 39) message on error or if the PUK is operative. Otherwise the workflow will continue with *ENTER_PIN* (page 38). For detailed information see message *ENTER_PUK* (page 39).

Changed in version 1.16.0: The parameter "value" must be omitted if the used *READER* (page 42) has a keypad.

- **value**: The Personal Unblocking Key (PUK) of the card. This must be 10 digits if the *READER* (page 42) has no keypad, otherwise this parameter must be omitted.

```
{
  "cmd": "SET_PUK",
  "value": "1234567890"
}
```

Note: This command is allowed only if the AusweisApp sends an initial *ENTER_PUK* (page 39) message. Otherwise you will get a *BAD_STATE* (page 35) message as an answer.

---

8 Messages

The AusweisApp (server) will send some proper messages (**msg**) to your application (client) during the whole workflow or as an answer to your *Commands* (page 22).

8.1 ACCESS_RIGHTS

This message will be sent by AusweisApp once the authentication is started by *RUN_AUTH* (page 23) and the AusweisApp got the certificate from the service.

If your application receives this message you can call *SET_ACCESS_RIGHTS* (page 26) to change some optional access rights or you can call *GET_ACCESS_RIGHTS* (page 25) to get this message again.

Also you can call *GET_CERTIFICATE* (page 27) to get the certificate to show this to your user.

The workflow will continue if you call *ACCEPT* (page 28) to indicate that the user accepted the requested access rights or call *CANCEL* (page 27) to abort the whole workflow.

- **error**: This optional parameter indicates an error of a *SET_ACCESS_RIGHTS* (page 26) call if the command contained invalid data.
- **chat**: Access rights of the provider.
  - **effective**: Indicates the enabled access rights of **optional** and **required**.
  - **optional**: These rights are optional and can be enabled or disabled by *SET_ACCESS_RIGHTS* (page 26).
  - **required**: These rights are mandatory and cannot be disabled.
- **transactionInfo**: Optional transaction information.
- **aux**: Optional auxiliary data of the provider.
  - **ageVerificationDate**: Optional required date of birth for AgeVerification as ISO 8601.
  - **requiredAge**: Optional required age for AgeVerification. It is calculated by AusweisApp on the basis of ageVerificationDate and current date.
  - **validityDate**: Optional validity date as ISO 8601.
  - **communityId**: Optional id of community.

```
{
  "msg": "ACCESS_RIGHTS",
  "error": "some optional error message",
  "chat":
        {
         "effective": ["Address", "FamilyName", "GivenNames", "AgeVerification
↪", "CanAllowed"],
         "optional": ["GivenNames", "AgeVerification", "CanAllowed"],
         "required": ["Address", "FamilyName"]
        },
  "transactionInfo": "this is an example",
  "aux":
       {
        "ageVerificationDate": "1999-07-20",
```

(continues on next page)

(continued from previous page)

```
        "requiredAge": "18",
        "validityDate": "2017-07-20",
        "communityId": "02760400110000"
       }
}
```

```
{
  "msg": "ACCESS_RIGHTS",
  "chat":
        {
         "effective": ["Address", "FamilyName", "GivenNames", "AgeVerification
↪"],
         "optional": ["GivenNames", "AgeVerification"],
         "required": ["Address", "FamilyName"]
        }
}
```

Values

New in version 1.22.0: The following access rights are possible now:

- CanAllowed
- PinManagement

New in version 1.20.0: The following write access rights are possible now:

- WriteAddress
- WriteCommunityID
- WriteResidencePermitI
- WriteResidencePermitII

The following access rights are possible:

- Address
- BirthName
- FamilyName
- GivenNames
- PlaceOfBirth
- DateOfBirth
- DoctoralDegree
- ArtisticName
- Pseudonym
- ValidUntil
- Nationality

- IssuingCountry
- DocumentType
- ResidencePermitI
- ResidencePermitII
- CommunityID
- AddressVerification
- AgeVerification
- WriteAddress
- WriteCommunityID
- WriteResidencePermitI
- WriteResidencePermitII
- CanAllowed
- PinManagement

See also:

TR-03110[6], part 4, chapter 2.2.3

TR-03127[7], chapter 3.2.2

8.2 API_LEVEL

This message will be sent if *GET_API_LEVEL* (page 22) or *SET_API_LEVEL* (page 22) is called.

It lists all **available** API levels that can be used and set by *SET_API_LEVEL* (page 22). Also it indicates the **current** selected API level.

New in version 1.24.0: Level **2** added.

- **error**: Optional error message if *SET_API_LEVEL* (page 22) failed.
- **available**: List of supported API level by this version.
- **current**: Currently selected API level.

```
{
  "msg": "API_LEVEL",
  "error": "optional error message like an invalid level",
  "available": [1,2,3,4],
  "current": 4
}
```

Your application should always set the compatible API level. The AusweisApp will support multiple API levels to give you enough time to add support for the new API.

[6] https://www.bsi.bund.de/EN/Publications/TechnicalGuidelines/TR03110/BSITR03110.html
[7] https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr03127/tr-03127.html

Even if you added support for the new API, your application should still support the old API level in case the user updates your application but does not update the AusweisApp. Otherwise you need to show a message to the user that they need to update the AusweisApp.

The API level will be increased for **incompatible** changes only. If we can add additional commands, messages or information without breaking the previous API you can check the application version with *GET_INFO* (page 22) to know if the current version supports your requirements.

This documentation will mark every API change with a flag like the following:

- New in version 1.10.0.
- Changed in version 1.10.0.
- Deprecated since version 1.10.0.

8.3 AUTH

This message will be sent by AusweisApp if an authentication is initially started. The next message should be *ACCESS_RIGHTS* (page 31) or *AUTH* (page 34) again if the authentication immediately results in an error.

If you receive an *AUTH* (page 34) message with a parameter **error** your command *RUN_AUTH* (page 23) was invalid and the workflow was not started at all.

- **error**: Optional error message if *RUN_AUTH* (page 23) failed.

```
{
  "msg": "AUTH",
  "error": "error message if RUN_AUTH failed"
}
```

If the workflow is finished the AusweisApp will send a message with a result and an url parameter to indicate the end of an authentication.

New in version 1.26.3: Parameter **reason** added.

- **result**: The final result of authentication.
  - **major**: Major “error” code.
  - **minor**: Minor error code.
  - **language**: Language of description and message. Language “en” is supported only at the moment.
  - **description**: Description of the error message.
  - **message**: The error message.
  - **reason**: Unique *Failure Codes* (page 49).
- **url**: Refresh url or an optional communication error address.

```
{
  "msg": "AUTH",
  "result":
          {
```

(continues on next page)

(continued from previous page)

```
            "major": "http://www.bsi.bund.de/ecard/api/1.1/resultmajor#ok"
          },
  "url": "https://test.governikus-eid.de/gov_autent/async?refID=_123456789"
}
```

```
{
  "msg": "AUTH",
  "result":
          {
           "major": "http://www.bsi.bund.de/ecard/api/1.1/resultmajor#error",
           "minor": "http://www.bsi.bund.de/ecard/api/1.1/resultminor/al/
↪common#internalError",
           "language": "en",
           "description": "An internal error has occurred during processing.
↪",
           "message": "The connection to the ID card has been lost. The␣
↪process was aborted.",
           "reason": "Card_Removed"
          },
  "url": "https://test.governikus-eid.de/gov_autent/async?refID=_abcdefgh"
}
```

8.4 BAD_STATE

Indicates that your previous command was send in an invalid state.

Some commands can be send to the server only if certain "state" is reached in the workflow to obtain the corresponding result. Otherwise the command will fail with *BAD_STATE* (page 35).

For example, you cannot send *GET_CERTIFICATE* (page 27) if there is no authentication in progress.

- **error**: Name of the received command that is invalid in this state.

```
{
  "msg": "BAD_STATE",
  "error": "NAME_OF_YOUR_COMMAND"
}
```

8.5 CERTIFICATE

Provides information about the used certificate.

- **description**: Detailed description of the certificate.
  - **issuerName**: Name of the certificate issuer.
  - **issuerUrl**: URL of the certificate issuer.
  - **subjectName**: Name of the certificate subject.
  - **subjectUrl**: URL of the certificate subject.

  - **termsOfUsage**: Raw certificate information about the terms of usage.
  - **purpose**: Parsed purpose of the terms of usage.
- **validity**: Validity dates of the certificate in UTC.
  - **effectiveDate**: Certificate is valid since this date.
  - **expirationDate**: Certificate is invalid after this date.


```
{
  "msg": "CERTIFICATE",
  "description":
                {
                 "issuerName": "Governikus Test DVCA",
                 "issuerUrl": "http://www.governikus.de",
                 "subjectName": "Governikus GmbH & Co. KG",
                 "subjectUrl": "https://test.governikus-eid.de",
                 "termsOfUsage": "Anschrift:\t\r\nGovernikus GmbH & Co. KG\r\
↪nAm Fallturm 9\r\n28359 Bremen\t\r\n\r\nE-Mail-Adresse:\thb@bos-bremen.de\t\
↪r\n\r\nZweck des Auslesevorgangs:\tDemonstration des eID-Service\t\r\n\r\
↪nZuständige Datenschutzaufsicht:\t\r\nDie Landesbeauftragte für Datenschutz␣
↪und Informationsfreiheit der Freien Hansestadt Bremen\r\nArndtstraße 1\r\
↪n27570 Bremerhaven",
                 "purpose": "Demonstration des eID-Service"
                },
  "validity":
              {
               "effectiveDate": "2016-07-31",
               "expirationDate": "2016-08-30"
              }
}
```


8.6 CHANGE_PIN

This message will be sent by AusweisApp if a change PIN workflow is initially started.

If you receive a *CHANGE_PIN* (page 36) message with a parameter **success** the workflow is finished. This could happen after a *SET_PIN* (page 29) command if the connection to the card failed. Also the parameter **success** is false after a *CANCEL* (page 27) command.

New in version 1.22.0: Support of CHANGE_PIN message.

New in version 1.26.4: Parameter **reason** added.

- **success**: Indicates with true that the PIN was successfully changed, otherwise false.
- **reason**: Unique *Failure Codes* (page 49).


```
{
  "msg": "CHANGE_PIN",
  "success": true
}
```

```
{
  "msg": "CHANGE_PIN",
  "success": false,
  "reason": "Card_Removed"
}
```

8.7 ENTER_CAN

Indicates that a CAN is required to continue workflow.

If the AusweisApp sends this message, you will have to provide the CAN of the inserted card with *SET_CAN* (page 30).

The CAN is required to enable the last attempt of PIN input if the retryCounter is **1**. The workflow continues automatically with the correct CAN and the AusweisApp will send an *ENTER_PIN* (page 38) message. Despite the correct CAN being entered, the retryCounter remains at **1**.

The CAN is also required, if the authentication terminal has an approved “CAN allowed right”. This allows the workflow to continue without an additional PIN.

If your application provides an invalid *SET_CAN* (page 30) command the AusweisApp will send an *ENTER_CAN* (page 37) message with an error parameter.

If your application provides a valid *SET_CAN* (page 30) command and the CAN was incorrect the AusweisApp will send *ENTER_CAN* (page 37) again but without an error parameter.

New in version 1.14.2: Support of “CAN allowed right”.

- **error**: Optional error message if your command *SET_CAN* (page 30) was invalid.
- **reader**: Information about the used card and card reader. Please see message *READER* (page 42) for details.

```
{
  "msg": "ENTER_CAN",
  "error": "You must provide 6 digits",
  "reader":
           {
            "name": "NFC",
            "insertable": false,
            "attached": true,
            "keypad": false,
            "card":
                   {
                    "inoperative": false,
                    "deactivated": false,
                    "retryCounter": 1
                   }
           }
}
```

Note: There is no retry limit for an incorrect CAN.

8.8 ENTER_PIN

Indicates that a PIN is required to continue the workflow.

If the AusweisApp sends this message, you will have to provide the PIN of the inserted card with *SET_PIN* (page 29).

The workflow will automatically continue if the PIN was correct. Otherwise you will receive another message *ENTER_PIN* (page 38). If the correct PIN is entered the retryCounter will be set to **3**.

If your application provides an invalid *SET_PIN* (page 29) command the AusweisApp will send an *ENTER_PIN* (page 38) message with an error parameter and the retryCounter of the card is **not** decreased.

If your application provides a valid *SET_PIN* (page 29) command and the PIN was incorrect the AusweisApp will send *ENTER_PIN* (page 38) again with a decreased retryCounter but without an error parameter.

If the value of retryCounter is **1** the AusweisApp will initially send an *ENTER_CAN* (page 37) message. Once your application provides a correct CAN the AusweisApp will send an *ENTER_PIN* (page 38) again with a retryCounter of **1**.

If the value of retryCounter is **0** the AusweisApp will initially send an *ENTER_PUK* (page 39) message. Once your application provides a correct PUK the AusweisApp will send an *ENTER_PIN* (page 38) again with a retryCounter of **3**.

- **error**: Optional error message if your command *SET_PIN* (page 29) was invalid.
- **reader**: Information about the used card and card reader. Please see message *READER* (page 42) for details.


```
{
  "msg": "ENTER_PIN",
  "error": "You must provide 6 digits",
  "reader":
           {
            "name": "NFC",
            "insertable": false,
            "attached": true,
            "keypad": false,
            "card":
                   {
                    "inoperative": false,
                    "deactivated": false,
                    "retryCounter": 3
                   }
           }
}
```

8.9 ENTER_NEW_PIN

Indicates that a new PIN is required to continue the workflow.

If the AusweisApp sends this message, you will have to provide the new PIN of the inserted card with *SET_NEW_PIN* (page 29).

New in version 1.22.0: Support of ENTER_NEW_PIN message.

- **error**: Optional error message if your command *SET_NEW_PIN* (page 29) was invalid.
- **reader**: Information about the used card and card reader. Please see message *READER* (page 42) for details.


```
{
  "msg": "ENTER_NEW_PIN",
  "error": "You must provide 6 digits",
  "reader":
           {
            "name": "NFC",
            "insertable": false,
            "attached": true,
            "keypad": false,
            "card":
                   {
                    "inoperative": false,
                    "deactivated": false,
                    "retryCounter": 3
                   }
           }
}
```


8.10 ENTER_PUK

Indicates that a PUK is required to continue the workflow.

If the AusweisApp sends this message, you will have to provide the PUK of the inserted card with *SET_PUK* (page 30).

The workflow will automatically continue if the PUK was correct and the AusweisApp will send an *ENTER_PIN* (page 38) message. Otherwise you will receive another message *ENTER_PUK* (page 39). If the correct PUK is entered the retryCounter will be set to **3**.

If your application provides an invalid *SET_PUK* (page 30) command the AusweisApp will send an *ENTER_PUK* (page 39) message with an error parameter.

If your application provides a valid *SET_PUK* (page 30) command and the PUK was incorrect the AusweisApp will send *ENTER_PUK* (page 39) again but without an error parameter.

If AusweisApp sends *ENTER_PUK* (page 39) with field “inoperative” of embedded *READER* (page 42) message set true it is not possible to unblock the PIN. You will have to show a message to the user that the card is inoperative and the user should contact the authority responsible for issuing the identification card to unblock the PIN. You need to send a *CANCEL* (page 27) to abort the workflow if card is operative. Please see the note for more information.

- **error**: Optional error message if your command *SET_PUK* (page 30) was invalid.
- **reader**: Information about the used card and card reader. Please see message *READER* (page 42) for details.


```
{
  "msg": "ENTER_PUK",
  "error": "You must provide 10 digits",
  "reader":
           {
            "name": "NFC",
            "insertable": false,
            "attached": true,
            "keypad": false,
            "card":
                   {
                    "inoperative": false,
                    "deactivated": false,
                    "retryCounter": 0
                   }
           }
}
```

Note: There is no retry limit for an incorrect PUK. But be aware that the PUK can only be used 10 times to unblock the PIN. There is no readable counter for this. The AusweisApp is not able to provide any counter information of PUK usage. If the PUK is used 10 times it is not possible to unblock the PIN anymore and the card will remain in PUK state. Also it is not possible to indicate this state before the user enters the correct PUK once. This information will be provided as field "inoperative" of *READER* (page 42) message.

8.11 INFO

Provides information about the AusweisApp.

Especially if you want to get a specific **Implementation-Version** to check if the current installation supports some additional *Commands* (page 22) or *Messages* (page 31).

Also you should check the *API_LEVEL* (page 33) as it will be increased for **incompatible** changes.

- **VersionInfo**: Structure of version information.
  - **Name**: Application name.
  - **Implementation-Title**: Title of implementation.
  - **Implementation-Vendor**: Vendor of implementation.
  - **Implementation-Version**: Version of implementation.
  - **Specification-Title**: Title of specification.
  - **Specification-Vendor**: Vendor of specification.
  - **Specification-Version**: Version of specification.

```
{
  "msg": "INFO",
  "VersionInfo":
                {
                 "Name": "AusweisApp2",
                 "Implementation-Title": "AusweisApp2",
                 "Implementation-Vendor": "Governikus GmbH & Co. KG",
                 "Implementation-Version": "2.0.0",
                 "Specification-Title": "TR-03124-1",
                 "Specification-Vendor": "Federal Office for Information␣
↪Security",
                 "Specification-Version": "1.4"
                }
}
```

8.12 INSERT_CARD

Indicates that the AusweisApp requires a card to continue.

If the AusweisApp needs a card to continue the workflow this message will be sent as a notification. If your application receives this message it should show a hint to the user.

The user must provide a physical card or your application needs to to provide a "virtual" card by calling *SET_CARD* (page 26).

After the user or your application inserted a card, the workflow will continue automatically, unless both the eID function and CAN allowed mode are disabled. CAN allowed mode is enabled if the AusweisApp is used as SDK and the certificate contains the CAN allowed right. In this case, the workflow will be paused until another card is inserted. If the user already inserted a card this message will not be sent at all.

This message will also be sent if there is no connected card reader.

- **error**: Optional detailed error message.

```
{
  "msg": "INSERT_CARD",
  "error": "Name cannot be undefined"
}
```

8.13 INTERNAL_ERROR

Indicates an internal error.

If your application receives this message you found a bug. Please report this issue to our support!

- **error**: Optional detailed error message.

```
{
  "msg": "INTERNAL_ERROR",
  "error": "Unexpected condition"
}
```

8.14 INVALID

Indicates a broken JSON message.

If your application receives this message you passed a broken JSON structure to the AusweisApp.

Please fix your JSON document and send it again!

- **error**: Detailed error message.

```
{
  "msg": "INVALID",
  "error": "unterminated string (offset: 12)"
}
```

8.15 READER

Provides information about a connected or disconnected card reader.

This message will be sent by the AusweisApp if a card reader was added or removed to the operating system. Also if a card was inserted into a card reader or removed from a card reader.

Your application can explicitly check for card reader with *GET_READER* (page 23).

If a workflow is in progress and a card with disabled eID function was inserted, this message will still be sent, but the workflow will be paused until a card with enabled eID function is inserted.

New in version 2.2.0: Parameter **card** signals an **unknown card** with an empty object (*API_LEVEL* (page 33) 3).

New in version 1.24.0: Parameter **insertable** added.

New in version 1.16.0: Parameter **keypad** added.

- **name**: Identifier of card reader.
- **insertable**: Indicates whether a card can be inserted via *SET_CARD* (page 26).
- **attached**: Indicates whether a card reader is connected or disconnected.
- **keypad**: Indicates whether a card reader has a keypad. The parameter is only shown when a reader is attached.
- **card**: Provides information about an inserted eID card. An empty object is used for an unknown card. Otherwise null.
  - **inoperative**: True if PUK is inoperative and cannot unblock PIN, otherwise false. This can be recognized if user enters a correct PUK only. It is not possbible to read this data before a user tries to unblock the PIN.
  - **deactivated**: True if eID function is deactivated, otherwise false. The scan dialog on iOS won't be closed if this is True. You need to send *INTERRUPT* (page 28) yourself to show an error message.
  - **retryCounter**: Count of possible retries for the PIN. If you enter a PIN with command *SET_PIN* (page 29) it will be decreased if PIN was incorrect.

eID card:

```
{
  "msg": "READER",
  "name": "NFC",
  "insertable": false,
  "attached": true,
  "keypad": false,
  "card":
         {
          "inoperative": false,
          "deactivated": false,
          "retryCounter": 3
         }
}
```

Unknown card (*API_LEVEL* (page 33) **3):**

```
{
  "msg": "READER",
  "name": "NFC",
  "insertable": false,
  "attached": true,
  "keypad": false,
  "card": {}
}
```

No card:

```
{
  "msg": "READER",
  "name": "NFC",
  "insertable": false,
  "attached": true,
  "keypad": false,
  "card": null
}
```

8.16 READER_LIST

Provides information about all connected card readers.

Changed in version 1.24.0: Parameter **reader** was renamed to **readers** with *API_LEVEL* (page 33) **2**.

- **readers**: A list of all connected card readers. Please see message *READER* (page 42) for details.

```
{
  "msg": "READER_LIST",
  "readers":
           [
             {
              "name": "Example reader 1 [SmartCard] (1234567) 01 00",
```

(continues on next page)

(continued from previous page)


```
            "insertable": false,
            "attached": true,
            "keypad": true,
            "card": null
          },

          {
            "name": "NFC",
            "insertable": false,
            "attached": true,
            "keypad": false,
            "card":
                  {
                    "inoperative": false,
                    "deactivated": false,
                    "retryCounter": 3
                  }
          }
        ]
}
```


8.17 STATUS

Provides information about the current workflow and state. This message indicates if a workflow is in progress or the workflow is paused. This can occur if the AusweisApp needs additional data like *ACCESS_RIGHTS* (page 31) or *INSERT_CARD* (page 41).

The messages will be sent by default if not disabled in *RUN_AUTH* (page 23) or *RUN_CHANGE_PIN* (page 24).

New in version 1.24.0: Support of STATUS message in *API_LEVEL* (page 33) **2**.

- **workflow**: Name of the current workflow. If there is no workflow in progress this will be null.
- **progress**: Percentage of workflow progress. If there is no workflow in progress this will be null.
- **state**: Name of the current state if paused. If there is no workflow in progress or the workflow is not paused this will be null.


```
{
  "msg": "STATUS",
  "workflow": "AUTH",
  "progress": 25,
  "state": "ACCESS_RIGHTS"
}
```

8.18 PAUSE

This message will be sent by AusweisApp to signal certain waiting conditions. E.g. if a card connection cannot be established or maintained due to a bad positioning of the card, this message will be send with *BadCardPosition* (page 45) as the cause. After the causing issue, denoted be the value of *cause* (page 45), was fixed, the next command should be *CONTINUE* (page 27).

New in version 2.2.0: The message *PAUSE* (page 45) was introduced in *API_LEVEL* (page 33) **3**. The SDK will halt until *CONTINUE* (page 27) was sent to acknowledge the condition.

- **cause**: The cause for the waiting condition.

```
{
  "msg": "PAUSE",
  "cause": "BadCardPosition"
}
```

The following list of *causes* (page 45) contains all conditions. Each condition is a stable and unique string to safely distinguish each waiting condition.

- **BadCardPosition**: Denotes an unstable or lost card connection. After fixing the issue you have to call *CONTINUE* (page 27) to go on with the workflow.

8.19 UNKNOWN_COMMAND

Indicates that the command type is unknown.

If your application receives this message you passed a wrong command to the AusweisApp.

Please fix your command and send it again!

Be aware of case sensitive names in *Commands* (page 22).

- **error**: Name of the unknown command.

```
{
  "msg": "UNKNOWN_COMMAND",
  "error": "get_INFo"
}
```

9 Workflow

This section shows some possible workflows as an example communication between your application and the AusweisApp.

The JSON structure can be identified by parameter **cmd** or parameter **msg** as described in section *Commands* (page 22) and section *Messages* (page 31).

- **cmd**: Commands are sent by your application.
- **msg**: Messages are sent by the AusweisApp.

9.1 Minimal successful authentication

The following messages and commands are the minimal iterations of a successful authentication.

We assume that the user already inserted a card into the connected card reader.

```
{"cmd": "RUN_AUTH", "tcTokenURL": "https://test.governikus-eid.de/
↪AusweisAuskunft/WebServiceRequesterServlet"}

{"msg": "AUTH"}

{"msg": "ACCESS_RIGHTS", "chat": {"effective":["FamilyName","GivenNames",
↪"DocumentType"],"optional":["GivenNames"],"required":["FamilyName",
↪"DocumentType"]}}

{"cmd": "ACCEPT"}

{"msg": "ENTER_PIN", "reader": {"attached":true,"card":{"inoperative":false,
↪"deactivated":false,"retryCounter":3},"keypad":false,"name":"NFC"}}

{"cmd": "SET_PIN", "value": "123456"}

{"msg": "AUTH","result": {"major":"http://www.bsi.bund.de/ecard/api/1.1/
↪resultmajor#ok"},"url":"https://test.governikus-eid.de/DEMO/?refID=123456"}
```

9.2 Successful authentication with CAN

The following messages and commands show possible iterations if the user enters an incorrect PIN and CAN twice before entering the correct CAN and PIN.

We assume that the user did not insert a card into the connected card reader.

```
{"cmd": "RUN_AUTH", "tcTokenURL": "https://test.governikus-eid.de/DEMO"}

{"msg": "AUTH"}

{"msg": "ACCESS_RIGHTS", "chat": {"effective":["DocumentType"],"optional":[],
↪"required":["DocumentType"]}}

{"cmd": "ACCEPT"}

{"msg": "INSERT_CARD"}

{"msg": "ENTER_PIN", "reader": {"attached":true,"card":{"inoperative":false,
↪"deactivated":false,"retryCounter":3},"keypad":false,"name":"NFC"}}

{"cmd": "SET_PIN", "value": "000000"}

{"msg": "ENTER_PIN", "reader": {"attached":true,"card":{"inoperative":false,
↪"deactivated":false,"retryCounter":2},"keypad":false,"name":"NFC"}}
```

(continues on next page)

(continued from previous page)


```
{"cmd": "SET_PIN", "value": "000001"}

{"msg": "ENTER_CAN", "reader": {"attached":true,"card":{"inoperative":false,
↪"deactivated":false,"retryCounter":1},"keypad":false,"name":"NFC"}}

{"cmd": "SET_CAN", "value": "000000"}

{"msg": "ENTER_CAN", "reader": {"attached":true,"card":{"inoperative":false,
↪"deactivated":false,"retryCounter":1},"keypad":false,"name":"NFC"}}

{"cmd": "SET_CAN", "value": "654321"}

{"msg": "ENTER_PIN", "reader": {"attached":true,"card":{"inoperative":false,
↪"deactivated":false,"retryCounter":1},"keypad":false,"name":"NFC"}}

{"cmd": "SET_PIN", "value": "123456"}

{"msg": "AUTH","result": {"major":"http://www.bsi.bund.de/ecard/api/1.1/
↪resultmajor#ok"},"url":"https://test.governikus-eid.de/DEMO/?refID=123456"}
```


9.3 Cancelled authentication

The following messages and commands show possible iterations if the user cancels the authentication.

We assume that the user did not connect the card reader.


```
{"cmd": "RUN_AUTH", "tcTokenURL": "https://test.governikus-eid.de/DEMO"}

{"msg": "AUTH"}

{"msg": "ACCESS_RIGHTS", "chat": {"effective":["DocumentType"],"optional":[],
↪"required":["DocumentType"]}}

{"cmd": "CANCEL"}

{"msg": "AUTH", "result": {"description":"The process has been cancelled.",
↪"language":"en","major":"http://www.bsi.bund.de/ecard/api/1.1/resultmajor
↪#error","message":"The process has been cancelled.","minor":"http://www.bsi.
↪bund.de/ecard/api/1.1/resultminor/sal#cancellationByUser"},"url":"https://
↪test.governikus-eid.de/DEMO/?errID=123456"}
```

9.4 Set some access rights

The following messages and commands show possible iterations if the user disables and enables an access right.

We assume that the user did not connect the card reader.

```
{"cmd": "RUN_AUTH", "tcTokenURL": "https://test.governikus-eid.de/DEMO"}

{"msg": "AUTH"}

{"msg": "ACCESS_RIGHTS", "chat": {"effective":["FamilyName","GivenNames",
↪"DocumentType"],"optional":["GivenNames"],"required":["FamilyName",
↪"DocumentType"]}}

{"cmd": "SET_ACCESS_RIGHTS", "chat": []}

{"msg": "ACCESS_RIGHTS", "chat": {"effective":["FamilyName","DocumentType"],
↪"optional":["GivenNames"],"required":["FamilyName","DocumentType"]}}

{"cmd": "SET_ACCESS_RIGHTS", "chat": ["GivenNames"]}

{"msg": "ACCESS_RIGHTS", "chat": {"effective":["FamilyName","GivenNames",
↪"DocumentType"],"optional":["GivenNames"],"required":["FamilyName",
↪"DocumentType"]}}

{"cmd": "CANCEL"}

{"msg": "AUTH", "result": {"description":"The process has been cancelled.",
↪"language":"en","major":"http://www.bsi.bund.de/ecard/api/1.1/resultmajor
↪#error","message":"The process has been cancelled.","minor":"http://www.bsi.
↪bund.de/ecard/api/1.1/resultminor/sal#cancellationByUser"},"url":"https://
↪test.governikus-eid.de/DEMO/?errID=123456"}
```

9.5 Minimal successful PIN change

The following messages and commands are the minimal iterations of a successful PIN change.

We assume that the user already inserted a card into the connected card reader.

```
{"cmd": "RUN_CHANGE_PIN"}

{"msg": "CHANGE_PIN"}

{"msg": "ENTER_PIN", "reader": {"attached":true,"card":{"inoperative":false,
↪"deactivated":false,"retryCounter":3},"keypad":false,"name":"NFC"}}

{"cmd": "SET_PIN", "value": "123456"}

{"msg": "ENTER_NEW_PIN", "reader": {"attached":true,"card":{"inoperative
↪":false,"deactivated":false,"retryCounter":3},"keypad":false,"name":"NFC"}}
```

(continues on next page)

(continued from previous page)

```
{"cmd": "SET_NEW_PIN", "value": "123456"}

{"msg": "CHANGE_PIN", "success": true}
```

10 Failure Codes

The AusweisApp will send failure codes indicating what went wrong and where it happened as well as how to solve it.

10.1 Helpful tips

The following tips will help you to prevent many of the failures mentioned in the next section. Also they assist you in receiving further guidance and help.

Check ID card position

Check the position of your ID card on the smartphone or card reader. Especially with smartphones, the field strength for the power supply of the ID card is not always sufficient. If you place your smartphone on your ID card, please also ensure that your surface is not electrically conductive, as this can then disrupt or prevent communication with the ID card. If all of the above does not work, please see *Replace ID card of card reader* (page 50).

Contact support

If the provided failure code did not help to resolve the issue, please contact the support (https://www.ausweisapp.bund.de/en/help-and-support), including the error code, situation description, and logfile, so that they can identify issues in your system configuration or AusweisApp. If you are using the AusweisApp you will find the logfile in the Help section.

Inform service provider

Directly notify the service provider if the failure code contained an incorrect TLS or service configuration. Usually the service provider contact information are available on the website on which you have started the authentication.

Fix connection problems

For any failure code that mentions connection issues in its cause, it is recommended to check your current connection. Verify an active internet connection, by opening e.g. https://www.ausweisapp.bund.de in the browser of your choice. This includes checking your firewall and antivirus configuration as well as your local network hardware. Ultimately the problem may be with your telecommunications provider, or the service provider. Please refer to the attached "Network_Error" for details. If you are using the AusweisApp, the diagnosis, which is located in the help section, may assist you in finding issues.

Replace ID card of card reader

It cannot be ruled out that your ID card is defective or, due to necessary updates, initially requires more power than your current smartphone or card reader can supply. If possible, try other card readers or smartphones. If the ID card still does not work you might need to replace it with a new one at your responsible authority.

10.2 Codes

- **User_Cancelled**
  The user canceled the workflow. In the SDK case, the user can also be a third-party application that has disconnected from the SDK.
  Possible Solutions: Complete the workflow without canceling.
- **Card_Removed**
  Possible causes for this failure are:
  1 Unstable NFC connection
  2 Removal of the ID card
  3 Removal of the card reader
  4 Cancellation of the remote access
  Possible Solutions:
  1 *Check ID card position* (page 49)
  2 The ID card has to be present on the reader during the whole workflow
  3 The card reader has to be attached during the whole workflow
  4 You must not cancel the remote access during the whole workflow
- **Parse_TcToken_Invalid_Url**
  An authentication was started according to TR-03124-1 section 2.2.1.1. However, no valid tcTokenURL was transmitted.
  Possible Solutions: *Inform service provider* (page 49).
- **Parse_TcToken_Missing_Url**
  An authentication was started according to TR-03124-1 section 2.2.1.1. However, the query "tcTokenURL" is missing.
  Possible Solutions: *Inform service provider* (page 49).
- **Get_TcToken_Invalid_Url**
  An authentication was started according to TR-03124-1 section 2.2.1.1. However, no valid tcTokenURL using the https scheme was transmitted.
  Possible Solutions: *Inform service provider* (page 49).
- **Get_TcToken_Invalid_Redirect_Url**
  The tcTokenURL call was answered with a redirect. The URL provided there is invalid or does not use the https scheme.
  Possible Solutions: *Inform service provider* (page 49).
- **Get_TcToken_Invalid_Certificate_Key_Length**
  The TLS certificate transmitted by the server when retrieving the tcToken uses an insufficient key length.
  Possible Solutions: *Inform service provider* (page 49).

- **Get_TcToken_Invalid_Ephemeral_Key_Length**
  The ephemeral key length generated by the TLS handshake to get the tcToken is insufficient.
  Possible Solutions: *Inform service provider* (page 49).
- **Get_TcToken_Invalid_Server_Reply**
  The server responded to the request for the tcToken neither with content nor with a forwarding.
  Possible Solutions: *Inform service provider* (page 49).
- **Get_TcToken_ServiceUnavailable**
  The server intended for providing the tcToken is temporarily unavailable.
  Possible Solutions: *Inform service provider* (page 49).
- **Get_TcToken_Server_Error**
  A server error 5xx occurred on requesting the tcToken.
  Possible Solutions: *Inform service provider* (page 49).
- **Get_TcToken_Client_Error**
  A client error 4xx occurred on requesting the tcToken.
  Possible Solutions: *Contact support* (page 49).
- **Get_TcToken_Empty_Data**
  The server responded to the request for the tcToken with empty content.
  Possible Solutions: *Inform service provider* (page 49).
- **Get_TcToken_Invalid_Data**
  The server responded to the request for the tcToken with content that does not comply with TR-03124-1 section 2.6.
  Possible Solutions: *Inform service provider* (page 49).
- **Get_TcToken_Network_Error**
  A network error occurred while retrieving the tcToken.
  Possible Solutions: *Fix connection problems* (page 49).
- **Certificate_Check_Failed_No_Description**
  TR-03112-7 section 3.6.4.1 requires a description of the service provider certificate. However, this was not transmitted by the service provider in the EAC1InputType.
  Possible Solutions: *Inform service provider* (page 49).
- **Certificate_Check_Failed_No_SubjectUrl_In_Description**
  TR-03124-1 section 2.7.3 requires that the service provider's URL is included in the description of the certificate. The URL does not exist.
  Possible Solutions: *Inform service provider* (page 49).
- **Certificate_Check_Failed_Hash_Mismatch**
  TR-03124-1 section 2.7.3 requires that the hash of the certificate description matches that stored in the certificate. These don't match.
  Possible Solutions: *Inform service provider* (page 49).
- **Certificate_Check_Failed_Same_Origin_Policy_Violation**
  TR-03124-1 section 2.7.3 requires that the tcTokenUrl has the same origin as the service provider's URL from the certificate description. This condition is not met.
  Possible Solutions: *Inform service provider* (page 49).
- **Certificate_Check_Failed_Hash_Missing_In_Description**

TR-03124-1 Section 2.7.3 requires that the hashes of all TLS certificates used are included in the description of the service provider certificate. This condition is not met.

Possible Solutions: *Inform service provider* (page 49).

- **Pre_Verification_No_Test_Environment**

  Occurs when the development mode of AusweisApp is activated and a genuine ID card is used.

  Possible Solutions: Disable developer mode. The use of genuine ID cards is not permitted with activated developer mode, as this is only intended to facilitate the commissioning of services with test ID cards.

- **Pre_Verification_Invalid_Certificate_Chain**

  A certificate chain was sent from the server that is unknown to AusweisApp.

  Possible Solutions: *Inform service provider* (page 49).

- **Pre_Verification_Invalid_Certificate_Signature**

  At least one signature in the certificate chain used by the server is incorrect.

  Possible Solutions: *Inform service provider* (page 49).

- **Pre_Verification_Certificate_Expired**

  The certificate chain used by the server is currently not valid.

  Possible Solutions: Make sure your system time is set correctly. If the problem persists, see *Inform service provider* (page 49).

- **Extract_Cvcs_From_Eac1_No_Unique_At**

  The server submitted a certificate chain that contained more than one terminal certificate.

  Possible Solutions: *Inform service provider* (page 49).

- **Extract_Cvcs_From_Eac1_No_Unique_Dv**

  The server transmitted a certificate chain containing more than one DV certificate.

  Possible Solutions: *Inform service provider* (page 49).

- **Extract_Cvcs_From_Eac1_At_Missing**

  The server transmitted a certificate chain that does not contain a terminal certificate.

  Possible Solutions: *Inform service provider* (page 49).

- **Extract_Cvcs_From_Eac1_Dv_Missing**

  The server transmitted a certificate chain that does not contain a DV certificate.

  Possible Solutions: *Inform service provider* (page 49).

- **Connect_Card_Connection_Failed**

  In order to communicate with the ID card, a connection must first be established. This process failed.

  Possible Solutions: *Check ID card position* (page 49).

- **Connect_Card_Eid_Inactive**

  The PIN of the card is deactivated. The card can currently only be used with the CAN for on-site reading.

  Possible Solutions: When your ID card was issued, the online ID card function (the PIN) was not activated or you had the function deactivated afterwards. You can have the function activated at the citizens' office (Bürgeramt) or activate it with the CAN at https://www.pin-ruecksetzbrief-bestellen.de.

- **Prepace_Pace_Smart_Eid_Invalidated**

The attempt to establish a connection with a PIN to a Smart-eID failed, because all PIN-attempts have been used.

Possible Solutions: The PIN is permanently disabled after 3 failed attempts. Please set up your Smart-eID again.

- **Establish_Pace_Channel_Basic_Reader_No_Pin**

  An attempt was made to establish a PACE-channel with a basic reader. However the PIN, CAN, or PUK could not be taken over after the user-input.

  Possible Solutions: *Contact support* (page 49).

- **Establish_Pace_Channel_Puk_Inoperative**

  An attempt was made to set up a PACE channel with the PUK to unlock the PIN. However, the PUK can no longer be used because it has already been used 10 times.

  Possible Solutions: The PIN can be unlocked with the PUK after three incorrect entries. However, this is only possible ten times and you have reached that limit. However you can set a new PIN at the citizens' office (Bürgeramt) or let it be set with the CAN at https://www.pin-ruecksetzbrief-bestellen.de.

- **Establish_Pace_Channel_Unknown_Password_Id**

  The establishment of a PACE channel was finished. However, an unsupported password type was used (PIN, CAN, PUK are supported).

  Possible Solutions: *Contact support* (page 49).

- **Establish_Pace_Channel_User_Cancelled**

  The user canceled the workflow on a comfort USB reader or a smartphone as a card reader with keyboard mode enabled.

  Possible Solutions: Complete the workflow without canceling.

- **Maintain_Card_Connection_Pace_Unrecoverable**

  An error occurred while setting up the PACE channel that was not due to user error.

  Possible Solutions: The connection to the ID card could not be established with the PIN, CAN, or PUK. The entered passwords have no influence on this. Please note *Check ID card position* (page 49).

- **Did_Authenticate_Eac1_Card_Command_Failed**

  The 4th card command of the terminal authentication according to TR-0110-3 section B.3 failed.

  Possible Solutions: *Check ID card position* (page 49).

- **Process_Certificates_From_Eac2_Cvc_Chain_Missing**

  When setting up the PACE channel with PIN or CAN, the ID card communicated which certificate it knew. However, the server sent a certificate chain that does not contain this certificate.

  Possible Solutions: *Inform service provider* (page 49).

- **Did_Authenticate_Eac2_Invalid_Cvc_Chain**

  When setting up the PACE channel with PIN or CAN, the ID card communicated which certificate it knew. However, the server sent a certificate chain that does not contain this certificate.

  Possible Solutions: *Inform service provider* (page 49).

- **Did_Authenticate_Eac2_Card_Command_Failed**

  A terminal or chip authentication card command according to TR-0110-3 sections B.2 and B.3 failed.

Possible Solutions: *Check ID card position* (page 49).

- **Generic_Send_Receive_Paos_Unhandled**
  A message was sent by the server in the PAOS communication during authentication, that could not be completely processed.
  Possible Solutions: *Contact support* (page 49).
- **Generic_Send_Receive_Network_Error**
  A network error has occurred in the PAOS communication during authentication.
  Possible Solutions: *Fix connection problems* (page 49).
- **Generic_Send_Receive_Tls_Error**
  An authentication error occurred in the PAOS communication during the TLS handshake. The TLS certificate is incorrect.
  Possible Solutions: *Inform service provider* (page 49).
- **Generic_Send_Receive_Service_Unavailable**
  The server intended for the PAOS communication during authentication is temporarily unavailable.
  Possible Solutions: *Inform service provider* (page 49).
- **Generic_Send_Receive_Server_Error**
  A server error 5xx occurred in the PAOS communication during authentication.
  Possible Solutions: *Inform service provider* (page 49).
- **Generic_Send_Receive_Client_Error**
  A client error 4xx occurred in the PAOS communication during authentication.
  Possible Solutions: *Contact support* (page 49).
- **Generic_Send_Receive_Paos_Unknown**
  An unknown message was sent by the server in the PAOS communication during authentication.
  Possible Solutions: *Inform service provider* (page 49).
- **Generic_Send_Receive_Paos_Unexpected**
  An unexpected message was sent by the server in the PAOS communication during authentication.
  Possible Solutions: *Inform service provider* (page 49).
- **Generic_Send_Receive_Invalid_Ephemeral_Key_Length**
  The symmetric key generated by the TLS handshake for PAOS communication is not long enough.
  Possible Solutions: *Inform service provider* (page 49).
- **Generic_Send_Receive_Certificate_Error**
  The TLS certificate for PAOS communication uses key lengths that are too small or is not included in the description of the service provider certificate.
  Possible Solutions: *Inform service provider* (page 49).
- **Generic_Send_Receive_Session_Resumption_Failed**
  Failed to resume TLS session during PAOS communication.
  Possible Solutions: *Contact support* (page 49).
- **Transmit_Card_Command_Failed**

During authentication, card commands transmitted in PAOS communication could not be correctly transmitted to the card.

Possible Solutions: *Check ID card position* (page 49).

- **Start_Paos_Response_Missing**

  The message "StartPaosResponse" from the server could not be evaluated because it does not exist.

  Possible Solutions: *Contact support* (page 49).

- **Start_Paos_Response_Error**

  The "StartPaosResponse" message from the server returned an error. The AusweisApp or the ID card did not behave as expected by the server.

  Possible Solutions: *Contact support* (page 49).

- **Check_Refresh_Address_Fatal_Tls_Error_Before_Reply**

  An error occurred during the TLS handshake when checking the return address after a successful authentication. The TLS certificate is incorrect.

  Possible Solutions: *Inform service provider* (page 49).

- **Check_Refresh_Address_Invalid_Ephemeral_Key_Length**

  The symmetric key generated by the TLS handshake when calling the return address is not long enough.

  Possible Solutions: *Inform service provider* (page 49).

- **Check_Refresh_Address_Service_Unavailable**

  The server providing the return address is temporarily unavailable.

  Possible Solutions: *Inform service provider* (page 49).

- **Check_Refresh_Address_Server_Error**

  A server error 5xx occurred on requesting the return address.

  Possible Solutions: *Inform service provider* (page 49).

- **Check_Refresh_Address_Client_Error**

  A client error 4xx occurred on requesting the return address.

  Possible Solutions: *Contact support* (page 49).

- **Check_Refresh_Address_Service_Timeout**

  The call to the return address did not provide an answer within 30 seconds.

  Possible Solutions: *Fix connection problems* (page 49).

- **Check_Refresh_Address_Proxy_Error**

  A proxy server was configured by the operating system or the settings of AusweisApp. This didn't work for checking the return address.

  Possible Solutions: *Fix connection problems* (page 49).

- **Check_Refresh_Address_Fatal_Tls_Error_After_Reply**

  When checking the return address after successful authentication, the TLS handshake could not be completed successfully.

  Possible Solutions: *Fix connection problems* (page 49).

- **Check_Refresh_Address_Unknown_Network_Error**

  A unknown error occurred when checking the return address after successful authentication.

  Possible Solutions: *Fix connection problems* (page 49).

- **Check_Refresh_Address_Invalid_Http_Response**
  The call to the return address did not result in forwarding.
  Possible Solutions: *Inform service provider* (page 49).
- **Check_Refresh_Address_Empty**
  The call to the return address led to a redirect but no URL was supplied.
  Possible Solutions: *Inform service provider* (page 49).
- **Check_Refresh_Address_Invalid_Url**
  The call to the return address led to a redirect, but no correct URL was supplied.
  Possible Solutions: *Inform service provider* (page 49).
- **Check_Refresh_Address_No_Https_Scheme**
  The call to the return address led to a redirect, but delivered an URL without https scheme.
  Possible Solutions: *Inform service provider* (page 49).
- **Check_Refresh_Address_Fetch_Certificate_Error**
  The server certificate could not be obtained after tracing all redirects.
  Possible Solutions: *Fix connection problems* (page 49).
- **Check_Refresh_Address_Unsupported_Certificate**
  The check of the return address after a successful authentication was interrupted because the server uses a TLS certificate with unsupported algorithms or key lengths.
  Possible Solutions: *Inform service provider* (page 49).
- **Check_Refresh_Address_Hash_Missing_In_Certificate**
  The server certificate of the return address is not included in the description of the service provider certificate.
  Possible Solutions: *Inform service provider* (page 49).
- **Browser_Send_Failed**
  On desktop systems, the web browser waits for a response from AusweisApp after starting authentication. However, for unknown reasons, the web browser connection to the browser is lost and the answer cannot be sent.
  Possible Solutions: If the problem occurs repeatedly and changing the browser does not help, please *Contact support* (page 49).
- **Generic_Provider_Communication_Network_Error**
  A network error occurred while communicating with a service provider. This only applies to services that are started from AusweisApp, such as self-authentication.
  Possible Solutions: *Fix connection problems* (page 49).
- **Generic_Provider_Communication_Invalid_Ephemeral_Key_Length**
  When communicating with a service provider, the symmetric key generated by the TLS handshake is not long enough. This only applies to services that are started from AusweisApp, such as self-authentication.
  Possible Solutions: *Inform service provider* (page 49).
- **Generic_Provider_Communication_Certificate_Error**
  When communicating with a service provider, the TLS certificate uses key lengths that are insufficient. This only applies to services that are started from AusweisApp, such as self-authentication.
  Possible Solutions: *Inform service provider* (page 49).

- **Generic_Provider_Communication_Tls_Error**

  An error occurred during the TLS handshake when communicating with a service provider. The TLS certificate is incorrect. This only applies to services that are started from AusweisApp, such as self-authentication.

  Possible Solutions: *Inform service provider* (page 49).

- **Generic_Provider_Communication_ServiceUnavailable**

  The server of the service provider is temporarily unavailable.

  Possible Solutions: *Inform service provider* (page 49).

- **Generic_Provider_Communication_Server_Error**

  A server error 5xx occurred in the communication with the service provider.

  Possible Solutions: *Inform service provider* (page 49).

- **Generic_Provider_Communication_Client_Error**

  A client error 4xx occurred in the communication with the service provider.

  Possible Solutions: *Contact support* (page 49).

- **Get_SelfAuthData_Invalid_Or_Empty**

  The authentication for the self-authentication was completed successfully, but the server then did not transmit the read data correctly.

  Possible Solutions: *Inform service provider* (page 49).

- **Change_Pin_No_SetEidPinCommand_Response**

  The AusweisApp sent a PIN change command to its core, but received an answer for a different command.

  Possible Solutions: *Contact support* (page 49).

- **Change_Pin_Input_Timeout**

  When changing a PIN, the user took too long to set the new PIN. Timeouts are currently only known from card readers with a PIN pad, which also affects smartphones as card readers with activated keyboard mode.

  Possible Solutions: Enter the PIN within 60 seconds.

- **Change_Pin_User_Cancelled**

  The user canceled the PIN change after entering the current valid PIN. Can only occur with card readers with a PIN pad, which also affects smartphones as card readers with activated keyboard mode.

  Possible Solutions: Carry out the PIN change without abortion.

- **Change_Pin_New_Pin_Mismatch**

  When changing a PIN, the user entered an incorrect confirmation of the new PIN. Can only occur with USB card readers with a PIN pad. Smartphone as a card reader with activated keyboard mode does not allow this behavior.

  Possible Solutions: Confirm the new PIN correctly.

- **Change_Pin_New_Pin_Invalid_Length**

  When changing a PIN, the user entered a new PIN with an incorrect length. Can only occur with USB card readers with a PIN pad. However, there is no known device/case that allows this possibility. Smartphone as a card reader with activated keyboard mode does not allow this behavior.

  Possible Solutions: *Contact support* (page 49).

- **Change_Pin_Unexpected_Transmit_Status**
  The command to change the PIN has been transmitted and answered. However, the answer is blank, unknown, or unexpected.
  Possible Solutions: *Check ID card position* (page 49).
- **Change_Pin_Card_New_Pin_Mismatch**
  Like Change_Pin_New_Pin_Mismatch but at a higher protocol level.
  Possible Solutions: Confirm the new PIN correctly.
- **Change_Pin_Card_User_Cancelled**
  Like Change_Pin_User_Cancelled but at a higher log level.
  Possible Solutions: Carry out the PIN change without abortion.
- **Change_Pin_Unrecoverable**
  The change PIN workflow encountered an error, which prevents the continuation of the workflow. This often indicates a problem with the card connection.
  Possible Solutions: *Check ID card position* (page 49).
- **Start_Ifd_Service_Failed**
  The IFD service according to TR-03112-6 appendix “IFD Service” could not be started. Either no suitable TLS certificate could be found/generated or the start of the TLS server failed. This applies to both remote access and the local service of AusweisApp on Android that is used through the SDK.
  Possible Solutions: *Contact support* (page 49).
- **Prepare_Pace_Ifd_Unknown**
  The establishment of a PACE channel was requested by the client on a smartphone as a card reader with activated keyboard mode. However, an unsupported password type was requested (PIN, CAN, PUK are supported).
  Possible Solutions: *Contact support* (page 49).
- **Establish_Pace_Ifd_Unknown**
  The establishment of a PACE channel was requested by the client on a smartphone as a card reader with activated keyboard mode. However, an unsupported password type was requested (PIN, CAN, PUK are supported).
  Possible Solutions: *Contact support* (page 49).
- **Enter_Pace_Password_Ifd_User_Cancelled**
  Occurs when the user canceled entering the PIN, CAN, or PUK on a smartphone acting as a card reader with keyboard mode enabled.
  Possible Solutions: *Contact support* (page 49).
- **Enter_New_Pace_Pin_Ifd_User_Cancelled**
  Occurs when the user has canceled entering the new PIN during a PIN change on a smartphone acting as a card reader with keyboard mode enabled.
  Possible Solutions: *Contact support* (page 49).
- **Check_Status_Unavailable**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Check_Applet_Internal_Error**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Install_Smart_User_Cancelled**

Is not yet included in the product and will only be relevant with version 2.0.0.

- **Delete_Smart_User_Cancelled**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Delete_Personalization_User_Cancelled**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Update_Support_Info_User_Cancelled**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Install_Smart_Service_Response_Fail**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Install_Smart_Service_Response_Unsupported**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Install_Smart_Service_Response_Overload**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Install_Smart_Service_Response_Maintenance**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Install_Smart_Service_Response_Nfc_Disabled**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Install_Smart_Service_Response_Integrity_Check_Failed**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Install_Smart_Service_Response_Not_Authenticated**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Install_Smart_Service_Response_Network_Connection_Error**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Update_Support_Info_Call_Failed**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Update_Support_Info_Service_Response_Fail**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Update_Support_Info_Service_Response_Unsupported**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Update_Support_Info_Service_Response_Overload**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Update_Support_Info_Service_Response_Maintenance**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Update_Support_Info_Service_Response_Nfc_Disabled**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Update_Support_Info_Service_Response_Integrity_Check_Failed**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Update_Support_Info_Service_Response_Not_Authenticated**
  Is not yet included in the product and will only be relevant with version 2.0.0.

- **Update_Support_Info_Service_Response_Network_Connection_Error**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Delete_Smart_Service_Response_Fail**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Delete_Smart_Service_Response_Unsupported**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Delete_Smart_Service_Response_Overload**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Delete_Smart_Service_Response_Maintenance**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Delete_Smart_Service_Response_Nfc_Disabled**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Delete_Smart_Service_Response_Integrity_Check_Failed**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Delete_Smart_Service_Response_Not_Authenticated**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Delete_Smart_Service_Response_Network_Connection_Error**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Delete_Personalization_Failed**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Get_Session_Id_Invalid**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Smart_ServiceInformation_Query_Failed**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Get_Challenge_Invalid**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Initialize_Personalization_Failed**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Smart_PrePersonalization_Wrong_Status**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Smart_PrePersonalization_Incomplete_Information**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Transmit_Personalization_Size_Mismatch**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Start_Paos_Response_Personalization_Empty**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Start_Paos_Response_Personalization_Invalid**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Finalize_Personalization_Failed**

Is not yet included in the product and will only be relevant with version 2.0.0.

- **Insert_Card_No_SmartReader**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Insert_Card_Multiple_SmartReader**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Insert_Card_Unknown_Eid_Type**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Insert_Card_Invalid_SmartReader**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Insert_Card_Missing_Card**
  Is not yet included in the product and will only be relevant with version 2.0.0.
- **Change_Smart_Pin_Failed**
  Is not yet included in the product and will only be relevant with version 2.0.0.

11 Simulator

The simulator provides a virtual card for testing purposes. It is enabled by default in all SDKs. This will be indicated by a *READER* (page 42) message with the parameter **name** being *Simulator*. This virtual card can be inserted by *SET_CARD* (page 26) or can be automatically used in *Automatic* (page 19) mode. The default values of that virtual card are hardcoded and can only be changed by the **files** and **keys** parameter in the **simulator** parameter.

Note: The simulated card can only be used in a test environment.

11.1 Filesystem

The content of the filesystem can be provided as a JSON array of objects. The `fileId` and `shortFileId` are specified in TR-03110_part4[8]. The `content` is an ASN.1 structure in DER encoding. All fields are hex encoded.

These are the default values if your application does not provide other values as **simulator** parameter in *SET_CARD* (page 26) or if you use the *Automatic* (page 19) mode of *Desktop* (page 18) or the *Container* (page 20) variant.

New in version 2.2.0: Parameter **content** in **keys** added.

Deprecated since version 2.2.0: Parameter **private** in **keys** will be removed in 2.3.0.

```
"files":
 [
    {"fileId": "0101", "shortFileId": "01", "content": "610413024944"},
    {"fileId": "0102", "shortFileId": "02", "content": "6203130144"},
    {"fileId": "0103", "shortFileId": "03", "content":
↪"630a120832303239313033331"},
    {"fileId": "0104", "shortFileId": "04", "content": "64070c054552494b41"},
    {"fileId": "0105", "shortFileId": "05", "content":
↪"650c0c0a4d55535445524d414e4e"},
    {"fileId": "0106", "shortFileId": "06", "content": "66020c00"},
    {"fileId": "0107", "shortFileId": "07", "content": "67020c00"},
    {"fileId": "0108", "shortFileId": "08", "content":
↪"680a120831393634303831332"},
    {"fileId": "0109", "shortFileId": "09", "content":
↪"690aa1080c064245524c494e"},
    {"fileId": "010a", "shortFileId": "0a", "content": "6a03130144"},
    {"fileId": "010b", "shortFileId": "0b", "content": "6b03130146"},
    {"fileId": "010c", "shortFileId": "0c", "content":
↪"6c30312e302c06072a8648ce3d0101022100a9fb57dba1eea9bc3e660a909d838d726e3bf623d52620282013
↪"},
    {"fileId": "010d", "shortFileId": "0d", "content": "6d080c064741424c4552"}
↪,
    {"fileId": "010f", "shortFileId": "0f", "content":
```

(continues on next page)

[8] https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03110/BSI_TR-03110_Part-4_V2-2.pdf

(continued from previous page)

```
↪"6f0a12083230313931313031"},
    {"fileId": "0111", "shortFileId": "11", "content":
↪"712d302baa120c1048454944453545241e1ba9e45203137ab070c054bc3964c4ead03130144ae0713053531
↪"},
    {"fileId": "0112", "shortFileId": "12", "content": "7209040702760503150000
↪"},
    {"fileId": "0113", "shortFileId": "13", "content":
↪"7316a1140c125245534944454e4345205045524d49542031"},
    {"fileId": "0114", "shortFileId": "14", "content":
↪"7416a1140c125245534944454e4345205045524d49542032"},
    {"fileId": "0115", "shortFileId": "15", "content":
↪"7515131374656c3a2b34392d3033302d31323334353637"},
    {"fileId": "0116", "shortFileId": "16", "content":
↪"761516136572696b61406d75737465726d616e6e2e6465"}
 ],
 "keys":
 [
    {"id": 1, "content":
↪"308202050201003081ec06072a8648ce3d02013081e0020101302c06072a8648ce3d0101022100a9fb57dba1e
↪"},
    {"id": 2, "content":
↪"308202050201003081ec06072a8648ce3d02013081e0020101302c06072a8648ce3d0101022100a9fb57dba1e
↪"}
 ]
```

All keys and also the files from *Advanced* (page 63) will always be present and can be overwritten as they are required for successful authentication. The other files will not exist until they are specified so it is possible to simulate a missing piece of personal data.

The keys are used to calculate the pseudonym. Key 1 is used to check the blacklist while key 2 is used to calculate the pseudonym for the service provider.

A new key can be generated with OpenSSL (convert base64 to hex after generation).

```
openssl genpkey -algorithm ec -pkeyopt ec_paramgen_curve:brainpoolP256r1
```

11.2 Advanced

In addition the following files and keys can be overridden to change the technical behavior of the virtual card. This does not affect the personal data and is only required for eID-Client or eID-Server development.

```
"files":
 [
    {"fileId": "2f00", "shortFileId": "1e", "content":
↪"61324f0fe828bd080fa00000167455349474e500f434941207a752044462e655369676e5100730c4f0aa000
↪"},
    {"fileId": "011c", "shortFileId": "1c", "content":
↪"3181c13012060a04007f000702020402020201020201 0d300d060804007f000702020202010230 12060a04007
↪"},
    {"fileId": "011d", "shortFileId": "1d", "content":
```

(continues on next page)

(continued from previous page)

```
↪"308209f606092a864886f70d010702a08209e7308209e3020103310f300d06096086480165030402030500308
↪"}
 ],
"keys":
 [
    {"id": 41, "content":
↪"308202050201003081ec06072a8648ce3d02013081e0020101302c06072a8648ce3d0101022100a9fb57dba1e
↪"}
 ]
```

See also:

ISO 7816-4:2005 8.2.1.1

TR-03110_part3[9], part 3: Section A.1.2. Storage on the Chip

[9] https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03110/BSI_TR-03110_Part-3-V2_2.pdf