ABAS ESSENTIALS SDK

Documentation

Version 1.1.13

Table of Contents

This is the technical documentation for abas Essentials SDK^. It is written by developers for developers. For a more abstract view of the product visit https://abas-essentials-sdk.com.

Introduction

abas Essentials SDK^ is a software development kit intended to be used for customizing abas ERP^.

abas Essentials SDK^ enables developers to handle all parts of an abas ERP customization in one project. As abas Essentials SDK^ uses the technology of JFOP server Apps for code deployment in abas ERP, abas Essentials SDK^ projects are called ESDK App^s.

The following abas ERP customizations can be automated using abas Essentials SDK^:

- Add logic and automated tests to the project using abas Java Objects (AJO) or other programming languages
- Create/update data objects, enumerations and/or named types in abas ERP
- Add/update/delete variables from tables of variables
- Create new additional tables of variables
- Import customized database screens
- Import Infosystems and Infosystem screens
- Copy any files to `$MANDANTDIR` or sub-directories
- Add/update lines to/in `fop.txt`
- Add lines to `mandantdir.env`
- Add lines to `logging.custom.properties`

ESDK Gradle Plugin

The core component of the ESDK^ toolset is the ESDK Gradle Plugin. It is the development tool for creating ESDK App^s.

As a Gradle Plugin it is easily integrable into IDEs such as abas Tools, Eclipse and Intellij IDEA. All components of your ESDK App^ are exported from the abas ERP client into your project in the IDE and can be either modified directly in the IDE or re-installed in the abas ERP client, edited and re-exported.

This way all components of your ESDK App^ are part of your local project and can be managed by version control.

As it is based on Gradle — a modern build automation tool — it is not only standardized in a way that

makes it possible to use it in different infrastructures (IDE and platform independent) but can also be adapted by the ESDK App developer him- or herself.

ESDK App Installer

The ESDK App Installer^ is the deployment tool for ESDK App^.

It can be used by any abas Customer to install an ESDK App^ in any of his abas ERP clients.

This way an ESDK App^ is developed once and can be distributed to any abas Customer. Using the ESDK App Installer^ is so straight-forward that the Customer's system administrator run the installation of the favored ESDK App^ without the help of the app's developer or any other party.

Help and Further Information

The ESDK Landing Page provides general information about ESDK^ and additional resources.

Help and troubleshooting articles can be searched for in the ESDK Knowledge Base alongside the ESDK Service Portal.

1. Project setup

Setting up a new ESDK App Project^ for development.

1.1. Prerequisites

For development with abas Essentials SDK^ we recommend the following software and tools:

1.1.1. Software on your development machine

To develop an app, the following technologies must already be installed on your development machine:

- JDK8, Java Development Kit (for example Amazon Corretto 8)
- Gradle, a build automation tool
- Docker, a software containerization platform

For further information take a look at the appropriate homepages.

1.1.2. Development tools

- an IDE (e.g. Eclipse, IntelliJ IDEA, or abas Tools);
- a command-line interface (bash, PowerShell, Terminal);
- an abas ERP client (version 2017r2n00 or above);
- a Nexus Artifact Server.

You can use a dockerized abas ERP client and Nexus Artifact Server or one or both as a self-hosted and self-administered instance.

1.1.3. abas ERP and Nexus Artifact Server as Docker Containers

We recommend developing with Docker containers.

To download the Docker images and start the erp and Nexus containers you can download the docker-compose.yml file and run

```
docker-compose up -d
```

The ESDK Project Builder generates a `docker-compose.yml` file for the abas ERP version you have chosen as "Development Version".

Alternatively you can run the following two commands and do the naming and port mapping manual-

ly:

```
docker run --init -d --name erp -p 2205:22 -p 8001:80 -p 6560:6550 -p
48592:48392 -e ABAS_GUI_PORT=48592 -h dockerbau
sdp.registry.abas.sh/abas/test:<version> run ①

docker run -d --name nexus -p 8081:8081 sonatype/nexus:oss
```

① Replace `<version>` with the version of abas ERP you want to develop against, e.g. 2017r4n16p36.
For (no longer supported) version 2016r4n13 you don't need the option `run`.

Using `docker logs erp -f` you can see all log messages during container start up.

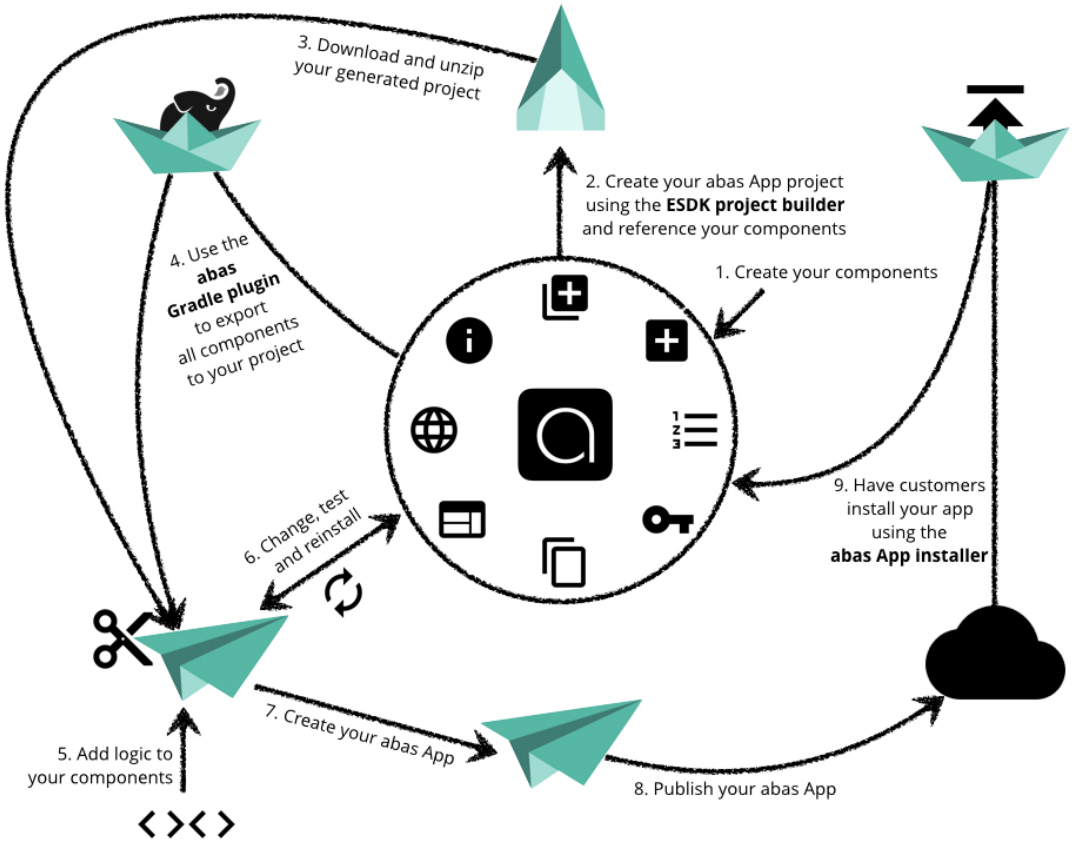
1.1.4. Self-hosted and self-administered instances

If you already have an abas ERP installation and/or a Nexus Artifact Server and want to use them for development, you can do so, too.

You need to make sure that your abas ERP installation can access the Nexus Artifact Server. Otherwise the `publishHomeDirJars` and `publishClientDirJars` tasks will not work. Often when using an existing abas ERP installation it does not have access to your local computer. If you want to use a dockerized Nexus Artifact Server you can also start it on the server your abas ERP installation runs on.

2. Setup ESDK development environment

Set up a development environment for a new or an existing ESDK App^.



2.1. Setup abas ERP client and Nexus Artifact Server

2.1.1. Log in to the abas Partner Docker Registry

To access the abas ERP Docker images you have to log in to the abas Partner Docker Registry.

```
docker login sdp.registry.abas.sh
```

Use your extranet credentials for authentication.

In order to be able to pull Docker images from the abas Partner Docker Registry you need to accept the License Agreement.

2.1.2. Start the containers

Start abas ERP and Nexus Artifact Server containers as described in the Docker chapter.

2.2. Download and configure the mini-GUI

When the abas ERP container is ready, open the following URL in the browser of your choice: http://<your-ip-address>:8001 Download `abasgui-mini-erp.tgz` and `wineks-erp.ini`. Unpack the `abasgui-mini-erp.tgz` archive, rename the `wineks-erp.ini` file to `wineks.ini` and copy it into the unpacked mini-GUI directory. Edit the `wineks.ini` file to look like this:

```
[wineks]
server   = erp
sprachen = Deutsch Englisch

[erp]
host=<your ip address> ①
useEksd=true
portEksd=48592 ②
client=/abas/erp
text=abasERP in Docker
```

① Enter your IP address, your IP address can be determined by executing `ifconfig` (Linux/Mac) or `ipconfig` (Windows) in your preferred command line interface.

② Enter the abas GUI port here, the port mapping is done in the `docker run` command you executed earlier, or is configured in the `docker-compose.yml` file.

2.3. Start abas ERP

Start abas ERP by running `wineks.exe`, or for newer abas ERP clients `abasgui.exe`.

You can only run `wineks.exe` or `abasgui.exe` on Windows. If you are on another operating system, you need to use a virtual machine to use the abas ERP user interface (not for ESDK App^ development in general).

3. Create your ESDK App

3.1. Register your App ID

Go to the ESDK Developer Portal. After logging in with your extranet credentials, you will see a list of all App ID^s already registered by your company. To add a new App ID^ press the red plus button in the upper right corner. Enter your desired App ID^ and click the **`[Create]`** button.

The web app to register App ID^s is tested with Google Chrome browser. Other browser may or may not work.

App ID^ creation is free of charge. To ensure every ESDK App^ uses a unique App ID^ we require App ID^ registration for all ESDK App^s. **If you do not register your App ID^, you cannot install your ESDK App^ using the ESDK App Installer^.** Since another App can register and use your App ID^, your App's additional variables might be overridden.

3.2. Create your components in abas ERP

You can create for example infosystems, additional databases, variables, database screens, data objects, translations, keys and enumerations.

If your ESDK App^ consists of only custom logic (e.g. EventHandlers), or you are not sure what components you will need yet, you can continue to the next step. For help with creating custom components in your abas ERP client, refer to your abas ERP Administrator.

3.3. Create a new ESDK project

After you finished creating your components in abas ERP, you need to create a new ESDK App Project^. You can do this locally with the Project Initializer or online using the Project Builder.

3.3.1. Project Initializer

The Project Initializer is a command-line tool to generate the initial project structure. See the external documentation on how to download and use the tool.

3.3.2. Project Builder

The ESDK Project Builder is an online application allowing you to fill in the appropriate data, such as connection data and the name and dependencies of your app. Reference all the components you previously created in the abas ERP client, then generate the project Zip file and download it to the development environment. Unzip the project to a directory of your choice.

Extranet credentials are required to log in and access the ESDK Project Builder. If you do not have extranet credentials, you can contact us.

3.4. Configure your Gradle project settings

There is a predefined template available to connect to the abas ERP client and Nexus Artifact Server in a Docker container. You may generate the settings on your development machine by running: `./initGradleProperties.sh` in your preferred command line interface to create the `gradle.properties` file.

Make sure that after you run `initGradleProperties.sh` every host value (nexus, edp and ssh) in the `gradle.properties` file is set to your developing machine's IP, or the IP of the machine the corresponding service runs on. On Windows with `Docker Desktop` you may use the string `host.docker.internal`.
Using `localhost` will not work!

Also keep in mind that the local IP address may change when connecting to or disconnecting from a network (e.g. logging into a VPN).

Starting with version 1.0 there is a new configuration option available: External Configuration

3.5. Import the project in your IDE

Import the project as a Gradle project in your IDE. In Eclipse for instance, this enables you to execute Gradle tasks directly from an Eclipse View.

When using abas Tools please make sure to use the latest version, at least version 1.7.3.

3.6. Specify project components

All components belonging to your app (e.g. tables, infosystems, screens, etc. created in step "Create your components in abas ERP" above) have to be specified in the `build.gradle` file.

The chapter Project components provides more details about this.

3.7. Export your abas ERP components

Use the Gradle task Exporting all Components from our ESDK Gradle Plugin^ to export all components specified in the step above from the abas ERP client to your project. You can either run the task right from your IDE or in your preferred command line interface (change to the directory where your app is located and execute `./gradlew exportAll`).

To get an overview about all available tasks, you can run `./gradlew tasks` on your command line interface. The ESDK Gradle Plugin^ tasks are listed in the groups abas Basic tasks, abas Help and Support tasks and abas Professional tasks.

3.8. Add logic

Once the `exportAll` step is finished, your abas ERP components are copied to your project and you

can start to add some logic.

3.8.1. Regenerate the AJO classes

To include classes for your newly created components run the `installAjo` Gradle task.

3.8.2. Provide abas specific dependencies

In order for your project to know about the abas specific dependencies, needed for development with JEDP and AJO, run the `publishAllJars` tasks. Afterwards refresh the Gradle project in your IDE.

3.8.3. Use additional dependencies

To add dependencies to your project they have to be in a Maven repository. This can either be a publicly available Maven repository like Maven Central, or a private one that runs locally or in your company's internal infrastructure.

You have to first add the repository to the `repositories` block in your `build.gradle`, then add the dependency to your `dependencies` block. Specify the Maven group, module name and version. For example, to add Google's Gson Library to your app, add the following to your `build.gradle`:

build.gradle

```
repositories {
    ...
    mavenCentral()
    ...
}
...
dependencies {
    ...
    implementation 'com.google.code.gson:gson:2.8.5'
    ...
}
```

For more information regarding dependency management refer to the Gradle Documentation.

How the shared option influences additional dependencies

In your `build.gradle` you can use the `shared` option to specify what configuration you want to use for `parent.delegation` of your JFOP server app. You can use `parent.delegation=true` or `parent.delegation=false`.

For ESDK App^s, the default value for `parent.delegation` is `false`. If you set `shared = true`, `parent.delegation` is set `true` and the default value for `parent` is set to `DEFAULT_SHARED` automatically if nothing else is specified. For simple EventHandler classes that only use the jar files in abas ERP standard delivery, this is fine.

You can specify an alternative value for your parent classpath by setting `parent=<YourClassPathName>` in your build.gradle file like in the following example:

```
esdk {
    app {
        ...
        shared = true
        parent = "DEFAULT"
        ...
    }
    ...
}
```

The `parent` option also allows you to create a dependency structure between multiple apps that are interdependent:

```
esdk {
    app {
        ...
        shared = true
        parent = "<YourAppId>"
        ...
    }
    ...
}
```

You should only change parent.delegation to `true` by setting `shared=true` in your `build.gradle`, if you explicitly need to share dependencies provided by another JFOP server app. The jar files in `$HOMEDIR/java/lib` are provided with the JFOP server app named `DEFAULT`.

If you are using third-party libraries, we highly recommend to let `shared` set to `false` and specify the app's dependencies explicitly. This keeps its classpath clean and avoids version collisions between third-party libraries that come with your app and the abas ERP standard delivery.

How to explicitly use a dependency

There are three configurations that define which dependencies the app needs when, and how they are provided:

- `provided` dependencies are needed to compile the app but are provided at runtime by the `s3j-fopserver_bootstrap.classpath` and thus not packaged with the app. They must not be listed in the app's classpath file. The ESDK Gradle Plugin^ checks that they are not used in the other dependency configurations.
- `implementation` dependencies are needed to compile the app and are explicitly listed in the app's classpath file. If they do not belong to group `de.abas.homedir` or `de.abas.clientdir`, they are packaged with the app and get installed in the JFOP server app's `lib` directory.

- `runtime` dependencies are not needed to compile the app but to execute it at runtime. Examples are logging frameworks, Apache Commons Collections or abas-jfop-base.

An example

If you want or have to explicitly use a JAR provided in `$HOMEDIR/java/lib` you can reference it in your `build.gradle`'s dependency block using the `provided`, `implementation` or `runtimeOnly` configuration. All libraries in `$HOMEDIR/java/lib` are uploaded to the Nexus server and can be referenced as following:

build.gradle

```
dependencies {
    provided "de.abas.homedir:abas-db-base:1.0.0"
    provided "de.abas.homedir:jedp:1.0.0"
    provided "de.abas.homedir:abas-jfop-runtime-api:1.0.0"
    provided "de.abas.homedir:abas-erp-common:1.0.0"
    provided "de.abas.homedir:abas-enums:1.0.0"

    implementation "de.abas.homedir:abas-axi2:1.0.0"
    implementation "de.abas.homedir:abas-axi:1.0.0"
    implementation "de.abas.homedir:abas-db-internal:1.0.0"
    implementation "de.abas.clientdir:abas-db:1.0.0-SNAPSHOT"
    implementation "de.abas.clientdir:abas-db-infosys:1.0.0-SNAPSHOT"

    implementation "de.abas.homedir:commons-collections:1.0.0"
    runtimeOnly "de.abas.homedir:abas-jfop-base:1.0.0"
    runtimeOnly "de.abas.homedir:jcl-over-slf4j:1.0.0"
    runtimeOnly "de.abas.homedir:slf4j-api:1.0.0"

    testImplementation "junit:junit:4.12"
    testImplementation "org.hamcrest:hamcrest-all:1.3"

    integTestImplementation "de.abas.homedir:abas-db-util:1.0.0"
}
```

All `provided` dependencies are automatically added to the `implementation` classpath (and thus to the deprecated `compile` classpath).

While missing `provided` and `implementation` dependencies get reported by the compiler or your IDE, missing `runtimeOnly` dependencies only are reported when executing the app. Typically, a `ClassNotFoundException` is thrown, mentioning the missing class. The shell script `jarversion.sh` — provided by abas ERP standard delivery — helps to find the right jar file:

```
jarversion.sh -c <fully-qualified class name>
```

jarversion.sh Example

```
$ jarversion.sh -c de.abas.erp.db.DbContext
Class: de.abas.erp.db.DbContext
Path: file:/abas/s3/java/lib/abas-db-base.jar!/de/abas/erp/db/DbContext.class
```

3.9. Use the esdk-client-api

The esdk-client-api enables you to use common functionality in your ESDK App^'s code, examples being getting properties such as your app's name or App ID^ during runtime.

3.9.1. Getting the esdk-client-api

To use the esdk-client-api add the following dependency to the `build.gradle` file of your ESDK App^:

```
dependencies {
    ...
    implementation "de.abas.esdk:client-api:1.0.2:all"
    ...
}
```

In order to resolve the esdk-client-api dependency you need to add the following repository:

```
repositories {
    ...
    maven { url "https://artifactory.abas.sh/artifactory/abas.maven-public/" }
    ...
}
```

For additional help refer to the esdk-client-api's Javadoc.

3.10. Add tests

To ensure that logic and setup of your app work as expected, tests should be added.

Projects set up with Project Initializer or Project Builder include the setup for unit testing with JUnit 4. The ESDK Gradle Plugin^ also adds support for integration tests.

The chapter about Verification tasks provides more details about testing and the provided Gradle tasks.

JUnit 5 support

The default configuration contains the dependencies for tests with JUnit 4. To write and execute tests with JUnit 5, the configuration must be extended:

build.gradle

```
tasks.withType(Test) {
    useJUnitPlatform() ①
    testLogging {
        events('passed', 'skipped', 'failed') ②
    }
}

dependencies {
    implementation "junit:junit:4.12" ③
    implementation 'org.junit.jupiter:junit-jupiter-api:5.7.0' ④
    testRuntimeOnly 'org.junit.vintage:junit-vintage-engine' ⑤
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine' ⑥
}
```

① Tell IDEs to us JUnit Platform test executors

② Log test execution events to be picked up by IDEs

③ JUnit 4 may still be used, e.g. to keep old tests until they are migrated to JUnit 5

④ Use JUnit Jupiter test api (introduced with JUnit 5)

⑤ Allow execution of JUnit 4 tests with JUnit Jupiter

⑥ Test engine to execute JUnit 5 tests

See the JUnit 5 User Guide for more information.

3.11. Synchronize your changes

The ESDK Gradle Plugin^ offers a range of Gradle tasks for transferring changes between your abas ERP development client and your local ESDK App Project^.

Tasks to transfer changes from your abas ERP development client into you project are **export** tasks. Tasks to transfer changes from your local project to your abas ERP development client are **import** or **install** tasks.

All ESDK App^ development specific tasks are grouped in either **abas basic** or **abas professional**.

The **abas basic** group contains all tasks necessary to get you started with ESDK App^ development:

- checkPreconditions: Check if your connection settings are correct.

- publishAllJars: Make home dir and client dir Jars available on the Nexus Artifact Server.
- exportAll: Export all configured components from your abas ERP client to your project.
- syncCode: Synchronize code changes.
- fullInstall: Install all your app components and code into an abas ERP client.

If you are a more advanced user, you can also use the tasks in **abas professional** to only install/import or export the components you changed. This saves you some time during development but since there are quite a view components, it is also a less structured view.

3.12. Create the ESDK App

Once your app is ready to be delivered, you can create a jar file containing all the components that belong to it.

The Gradle task `createEsdkAppJar` will pack all necessary files and information in a jar file.

```
./gradlew createEsdkAppJar
```

3.13. Install your app in another abas ERP client

Install your app from the cloud or locally using the ESDK App Installer.

4. Nexus

Once you start developing an ESDK App^, you need all AJO dependencies to be available in the Sonatype Nexus Artifact server, so that your IDE has access to these dependencies and you can develop logic for your ESDK App^ in AJO.

Previously samba shares where needed to access `$MANDANTDIR/java/lib` and `$HOMEDIR/-java/lib` from your IDE. This is no longer necessary. All dependencies in `$HOMEDIR/java/lib` and `$MANDANTDIR/java/lib` are published to a repository on the Sonatype Nexus Artifact server.

Once the AJO artifacts are available on Sonatype Nexus, no connection to the client is needed at all for writing code and compiling against the AJO libraries.

4.1. Configuration

The Nexus Artifact Server can be run in a Docker container (as suggested in Setup ESDK development environment) or deployed and maintained by yourself.

Both Nexus Artifact Server versions — version 2 and 3 — are supported. Nexus Server version 2 is the default.

The connection properties of your Nexus container need to be configured in the gradle.properties file

in your project. For more details refer to Setup.

For your project to get the libraries from your Nexus, you will need to indicate two repositories. The first repository contains all the `HOMEDIR` libraries ($HOMEDIR/java/lib). The second repository contains the `CLIENTDIR` libraries, mainly `abas-db.jar` and `abas-db-infosystem.jar` ($MANDANTDIR/abasbase/java/lib).

For Sonatype Nexus **version 2** specify:

```
repositories {
    maven { url
"http://$NEXUS_HOST:$NEXUS_PORT/nexus/content/repositories/$NEXUS_NAME-SNAPSHOT"
}
    maven { url
"http://$NEXUS_HOST:$NEXUS_PORT/nexus/content/repositories/$NEXUS_NAME" }
}
```

In case of using the Nexus url instead of Nexus host and port, specify:

```
repositories {
    maven { url "$NEXUS_URL/nexus/content/repositories/$NEXUS_NAME-SNAPSHOT" }
    maven { url "$NEXUS_URL/nexus/content/repositories/$NEXUS_NAME" }
}
nexus {
    nexusUrl = NEXUS_URL
    nexusRepoName = NEXUS_NAME
    nexusPassword = NEXUS_PASSWORD
}
```

For Sonatype Nexus **version 3** specify:

```
repositories {
    maven { url "http://$NEXUS_HOST:$NEXUS_PORT/repository/$NEXUS_NAME-SNAPSHOT"
}
    maven { url "http://$NEXUS_HOST:$NEXUS_PORT/repository/$NEXUS_NAME" }
}
```

In case of using the Nexus url instead of Nexus host and port, specify:

```
repositories {
    maven { url "$NEXUS_URL/repository/$NEXUS_NAME-SNAPSHOT" }
    maven { url "$NEXUS_URL/repository/$NEXUS_NAME" }
}
nexus {
    nexusUrl = NEXUS_URL
    nexusRepoName = NEXUS_NAME
    nexusPassword = NEXUS_PASSWORD
}
```

Make sure the `build.gradle` file in your project actually uses the `NEXUS_VERSION` property. It should contain the following line:

```
nexusVersion = NEXUS_VERSION
```

5. Project structure

The project resources can be found in the following directory and file structure.

```
yourAppProject
 ├──── build.gradle ①
 ├──── docker-compose.yml ②
 ├──── Dockerfile-erp-overrides ③
 ├──── gradle.properties ④
 ├──── gradle.properties.template ⑤
 ├──── gradlew ⑥
 ├──── gradlew.bat ⑦
 ├──── gradle ⑧
 ├──── README.md ⑨
 ├──── .gitignore ⑩
 ├──── Jenkinsfile ⑪
 └──── src
     ├──── integTest ⑫
     ├──── main ⑬
     │    ├──── java ⑭
     │    └──── resources ⑮
     │        ├──── advancedDbScreens ⑯
     │        ├──── base ⑰
     │        ├──── data ⑱
     │        ├──── dbscreens ⑲
     │        ├──── enums ⑳
     │        ├──── enumsAfterVarreorg
     │        ├──── IS
     │        ├──── keys
     │        ├──── namedTypes
     │        ├──── vartab
     │        ├──── fop.json
     │        ├──── logging.custom.properties
     └──── test
```

① Project build configuration

② ERP and Nexus as Docker containers

③ Override settings of standard erp docker images

④ Generated with initGradleProperties.sh and gradle.properties.template

⑤ Template for your local gradle.properties file

⑥ Wrapper to execute Gradle tasks

⑦ Wrapper to execute Gradle tasks for windows

⑧ Directory for Gradle distribution

⑨ Introduction how to setup a project

⑩ Configuration file for Git to specify what files should be ignored

⑪ Template for a Jenkins Job configuration

⑫ Your integration tests

⑬ Project source code and resources

⑭ Your source code

⑮ Your project resources

⑯ DB screens to override standard screens completely when installed in the client

⑰ Files to be transferred to the client

⑱ Data to import to the client

⑲ DB screens to be installed on the client

⑳ Enumerations to be imported

Enumerations to be imported after varreorg of imported vartabs

Infosystems which will be installed on the client

Keys which will be installed on the client

Named types which will be installed on the client

Variable table changes and custom databases

fop.txt changes

Specify your ESDK App^s log output

Your tests (e.g.: JUnit tests)

All files need to be saved in UTF-8 encoding. The only exception are files in `src/main/resources/base` that are explicitly needed in s3 encoding in the abas ERP client, such as FOPs.

6. Project components

You can specify all components belonging to your app in the `app{ … }` section of the `build.gradle` file.

6.1. Example for GeoLocation demo app

This is an example how this look like for our Geolocation (G30LO) demo app.

```
esdk {
    app {
        name = "g30l0"
        vendorId = "ag"
        appId = "g30l0"
        shared = false
        workdirs = [ "ow1" ]
        infosystems = [ "IS.OW1.G30L0" ]
        tables = [ "Kunde" ]
        screens = [:]
        data = []
        enums = []
        namedTypes = []
        languages = "DA"
        essentialsVersions = ["2018r4-2018r4"]
    }
    ...
}
```

You can find the complete app source at GitHub: G30L0

6.2. All supported project components

```
esdk {
    app {
        name = "" ①
        vendorId = "" ②
        appId = "" ③
        appIdForTables = true ④
        shared = false ⑤
        infosystems = [""] ⑥
        tables = [""] ⑦
        data = [""] ⑧
        keys = [""] ⑨
        enums = [""] ⑩
        namedTypes = [""] ⑪
        screens = [""] ⑫
        advancedScreens = [""] ⑬
        essentialsVersions = [""] ⑭
        preconditions = [""] ⑮
        languages = "" ⑯
        workdirs = [""] ⑰
    }
    ...
}
```

① The name for your ESDK App^

② Your vendor ID

③ Your App ID^

④ Whether to use the App ID as prefix in additional variables in variable tables

⑤ Whether your JFOP server app uses parentDelegation

⑥ Infosystems belonging to the app

⑦ Standard and additional tables

⑧ Data files to be imported during the installation

⑨ Keys

⑩ Enumerations

⑪ Named Types

⑫ Customized standard and additional database screens

⑬ Customized standard database screens to override screen completely

⑭ abas ERP version range(s) your ESDK App^ is compatible to

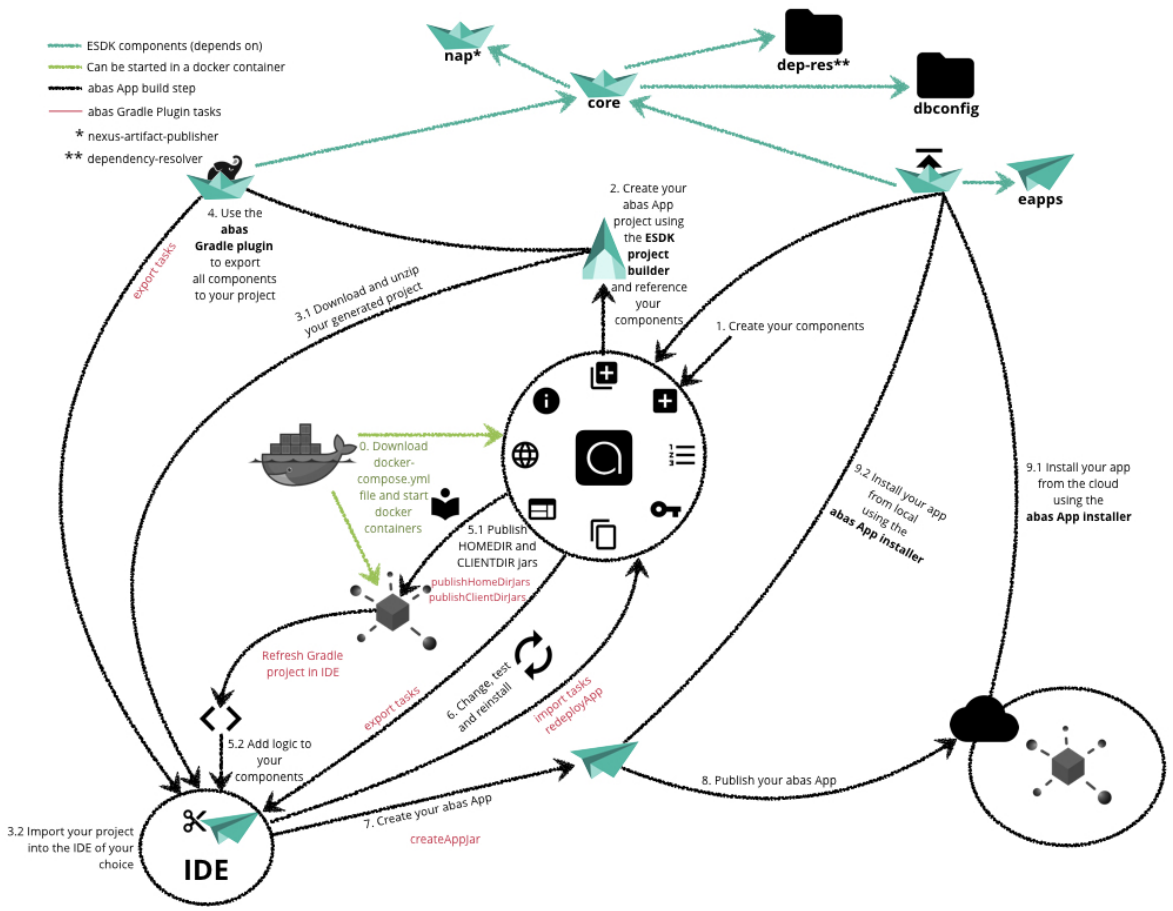
⑮ Preconditions to be met prior to installation

⑯ Languages

⑰ Workdirs

7. Task overview

The following picture gives an overview about the most relevant ESDK^ tasks.



The tasks are grouped in **basic**, **help and support**, as well as **professional** ESDK^ tasks.

The basic tasks are a set of selected combined tasks to simplify the app development. Use the professional tasks if you want to make specific, selective changes. The help and support tasks give information about the app project and its configuration.

8. abas Basic tasks

8.1. Checking Development and Installation Preconditions

The `checkPreconditions` Gradle task prints information about the local development environment and checks the preconditions for exporting components from and installing to an ESDK App^ in an abas ERP client.

First, all connection settings are tested:

- Is the abas ERP client accessible via EDP and has a version supported by ESDK
- Is the abas ERP client accessible via SSH
- Is the Nexus Artifact Server reachable
- Is the abas environment set
- Are the user rights sufficient

You can also configure additional checks for your ESDK App^:

- Check for an abas ERP version range
- Check for specific working directories

Execute `gradlew checkPreconditions --info` to get more detailed information.

8.2. Publishing all JARs

The `publishAllJars` task executes both, `publishHomeDirJars` and `publishClientDirJars` to the Nexus server to get you ready for development.

8.3. Exporting all Components

The `exportAll` task runs all export tasks so that all Project components are exported at once.

For more information about the dependent tasks read each task's documentation.

The export tasks run by `exportAll` are:

- exportMetaData
- exportData
- exportDictionaries
- exportEnums
- exportIS
- exportKeys
- exportNamedTypes
- exportScreen
- exportVartab

8.4. Installing the ESDK App

The task `fullInstall` is the main installation routine for installing an ESDK App^ using the ESDK Gradle Plugin.

8.4.1. Setup

Before the installation can be started, there are a few prerequisites.

The abas ERP client needs to be running and accessible via both SSH and EDP. Further the Sonatype

Nexus Artifact server needs to be running and username and password of a user with read and write rights need to be available.

If the abas ERP client and/or the Sonatype Nexus Artifact server are supposed to run in a Docker container, the Docker container(s) must be started and running before the ESDK App^ installation is started.

In every ESDK App^ a file named `gradle.properties` needs to be created right in the root folder of the project. In this file all properties regarding credentials and host names or ports must be defined.

You can use the shell script `initGradleProperties.sh` to generate that file from `gradle.properties.template`, both shipped with the project generated by the Project Initializer or the Project Builder. See also Configure your Gradle project settings.

gradle.properties

```
ABAS_HOMEDIR=/abas/s3 ①
ABAS_CLIENTDIR=/abas/erp ②
ABAS_CLIENTID=erp ③

EDP_CLIENT=/abas/erp ④
EDP_USER= ⑤
EDP_PASSWORD=sy ⑥
EDP_HOST=<abas IP address> ⑦
EDP_PORT=6550 ⑧

NEXUS_HOST=<nexus IP address> ⑨
NEXUS_PORT=8081 ⑩
NEXUS_URL=https://your-custom-url ⑪
NEXUS_NAME=abas-essentials-libs ⑫
NEXUS_USER_NAME=admin ⑬
NEXUS_PASSWORD=admin123 ⑭
NEXUS_VERSION=2 ⑮

SSH_HOST=<abas IP address> ⑯
SSH_PORT=22 ⑰
SSH_USER=erp ⑱
SSH_PASSWORD=none ⑲
SSH_KEY= ⑳

installType=SSH

version=0.0.1
```

① Absolute path to `$HOMEDIR` in the abas ERP client

② Absolute path to `$MANDANTDIR` in the abas ERP client

③ Name of abas ERP client as in the `mandantdir.env`

④ The EDP client (in most cases same as ABAS_CLIENTDIR)

⑤ When Linux logins are used for authentication in abas ERP, the username has to be specified here

⑥ Password used for authentication in abas ERP

⑦ Host name or IP address of the server abas ERP is running on

⑧ EDP port (if it is not reconfigured, `6550` is the default port for EDP)

⑨ Host name or IP address of the server Sonatype Nexus is running on

⑩ Port the Sonatype Nexus Artifact server communicates on

⑪ Alternative to specifying `NEXUS_HOST` and `NEXUS_PORT`. If you are using a self-hosted Sonatype Nexus Artifact server you can specify its base URL with this property. If you use this feature, you have to update your build.gradle file manually.
Please refer to Nexus.

⑫ Name of the repository the Java dependencies will be stored in on Sonatype Nexus

⑬ Username to authenticate to Sonatype Nexus (`admin` is the default username after Sonatype Nexus is installed in a Docker container)

⑭ Password to authenticate to Sonatype Nexus (`admin123` is the default password after Sonatype Nexus is installed in a Docker container)

⑮ Version of Sonatype Nexus (2 is the default value, optionally you can configure to use Sonatype Nexus Version 3)

⑯ Host name or IP address of the server abas ERP is running on

⑰ ssh port (if it is not reconfigured 22 is the default port for ssh)

⑱ Needs to be a Linux user known to abas ERP (usually `s3` or the login for a specific client of the abas ERP instance)

⑲ Password required to sign in via ssh to the server abas ERP is running on using the specified user (it can't be empty — if no password is assigned just put any string here)

⑳ Instead of a password, an ssh key file can be used for authentication. The path to the file needs to be specified here.

Installation type, only `SSH` and `LOCAL` are valid (usually one wants to use SSH)

Mandatory version of the ESDK App^ (must be a valid semantic version number)

Supported login mechanisms for SSH are public key and password based logins.

8.4.2. Full Install

Once the `$HOMEDIR/java/lib` jars are uploaded successfully to an available Sonatype Nexus Artifact server, the task `fullInstall` can be run.

This task installs the ESDK App^ in the abas client specified during setup. The task `fullInstall` consists of (and therefore executes) the following tasks in the mentioned order:

- processAppResources
- `processResources`
- checkPreconditions
- importEnums
- importNamedTypes
- reorgNamedTypes
- installVartabs
- varreorg
- importEnumsAfterVarreorg
- enumReorgAfterVarreorg
- installBaseFiles
- createWorkdirs
- installIS
- importKeys
- keyReorg
- installFopTxt
- installAjo
- publishClientDirJars
- publishHomeDirJars
- createJfopServerApp
- installJfopServerApp
- redeployJfopServerApp
- importMetaData
- importData
- installScreens
- importDictionaries
- installLog
- installMandantdirEnv

After running `./gradlew fullInstall` your app will be installed in the specified client and ready to use as configured by you.

8.5. Synchronizing the abas Client with Local Changes

The task `syncCode` bundles the execution of several tasks to generate and publish your app's AJO classes.

These tasks (and, of course, their dependent tasks) are called: `installAjo`, `publishClientDirJars` and `redeployJfopServerApp`.

8.6. Creating the ESDK App JAR

The task `createEsdkAppJar` (deprecated name: `createAppJar`) creates the app JAR which can be installed in another abas ERP client.

You can find the generated app JAR under the build directory: `<your-project-dir>/build/libs/<appName>-<version>-standalone.jar`

In abas Tools the `build` directory is filtered out by default. Adapt the filter settings to display it.

8.7. Packing your ESDK App

You can pack your ESDK App Project^ into a ZIP file, to provide it to another person for building/testing.

You should use the `packEsdkApp` task in the **abas basic** task group or run `./gradlew packEsdkApp` to create this ZIP archive.

The ZIP archive can then be found in `<your-project-dir>/build/esdk-app/`.

Make sure your project's version is updated in the **`gradle.properties.template`** file. During development, you might configure the version in your `gradle.properties` file only. However, this file is only for local development and will not be packed into the ZIP file.

Use this step in a Continuous Integration Pipeline to automatically build your project.

If you want to permanently work with another person on the project, consider using a Git Server such as Github or Bitbucket.

8.8. Installing the ESDK App using the ESDK App Installer

The `installEsdkApp` task installs the ESDK App^ on the client using the ESDK App Installer^.

The ESDK App^ needs to be available. It can be built from the ESDK App Project^ using the task createEsdkAppJar. The task `installEsdkApp` then performs the installation process with the ESDK App Installer^, that is likely to be used on a customer's client.

This task depends on the task installEsdkAppInstaller which installs the ESDK App Installer^ on the client.

9. abas Help and Support tasks

9.1. Information about the App Project

The task `esdkProjectInfo` prints version information about the app, Gradle, abas ERP and the ESDK plugin, as well as the configuration data used to connect to abas ERP and Nexus. It also shows which Docker image is used (if this information can be discovered).

9.2. ESDK Version Information

The Gradle task `esdkVersion` displays the version of the ESDK^ plugin.

9.3. Submitting the ESDK App to Receive Support

To submit your ESDK App for support, you need to create a ZIP archive from your ESDK App Project^.

We ask you to use the task `packProjectForSupport` in the **help and support** task group or run `./gradlew packProjectForSupport` to create the ZIP archive.

The ZIP archive can then be found in `<your-project-dir>/build/esdk-app/`.

9.4. Print Docker Image Tags

The task `printDockerTags` will print all tags of the Docker images `abas/test` in `sdp.registry.abas.sh` that are applicable for the defined abas ERP version ranges of your project.

To execute the task you must provide the abas docker registry credentials.

```
./gradlew printDockerTags -PdockerUser=<username> -PdockerPassword=<password>
```

In order to execute this task and to use these Docker images you need to have access to the `sdp.registry.abas.sh` Docker Registry. See Log in to the abas Partner Docker Registry on how to achieve this.

9.5. Print Project Version

The `printVersion` task prints the current version of your project. The version of your project can be stored in either one of the following files:

- `settings.gradle`
- `gradle.properties`
- `build.gradle`

Regardless of which of these files you use to set the version, the `printVersion` task will print it to the console.

9.6. Convert Vartab Files

With ESDK version 1.0, the format of variable table definition files changed from `.schm` to `.json`.

To upgrade an app using ESDK versions < 1.0, the all `.schm` files in `src/main/resources/-vartab` have to be converted to the new format. The task `convertVartab` supports in performing this job.

```
./gradlew convertVartab
```

If your app contains additional variable tables, you need to provide their mapping to the task. The new variable table format does not reference additional variable tables by number but by classname and resolves the actual number only on import. Therefore, no file manipulation is needed for variable table files prior to installation anymore.

An example: If your app contains the additional variable tables with classnames `Replacement` and `TestDb` you need to first find out, what number they were exported with. Go to their `.schm` files:

Replacement.schm

```
database master Replacement {
    number 19
    classname Replacement
...
```

TestDb.schm

```
database master TestDb {
    number 15
    classname TestDb
...
```

As shown above, Replacement was exported with database number 19 and TestDb with database

number 15.

You now need to reference these mappings when converting to the new format as follows:

```
./gradlew -PaddDbs=[15:TestDb,19:Replacement] convertVartab
```

Any reference field for these databases in other databases will then use the classname.

Customer.json

```
{
  "name": "Kunde",
  "number": 0,
  "isMasterData": true,
  "classname": "Customer",
  "groups": [{
    "name": "Kunde",
    "number": 1,
    "classname": "Customer",
    "headFields": [{
      "name": "testreference",
      "newName": "testreference",
      "type": "P{TestDb}:0", ①
      "description": "Reference additional database",
      "screenVisibility": "Editable",
      "showPriority": "A",
      "changePriority": "A"
    }]
  }]
}
```

① reference to additional database `TestDb`

10. abas Professional tasks

The professional tasks do all the detailed work that is necessary to get the app up and running. While they are bundled up in abas Basic tasks, you, as an app developer, need to know the details for development. The following sections describe all professional tasks belonging to the ESDK Gradle Plugin^ and provide examples on how to achieve the given aspect in your app.

10.1. Transfer files

10.1.1. installBaseFiles

The Gradle task `installBaseFiles` tries to copy all the files and subfolders present in the `base` folder of your resources into the target abas client directory.

You can use the base folder to store configuration files, FOPs, and other types of files which are not affected by the other tasks of the ESDK Gradle Plugin.

Configuration file

You can add a file named `.config` (case sensitive) at the root of the base and/or in any of the sub-folders. In this file, you can specify a list of files (present in the same folder the `.config` file is) with special things you would like to do to the files. The options supported are

- `ro` : Make the file read only.
- `rw` : Make the file readable & writable by everybody.
- `x` : Make the file executable.
- `crlf` : Apply `crlf-`, which will remove MS Windows carriage return character from each end of line. ONLY do this on text files, NEVER on binaries.
- `nooverwrite` : This will tell the program to not overwrite the file if it's already present in the destination folder.

The option values must be separated by a comma.

Example:

```
config.properties=crlf,ro
```

This will make the file `config.properties` read only and will remove the carriage return character from every line of that file.

There is no export task for baseFiles, as they are all individual and need to be managed from the project itself.

10.2. Enumerations

Enumerations and named types are referenced in variable tables, screens and Infosystems by their identity number. This identity number can change upon installation depending on the client the ESDK App^ is being installed in.

To make enumeration and named type references work an enumeration/named type's original identity number is mapped to the new identity number it got during installation.During the installation process all dependent object's import files are adapted to include the new instead of the old identity number.

In order for this mapping and replacement to work, it is very important that dependent objects - variable tables, screens and Infosystems - are exported at the same time from the same client as enumerations and named types.

This ensures that the identity numbers exported for the enumerations and named types matches the ones used in the references in variable tables, screens and Infosystems.

When experiencing problems with Enumeration/Named Type references therefore check if the identity numbers from the XML file(s) of the Enumerations/Named Types match the ones in the variable table, screen and Infosystem export files.

10.2.1. exportEnums

Using the `exportEnums` task it is possible to export existing enumerations from the client specified in the `gradle.properties` file.

To export an existing enumeration you need to enter the enumeration's classname in an `enums` list in the app section of the `build.gradle` as following:

```
esdk {
    app {
        ...
        enums = [ "ClientUsage", "AbcPriority" ]
    }
}
```

We recommend using a unique classname for your enumeration. This can best be assured by using your App ID^ as namespace.

E.g. for an App ID^ spare and an enumeration with search word FORMAT you could use `SpareFormat` as classname.

To export all enumerations specified in the `build.gradle` file to a file named `enums.xml` in `src/main/resources/enums`, run:

```
./gradlew exportEnums
```

Advanced enumerations

The task `exportEnums` only supports enums with enumeration elements of type `ValueSet:Identifier (109:1)` and `ValueSet:ValueSetIdentifier (109:2)` by default.

Enumerations using other types of enumeration elements can also be exported, but you need to specify the according dependent selection. To do so, configure an `enums.json` file in `src/main/resources/enums` with dependent selections as needed. E.g. for a customer-dependent selection:

enums.json

```
{
  "dependents" : [
    {
      "dbnr" : 0,
      "grnr" : 1,
      "headfields" : ["id", "guid", "nummer", "such", "name"],
      "tablefields" : []
    }
  ]
}
```

For more information on data export and dependent selections refer to exportData.

When using references to tables of variables created during the installation process, refer to importEnumsAfterVarreorg.

10.2.2. importEnums

The Gradle task `importEnums` imports all XML files in the directory `src/main/resources/enums` of an ESDK App^.

Normally only one import file is needed, containing all necessary enumerations.

To identify an already existing enumeration, every enumeration exported with the `exportEnums` task gets a `guid` (if it does not already have one) — a 38 digit number uniquely identifying the specific object among all other objects (regardless which database or group). The `importEnums` task then creates a new enumeration with that `guid` and updates exactly this if changes are necessary on further invocations.

To import all enumerations in `src/main/resources/enums` run:

```
./gradlew importEnums
```

10.2.3. importEnumsAfterVarreorg

The Gradle task `importEnumsAfterVarreorg` imports all XML files in the directory `src/main/resources/enumsAfterVarreorg` of an ESDK App^.

This task works similar to importEnums, but is used to import enumerations depending on a customized table of variables. It is called after the table of variables was updated with your extensions, and the reorganization completed.

To import all enumerations in `src/main/resources/enumsAfterVarreorg` run:

```
./gradlew importEnumsAfterVarreorg
```

10.2.4. enumReorg

After importing enumerations a reorganization is needed to be able to use the newly imported or updated enumerations.

To execute the enumerations reorganization run:

```
./gradlew enumReorg
```

Both `namedTypesReorg` and `enumReorg` run the same reorganization command on the client. Therefore, if your project consists of both, enumerations and named types, there is no need to run both reorganization tasks. After running both import tasks just run `namedTypesReorg`.

10.2.5. enumReorgAfterVarreorg

After importing enumerations a reorganization is needed to be able to use the newly imported or updated enumerations. The `enumReorgAfterVarreorg` task reorganizes the enumerations that were imported by `importEnumsAfterVarreorg` — just after a reorganization of the tables of variables has taken place.

To execute the enumerations reorganization run:

```
./gradlew enumReorgAfterVarreorg
```

10.3. Metadata

10.3.1. exportMetaData

The metadata export works analogously to exportData but exports to `src/main/resources/metaData` and expects JSON configuration files to be present in `src/main/resources/metaData` as well.

Metadata makes sense to use for data that configures the system prior to other data imports, e.g. for number ranges for data imported during importData.

You can limit the JSON files considered for export by using the 'metaData' property in the 'esdk' section of the `build.gradle` file of an ESDK App Project^ like this:

```
esdk {
    app {
        //...
        metaData = [ "mydata.json" ]
    }
}
```

This way, only the listed JSON files will be considered for data export.

10.3.2. importMetaData

The Gradle task `importMetaData` tries to import all XML files in the directory `src/main/resources/metaData` of an ESDK App^.

It uses the same import logic as importData and is meant for importing data objects that configure the system and need to be present for before importData runs, e.g. number ranges for objects imported by importData.

The task `importMetaData` therefore always runs before importData.

To import all XML files in `src/main/resources/metaData` of an ESDK App Project^ run:

```
./gradlew importMetaData
```

10.4. Data

10.4.1. exportData

General Description

One or more JSON formatted files can be added to `src/main/resources/data` to describe what data should be exported.

A JSON export data specification needs to contain at least one entry for `roots`. Entries in `roots` define the objects that actually should be exported, as opposed to `dependents`, which specify what parts of referenced objects should be exported. This means you need to add a `dependents` entry for every field specified in your `roots` entry referencing a different database.

You can limit the JSON files considered for export by using the 'data' property in the 'esdk' section of the `build.gradle` file of an ESDK App Project^ like this:

```
esdk {
    app {
        //...
        data = [ "mydata.json" ]
    }
}
```

This way, only the listed JSON files will be considered for data export.

An example for an export specification that contains only one `roots` entry looks as follows:

```
{
  "roots" : [
    {
      "dbnr" : 0,
      "grnr" : 1,
      "headfields" : [ "id", "guid", "such", "name" ],
      "tablefields" : [],
      "criteria" : "id=(167,0,0)"
    }
  ]
}
```

Running the `exportData` task will check if there are undefined dependencies such as reference fields. E.g. for the following JSON export data description the `exportData` task will fail as it contains the reference field `staat`.

```
{
  "roots" : [
    {
      "dbnr" : 0,
      "grnr" : 1,
      "headfields" : [ "id", "guid", "such", "name", "staat" ],
      "tablefields" : [],
      "criteria" : "id=(167,0,0)"
    }
  ]
}
```

```
./gradlew exportData

 You have to give a DependentSelection for 97:1
 To define it in your 'src/main/resources/enums/enums.json' data export file,
add:
    "dependents" : [
        {
            "dbnr" : 97
            "grnr" : 1
            "headfields" : [ "id", "guid", such", "name" ]
            "tablefields" : []
        }
    ]
```

As indicated above, in case of exporting with reference fields, a `dependents` entry is needed to define what to export from the object the reference field points to, e.g.:

```
{
  "roots" : [
    {
      "dbnr" : 0,
      "grnr" : 1,
      "headfields" : [ "id", "guid", "such", "name", "staat" ],
      "tablefields" : [],
      "criteria" : "id=(167,0,0)"
    }
  ],
  "dependents" : [
    {
      "dbnr" : 97,
      "grnr" : 1,
      "headfields" : [ "id", "guid", "such", "name" ],
      "tablefields" : []
    }
  ]
}
```

Configuration Options

The table below shows all possible configuration options and their purpose.

Table 1. Configuration Options

Option	Purpose	Optional	Possible Values	Default Value
`dbnr`	Number of the database to export from.	required	`0`	-

Option	Purpose	Optional	Possible Values	Default Value
`grnr`	Number of the group to export from.	required	`1`	-
`headfields`	List of fields to export found in the head of the object.	required (must at least contain field `id` and the field(s) used as `identifiers`)	`[ "id", "guid", "name" ]`	-
`tablefields`	List of fields to export found in the table of the object.	optional (if used it must at least contain field `zid` and any field used as `rowIdentifiers`)	`[ "zid", "elex", "anzahl" ]`	-
`criteria`	Only for roots: abas ERP selection string used to limit the objects added to the XML file on export.	optional	`such==DEMO`	`""`
`identifiers`	Specify identifiers to determine if an object is already there or needs to be created anew. For more information see Object Identification.	optional	`[ "field1", "field2" ]`	`[ "guid" ]`
~~`identifier`~~	**Deprecated, use `identifiers` instead.** Specify identifying field to determine if an object is already there or needs to be created anew. For more information see Object Identification.	optional	`field`	`guid`

Option	Purpose	Optional	Possible Values	Default Value
`importonce`	Import object on every app installation or only once. For more information see Configuring the Import Behaviour.	optional	`true`, `false`	`false`
`standard`	Objects marked as "standard" are assumed to be "already there" and not imported. For more information see References to Objects created during the Installation.	optional	`true`, `false`	`false`
`importmode`	Whether to fail or not on importing fields whose values cannot be set. For more information see Configuring the Import Behaviour.	optional	`tolerant`, `normal`	`normal`
`rowimportmode`	Start import of table rows with a cleaned table, simply add all new rows after the existing ones, or update existing rows. For more information see Configuring the Import Behaviour.	optional	`clearBeforeImport`, `alwaysAdd`, `updateMatching`	`clearBeforeImport`

Option	Purpose	Optional	Possible Values	Default Value
`rowIdentifiers`	Table fields used to identify table rows when `rowimportmode` `updateMatching` is used. For more information see Configuring the Import Behaviour.	optional	`["elex", "such", "name"]`	`[]`

Object Identification

Each selection, no matter if root or dependent, needs to have at least `id` and the field used as identifier as head fields and, if any table fields are defined, `zid` must be one of them.

If the `guid` field is not filled in the object that is exported, it will be filled with a generated guid in the export file (not in the object that is exported).

Customized Object Identification

The identifying fields (`identifiers`) are used to determine if an object already exist and has to be updated, or if it does not exist and needs to be created. If multiple identifying fields are specified, all of their field values have to match to identify the object as to be updated.

By default objects are identified by `guid`. GUID stands for Globally Unique Identifier and is represented by 32 random hexadecimal digits.

Currently, the `guid` field present in every data object is not automatically filled. Therefore, when an object is exported through the data export, and the `guid` field is not filled, a GUID is generated and entered as field value for field `guid` in the exported XML file.

A GUID is only generated if the field `guid` is exported and is not already filled with a different value. The `guid` field does not have to be an identifying field for the value to be generated.

During import the `guid` field will only be set if the object is created anew, since the `guid` field is only writable during object creation.

Object identification can be customized by using the `identifiers` option:

```
{
  "roots" : [
    {
      "dbnr" : 0,
      "grnr" : 1,
      "identifiers" : [ "such" ],
      "headfields" : [ "id", "guid", "such", "name", "staat" ],
      "tablefields" : [],
      "criteria" : "id=(167,0,0)"
    }
  ],
  "dependents" : [
    {
      "dbnr" : 97,
      "grnr" : 1,
      "identifiers" : [ "nummer" ],
      "headfields" : [ "id", "nummer", "guid", "such", "name" ],
      "tablefields" : []
    }
  ]
}
```

The `identifier` then shows up as an attribute to each `RecordSet` tag:

```
<RecordSet tableNumber="0:1" tableName="Kunde:Kunde" identifier="such" standard
="false">
```

Referencing Standard Objects

Standard objects are objects that are part of the abas ERP standard delivery. They already exist in the targeted installation and should be marked as such by adding the `standard` option to the selection:

```
{
  "roots" : [
    {
      "dbnr" : 88,
      "grnr" : 3,
      "identifiers" : [ "such" ],
      "headfields" : [ "id", "such", "name",  "aktiv", "kanal", "arb", "laytyp",
"kontext", "layname", "genlayout", "genlayname" ],
      "tablefields" : [],
      "criteria" : "such==ISJASOW1AUO"
    }
  ],
  "dependents" : [
    {
      "dbnr" : 65,
      "grnr" : 1,
      "standard" : true,
      "identifiers" : [ "nummer" ],
      "headfields" : [ "id", "nummer" ],
      "tablefields" : []
    },
    {
      "dbnr" : 88,
      "grnr" : 6,
      "standard" : true,
      "identifiers" : [ "such" ],
      "headfields" : [ "id", "such" ],
      "tablefields" : []
    }
  ]
}
```

The `standard` option then shows up as an attribute to each `RecordSet` tag:

```
<RecordSet tableNumber="65:1" tableName="Infosystem:Infosystem" identifier=
"nummer" standard="true">
```

References to Objects created during the Installation

For data objects with dependencies on objects created during the installation process — such as Infosystems or additional tables of variables — a customized identifier combined with the `standard` option can be used to fill these fields during data import.

For example a call parameter with an additional table of variables as source context and an Infosystem as target context could be configured for export as follows:

```
{
  "roots" : [
    {
      "dbnr" : 87,
      "grnr" : 9,
      "headfields" : [ "id", "guid", "such", "zielobj", "kontexttyp", "aufrktxt"
],
      "tablefields" : [ "zid", "zielaktion", "zielvar", "aufrwtyp", "aufrwert"
],
      "criteria" : "nummer==100268"
    }
  ],
  "dependents" : [
    {
      "dbnr" : 12,
      "grnr" : 26,
      "identifiers" : [ "vgrtxt14" ],
      "standard" : true,
      "headfields" : [ "id", "vgrtxt14" ],
      "tablefields" : []
    },
    {
      "dbnr" : 65,
      "grnr" : 1,
      "standard" : true,
      "identifiers" : [ "classname" ],
      "headfields" : [ "id", "classname" ],
      "tablefields" : []
    }
  ]
}
```

Configuring the Import Behaviour

There are three additional configuration options.

The first one is `importonce`, with the possible values `false` (default) or `true`.
Setting this option to `true` defines that objects are only imported once, and that they are not updated if already present.

The second one is `importmode`, with allowed values `normal` (default) or `tolerant`.
It is used to control the handling of password and readonly fields:

- In `normal` mode, password and readonly fields are exported (an empty field will be exported) but will fail on import.
- In `tolerant` mode, password, empty and readonly fields are ignored during export and import. In this case a warning will be displayed, listing the ignored field names.

The third option is `rowimportmode`, with the supported values being `clearBeforeImport` (default), `alwaysAdd` or `updateMatching`.
It specifies how object rows are handled when importing object data:

- `clearBeforeImport`: All object rows are deleted on import — even if the import XML file doesn't contain any rows to import.
- `alwaysAdd`: No rows are deleted. Rows specified in the import file are appended on import.
- `updateMatching`: No rows are deleted. Rows specified in the import file are updated if they match existing rows, and appended otherwise.

For reasons of backwards compatibility, the default value for `rowimportmode` is `clearBeforeImport`. Set it to `alwaysAdd` if you want existing rows to be left untouched.

For the "permissions" table "Erlaubnis:Erlaubnis" (85:1) no rows are ever added, regardless of the `rowimportmode`. Still, it is advisable to use `rowimportmode`: `alwaysAdd` to avoid warnings regarding rows not being able to delete.

Configuration example

Example JSON Snippet for Import Behaviour Options

```
{
  "roots" : [
    {
      "dbnr" : 2,
      "grnr" : 1,
      "identifiers" : [ "such", "name" ],
      "importonce" : true,
      "importmode" : "tolerant",
      "headfields" : [ "id", "guid", "such", "name" ],
      "tablefields" : [ "zid", "elem", "elanzahl" ],
      "criteria" : "such==DEMO",
      "rowimportmode" : "alwaysAdd"
    }
  ]
}
```

The `importOnce`, `importMode` and `rowImportMode` options then show up as attributes in the `RecordSet` tag:

Resulting XML Snippet as generated by `exportData`

```
<RecordSet tableNumber="2:1" tableName="Teil:Artikel" importOnce="true"
importMode="tolerant" rowImportMode="alwaysAdd">
```

updateMatching example

If `rowImportMode` is set to `updateMatching`, you can define which table fields will be used to identify the matching rows with the `rowIdentifiers` attribute.

The identifying fields cannot be updated during the import.

Example using `rowimportmode: "updateMatching"`

```
{
  "roots" : [
    {
     "dbnr" : 2,
     "grnr" : 1,
     "importonce" : true,
     "importmode" : "normal",
     "headfields" : [ "id", "guid", "such", "name" ],
     "tablefields" : [ "zid", "elex", "such", "name", "anzahl" ],
     "rowimportmode" : "updateMatching",
     "rowIdentifiers" : ["elex", "such", "name"]
    }
  ],
  "dependents" : [
    {
    "dbnr" : 7,
    "grnr" : 0,
    "standard" : false,
    "identifier" : "guid",
    "headfields" : [ "id", "guid", "such", "name" ],
    "tablefields" : []
    }
  ]
}
```

The `rowIdentifiers` then show up as attributes in the `RecordSet` tag:

```
<RecordSet tableNumber="2:1" tableName="Teil:Artikel" standard="false"
rowIdentifiers="elex,such,name" importOnce="true" importMode="normal"
rowImportMode="updateMatching">
```

The row update behavior is as follows:

- If a row already exists and can be identified uniquely: this row is updated.
- If `importMode` is `tolerant`: the first matching row is updated.
- If `importMode` is `normal` and there is more than one matching row: the import will fail.
- If no matching row is found: a new row is appended to the end of the table.

If a row is updated, only values for fields provided in the xml source file will be updated. Values of other fields in this row won't be changed.

Call Parameter Lists

Currently, it is not possible to fully automatically export call parameter list objects. To later successfully import call parameter lists, they need to be in a `<RecordSet>` with attribute `tableNumber` set to `87:21` and the `<RecordSet>` element needs to contain the `<Head>` and `<Row>` elements.

Also, the import needs the reference field `aufruf` to be present (for standard call parameter lists you can also reference the objects in `aufruf` as standard objects).

Here is an example of a `data.xml` file containing call parameter list objects that can be imported successfully:

data.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ABASData>
    <RecordSet tableNumber="87:21" tableName=
"Datenexport:APListe" identifier="nummer" standard="false"
importOnce="false" importMode="normal">
        <Record id="(1595,87,0)">
            <Head>
                <Field name="id" abasType="ID87">
(1595,87,0)</Field>
                <Field name="nummer" abasType="NK87">
60000</Field>
            </Head>
            <Row number="1">
                <Field name="zid" abasType="IDZ87:23"
>(1596,87,0)</Field>
                <Field name="aufruf" abasType="ID87:9"
>(1585,87,0)</Field>
                <Field name="aktiv" abasType="B1">1</Field>
            </Row>
            <Row number="2">
                <Field name="zid" abasType="IDZ87:23"
>(1597,87,0)</Field>
                <Field name="aufruf" abasType="ID87:9"
>(1588,87,0)</Field>
                <Field name="aktiv" abasType="B1">0</Field>
            </Row>
        </Record>
    </RecordSet>
    <RecordSet tableNumber="87:21" tableName=
"Datenexport:APListe" identifier="nummer" standard="false"
importOnce="false" importMode="normal">
        <Record id="(1585,87,0)">
            <Head>
```

```
                <Field name="id" abasType="ID2">
(1585,87,0)</Field>
                <Field name="nummer" abasType="NK2">
60002</Field>
            </Head>
            <Row number="1">
                <Field name="zid" abasType="IDZ87:23"
>(1586,87,0)</Field>
                <Field name="aufruf" abasType="ID87:9"
>(1408,87,0)</Field>
                <Field name="aktiv" abasType="B1">0</Field>
            </Row>
        </Record>
    </RecordSet>
    <RecordSet tableNumber="87:21" tableName=
"Datenexport:APListe" identifier="nummer" standard="true"
importOnce="false" importMode="normal">
        <Record id="(1588,87,0)">
            <Head>
                <Field name="id" abasType="ID2">
(1588,87,0)</Field>
                <Field name="nummer" abasType="NK2">
60003</Field>
            </Head>
        </Record>
    </RecordSet>
    <RecordSet tableNumber="87:9" tableName=
"Datenexport:Aufrufparameter" identifier="nummer" standard="
true" importOnce="false" importMode="normal">
        <Record id="(1408,87,0)">
            <Head>
                <Field name="id" abasType="ID2">
(1408,87,0)</Field>
                <Field name="nummer" abasType="NK2">
52686</Field>
            </Head>
        </Record>
    </RecordSet>
</ABASData>
```

10.4.2. importData

The Gradle task `importData` tries to import all XML files in the directory `src/main/resources/-data` of an ESDK App^.

Each XML import file needs to contain all object data required for the import. This includes all dependent objects (the objects' reference fields point to).

The importer determines the correct import order automatically. It considers all objects from all XML files in `src/main/resources/data` when determining the import order.

We recommend using a unique identifier for your data objects wherever possible. This can best be assured by using your App ID^ as namespace.

To import all XML files in `src/main/resources/data` of an ESDK App Project^ run:

```
./gradlew importData
```

Call Parameter Lists

To identify call parameter lists uniquely, more than a single attribute is necessary. In this case, provide all field names in the `RecordSet` XML node's `identifiers` (not `identifier`!) attribute, separated by comma.

Example: `identifiers="aufrktxt,kontexttyp,auftab,buttonfeld,lieferumfang,nurkopf"`.

Then, in the `Record.Head` node, list `Field` nodes for these identifier fields with values that together identify the object uniquely.

10.5. Infosystems

10.5.1. exportIS

The task `exportIS` exports all infosystems that are referenced in the `infosystems` property in the esdk closure in your `build.gradle` file:

```
esdk {
    app {
        ...
        infosystems = ["IS.OW1.TESTINFO", "IS.OW1.REPLACEMENTCATALOGUE"]
    }
}
```

The infosystem(s) get exported to the `isrein` folder in your abas ERP client using `infosys_export.sh`. They are then transferred to the `IS` folder in the source folder `src/main/resources` of your ESDK App Project^.

The main infosystem file is converted from S3 charset to UTF-8 in this process.

10.5.2. installIS

The task `installIS` installs all infosystems within the `IS` folder in the source folder `src/main/resources` of your ESDK App Project^.

The infosystem(s) get transferred to your client and installed using the shell scripts `infosysimport.sh` — if the infosystem does not exist yet — or `infosys_upgrade.sh` if the infosystem does exist and needs to be updated.

The main infosystem file is converted from UTF-8 to S3 charset in this process.

The identity number used when exporting will be removed during the import, so the infosystem can be imported even if the identity number is already in use in the target client.

10.6. Keys

"Key" is the abas term for what is more commonly known as a "database index".

10.6.1. exportKeys

Using the `exportKeys` task it is possible to export existing keys from the client specified in the `gradle.properties` file.

To export an existing key you need to enter the key's `bname` in a `keys` list in the `app` section of the `build.gradle` as following:

```
esdk {
    app {
        // ...
        keys = [ "test_key_1", "test_key_2" ]
    }
}
```

To export all keys specified in the `build.gradle` file to a file named `keys.xml` in `src/main/resources/keys` run:

```
./gradlew exportKeys
```

10.6.2. importKeys

The Gradle task `importKeys` imports all XML files in the directory `src/main/resources/keys` of an ESDK App^.

Normally only one import file is needed, containing all necessary keys.

To identify an already existing key, every key exported with the `exportKeys` task gets a `guid` (if it does not already have one), a 38 digit number uniquely identifying the specific object among all other objects (regardless which database or group). The `importKeys` task then creates a new key with that

`guid` and updates exactly this if changes are necessary on further invocations.

We recommend using a unique key name (field `schlname`/`keyName`) for your key. This can best be assured by using your App ID^ as namespace. E.g. for an App ID^ `spare` and a key with search word `CUSTCONT` you could use `SpareCustomerContact` as key name.

If a similar key already exists in the target client a `TODO:` entry with the affected key is included in the installation log file.

To import all keys in `src/main/resources/keys` of an ESDK App Project^ run:

```
./gradlew importKeys
```

10.6.3. keyReorg

The Gradle task `keyReorg` reorganizes the key table, thus making the key available or updating it.

10.7. Named types

Enumerations and named types are referenced in variable tables, screens and Infosystems by their identity number. This identity number can change upon installation depending on the client the ESDK App^ is being installed in.

To make enumeration and named type references work an enumeration/named type's original identity number is mapped to the new identity number it got during installation.During the installation process all dependent object's import files are adapted to include the new instead of the old identity number.

In order for this mapping and replacement to work, it is very important that dependent objects - variable tables, screens and Infosystems - are exported at the same time from the same client as enumerations and named types.

This ensures that the identity numbers exported for the enumerations and named types matches the ones used in the references in variable tables, screens and Infosystems.

When experiencing problems with Enumeration/Named Type references therefore check if the identity numbers from the XML file(s) of the Enumerations/Named Types match the ones in the variable table, screen and Infosystem export files.

10.7.1. exportNamedTypes

By using the `exportNamedTypes` task it is possible to export existing named types from the client

specified in the `gradle.properties` file.

To export an existing named type you need to enter the named type's class name in the `namedTypes` list in the `app` section of the `build.gradle` as following:

```
esdk {
    app {
        ...
        namedTypes = [ "InstallationDate", "DisclaimerText" ]
    }
}
```

We recommend using a unique class name for your named types. This can best be assured by using your App ID^ as namespace. E.g. for an App ID^ `spare` and an named type with search word `INSTDATE` you could use `SpareInstallationDate` as class name.

To export all named types specified in the `build.gradle` file to a file named `namedTypes.xml` in `src/main/resources/namedTypes`, run:

```
./gradlew exportNamedTypes
```

10.7.2. importNamedTypes

The Gradle task `importNamedTypes` imports all XML files in the directory `src/main/resources/namedTypes` of an ESDK App^.

Normally only one import file is needed, containing all necessary named types.

To identify an already existing named type, every named type exported with the `exportNamedTypes` task gets a `guid` (if it does not already have one), a 38 digit number uniquely identifying the specific object among all other objects (regardless which database or group). The `importNamedTypes` task then creates a new named type with that `guid` and updates exactly this if changes are necessary on further invocations.

To import all named types in `src/main/resources/namedTypes` of an ESDK App Project^ run:

```
./gradlew importNamedTypes
```

10.7.3. namedTypesReorg

After importing named types, a reorganization is needed to be able to use the newly imported or updated named types.

To execute the named types reorganization run:

```
./gradlew namedTypesReorg
```

Both `namedTypesReorg` and `enumReorg` run the same reorganization command on the client. Therefore, if your project consists of both, enumerations and named types, there is no need to run both reorganization tasks. After running both import tasks just run `namedTypesReorg`.

10.8. Table of Variables (Vartab)

10.8.1. General

This section describes the import and export of tables of variables (schema of database groups).

New variable table format

Since ESDK version 1.0 a new json based variable table format is used. This documentation uses the new format.

For information on how to update from an ESDK version < 1.0 refer to Convert Vartab Files.

Shell Groups

If you export variables from a shell group, there will be a duplicate entry exported in the `.json` file — one for the shell group and one for the origin of the shell. To use the `.json` file for import please remove the entry for the shell group from the `.json` file manually.
For more information refer to Handling Shell Groups during export.

10.8.2. installVartab

In a file with the suffix `.json` in `src/main/resources/vartab` all changes for one table of variables can be defined. It is possible to add custom variables, as well as to update and delete them.

The `.json` file needs to contain all variables of a table of variables relevant for the ESDK App^ this `.json` file is for. There must be one `.json` file for each table of variables that needs changes for the ESDK App^.

All variables belonging to an ESDK App^ have to be prefixed with the `appId` in the following fashion:

App Variable Namespace

The `vendorId` and `appId` of an ESDK App^ are defined in the `build.gradle` of the app project. They are stated in the `esdk.app` section as following:

build.gradle

```
esdk {
    app {
        name="myApp"
        vendorId="vv"
        appId="aaaaa"
    }
}
```

A variable for an ESDK App^ must have the following format in the table of variables:

```
<vendorId>y<appId><variable_name> ①
```

① The `vendorId` follows this formatting convention: `[a-z][a-z0-9]`
Note that technically the `vendorId` is not needed. It is just intended to be used for a better visual clustering of "your" fields in the table of variables. Currently, when importing variables, the prefix `xx` is always used, regardless what `vendorId` is specified.

The `appId` follows this formatting convention: `[a-z0-9]{5}`
It is the primary identifier of your app and defines the namespace for all of its individual variables.

The actual variable name can have the following format: `[a-z0-9]{1,12}`
From abas ERP version 2019r4 on, names of individual variables are case-sensitive and can be up to 72 places long. They can have the following format: `[a-zA-Z0-9]{1,72}`

To simplify transforming exiting customizations into ESDK App^s, the App ID^ prefix for variables can be switched off.All custom variables (variables starting with y) will be considered part of the ESDK App^.

To switch off prefixing add the following option to your `build.gradle` file:

build.gradle

```
esdk {
    // ...
    appIdForTables = false
    // ...
}
```

Disabling prefixing the custom variables should only be done to turn exiting customizations into ESDK App^s. It is not necessary to disable the prefix, if the customization does not include unprefixed variables in an existing client.

We strongly advise against disabling the prefix if it is not necessary!

When using an ESDK App^ with the prefix disabled, make sure every developer working on the client(s) this app is installed in knows about it.The app needs to contain all custom variables of the variable tables it makes changes in. If it does not contain a variable and the app is installed again, the variable that was not contained will be removed during app installation.

This means any changes to the variable tables should be through the ESDK App Project^ and installation of the resulting ESDK App^.

Adding variables to tables of variables

In the `.json` file only the actual variable name (`<variable_name>` above) has to be inserted. The prefix `<vendorId>y<appId>` is generated by ESDK automatically.

The `.json` file needs to have the following format:

Kunde.json

```
{
  "name": "Kunde",
  "number": 0,
  "isMasterData": false,
  "classname": "Customer",
  "groups": [{
    "name": "Kunde",
    "number": 1,
    "classname": "Customer",
    "headFields": [{
      "name": "testhead",
      "newName": "testhead",
      "type": "GL40",
      "description": "Test field head edited",
      "screenVisibility": "Editable",
      "showPriority": "A",
      "changePriority": "A"
    }, {
      "name": "testref",
      "newName": "testref",
      "type": "P{TestDb}:2",
      "description": "Test reference field head",
      "screenVisibility": "Editable",
      "showPriority": "A",
      "changePriority": "A"
    }]
  }]
}
```

To define table variables follow this example of an additional database file:

TestDb.json

```
{
  "name": "TestDb",
  "number": 25,
  "isMasterData": false,
  "classname": "TestDb",
  "groups": [{
    "name": "TestStructure",
    "number": 0,
    "classname": "TestStructure",
    "headFields": [{
      "name": "testhead",
      "newName": "testhead",
      "type": "GL30",
      "description": "Test field head",
      "screenVisibility": "Editable",
      "showPriority": "A",
      "changePriority": "A"
    }],
    "tableFields": [{
      "name": "testtable",
      "newName": "testtable",
      "type": "GL30",
      "description": "Test field table",
      "screenVisibility": "Editable",
      "showPriority": "A",
      "changePriority": "A"
    }]
  }]
}
```

The value for `groupNumber` is `0` by default and is therefore left out in the second example.

Skip fields, especially buttons, need to have `skip` in the field definition:

```
{
  "name": "testbutton",
  "newName": "testbutton",
  "isSkip": true,
  "type": "BU3",
  "description": "Test Button",
  "screenVisibility": "ViewEditable",
  "showPriority": "A",
  "changePriority": "A"
}
```

To add an alias field for an app field use the following syntax:

```
[{
  "name": "testhead",
  "newName": "testhead",
  "type": "GL30",
  "description": "Test field head",
  "screenVisibility": "Editable",
  "showPriority": "A",
  "changePriority": "A"
}, {
  "name": "testhead",
  "newName": "testalias",
  "isAlias": true,
  "type": "GL30",
  "newType": "GL40",
  "description": "Test field head alias",
  "screenVisibility": "Editable",
  "showPriority": "A",
  "changePriority": "A"
}]
```

You can also include alias fields for non-app fields, e.g. an alias for field 'name' (english field name 'description') looks as follows:

```
{
  "name": "name",
  "newName": "namealias",
  "isAlias": true,
  "type": "GL45",
  "newType": "GL100",
  "description": "test for non-app alias",
  "screenVisibility": "Editable",
  "showPriority": "A",
  "changePriority": "A"
}
```

The original field is then not contained in the exported variable table description.

If the above examples are installed via the `installVartab` Gradle task, the variables are added to the according tables of variables if they don't already exist.

Edit existing variables in tables of variables

If a variable already exists, it is checked during `installVartab` whether there are any changes. Only if there are ones, the changes will be applied and result in one or more "VVAR" (prepared Vartab) records which will then require reorganization via `varreorg`.

For example, to change the `testhead` field in table of variables `TestDb` from type `GL30` to `GL40`

run `installVartab` on the following file:

TestDb.json

```
{
  "name": "TestDb",
  "number": 25,
  "isMasterData": false,
  "classname": "TestDb",
  "groups": [{
    "name": "TestStructure",
    "number": 0,
    "classname": "TestStructure",
    "headFields": [{
      "name": "testhead",
      "newName": "testhead",
      "type": "GL40",
      "description": "Test field head",
      "screenVisibility": "Editable",
      "showPriority": "A",
      "changePriority": "A"
    }],
    "tableFields": [{
      "name": "testtable",
      "newName": "testtable",
      "type": "GL30",
      "description": "Test field table",
      "screenVisibility": "Editable",
      "showPriority": "A",
      "changePriority": "A"
    }]
  }]
}
```

Delete variables from tables of variables

A `.json` file needs to contain all variables in that table of variables belonging to that ESDK App^.

Therefore, all variables in the App Variable Namespace that are not defined in the `.json` file for the table of variables will be deleted.

For example, to delete the field `testtable` from the table of variables `TestDb` run `installVartab` on the following file:

TestDb.json

```
{
  "name": "TestDb",
  "number": 25,
  "isMasterData": false,
  "classname": "TestDb",
  "groups": [{
    "name": "TestStructure",
    "number": 0,
    "classname": "",
    "headFields": [{
      "name": "testhead",
      "newName": "testhead",
      "type": "GL40",
      "description": "Test field head",
      "screenVisibility": "Editable",
      "showPriority": "A",
      "changePriority": "A"
    }]
  }]
}
```

Renaming variables with `installVartab` will lead to **data loss without certain precautions**. If you want to rename your variable in a productive system and don't want to lose your data, you need to do it in two iterations (two versions):

1. Add a new variable with the new name and write all data from the old variable to the new variable. At this point, data transfer from one field to the other is a manual step after app installation.
2. Delete the old variable.

Just replacing the variable will lead to the old variable being deleted and a new variable being created. The deleted variable will lose all data.

Alternatively, if the type is compatible, you can use an alias field and keep the old variable.

Installation order

The installation imports one `json` file at a time in alphanumerical order. By changing the `json` file's name after exporting it, you can influence the import order. This may be necessary to ensure all objects are imported in the correct order.

During import, custom vartab's numbers may be changed when the number given in the import `json` files is already taken. If this happens, the related references in all dependent files (enums, screens) have to be adapted to point to the new database number.
Since ESDK version 1.1.12, the processing order of these dependent files has been optimized, so that it is often no longer necessary to set the import order manually.

Supported fields in json variable table files

In the variable table file following fields are supported:

Object	Required Fields	Optional Fields
Database	`name`, `number`, `isMasterData`, `classname`, `groups`	
Group	`name`, `number`, `classname`	`headFields`, `tableFields`
Field	`name`, `type`, `description`	`label`, `heading`, `screenVisibility`, `showPriority`, `changePriority`, `isSkip`, `isAlias`, `newName`, `newType`

10.8.3. exportVartab

The `exportVartab` task exports all custom `tables` of variables as `.json` files, that are defined in the `app` section of the `build.gradle` file. `exportVartab` considers groups.

The groups can be specified using:

- Standard databases: the German group name or classname
- Custom databases: the classname

If no groups are specified all groups containing fields that belong to the ESDK App are exported.

build.gradle

```
esdk {
    app {
        name = "sparePartsCatalogueApp"
        vendorId = "ag"
        appId = "spare"
        shared = false
        infosystems = [ "IS.OW1.REPLACEMENTCATALOGUE" ]
        tables = [ "Replacement", "Teil:Artikel,Fertigungsmittel" ] ①
        data = [ "data.json" ]
    }
}
```

① Add all table of variables names to the array named `tables`. For multiple tables of variables just add Strings separated by comma, e.g. `[ "Replacement", "Teil:Artikel,Fertigungsmittel" ]`. To separate the database from the groups use a colon and add the groups separated by commas.
Note that for standard databases, currently only their German names are supported.

The exported `.json` files can be found in `src/main/resources/vartab` in a directory named as the class name of the table of variables that was exported.

Handling Shell Groups during export

Shell Groups, meaning groups that quote another group for head or table schema, cannot be edited. There are two different types of Shell Groups:

1. Shell groups which quote head and table: These shell groups cannot be edited at all and should not be exported or should be deleted from the `.json` file after exporting. Instead, the quoted groups should be considered for export.
2. Shell groups which quote head but not table or vice versa: It makes only sense to export these shell groups if editing the non-quoted part is intended.

Whenever a whole database is specified for export, e.g. `Verkauf`, all groups including the shell groups are exported. The non-editable shell groups need to be removed manually. Alternatively, each group can be specified for export individually, thereby excluding shell groups, e.g. `Verkauf:Verkauf,Position`.

10.8.4. varreorg

The `varrorg` task checks if tables of variables prepared for reorganization exist (so-called "VVAR" objects). If VVAR objects exist, a table of variables reorganization is executed. Otherwise, an appropriate message is displayed and the table of variables reorganization is skipped.

10.9. Screens

10.9.1. exportScreens

The `exportScreens` Gradle task exports screens from the abas ERP client into the ESDK App Project^.

Additional Database Screens and App Tabs in Standard Database Screens

To specify what screens to export, add a `screens` attribute to your `build.gradle`'s `app` section and specify the Database and Group class name of the table the screen is for, as well as the priorities you want to export:

build.gradle

```
esdk {
    app {
        screens=["Customer:Customer": ["A", "D"], "TestDb:TestStructure": ["A"]]
    }
}
```

For customized screens for standard databases you can also specify the screen number instead of Database and Group class names:

build.gradle

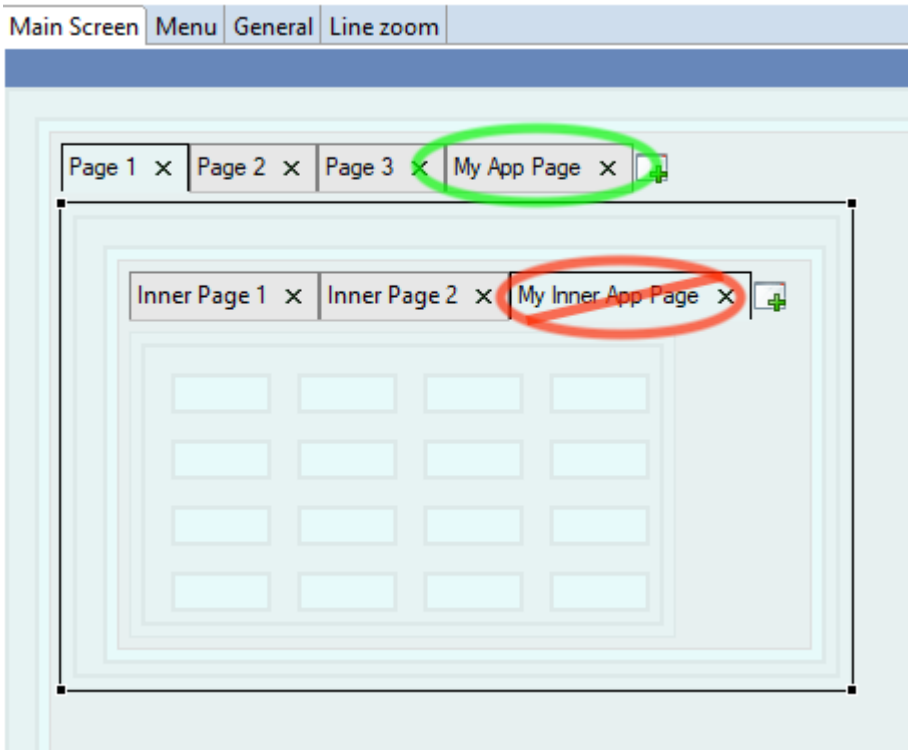
```
esdk {
    app {
        screens=["77": ["A"]]
    }
}
```

For customized standard database screens only the app tab(s) and menu buttons of your app will be exported.

Add your app tab(s) to the first tab container in the main screen or the first tab container in the line zoom, depending on whether you want to add head or table fields. Use the Screen Designer which is bundled with abas Tools to do so.

Only menu buttons containing your App ID^ in their field name will be exported.

You cannot add tabs to inner tab pages:



To identify your tab, set the key attribute to `<classname>$<appid>_<count>`, e.g.: `product$train_0`. If you are using abas Tools to add a tab to your screen, the key attribute will be generated as `<classname>$<count>`, e.g. `product$0`. So in this case you just have to add `<appid>_`.

If no tab container exists in the main screen or line zoom, add a tab container around the outermost layout in the screen. Put all existing content in the first tab and add your app tab with your fields after that. When installing, a tab container with a general tab will be added automatically around the outermost layout, and your app tab will be added after the general tab.

If no line zoom exists, add a line zoom and an app tab in that line zoom. When installing, a line zoom will be created and your app tab will be added to the tab container in that line zoom.

For customized additional database screens the complete screen is exported.

If there is no customized screen for your database, or no key identifier is set for your app for standard databases, the export will fail.

Overriding Standard Database Screens

If you want to override standard screens completely, you can use the `advancedScreens` attribute in the `app` section of your `build.gradle` file. Specify standard screens that should be overridden

completely as follows:

build.gradle

```
esdk {
    app {
        advancedScreens=["77": ["A"], "Customer:Customer": ["A", "D"]]
    }
}
```

Using the `advancedScreens` configuration might lead to the loss of other changes in the previously already customized screen (such as changes made by other apps or the customer himself).

10.9.2. installScreens

The `installScreens` Gradle task installs all customized screens that are part of the ESDK App^.

Additional Database Screens and App Tabs in Standard Database Screens

All screens to import must be present in `src/main/resources/dbscreens` in the `.screen` format.

Overriding Standard Database Screens

All advanced screens to import must be present in `src/main/resources/advancedDbScreens` in the `.screen` format.

10.9.3. upgradeScreens

The `upgradeScreens` Gradle task upgrades all customized screens that are part of the ESDK App^ to the format of the used ESDK version.

For using the `upgradeScreens` task the screens must be exported already.

10.10. Language support

10.10.1. exportDictionaries

The Gradle task `exportDictionaries` exports all screen and vartab texts for all infosystems and tables of variables in the ESDK App Project^, as well as the individual texts.

The screen texts will be exported into `msg.ma` files per infosystem / table of variables in either `src/main/resources/IS` or `src/main/resources/vartab`. The vartab texts will be exported into `msg.vt` files per table of variables in `src/main/resources/vartab`. Individual texts will be

exported into a `msg.ku` file in `src/main/resources/vartab`.

To export all screen and vartab texts run:

```
./gradlew exportDictionaries
```

The exported `msg.ma` and `msg.vt` files will include all languages specified in the app section of the `build.gradle` of the ESDK App Project^. The exported `msg.ku` file will include all languages. All texts that are already available in one of the specified languages will be included.

You can add all missing texts in all specified languages to the `msg.ma`, `msg.vt` and `msg.ku` files manually. These can then be imported using the `importDictionaries` task.

10.10.2. importDictionaries

The Gradle task `importDictionaries` imports all `msg.ma`, `msg.vt` and `msg.ku` files in the ESDK App Project^ into the dictionaries of the abas ERP client.

The `msg.ma` files containing translations are either found in `src/main/resources/IS` or `src/main/resources/vartab` as they either belong to an infosystem or to a table of variables. `msg.vt` and `msg.ku` files containing translations are found in `src/main/resources/vartab`.

The languages to import must be specified in the `app` section of the `build.gradle` of the ESDK App Project^ as following:

```
esdk {
    app {
        ...
        languages = "DA" ①
    }
}
```

① Specify all languages as found in the Configuration data object in the abas ERP client e.g. `D` for German and `A` for American English.

To import all dictionaries for all specified languages run:

```
./gradlew importDictionaries
```

To import only dictionaries of a given type for all specified languages run:

```
./gradlew importMsgMaDictionaries
```

or

```
./gradlew importMsgVtDictionaries
```

or

```
./gradlew importMsgKuDictionaries
```

Only languages that are installed in the abas ERP client will be considered.

10.11. Working directories

10.11.1. createWorkdirs

The Gradle task `createWorkdirs` creates working directories if missing and activates inactive ones needed by the ESDK App^ for the user who runs the installation.

All needed working directories can be defined in the `build.gradle`:

```
esdk {
    app {
        ...
        workdirs = [ "ow1", "owdir" ]
    }
}
```

After installing, it might be necessary to manually configure these working directories for additional users.

10.12. Generate AJO classes

10.12.1. installAjo

The Gradle task `installAjo` runs `ajo_install.sh` on the abas ERP client to generate AJO classes for all components that belong to the app project.

10.13. Change fop.txt

In an abas ERP client, the file `fop.txt` (located in the client's directory) contains all EventHandler registrations for database screens.

10.13.1. installFopTxt

The `installFopTxt` task adds all lines to `fop.txt` specified in the file `fop.json`.

The `fop.json` has to be added to your project here: `<your-project-dir>/src/main/resources/fop.json`. In this file, you can specify any entries you need to add to the `fop.txt` using the following JSON format:

fop.json Example for a Standard Database using AJO identifiers

```
[
  {
    "databaseName" : "(Sales)",
    "groupName" : "(PackingSlip)",
    "editorMode" : "neu",
    "event" : "maskende",
    "key" : "*",
    "field" : "*",
    "headOrTable" : "K",
    "isContinue" : "[C]",
    "handler" : "java:de.abas.example.YourClass@appid"
  }
]
```

Instead of specifying `databaseName` and `groupName` you can also use the screen number `screenNumber`. However, this is **not recommended for additional tables**, since additional tables are installed in the next free additional database. So the screen number changes depending on the client the app is installed in.

fop.json Example for a Standard Database using the screen number

```
[
  {
    "screenNumber": "35",
    "editorMode" : "*",
    "event" : "maskpruef",
    "key" : "*",
    "field" : "*",
    "headOrTable" : "K",
    "isContinue" : "[C]",
    "handler" : "java:de.abas.example.YourClass@appid"
  }
]
```

fop.json Example for an Additional Table

```
[
  {
    "databaseName" : "GermanDatabaseName",
    "groupName" : "GermanGroupName",
    "editorMode" : "neu",
    "event" : "maskende",
    "key" : "*",
    "field" : "*",
    "headOrTable" : "K",
    "isContinue" : "[C]",
    "handler" : "java:de.abas.example.YourClass@appid"
  }
]
```

If you want to add a `fop.txt` entry for every database and/or group, you can leave `databaseName` and/or `groupName` empty.

The `fop.txt` file is extended by the app's entries. A comment brace is factored around all `fop.txt` lines belonging to one app to identify an app's scope of lines, be able to add new lines for that app, update existing lines, and delete old lines automatically.

All `fop.txt` entries wrapped by the comment brace are subject to be changed by the ESDK App Installer^. Do not add additional lines to this section — they will be removed during the next ESDK App update. Also, lines in `fop.txt` may not be sorted because then the installer will not be able to recognize the related entries anymore.

`fop.txt` lines for an app that was installed with an ESDK version before 0.4.11 will not get updated or deleted automatically. These lines either have to be maintained manually or manually moved inside the comment brace for the app when installing the app with this or a newer version. The latter has only to be done once.

10.14. Configure logging

10.14.1. installLog

The `installLog` task adds a logging configuration to the abas ERP client's `logging.custom.properties` file in `$MANDANTDIR/java/log/config/`. All your ESDK App^s log output of the defined log level can then be found in the file specified in `logging.custom.properties` in your ESDK App Project^.

Updating the `logging.custom.properties` file is currently not supported. If you change lines and rerun `installLog` the changed lines will be imported but not replaced. Therefore, changes to your log configuration have to be done manually after `installLog` has run.

10.15. Change mandantdir.env

10.15.1. installMandantdirEnv

The task `installMandantdirEnv` inserts all properties of the `mandantdir.env` file in the source folder `src/main/resources` of your ESDK App Project^ to the `mandantdir.env` file in your development client. The custom properties will be inserted after the `BEGIN` section of your client.

Custom properties will be inserted if the properties key does not exist yet. To update a value of an existing properties key, you need to manually delete that key from the `mandantdir.env` file of your client, prior to running `installMandantdirEnv`.

10.16. Publish JARs

10.16.1. publishClientDirJars

The `publishClientDirJars` task publishes all java artifacts in `$MANDANTDIR/abasbase/-java/lib/` as SNAPSHOT versions into the defined Sonatype Nexus repository.

These artifacts contain the generated AJO classes and have to be re-published again if changes to the database schema (vartabs, enumerations, keys, infosystems) have been made and `installAjo` was executed.

10.16.2. publishHomeDirJars

The `publishHomeDirJars` task publishes all java artifacts in `$HOMEDIR/java/lib/` as Release versions into the defined Sonatype Nexus repository.

These artifacts remain unchanged as long as the version of abas ERP is not updated.

10.16.3. removeNexusFromClient

For publishing the java artifacts from home dir and client dir, a small Gradle project named `nexus` is copied to the client directory.

The task `removeNexusFromClient` removes this `nexus` project from the abas ERP client directory. This is needed in case of changed connection data of the Nexus artifact server.

10.17. The JFOP server app

This is a jar artifact containing all logic belonging to the ESDK App^. It also contains a descriptor for its classpath, and an optional JFOP server app it depends on (the "parent" app). Its internal structure is described in the online help. However, you don't need to handle this manually — the ESDK Gradle Plugin does it for you.

10.17.1. createJfopServerApp

The task `createJfopServerApp` (deprecated name: `createApp`) creates the JFOP server app. It bundles the app's logic together with the app descriptors and creates the file and directory structure the JFOP server needs to identify it as a JFOP server app.

10.17.2. installJfopServerApp

The task `installJfopServerApp` (deprecated name: `installApp`) installs the JFOP server app on the abas ERP client.

10.17.3. redeployJfopServerApp

The `redeployJfopServerApp` task (deprecated name: `redeployApp`) redeploys the JFOP server App and thus makes it accessible to the user. It creates and installs it if it is not up-to-date.

10.18. Installing the ESDK App Installer

10.18.1. installEsdkAppInstaller

The `installEsdkAppInstaller` task installs the ESDK App Installer^ with the same version as the currently used ESDK Gradle Plugin^'s version on the client.

All versions of the ESDK App Installer^ installed using the task `installEsdkAppInstaller` are located in `$MANDANTDIR/esdk-app-installers`.

11. Documentation tasks

Adding documentation for your ESDK App^.

11.1. Writing ESDK App documentation

An app's documentation is written in a plain text format called Asciidoctor.

To add documentation for your ESDK App^ create a `documentation` folder in your ESDK App Project^ directory. Within the `documentation` folder, add a source folder `src/main/asciidoc` and create a file `index.adoc` within that source folder.

Add the following structure to the `index.adoc` file (the Project Builder and the Project Initializer already generate an example structure):

index.adoc

```
:docinfo:
= <name of your app>
Documentation
:nofooter:

This document describes the <name of your app> app.

== About

<name of your app> +
(C) <name of your company>

https://your-website.com/
```

You write your ESDK App^'s documentation by adding `.adoc` files to the `documentation/src/main/asciidoc` folder and referencing them in your `index.adoc` file.

For example, add a file `subpage.adoc` with the following content:

subpage.adoc

```
== <Headline of your subpage>
This is a subpage about subcontent

Content goes here!
```

You can include this file in your `index.adoc` by adding `include::subpage.adoc[]` before `== About`:

index.adoc

```
...
This document describes the <name of your app> app.

include::subpage.adoc[]

== About
...
```

The documentation's HTML5 rendering is provided by Asciidoctor. Refer to the Asciidoctor documentation for help on how to format your documentation with headings, lists, images, etc.

11.1.1. Rendering and viewing the ESDK App documentation

To render the documentation run

```
./gradlew esdkAppDocumentation
```

or execute the `esdkAppDocumentation` task from the **documentation** task group in your IDE. Afterwards you can view the documentation by opening `<you-project-dir>/build/asciidoc/html5/index.html` in a browser of your choice.

11.2. Packaging the ESDK App Documentation

You can create a ZIP archive from your rendered documentation by running

```
./gradlew packEsdkAppDocumentation
```

or executing the `packEsdkAppDocumentation` task from the `documentation` task group in your IDE.

The ZIP archive can then be found in `<you-project-dir>/build/esdk-app/`.

12. Verification tasks

12.1. verify

The task `verify` depends on the tasks that execute the unit tests (`test`) and the integration tests (`integTest`) and thus runs all tests.

In order to identify the scope a test is executed on, it is very advisable to make the distinction between unit tests and integration tests:

	Unit Tests	**Integration Tests**
Scope	small, local	wide, potentially involving all components of the system
Source Set	`src/test/`	`src/integTest/`
Gradle Task	`test` (comes with the Java Plugin, is a dependency of `build`)	`integTest` (added by the ESDK Gradle Plugin^)
Speed	fast, run by the programmer on every change	slow, only a subset is run by the programmer; the full set is run by a Continuous Integration server on every code check-in

	Unit Tests	Integration Tests
Further Reading	UnitTest, Martin Fowler, 5 May 2014	IntegrationTest, Martin Fowler, 16 January 2018

12.2. integTest

The task `integTest` runs all integration tests.

You may use the ESDK testing utilities to setup integration tests easily. This library is tailored for use with JUnit 4. If you are using JUnit 5 or another testing framwork, it still has some useful helper methods, but the support is not as sophisticated as for JUnit 4.

After importing the app project into IntelliJ IDEA, the `integTest` directory is not marked as "test source" directory.

To fix this, add the following to `build.gradle`:

build.gradle

```
plugins {
    // other plugins
    id 'idea'  ①
}

idea {
    module {
        testSourceDirs += project.sourceSets.integTest.
allSource.srcDirs ②
        testResourceDirs += project.sourceSets.integTest
.resources.srcDirs ③
    }
}
```

① Adds the Gradle IDEA Plugin

② Marks `integTest\java` directory as test sources directory

③ Marks `integTest\resources` directory as test resources directory

12.3. addJacocoToAppClasspath

The task `addJacocoToAppClasspath` configures the JFOP server for Jacoco by adding the Jacoco agent to the ESDK App^'s classpath and configuring a JFOP server instance for the user who executes the task. The agent keeps track of the code that is executed on the server side while the integration tests are run.

12.4. instrumentJfopServer

The task `instrumentJfopServer` redeploys the ESDK App^ in the JFOP server and depends on the task `addJacocoToAppClasspath`.

12.5. retrieveServerSideCoverage

`retrieveServerSideCoverage` dumps the code coverage statistics (written by the Jacoco agent on an instrumented JFOP server instance) into a `coverage-serverside.exec` file in `$MANDANT-DIR/esdk/test-infrastructure/reports` and copies it to the local ESDK App Project^.

12.6. calculateCodeCoverage

Code coverage means the percentage of production code that is covered by tests.

To calculate the overall code coverage for your ESDK App^ you need to execute the Gradle task `calculateCodeCoverage`.

It considers all `.exec` files in `build/jacoco` (some for the integration tests — downloaded from the JFOP Server by retrieveServerSideCoverage, some for the unit tests — generated locally) to calculate the code coverage and generates an HTML report.

The HTML code coverage report is generated and can be viewed by opening `build/reports/jacoco/html/calculateCodeCoverage/html/index.html`.

12.7. codeCoverageVerification

The task `codeCoverageVerification` verifies that the static code metrics of all tests together meet our recommendations.

Your ESDK App^ has to meet the following requirements to pass:

Group	Name	Requirement
Formal Requirements	Version	Valid semantic version number, which is not a SNAPSHOT version (ending in `-SNAPSHOT`) and defined in `gradle.properties.template`.
	Gradle Wrapper	A Gradle Wrapper compatible with the ESDK Gradle Plugin must be available.
	App ID	The App ID^ must be valid and registered via the ESDK Developer Portal. Refer to Register your App ID.
	Size	The packed ESDK App^ ZIP file cannot exceed 50MB.
	Programming Language	Any JVM-based programming language that can be tested automatically and is runnable in the JFOP server.
	Documentation	Documentation must be available for the ESDK App^ in Asciidoc format. Refer to Writing ESDK App documentation.
	abas ERP Version	abas ERP Versions, provided for this app, must still be supported by abas.
Static Code Metrics	Cyclomatic Complexity	Must be less than 21.
	Maximum Method Length	Max LOC/functions must be less than 31.
Dynamic Code Metrics	Tests	Test types are not restricted. You are free to use Unit Tests, Integration Tests and/or End-to-End Tests. All tests must pass.
	Test Coverage	Test Coverage must be greater than or equal to 80%.

The task passes if all static code metrics requirements are met. If one or more requirements are not met it will fail and display a message stating what was missed.

13. Other tasks

13.1. prepareVartab

The task `prepareVartab` prepares the vartab import file for installation.

13.2. processAppResources

`processAppResources` copies all resources files to `${project.buildDir}/abas/resources`.

14. Gradle Plugin Settings

14.1. Configuring the SSH connection timeout

The default timeout for connecting to your abas ERP client via SSH is 10 seconds.

To configure the timeout add the `timeout` property to your `ssh` closure in your `build.gradle` and set it to the desired value (e.g. `5000` for 5 seconds):

build.gradle

```
ssh {
    host = SSH_HOST
    port = SSH_PORT.toInteger()
    user = SSH_USER
    password = SSH_PASSWORD
    timeout = 5000
}
```

Alternatively, you can add it as a property to your `gradle.properties` and reference it in your `build.gradle`:

gradle.properties

```
SSH_HOST=yourhost
SSH_PORT=2205
SSH_USER=erp
SSH_PASSWORD=none
SSH_TIMEOUT=5000
```

build.gradle

```
ssh {
    host = SSH_HOST
    port = SSH_PORT.toInteger()
    user = SSH_USER
    password = SSH_PASSWORD
    timeout = SSH_TIMEOUT.toInteger()
}
```

14.2. EDP Log Files

By default, the ESDK Gradle Plugin^ logs all EDP commands to `build/logs/edp.log`.

The logging directory can be changed by setting `logFileLocation` in the `abas` section of your `build.gradle` file:

build.gradle

```
abas {
    ...
    logFileLocation = "$buildDir/logging"
}
```

For example, you can redirect the EDP log output to a separate directory with the called task's name within `build/logs` as follows:

build.gradle

```
abas {
    ...
    gradle.taskGraph.whenReady {
        logFileLocation = "$buildDir/logs/${gradle.taskGraph.getAllTasks().get
(gradle.taskGraph.getAllTasks().size() - 1).name}"
    }
}
```

When calling `./gradlew checkPreconditions` this would generate a log file `build/logs/checkPreconditions/edp.log`.

14.3. Command execution and performance monitoring

Many Gradle Plugin tasks open an ssh shell to the abas ERP client and execute commands on the client machine.

The commands executed and their execution times can be logged to the console. This helps to understand which actions are performed in abas ERP and to identify possible performance bottlenecks.

To enable command logging, add the parameter `--info` to the gradle call. For example:

```
./gradlew installIS --info
```

A typical output:

```
> Task :installIS
Task ':installIS' is not up-to-date because:
  Task has not declared any outputs despite executing actions.
Installing ow1/G30L0

Commands executed in this ssh session:
 Duration | Command
    78 ms   *** ssh connection ①
    59 ms   Upload: IS.OW1.G30L0 -> /abas/erp/isrein/ ②
    62 ms   Upload: OW1.G30L0.msg.ma -> /abas/erp/isrein/
    29 ms   Upload: INDEX -> /abas/erp/isrein/screen.OW1.G30L0/
    32 ms   Upload: menu.xml -> /abas/erp/isrein/screen.OW1.G30L0/
    32 ms   Upload: Resources.language -> /abas/erp/isrein/screen.OW1.G30L0/
    31 ms   Upload: Resources_en_US.language ->
/abas/erp/isrein/screen.OW1.G30L0/
    33 ms   Upload: screen_OW1.G30L0_M.xml -> /abas/erp/isrein/screen.OW1.G30L0/
    54 ms   Upload: screen_OW1.G30L0_T.xml -> /abas/erp/isrein/screen.OW1.G30L0/
   263 ms   Upload: screen.OW1.G30L0 -> /abas/erp/isrein/
    23 ms   crlf- /abas/erp/isrein/IS.OW1.G30L0
    20 ms   crlf- /abas/erp/isrein/OW1.G30L0.msg.ma
    19 ms   crlf- /abas/erp/isrein/screen.OW1.G30L0/INDEX
    19 ms   crlf- /abas/erp/isrein/screen.OW1.G30L0/menu.xml
    20 ms   crlf- /abas/erp/isrein/screen.OW1.G30L0/Resources.language
    18 ms   crlf- /abas/erp/isrein/screen.OW1.G30L0/Resources_en_US.language
    19 ms   crlf- /abas/erp/isrein/screen.OW1.G30L0/screen_OW1.G30L0_M.xml
    19 ms   crlf- /abas/erp/isrein/screen.OW1.G30L0/screen_OW1.G30L0_T.xml
   312 ms   edpexport.sh -l 65:1 -p *** -f such -k arb==ow1 | grep G30L0; echo
$?
  9592 ms   infosys_upgrade.sh -p *** -w OW1 -s G30L0
   239 ms   edpexport.sh -l 65:1 -p*** -f nummer,such,arb -k such==G30L0 | grep
ow1
-----------
 10973 ms total ③
```

① Time to open the ssh session

② Executed command and its execution time in milliseconds

③ Total time used for commands in this ssh session (including ssh session creation time)

15. Compatible Essentials Versions

15.1. App Compatibility

To check for an abas ERP version number, you need to add a version range to the `essentialsVersions` list in the `app` section of the `build.gradle` as following:

```
esdk {
    app {
        ...
        essentialsVersions = ["2017r1n00-2017r4n17"]
    }
}
```

Multiple version ranges are supported, e.g.: `essentialsVersions = ["2017r4n13-2017r4n15", "2018r1n00-2018r4n17"]`

These version ranges define your app's compatibility with abas ERP.

To create a new standalone app JAR using the `createEsdkAppJar` task you have to specify at least one abas ERP version range. Otherwise, the following error message will be displayed:

```
abas ERP version compatibility check failed: no compatible abas
ERP versions specified
```

Installing existing standalone app JARs without abas ERP version compatibility ranges are supported but will lead to the following warning:

```
abas ERP version compatibility check failed: no
'essentialsVersions' compatibility specified in your
'build.gradle'
```

15.2. ESDK Compatibility

Currently, the ESDK toolset is **compatible with abas ERP version 2017r2n00 and later**. This minimum version is checked by the ESDK Gradle Plugin^ as well as the App Installer.

15.3. Overriding the Version Check

Sometimes it might be necessary to override the compatibility check.

Only override the minimum Essentials version check if you do exactly know what you are doing.

Installing an app on a too-old abas ERP installation may have unintended side effects.

Installing an app on an abas ERP installation the app was not built for might lead to ClassCastExceptions and MethodNotFoundErrors at runtime.

15.3.1. App Installer

To override the version check on app installation, provide the app installer argument `--skip-essentials-versioncheck`. See Download and Usage for details.

15.3.2. Gradle Plugin

To override the version check when using the ESDK Gradle Plugin^ to develop an app for an abas ERP version that is officially not supported, set the following property in your app project's `build.gradle`:

```
esdk {
    ...
    allowUnsupportedEssentialsVersions = true
    ...
}
```

16. ESDK APIs and Libraries

To make common tasks less time-consuming and error-prone, we provide a set of some small APIs. All of them are available in the Maven repository `https://artifactory.abas.sh/artifactory/abas.maven-public/`.

Library Name	Purpose	How to use it	Javadoc
client-api	`de.abas.esdk:client-api:1.0.2:all`		
	License checking, reading app properties	See Use the esdk-client-api for an example	Client API Javadoc
test-utils	`de.abas.esdk.test.util:esdk-test-utils:0.0.6`		
	Integration testing using the app project's application properties.	See integTest and a newly generated project by the ESDK Developer Portal's project initializer	Testing Utilities Javadoc
versionchecker	`de.abas.esdk.versionchecker:versionchecker:0.10.25`		
	Reading abas ERP versions, comparing them or check version matches	The Javadoc provides some examples	Version Checker Javadoc

17. ESDK App Installer

Using the ESDK App Installer^ to install ESDK App^s.

17.1. Preface

The ESDK App Installer^ is an application able to install an ESDK App^ fully automated. The ESDK App^ needs to be available as a standalone JAR file. This JAR file is created by executing the Gradle task createEsdkAppJar after the app was previously installed using the fullInstall task.

17.2. Download and Usage

The current version of the ESDK App Installer^ can be downloaded here as a ZIP file.

- Choose the current version of the ESDK App Installer^ and download it.
- Copy the downloaded ZIP file into the client directory of the abas ERP client you want your app installed on.
- Unzip the file.
- Change into the unzipped folder.
- Change into the bin folder.

The ESDK App Installer^ is ready to be used to install an ESDK App^ in this abas ERP client.

From an ESDK App Project^ you can also use the ESDK Gradle Plugin^ task `installEsdkApp` to install you ESDK App^ using the ESDK App Installer^.

```
ESDK App Installer, Version 1.1.13

usage: ./esdk-app-installer [-a <artifact>] [--eapps-only] [-f] [-h] [-l <arg>]
[--maven-repo-password <maven-repo-password>] [--maven-repo-user <maven-repo-
user>] [--modular <modular>] [-p
       <edp-password>] [-q] [-s] [--skip-essentials-versioncheck] [-u <edp-
user>] [-v] [--version] [--yes-i-have-a-backup]
    -a,--artifact <artifact>                        Archive path or link to
nexus artifact
       --eapps-only                                 Only installs the
essentialsApps infosystem. All other installation actions will be ignored.
    -f,--force                                      Forces installation of given
ESDK App, even if the same or a higher version is already installed.
    -h,--help                                       show usage
    -l,--languages <arg>                            App installation languages
(e.g. English and French: EF)
       --maven-repo-password <maven-repo-password>  maven repository password
       --maven-repo-user <maven-repo-user>          maven repository user
       --modular <modular>                          Only installs specified
components of the ESDK App:
                                                    enums: installs enumerations
and named types only
                                                    vartabs: installs variable
table changes and their screens only
                                                    infosystems: installs
infosystems only
                                                    code: installs the JFOP
server app only
                                                    keys: installs keys only
                                                    data: installs data objects
only
    -p,--edp-password <edp-password>                EDP password
    -q,--quiet                                      quiet log output
    -s,--skip                                       do not install
essentialsApps infosystem
       --skip-essentials-versioncheck               Skips the check if this ESDK
App Installer and the App to be installed are compatible with the target abas
ERP version.
    -u,--edp-user <edp-user>                        EDP user
    -v,--verbose                                    verbose log output
       --version                                    display version of ESDK App
Installer
       --yes-i-have-a-backup                        Backup of the ERP System was
created

For more information visit: https://documentation.abas.cloud/en/esdk/#esdk-app-
installer
```

The ESDK App Installer^ requires an Internet connection to install ESDK App^s.
If the Internet connection is to be used via an HTTP proxy, its data can be set in the HTTP_PROXYHOST and HTTP_PROXYPORT environment variables (for example, in `homedir.env`).

To install an app that is available on an artifact server execute:

```
./esdk-app-installer -a <link to the app on an artifact server> [-u abasuser] -p abaspwd --maven-repo-user mavenRepoUser --maven-repo-password mavenRepoPwd
```

e.g.:

```
./esdk-app-installer -a https://artifactory.abas.sh/artifactory/abas.esdk.snapshots/de/abas/esdk/app/abasag-sparePartsCatalogue/0.4-SNAPSHOT/abasag-sparePartsCatalogue-0.4-20170520.095006-1.jar -p sy --maven-repo-user myuser --maven-repo-password C0mpl1c4t3d
```

To install an app that is available locally execute:

```
./esdk-app-installer -a <path to local app JAR file> [-u abasuser] -p abaspwd
```

e.g.:

```
./esdk-app-installer -a /abas/erp/sparePartCatalogueApp-standalone-app.jar -p sy
```

The ESDK App Installer^ reads the abas ERP client directory and some other values from the current environment. If the environment is not set up properly, the installer fails with messages complaining that `MNAME` is missing.
In this case, apply the ERP environment first. In the Essentials client directory, call: `eval $(sh ./denv.sh)`

All non-optional arguments except for the path to the local app JAR file or link to the app on an artifact repository can also be supplied interactively. Executing

```
./esdk-app-installer -a <path to local app JAR/link to the app on an artifact server>
```

e.g.

```
./esdk-app-installer -a https://artifactory.abas.sh/artifactory/abas.esdk.snapshots/de/abas/esdk/app/abasag-sparePartsCatalogue/0.4-SNAPSHOT/abasag-sparePartsCatalogue-0.4-20170520.095006-1.jar
```

```
./esdk-app-installer -a /abas/erp/sparePartCatalogueApp-standalone-app.jar
```

will prompt you for all other necessary arguments. All password inputs will be hidden.

The argument `abasuser` is optional. If not supplied it will be read from the environment.

To install the app in specific languages, use the `-l` / `--languages` argument with the language(s) to install:

```
./esdk-app-installer -a <path to local app JAR/link to app on an artifact server> -l <language(s)>
```

e.g.

```
./esdk-app-installer -a https://artifactory.abas.sh/artifactory/abas.esdk.snapshots/de/abas/esdk/app/abasag-sparePartsCatalogue/0.4-SNAPSHOT/abasag-sparePartsCatalogue-0.4-20170520.095006-1.jar -l A
```

```
./esdk-app-installer -a /abas/erp/sparePartCatalogueApp-standalone-app.jar -l A
```

Only languages contained in the app JAR will have texts, other language arguments will have no effect.

17.3. Use with abas ERP as a Docker container

The Gradle task `installEsdkAppInstaller` automatically downloads and unpacks an ESDK App Installer to the container or server whose connection is specified in the `gradle.properties` file. The task `installEsdkApp` uses that to install the app.

1. Download the installer to your local machine
2. Copy the installer into the abas ERP container
   ```
   docker cp <path to downloaded installer>/installer.1.1.13.zip erp:/abas/erp/
   ```
3. Create your app JAR
4. Copy the app JAR into the abas ERP container
   ```
   docker cp <path to you App JAR>/<yourappname>.<version>-standalone-app.jar erp:/abas/erp/
   ```
5. Run `bash` with the user `erp` inside the abas ERP container (named `erp`)
   ```
   docker exec -u erp -it erp bash
   ```
6. Set the client environment
   ```
   cd /abas/erp && eval $(sh denv.sh)
   ```
7. Unzip the installer

```
unzip installer-1.1.13.zip
```

8. Run the installer with the required options
```
esdk-app-installer-1.1.13/bin/esdk-app-installer -a <yourapp-name>.<version>-standalone-app.jar
```

To exit the container type `exit`.

Alternatively, download the installer by executing the following command in your Docker container:

```
wget -qO- 'https://artifactory.abas.sh/artifactory/abas.maven-public/de/abas/esdk/installer/1.1.13/installer-1.1.13.zip' -O app-installer-tmp.zip && unzip -q app-installer-tmp.zip && rm app-installer-tmp.zip
```

17.4. Logging

The ESDK App Installer^ logs to the console and to a file. The log file can be found in `$MANDANTDIR/esdk-installations/<appId>/<version>` and is named `installation.log`. If the same version of an ESDK App^ is installed again, the previously existing `installation.log` is kept. It gets an index that is incremented each time a new `installation.log` is created in the same directory.

Additionally, the EDP communication is logged to `edp.log` within `$MANDANTDIR/esdk-installations/<appId>/<version>`.

For an ESDK App^ with App ID^ `train` and version `0.12.4` that was installed three times the log directory structure and files would look like this:

```
ls $MANDANTDIR/esdk-installations/train/0.12.4/
edp.log  installation-2.log  installation-1.log  installation.log
```

The default log level is INFO and informs about each step the installer is executing, e.g.:

```
Checking preconditions for train 0.12.4
Starting app installation for train 0.12.4
Installing JFOP server app trainingApp
Installing enumerations [/tmp/esdk:427219964796100830 8app/enums/enums.xml]
Running enum reorg
Installing vartabs CustomTestDb.json
Running varreorg
Installing enumerations
[/tmp/esdk:427219964796100830 8app/enumsAfterVarreorg/enum.xml]
Running enum reorg
Installing database screens 0_a.screen
Installing fop.txt changes
Installing logging.custom.properties changes
Installing files from base
Installing infosystems [[TESTINFO]]
Regenerating AJO classes for [[TESTINFO]] and enumerations
Installing data objects from [/tmp/esdk:427219964796100830 8app/data/data.xml]
Redeploying JFOP server app trainingApp
Installing keys [/tmp/esdk:427219964796100830 8app/keys/keys.xml]
Running key reorg
Installation of train 0.12.4 successfully completed!
```

To only display warning and error messages, the installer can be run with the quiet option `-q` / `--quiet`:

```
./esdk-app-installer -a <path to local app JAR/link to app on an artifact server> -q
```

To get more detailed log messages, the installer can be run with the verbose option `-v` / `--verbose`:

```
./esdk-app-installer -a <path to local app JAR/link to app on an artifact server> -v
```

17.5. Installation of Infosystem ESSENTIALSAPPS

The Infosystem ESSENTIALSAPPS gives an overview of all installed apps in the abas ERP client. It is an app itself and gets installed whenever the ESDK App Installer^ is used.

If installed, the Infosystem ESSENTIALSAPPS is used to determine whether an ESDK App^ is already installed in the same or a higher version. By default, if the same or a higher version of the ESDK App^ is already installed, the installation is skipped.

The installation can be forced with the `-f`/`--force` option.

The installation of the infosystem ESSENTIALSAPPS app can be skipped with the `-s`/`--skip` option of the ESDK App Installer^.

```
./esdk-app-installer -a <path to local app JAR/link to app on artifact server> -s
```

e.g.

```
./esdk-app-installer -a https://artifactory.abas.sh/artifactory/abas.esdk.snapshots/de/abas/esdk/app/abasag-sparePartsCatalogue/0.4-SNAPSHOT/abasag-sparePartsCatalogue-0.4-20170520.095006-1.jar -s
```

```
./esdk-app-installer -a /abas/erp/sparePartCatalogueApp-standalone-app.jar -s
```

17.6. Permissions

The ESDK App Installer^ needs

- a shell user with abas environment matching the abas client the ESDK App^ is supposed to be installed in and
- an abas user (i.e. a Password Definition record in abas ERP^) with permissions to create and edit any object that is part of that ESDK App^.

We recommend to configure a dedicated abas user to install ESDK App^s on customers' live systems.

In the abas Docker images named `test` in the `sdp` registry, the abas user with password definition identity number 26 has sufficient rights for installing apps. The shell user `erp` should be used for running the ESDK App Installer^.

In detail, the following permissions are needed for ESDK App^ installation.

Usage	Details	Shell Permissions	abas ERP Permissions
General	Some precondition checks are executed to ensure basic permission needs are fulfilled.	The script `edpexport.sh`, command `ls` on files in `$MANDANTDIR` and subdirectories such as `$MANDANTDIR/java/abasbase`, `$MANDANTDIR/isrein`, `$MANDANTDIR/java/apps`, `$GRADLE_USER_HOME`, etc.	Access to the client's version information via EDP.

Usage	Details	Shell Permissions	abas ERP Permissions
Base Files	Creates the needed directory structure and transfers the files. It uses a temporary directory, which is created and deleted, and may do encoding changes.	The command `mkdir` for all directories that are to be created for the directory structure of the base files as well as temporary directories in `$MANDANTDIR`. The command `rm` for deleting previously created temporary directories in `$MANDANTDIR`. The command `cp` to copy files from the temporary directory to their target location. The command `chmod` to change file permissions according to configuration in the `.config` file. The command `echo`. The script `s3_conv` for converting file encoding to s3. The script `crlf-` to convert line endings.	
Data	Data is created and updated via EDP in the abas database.		Permissions to create and edit any object that is contained in the app's data XML file(s).
Enumerations	Enumerations are created and updated via EDP.		Permissions to create and edit enumerations and objects that are referenced in the enumerations.
Named Types	Named Types are created and updated via EDP.		Permissions to create and edit Named Types and objects that may be referenced in each Named Type.

Usage	Details	Shell Permissions	abas ERP Permissions
Keys	Keys are created and updated via EDP.		Permissions to create and edit Keys and objects that may be referenced in each Key.
Infosystems	Infosystems are imported via shell commands.	Permissions to run the scripts `edpexport.sh`, `infosysimport.sh` and `infosys_upgrade.sh` and permission to create the directory structure in `$MANDANTDIR/isrein`	
Variable tables	EDP is used for altering the tables. The Table of Variables reorganization is triggered via a shell command.	Permission to run the script `varreorg`	Permissions to edit Tables of Variables, to add, change and remove custom fields from the table. Permissions to register Additional Tables of Variables in the Company Data object.
AJO Generation	AJO Generation is triggered via a shell command.	Permissions to run the script `ajo_install.sh`.	
JFOP server app deployment	JFOP server app deployment is triggered via a shell command.	Permissions to run the script `jfopserver_cmds.sh` and to create the JFOP server app's directory structure in `$MANDANTDIR/java/apps`.	
Screens	Screens are identified via EDP queries and regenerated upon import.	Permissions to regenerate screens.	Permissions to query Tables of Variables to determine table names and screen numbers.

Usage	Details	Shell Permissions	abas ERP Permissions
Working Directories	Working Directories are registered via EDP.		Permissions to edit the installing users' Password Definition and the referenced Working Directory Permission object.
Dictionaries	Dictionary files are extended and the dictionaries are regenerated via shell commands.	Permissions to run `mkdir` and `rm` to create and delete a temporary directory in `$MANDANTDIR` and `cp` to copy from the temporary directory into the target directory. Permissions to run commands `s3_conv` and `crlf-`. Permissions to run the scripts `edpexport.sh`, `trans_tool.sh`, `trans_smart_do.sh` and `trans.sh`.	
fop.txt	Changes to the fop.txt file are transferred via shell commands.	Permissions to run `mkdir` and `rm` to create and delete a temporary directory in `$MANDANTDIR` and `cp` to copy from the temporary directory into the target directory. Permission to run commands `awk`, `sed`, `cat` and `grep` to alter the temporary fop.txt files. Permission to run command `echo`.	
Logging		Permissions to alter `$MANDANTDIR/java/log/config/logging.custom.properties`	

Usage	Details	Shell Permissions	abas ERP Permissions
mandantdir.env		Permissions to edit the file `$HOMEDIR/mandantdir.env` and run `envmake -B`.	

18. Demo Projects

18.1. Training App

The demo project `trainingApp` demonstrates all ESDK^ features using the latest ESDK^ version (this can also be a SNAPSHOT version).

Repository on Github

The `trainingApp` is for presenting features and therefore not all components make sense together as they are only intended to show how to use the specific ESDK^ features.

18.2. Geolocation App

The demo project `g30l0` consists of a small Infosystem that resolves customers' geo locations using an Open Street Maps web service.

Repository on Github

It includes unit tests using the mocking framework mockito and integration tests using the ESDK Testing Utilities.

To demonstrate how Kotlin can be used as an alternative programming language on the JFOP Server, there is a Kotlin port also: Repository on Github

18.3. Spare Parts Catalogue App

This example app implements an Infosystem for importing spare parts as a replacement catalogue from a CSV file. It also gives an example of how messages for multiple operating languages can be handled via properties files.

Repository on Github

18.4. Further Demo Projects

Additionally, there are several further projects in the ESDK Team Account on Github, e.g. small projects that were used in presentations during the ESDK Developer Days.

19. Infosystem ESSENTIALSAPPS

Use the infosystem ESSENTIALSAPPS to get information about all installed apps on your abas ERP installation.

The infosystem gives you information about:

- Error during installation (icon): Indicates if the current installation was successful
- App ID^
- App Version: The version of the app which is currently installed
- Installation date
- Installation duration
- Licensing (icon): Indicates whether the app has a valid license (green), does not use licensing (blue), is not licensed (red), or the license controller is not available (grey)
- abas ERP version compatibility: The abas ERP version ranges the app is compatible with
- abas ERP version compatibility (icon): Shows if the app is compatible with the actual abas ERP version
- Components (Button): Lists all components belonging to the app
- Installation Log File (Button): Shows the installation log file of the current version
- Installation Remarks: Shows error messages and information about the installation process

20. Submitting your ESDK App

20.1. Submitting your ESDK App for Support

You have the possibility to report any requests for missing features, or get help on any issues you are experiencing with abas Essentials SDK^ by raising a request for support. You will contact the ESDK^ development team directly and we will get back to you as soon as possible.

Appendix A: Upgrading an ESDK App

This section describes steps to perform when upgrading an ESDK App^ to a higher ESDK Plugin version.

To upgrade an existing app, just change the version number of the ESDK Plugin in `build.gradle`. Then check below whether any additional steps are to be performed for certain versions.

Starting from version 1.1.1 copies of this documentation are available in the Maven repository https://artifactory.abas.sh/artifactory/abas.maven-public/de/abas/esdk/-documentation/

A.1. Upgrading from 1.1 to 1.1.4

A.1.1. Gradle settings for existing ESDK App projects

ESDK libraries are not hosted on Bintray / JCenter anymore. Therefore, the abas.public-maven repository has to be configured in the Gradle settings.

Add a file `settings.gradle` to the app project's root directory with this content:

```
pluginManagement {
    repositories {
        maven {
            url = "https://artifactory.abas.sh/artifactory/abas.maven-public/"
        }
        gradlePluginPortal()
    }
}
```

For Gradle projects using Kotlin DSL, the file name is `settings.gradle.kts` and the content

```
pluginManagement {
    repositories {
        maven {
            url = uri("https://artifactory.abas.sh/artifactory/abas.maven-
public/")
        }
        gradlePluginPortal()
    }
}
```

If the file `settings.gradle(.kts)` already exists in the project, the above content should be placed at the beginning of the file.

A.2. Upgrading from 1.0 to 1.1

A.2.1. Gradle version

ESDK projects must use Gradle version 6.8.x

```
gradlew wrapper --gradle-version 6.8.2
```

A.3. Upgrading from 0.14 to 1.0

A.3.1. Table of Variables

The Table of Variables definition files have changed from `.schm` to `.json` format.

All `*.schm` files have to be converted to the new format. There is a Gradle task named Convert Vartab Files that performs this job.

For details about the new format, and the possibilities it offers, see Table of Variables (Vartab).

A.3.2. Export Vartab

exportVartab now only resolves the database using the database's classname (e.g. PrintParameter instead of Druckparameter) in order to make the export language independent.

A.3.3. Licensing

Licensing is discontinued by abas Product Management and should be removed from all ESDK apps using it.

A.4. Upgrading from 0.13 to 0.14

A.4.1. Gradle version

ESDK projects must use Gradle version 6.7.x

```
gradlew wrapper --gradle-version 6.7
```

A.4.2. Configuration properties

The configuration property `entryPoints` was removed.

To our knowledge this property was never used in client projects. Should this be the case, it should be removed from the `esdk` section in `build.gradle`.

A.5. Upgrading from 0.12 to 0.13

A.5.1. Gradle version

ESDK projects must use Gradle version 6.5. <<<

Appendix B: Experimental Features

B.1. External Configuration

Currently, ESDK App^ configuration settings are declared as part of Gradle build settings. This has several drawbacks:

- Every change to an ESDK app property triggers a reload of the Gradle project.
- The configuration settings format depends on the Gradle configuration format used (Groovy or Kotlin DSL).
- It's hard to access the settings with an external tool. For this reason there is not ESDK app settings wizard integrated in abas Tools, for example.

To work around these drawbacks we offer an alternative way to configure ESDK Apps in one or more external configuration files. These files use the HOCON (Human-Optimized Config Object Notation) format, providing further advantages:

- HOCON format is similar to JSON, but less pedantic and more convenient as a human-editable format.
- Settings can be split into several files (e.g. defaults and environment specific settings), can be retrieved from environment variables, or can be referenced from other properties.
- Durations can be specified including units, making it clear to which duration a value is resolved. For example `10s` for `timeout` is clear, while just `10` could also be 10 ms or 10 minutes.

External configuration is an experimental feature!
It is fully functional in the sense that it can replace the original configuration format. However, we reserve the right to change the configuration syntax or structure as we evaluate more real-world use cases.

To use external configuration, only two steps are needed:

1. Declare external ESDK App settings in HOCON format.
2. Remove settings from Gradle build configuration.

B.1.1. External Configuration Settings

Main configuration file

The main configuration file is located in the project root directory and called `application.conf`. This file will contain the app settings in HOCON format.

We will use the Training App as example. The adapted version can be found on this repository branch on Github.

application.conf

```
esdk {
    app { ①
        name = trainingApp ②
        supported-erp-versions = [ ③
            "2017r1n00-2017r4n16",
            "2018r1n00-2018r4n16",
            "2019r1n00-2019r4n16"
        ]
        namespace { ④
            vendor-id = ag ⑤
            app-id = train ⑥
        }
        export { ⑦
            infosystems = ["IS.OW1.TESTINFO"] ⑧
            tables = [TestDb, Teil] ⑨
            data = ["data.json"] ⑩
            meta-data = [mydata.json] ⑪
            keys = ["2738"] ⑫
            enums = [ ⑬
                Importfileformat,
                Importfileformat2,
                Importfileformat3,
                ThenSteps
            ]
            named-types = [TestRealNamedType] ⑭
            screens { ⑮
                "77" = [A],
                "Customer:Customer" = [A],
                "Operation:Operation" = [A],
                "Pricing:Pricing" = [A],
                "Sales:BlanketOrder" = [A],
                "TestDb:TestStructure" = [A]
            }
            advanced-screens { ⑯
                "75" = [A]
            }
        }
        installation { ⑰
            languages = DEA ⑱
```

```
            preconditions = ["workDirs=ow1"] ⑲
            workdirs = [ow1, owbi] ⑳
        }
    }
    erp {
        //default for Docker Desktop (Windows)
        host = "host.docker.internal"
        //if env variable HOSTNAME is set it will override the value above
        host = ${?HOSTNAME}
        edp {
            port = 6569
            password = sy
        }
        ssh {
            port = 2214
            user = erp
            password = none
            connection-timeout = 10s
        }
    }
    nexus {
        host = ${esdk.erp.host}
        port = 8090
    }
}
// if file exists, its settings will override any values from above
include file("application-overrides.conf")
```

① This block contains all ESDK app settings

② The ESDK app name

③ A list of abas ERP version range(s) this ESDK App is compatible to

④ This block contains settings building the app's namespace

⑤ ID of app vendor or creator

⑥ Unique app id as registered in the ESDK Developer Portal
See Register your App ID

⑦ This block contains settings used to export ERP (meta-)data to the ESDK app project.
All entries are optional and default to empty if not provided.

⑧ A list of names of infosystems provided by this app.
See exportIS

⑨ A list of standard and additional tables provided by this app.
See exportVartab

⑩ A list of files describing data to export.
See exportData

⑪ A list of files describing meta-data to export.

See exportMetaData

⑫ A list of keys to export.
See exportKeys

⑬ A list of enum class names to export.
See exportEnums

⑭ A list of named type's class names to export.
See exportNamedTypes

⑮ A map of screens to export. Mapping is "Database and Group class name" : "priorities".
See exportScreens

⑯ A map of standard screens overrides to export. Mapping is "Database and Group class name" : "priorities".
See exportScreens

⑰ This block contains settings used during app installation with the provided Gradle tasks

⑱ All languages to consider while importing texts into ERP dictionaries.
See importDictionaries

⑲ Preconditions to be met prior to installation.
See Checking Development and Installation Preconditions

⑳ Working directories to create or activate on the abas ERP client.
See createWorkdirs

This block contains erp host configuration properties

HOCON files can contain comments (separate line or inline)

Host name or IP address of the server abas ERP is running on

This value will be used (and override the one given in the line above) if an environment variable called `HOSTNAME` is defined

This block contains settings to access ERP via EDP

The EDP port

The EDP password

This block contains settings to access ERP via SSH

The SSH port

The SSH user

The SSH password

The SSH connection timeout. Can also be specified in other units like `ms` or `minutes`.
See Duration format

This block contains settings of the host running Nexus

The Nexus host. Here the same value provided in the `erp` section above is used

The nexus port

If a file named `application-overrides.conf` exists (next to `application.conf`) it is parsed.
Any values provided there will override those given above.

What is the correct `host`?

When a service (abas ERP or Nexus) is running in a docker container, choosing the correct value for `*.host` properties can be tricky. The following table lists common options:

Environment	**Value for** `*.host`
Service is running in a docker container on local Windows machine (Docker Desktop)	`host.docker.internal`
Service is running in a docker container on local Windows machine (Docker Toolbox)	Result of "docker-machine ip default"
Service is running in a docker container on local Linux machine	Value of "hostname" or "hostname -I"
Service is running in a docker container on a remote machine	Host name or IP address of machine hosting docker
Service is running on a dedicated local or external machine	Host name or IP address of machine hosting the service

Never use `localhost` in a Docker setting because then calls from ERP host to Nexus host will not succeed!

Configuration include

Configuration includes are very useful to provide common configurations for all installations or developers (in `application.conf`) and also have a copy with installation specific settings on each machine (in `application-overrides.conf`).

Here is an example of a developer specific include configuration. The structure is the same as in `application.conf` above, but only specifying the relevant properties.
In this instance the ERP client host name is specified.

application-overrides.conf

```
esdk {
  erp {
    host = "tec-34"
  }
}
```

Only `application.conf` is packed into the ESDK app archive on Packing your ESDK App and Submitting the ESDK App to Receive Support.

Configuration defaults

Some settings are identical in most projects and are therefore provided as default.
Still, it is possible to provide alternative values. Look up the value below and be sure to include the override in the correct position within the configuration structure.

reference.conf

```
// HOCON configuration defaults.
// Values configured here will be used if not specified otherwise in
`application.conf`.
esdk {
    app {
        namespace {
            use-for-tables = true ①
        }
        installation {
            classpath { ②
                shared = false ③
                parent = "DEFAULT_SHARED" ④
            }
            install-type = SSH ⑤
            allow-unsupported-erp-versions = false ⑥
        }
    }
    erp {
        home-dir = "/abas/s3" ⑦
        client-dir = "/abas/erp" ⑧
        client-id = "erp" ⑨
        edp {
            port = 6560 ⑩
            user = "" ⑪
        }
        ssh {
            port = 2205 ⑫
            user = erp ⑬
            password = none ⑭
            key-file = "" ⑮
            connection-timeout = 10000ms ⑯
        }
    }
    nexus {
        port = 8081 ⑰
        repository-name = abas-essentials-libs ⑱
        user = admin ⑲
        password = admin123 ⑳
        version = "2"
    }
}
```

① Determines whether the app-id should be used to prefix App variables in variable tables.
This should only be set to `false` if existing customizations should be turned into an ESDK App.
See App Variable Namespace

② This block contains settings affecting the JFOP server classpath

③ Sets the `parent.delegation` mode of the JFOP server app
See How the shared option influences additional dependencies

④ Sets an alternative value for the parent classpath. This value is only applied when `shared` is `true`
See How the shared option influences additional dependencies

⑤ Installation type. One of `SSH` or `LOCAL`

⑥ Set to `true` to install an app in an abas ERP version that is officially not supported

⑦ Absolute path to `$HOMEDIR` in the abas ERP client

⑧ Absolute path to `$MANDANTDIR` in the abas ERP client

⑨ Name of abas ERP client as in the `mandantdir.env`

⑩ EDP port

⑪ EDP user

⑫ SSH port

⑬ SSH user

⑭ SSH password

⑮ Path and name of SSH key file to use instead of a `password`

⑯ SSH connection timeout in milliseconds

⑰ Nexus port

⑱ Name of the repository the Java dependencies will be stored in on Sonatype Nexus

⑲ Nexus user name

⑳ Nexus user password

Nexus version. 2 or 3 are supported

B.1.2. Gradle Build Configuration

To activate external configuration settings all ESDK App settings have to be removed from the Gradle build configuration.

gradle.properties

All properties used in an `esdk { … }` (`build.gradle`) or an `esdk.apply { … }` (`build.gradle.kts`) block can be removed.
Typically this are:

- ABAS_*
- EDP_*
- NEXUS_*
- SSH_*

`NEXUS_*` properties may also be used in other Gradle configurations. They can still be removed from `gradle.properties`. The ESDK Gradle Plugin^ will read the values from external configuration settings and inject Gradle project extra properties accordingly.

Table 2. Supported properties and their mapping to Gradle properties

External configuration	Gradle
`esdk.nexus.host`	`NEXUS_HOST`
`esdk.nexus.port`	`NEXUS_PORT`
`esdk.nexus.repository-name`	`NEXUS_NAME`
`esdk.nexus.user`	`NEXUS_USER_NAME`
`esdk.nexus.password`	`NEXUS_PASSWORD`
`esdk.nexus.version`	`NEXUS_VERSION`

build.gradle(.kts)

build.gradle

Remove the `esdk { ... }` block.

build.gradle.kts

Remove the `esdk.apply { ... }` block and all property definitions that are only used within this block (e.g. `val ABAS_HOMEDIR: String by project`).

Appendix C: Step-by-step documentation for Windows users

This documentation describes step by step how to set up an ESDK development environment on Windows and create a simple ESDK App^.

C.1. Project setup on Windows

For development with abas Essentials SDK^ on Windows, the example given below uses the following software and tools:

C.1.1. Java Development Kit JDK8

- Step 1: Download JDK8, for example the **Open Java development kit JDK8 Corretto** from Amazon Corretto 8
- Step 2: Check the installed version by using the command `java -version` in a command line interface.

- Step 3: Check if the JDK binary is set by running `echo %PATH%` in a command line interface.
- Step 4: Check if the environment variable `JAVA_HOME` is set by running `echo %JAVA_HOME%` in a command line interface.

C.1.2. Docker for Windows

- Step 1: Download the latest **stable** version of Docker Desktop.
- Step 2: Start the Docker Desktop Installer and follow the instructions.

C.1.3. abas Tools

In the following example, we will work with the abas ESDK Developer of abas Tools. You will find more information about how to use the abas ESDK Developer in the PDF documentation abas Tools ESDK Developer. However, you can also use another IDE, such as IntelliJ or Eclipse.

 Download the current version of abas Tools, Distribution for ESDK users for 64-Bit Java Windows, from the abas download area. More documentation about abas Tools is available there.

C.1.4. Git Bash

Download the latest version of Git Bash for Windows from the official Git website.

C.1.5. abas ERP client and Nexus Artifact Server as Docker containers

- Step 1: Pull the images of the abas ERP client and Nexus from Docker.
  - Log in to the Docker registry using the command `docker login sdp.registry.abas.sh` and your extranet credentials in a Git Bash.
  - Download the docker-compose.yml file.
  - Open the file in abas Tools using the menu `File` > `Open File ...` in the menu bar.
  - Specify the image you want to be pulled in the line `image: sdp.registry.abas.sh/abas/test:<PLEASE_ENTER_A_VERSION>`. Replace <PLEASE_ENTER_A_VERSION> with the abas ERP version you want to develop against.
  - In the directory where the `docker-compose.yml` file is stored, open a Git Bash and run the command `docker-compose up -d` to pull and start the images of the chosen abas ERP version and Nexus.
- Step 2: Download and configure the abas ERP mini-GUI.
  - Open the following URL in a browser: http://<your-ip-address>:8001
  - Download the file `abasgui-mini-erp.tgz`.

- In a dedicated directory, unpack first the `.tgz` archive, then the `.tar` file.
- Download the file `wineks-erp.ini`.
- Rename the file `wineks-erp.ini` to `wineks.ini`.
- Copy the file `wineks.ini` into the unpacked `mini-GUI` directory.
- Open the file `wineks.ini` in abas Tools using the menu `File` > `Open File ...` in the menu bar.
- Change the following values:
  - In the line `host=`, enter the string `host.docker.internal`.
  - In the line `portEksd=`, enter the abas GUI test port as mapped in the `docker-compose.yml` file. The test port 48592 is generally used.
- Save your changes.

- Step 3: Start the dockerized abas ERP client.
  - Run `abasgui.exe`.
  - In the `Password` text field of the login window, enter the password of the abas ERP user and confirm it by clicking the **`[Log-on]`** button.

C.2. Create your ESDK App

- Step 1: Register a unique app ID for your ESDK App^.
  - In Google Chrome, open the ESDK Developer Portal.
  - Log in with your extranet credentials.
  - Click the red plus icon in the upper right corner to register the app ID.
  - Enter your App ID^.
  - Click the **`[Create]`** button.

C.3. Create a new component in your dockerized abas ERP client

In the following example, we will create a new infosystem.

- Step 1: On the `Standard` page of the abas ERP command overview, go to the `Infosystem` folder and run the `Set up infosystem` database command.
- Step 2: In the toolbar, click the plus icon to trigger the creation of a new infosystem.
- Step 3: On the `General` page, fill in the fields `Search word`, `Description` and `Workspace`. The workspace of a customized infosystem should be the user-defined `OW1` workspace.
- Step 4: On the `Output, Java and Help` page, fill in the `Java class name` field. It can be based on the search word of your new infosystem.

- Step 5: On the `Variables` page, add a variable to the header section by clicking the **`[Attach variable to header section]`** button of the table. In the row of the new variable, fill in the fields as follows:
  - Field Name: `istesthead`
  - Type: `GL30`
  - Screen: `true`
  - Meaning: `Test Head`
- Step 6: On the `Variables` page, add a variable to the table section by clicking the **`[Attach variable to table section]`** button of the table. In the row of the new variable, fill in the fields as follows:
  - Field Name: `istestrow`
  - Type: `GL30`
  - Screen: `true`
  - Meaning: `Test Row`
- Step 7: Save your infosystem and confirm the automatic generation of the new screen description.

C.4. Initialize an ESDK App project using the ESDK Project Builder

- Step 1: Log in to the online ESDK Project Builder using your extranet credentials.
- Step 2: Fill in the fields of the form as follows:
  - Development Version: Select the abas ERP version you want to develop against.
  - App ID: Enter the unique App ID^ you have registered before.
  - App Name: Enter the app name. The first character of the app name must be a lowercase letter.
  - Vendor ID: Enter two lowercase letters.
  - Package Name: Enter the name of the package according to the naming convention `de.<company>.esdk.<appname>`. Use your company name here. The whole package name must be in lowercase.
  - Configure Essentials Version Ranges: Select a version range and click the plus icon to add these version range to the `essentialsVersion` list in the `app` section of the `build.gradle` file and set the app's compatibility with abas ERP in your ESDK App Project^.
  - Languages: Select in which language the ESDK App^ will be available. German and American English are set by default.
  - Add Infosystems: First, enter the working directory specified in the field `Workspace` of the `Set up infosystem` database command. Afterwards, enter the search word of your infosystem and click the plus icon. The name of the infosystem folder will be displayed in the row

below.

- abas Host and Nexus Host: Use the default host value `host.docker.internal`.
- Nexus Version: Make sure you are using version 2.

- Step 3: Click the **`[Create]`** button to generate the basic ESDK project structure of your ESDK App Project^ in a ZIP file. The generated ZIP file is stored in your `Downloads` folder.
- Step 4: Create a main folder for your ESDK App^ and unpack the ZIP file to this dedicated folder.

C.5. Configure the Gradle project settings

- Step 1: In the folder of your unpacked ESDK App Project^, generate the `gradle.properties` file by running the shell script <`./initGradleProperties.sh`> in a Git Bash.
- Step 2: Open the generated `gradle.properties` file in abas Tools using the menu `File` > `Open File …` in the menu bar and check the connection values. Make sure the host value `host.docker.internal` is specified in the rows `EDP_HOST`, `NEXUS_HOST` and `SSH_HOST` or change it if necessary.

C.6. Import the project in abas Tools

- Step 1: Import the Gradle project.
  - In abas Tools, go to `Window` > `Perspective` > `Open Perspective` > `Other`, and select `ESDK Developer` to open the ESDK Developer perspective.
  - Click `File` in the menu bar.
  - Select `Import` > `Gradle` > `Existing Gradle project`.
  - Follow the wizard instructions.
- Step 2: Set the connection from abas Tools to the dockerized abas ERP client.
  - Open the `abas ERP` connector dialog via `Window` > `Preferences` > `abas Tools Suite` > `Connectors` > `abas ERP`. The connection is pre-configured.
  - Check if the EDP settings such as `connection name`, `host name`, `client directory`, `port` and `password` are correct. Make sure the host value is identical to that of the files `wineks.ini` and `gradle.properties`.
  - Test the connection by clicking the **`[Test]`** button on the right-hand side. A confirmation message `Connection test is successful.` is displayed.
  - Confirm the connection setup by clicking **`[Apply and Close]`**. The active connection is displayed in the lower right corner of the status bar.

For more information about the connection configuration, see the documentation **abas Tools Connector Guide** under **Help** > **Help contents**.

C.7. Check the connection properties

Make sure the prerequisites, such as `connection properties`, `appId`, accessibility of Nexus Artifact Server, are fulfilled by double clicking the Gradle task `checkPreconditions` in the **abas basic** task group of the **Gradle Tasks** view.

The Gradle task `checkPreconditions` may fail due to a deprecated Gradle version. In this case, the Gradle version must be updated. To do so, open the terminal by right-clicking the project name in the `Package Explorer` and selecting `Show in` > `Terminal` in the context menu. In the terminal view opened below the editor area, run the Gradle task `gradlew wrapper --gradle-version 6.8.2`. Run the Gradle task `checkPreconditions` again and check if it has been successfully executed.

C.8. Resolve the dependencies

Run the Gradle task `publishAllJarFiles` in the **abas basic** task group of the **Gradle Tasks** view to provide the abas specific dependencies. Afterwards, refresh the Gradle project by right-clicking the Gradle project name in the Package Explorer and selecting `Gradle` > `Refresh Gradle project` in the context menu.

C.9. Export the infosystem into the local project

Export your new infosystem from the abas ERP client into your local ESDK App Project^ by double-clicking `exportAll` in the **abas basic** task group of the **Gradle Tasks** view.

C.10. Add logic to your project

In the following example, we will implement an Event Handler in the infosystem created before.

C.10.1. Create an Event Handler

- Step 1: Add the Event Handler class.
  - In the `src/main/java` folder of your local ESDK App Project^ in the Package Explorer, place the cursor on the package.
  - Click the `Create Event Handler for Infosystem` icon in the toolbar. Alternatively, you can use the shortcut `SHIFT+F7` or right-click the package name and select ESDK > `Create Event Handler for Infosystem` in the context menu.
  - Enter the name of your infosystem. The wizard suggests a name created by using the AJO class name and appending the suffix `EventHandler` in the field `Type name`.
  - Click **`[OK]`** to create the new source file in the selected package.

- Run the Gradle task `syncCode` in the **abas basic** task group of the **Gradle Tasks** view.
- Refresh the Gradle project by right-clicking the Gradle project name in the Package Explorer and selecting `Gradle` > `Refresh Gradle project` in the context menu.

C.10.2. Implement the Button after field event

- Step 1: Open the `Add Events` dialog by right-clicking the Event Handler source file you have just created and selecting ESDK > `Add events` in the context menu. Alternatively, you can open the source file in the Java editor and use the shortcut CTRL+U.
- Step 2: In the `Header fields` section, open the `start` node and select the `Button After` event. Click OK.
- Step 3: Complete the event method.

```
public class HelloWorldAppEventHandler {

    @ButtonEventHandler(field="start", type = ButtonEventType.AFTER)
    public void startButtonAfter(ButtonEvent event, ScreenControl screenControl,
DbContext ctx, HelloWorldApp head) throws EventException {

        String testhead = head.getTesthead();
        head.table().clear();
        HelloWorldApp.Row row = head.table().appendRow();
        row.setTestrow("Hello, " + testhead  + "!");
    }
}
```

- Step 4: Save the Event Handler and run the Gradle task `syncCode` in the **abas basic** task group of the **Gradle Tasks** view.
- Step 5: Refresh the Gradle project by right-clicking the Gradle project name in the Package Explorer and selecting `Gradle` > `Refresh Gradle project` in the context menu.

C.10.3. Register the Event Handler in the infosystem in your dockerized abas ERP client

- Step 1: In the abas ERP command overview, go to the `Infosystem` folder, run the `Set up infosystem` database command and open the infosystem in `Edit` mode.
- Step 2: Go to the `Variables` page and double-click the line `isbstart` to open the line zoom.
- Step 3: On the `EFOPS` page, fill in the `Button (after)` field as follows:

```
java:<qualifiedname>@<AppID>
```

You get the qualified name of the Event Handler class by right-clicking the class name of the public

class in the EventHandler.java editor and selecting `Copy Qualified Name` in the context menu.

You can find your app ID in the line `appId` of your `build.gradle` file.

- Step 4: Save your changes in the `Set up infosystem` screen.

C.10.4. Test your infosystem in your dockerized abas ERP client

- Step 1: Open your infosystem.
- Step 2: Enter `world` in the `Test Head` field.
- Setp 3: Click the Start icon.

`Hello, world` is displayed in the table row.

C.10.5. Upload the full logic in your local ESDK project

Run the Gradle task `exportIS` in the **abas professional** task group of the **Gradle Tasks** view to load the full logic of the infosystem in your local ESDK App Project^.

C.11. Create the app jar file

In abas Tools, create the app jar file to install your ESDK App^ in another abas ERP client.

- Step 1: Make sure the compatible abas ERP version range is set in the `build.gradle` file.
- Step 2: Run the Gradle task `createEsdkAppJar` in the **abas basic** task group of the **Gradle Tasks** view.

C.12. Install and test your ESDK app in a new dockerized abas ERP client

- Step 1: Make sure the dockerized abas ERP client is closed.
- Step 2: Stop and remove the currently running Docker container.

```
docker-compose down
```

- Step 3: Create and start the new Docker container.

```
docker-compose up -d
```

- Step 4: Open the dockerized abas ERP client using the mini-GUI.
- Step 5: In abas Tools, install the ESDK App Installer^ into the abas ERP client by running the Gradle task `installEsdkAppInstaller` in the **abas professional** task group of the **Gradle Tasks** view.

- Step 6: Install the ESDK App^ into the dockerized abas ERP client by running the Gradle task `installEsdkApp` in the **abas basic** task group of the **Gradle Tasks** view.
- Step 7: Open and test the infosystem in your dockerized abas ERP client.

Glossary

This glossary is meant to clarify the naming of the components that relate to abas Essentials SDK^.

abas Essentials SDK

The Software Development Kit (SDK) for customizing abas ERP^. It mainly consists of

- The ESDK Gradle Plugin: A set of Gradle tasks for developing, installing and testing ESDK App^s.
- The ESDK App Installer: An application for installing an ESDK App^ fully automated.

abas ERP

The core product of abas Software GmbH. It is also known as abas Essentials.

App ID

Short for ESDK App ID^.

ESDK

Short for abas Essentials SDK^. Please use only upper case letters, **not** eSDK.

ESDK App

The result of an ESDK App Project^. The finished and automatically installable customizing for abas ERP developed using abas Essentials SDK^.

Please do not use any other names, **not** ESDKs, ESDK package, etc.

ESDK App ID

The ESDK App^'s unique name space.

ESDK App Installer

The installation tool to install an ESDK App^ in an abas ERP client.

ESDK App Project

The project structure to develop an ESDK App^.

ESDK Gradle Plugin

The core component of the ESDK^ toolset. It is the development tool for creating ESDK App^s.

About

abas Essentials SDK 1.1.13, 2021-10-07

Subject to change

https://abas-essentials-sdk.com/