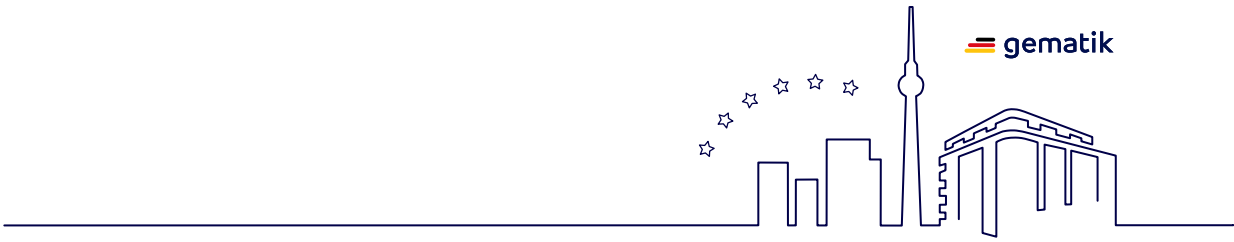
ERPione

gematik GmbH

Version 0.9.1, 2025-05-16



Contents

Scope 2
Eponym 2
Architecture 2
Backend 2
Frontend 2
Proxy 2
ERPione CLI 3
Installation 3
Docker 3
Nativ 4
From source 4
cURL 5
Configuration 5
API-Key 5
Default Environment 5
Use cases 6
Help 6
Issue prescriptions 7
Data Matrix Codes 7
Accept prescriptions 8
Dispense prescriptions 8
Close prescriptions 9
Abort prescriptions 9
Create charge items 10
Change charge items 10
Service information 11
Appendix A: Release Notes 12


ERPione

Scope

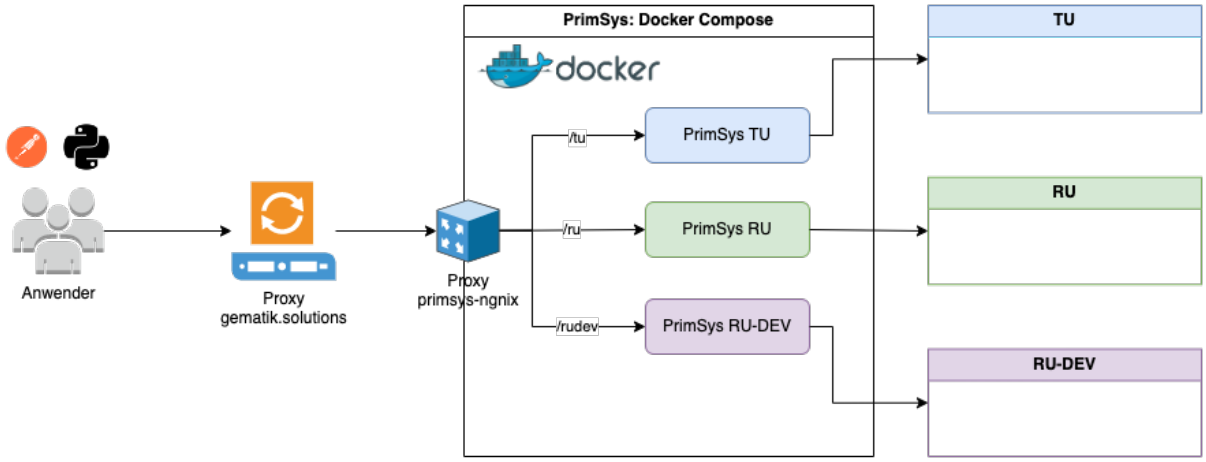
As a national competence center for digital health, gematik GmbH provides a lot of services and applications for testing the telematics infrastructure (TI) services. *ERPione* is one of them. It is a software solution that enables the convenient creation of e-prescriptions in a test environment. It is designed to be used by developers and testers who want to test the integration of e-prescriptions into their applications.

Eponym

The name *ERPione* is a combination of ERP - short for e-prescription (E-Rezept | Gematik) - and the Greek goddess Epione. As the goddess of pain relief and the mother of Hygieia, the patron saint of pharmacists, she is an ideal namesake.

Architecture

The architecture of *ERPione* follows a modular client-server approach, separating responsibilities between backend, frontend, and proxy components to ensure scalability, maintainability, and clear abstraction of technical details.



Backend

The backend handles all tasks such as communicating with the TI, processing FHIR or creating cryptographic signatures.

Frontend

The frontend itself is relatively lean (Thin Client). It therefore contains little to no domain knowledge, e.g. about FHIR or the e-prescription itself, but merely serves as an intermediary between the user and the backend service.

Proxy

The proxy is almost transparent to the user, but performs two essential tasks: routing to the appropriate environments and access control to the actual service via API keys.

ERPione CLI

The *ERPione* CLI is a command line interface that allows you to interact with the *ERPione* application from the terminal. It provides a set of commands for managing and configuring the application, as well as for performing various tasks.

Installation

The *ERPione* can be used in three variants. In the simplest case, *ERPione* can be used directly as a native and executable application for the respective operating system.

The container variant uses Docker for execution and also offers additional features such as a manpage, the oh-my-zsh and the integration of erp-cli-fhir.

The last variant for execution directly via the source files is possible, but is only recommended for developers and enthusiasts.

Translated with www.DeepL.com/Translator (free version)

If you have already installed Docker, the container variant should be your preferred choice.

Docker

The *ERPione* can be loaded directly as a Docker image from DockerHub with docker pull:

```
docker pull gematik1/erpione
```

The image can now be started with the following command using docker run:

```
docker run -it --name ERPione \
      --env ERPIONE_API_KEY=$ERPIONE_API_KEY \
      -v $FULL_HOST_PATH:/home/gematik/shared \
      gematik1/erpione
```

- With `-it` (abbreviated for `--interactive` and `--tty`), the container is started in an interactive TTY session and the user lands directly in the container.
- The container name `ERPione` is freely selectable
- With `--env ERPIONE_API_KEY=$ERPIONE_API_KEY` the API key is stored in the container as an environment variable. Here, `$ERPIONE_API_KEY` must be replaced with the actual API key if the environment variable is **not** stored on the host computer
- With `-v $FULL_HOST_PATH:/home/gematik` a Volume is included in the container (quasi a shared folder between the container and the host computer) to enable the exchange of files between the container and the host computer. If this is not required, this argument can also be omitted.

The `-it` switch means that the container is **not** shut down when it is exited. As

long as the container is running, you can always dial into the container again using docker exec.

```
docker exec -ti ERPione /bin/zsh
```

- With `-it` (abbreviated for `--interactive` and `--tty`), the container is started in an interactive TTY session and the user lands directly in the container.
- The container name `ERPione` that was previously selected for `docker run
- The last argument `/bin/zsh` specifies which shell is to be used.

Nativ

In principle, no special commissioning is necessary. *ERPione* can be used directly as an executable application without any installation. However, for the user's convenience, it is recommended to initially set up the API-Key in the environment variables.

The executable applications must be generated separately for each operating system. Currently, *ERPione* is provided native executable for the following operating systems:

Translated with www.DeepL.com/Translator (free version)

- Windows
- Linux
- macOS

From source

This variant is only recommended for developers and Python enthusiasts and requires the Python toolchain for execution

- Python 3.11
- pip
- ideally venv or virtualenv

A *virtual environment* can be created and activated with the following call from the main directory of *ERPione*:

```
python -m venv ERPione_venv

source ERPione_venv/bin/activate
```

The dependencies can then be installed within the *virtual environment* using pip:

```
pip install -r requirements.txt
```

cURL

cURL is a command-line tool for transferring data using various network protocols. It is widely used for testing APIs and web services. In the context of *ERPione*, cURL can be used to interact with the RESTful API endpoints provided by the application.

Configuration

API-Key

The API key is required for each *ERPione* request. To avoid having to specify it for each operation, it is possible to store the API key permanently as environment variable.

To do this, the API key must be stored under the `ERPIONE_API_KEY` environment variable in the operating system. This procedure differs from operating system to operating system. There are numerous instructions on the Internet, e.g., for Windows and Linux.

Default Environment

The environment (TU, RU, RU-DEV) can be specified separately for each request using the `--env` switch. To avoid having to specify this switch for each individual call, a default environment can be defined using the environment variable `ERPIONE_ENV`. The default environment can then be overridden with `--env'.

Use cases

The *ERPione* CLI is a command line interface that allows you to interact with the *ERPione* application from the terminal. It provides a set of commands for managing and configuring the application, as well as for performing various tasks.

Help

The `-h` (or `--help`) argument is the first port of call to get initial help on the CLI arguments.

For example, you can use the following help to get an overview of the available subcommands:

```
erpione --help
```

The appropriate help for a particular subcommand is then displayed as follows:

structure

```
erpione SUBCOMMAND --help
```

example

```
erpione prescribe --help
usage: erpione prescribe [-h] [-e {tu,ru,rudev}] [-s SERVER] [--api-key API_KEY | --api-key
-file API_FILE] [--actor ACTOR] [--dmc] bundles_path

positional arguments:
  bundles_path          Path to KBV-Bundles to be prescribed

options:
  -h, --help            show this help message and exit
  -e {tu,ru,rudev}, --env {tu,ru,rudev}
                        Specification the Environment for this call. This argument will
overwrite the default environment configured via ERPIONE_ENV
  -s SERVER, --server SERVER
                        The URL of the PrimSys-REST server (default: https://erpps-
test.dev.gematik.solutions/)
  --api-key API_KEY     API-Key for accessing the service
  --api-key-file API_FILE
                        Path to file where the API-Key is stored
  --actor ACTOR         ID or name of the actor
  --dmc                 Download Data-Matrix-Codes
```

Most subcommands have a common base of switches for the *ERPione*:

Table 1. common arguments

short	long	Description
-e	--env	Selects the appropriate TI environment. This switch overrides the default_environment. If the environment is not specified by either `-e` or standard_environment, then the default on the erpione-proxy is used.
-s	--server	The default corresponds to the internal hosting within gematik and generally does not need to be changed.
	--api-key	Can be used to override the environment variable or set one if no environment variable is set.
	--api-key -file	Can be used to overwrite the environment variable with an API key from a file, or set one if no environment variable is set.

Issue prescriptions

The following command can be used to issue e-prescriptions directly from a KBV bundle template:

erpione

```
erpione prescribe ~/examples/gkv_bundle_01.xml
```

curl

```
curl -H "apikey: $ERPIONE_API_KEY" \
  -H "Content-Type: application/xml" \
  https://erpps-
test.dev.gematik.solutions/ru/doc/6a7f66bc2bb0f4cd76deaa260abbb484/xml/prescribe \
  -d "@~/examples/gkv_bundle_01.xml" | jq
```

In some cases, it is necessary to issue many prescriptions at once. This would be very tedious to do with curl, or would have to be done with a script. The *ERPione* provides a built-in mechanism for this:

```
erpione prescribe ~/examples/
```

You just have to specify the directory instead of a specific file. The *ERPione* will iterate through the given directory and create a prescription for each template it finds.

Data Matrix Codes

If a Data Matrix Code (DMC) is required for the e-prescription output, the `--dmc` switch can be used. This means that the DMC is also downloaded for each e-prescription and stored as a PNG image file in the output directory with the naming scheme `dmc_<PRESCRIPTION_ID>.png`.

```
erpione prescribe --dmc ~/examples/gkv_bundle_01.xml
```

Accept prescriptions

E-prescriptions issued via *ERPione* can be accepted as a pharmacy with the following command:

erpione

```
erpione accept --taskid [TID]
```

curl

```
curl -X POST \
  -H "apikey: $ERPIONE_API_KEY" \
  https://erpps-
test.dev.gematik.solutions/ru/pharm/[PHARMACY_ID]/accept?taskId=[TID]&ac=[AC] | jq
```

The `--accesscode` parameter is optional in this case because *ERPione* already knows it. The situation is different for e-prescriptions that were not issued via *ERPione* or for older e-prescriptions that are no longer in the internal memory of *ERPione*. In this case, the AccessCode must be specified explicitly, similar to the curl approach:

```
erpione accept -t [TID] --accesscode [AC]
```

Accepted e-prescriptions are set to `in process` status in the FD and are *"reserved"* for that pharmacy. More precisely, *"reserved"* for all pharmacies that know the generated **Secret**.

Forgotten secrets can be a problem because such e-prescriptions cannot be processed further and are automatically removed from the FD after 90 days.

To prevent this, the e-prescriptions should be further processed (e.g., dispensed or deleted) after acceptance if possible

Dispense prescriptions

To dispense an e-prescription, the `accept` command must first be executed so that the FD can set the e-prescription to the ìn-progress` status and generate a secret. The e-prescription can then be dispensed using the following command:

```
erpione dispense -t [TID]
```

As with `accept`, the secret is only required for *externally* created or accepted e-prescriptions.

It is also possible to specify one or more substitute medicines at the time of dispensing. The `--body-file` option can be used to submit a JSON file containing the replacement drug.

```
erpione dispense -t [TID] --body-file ~/examples/dispense_body.json
```

The content of `dispense_body.json` could be something like the following, if more than one replacement drug is dispensed for this prescription

```
[
  {
    "type": "PZN",
    "category": "00",
    "standardSize": "NB",
    "supplyForm": "AMP",
    "amount": 1,
    "pzn": "09385450",
    "name": "Vitadol Gold Beutel 1000 mg",
    "batch": {
      "lotNumber": "123123",
      "expiryDate": "26/02/2024"
    }
  },
  {
    "type": "PZN",
    "category": "00",
    "standardSize": "NB",
    "supplyForm": "AMP",
    "amount": 1,
    "pzn": "17975846",
    "name": "Cbd 10 Cannbio Gold Edition 10 ml",
    "batch": {
      "lotNumber": "123123",
      "expiryDate": "26/02/2024"
    }
  }
]
```

Close prescriptions

The `close` command is used to finally close an e-prescription. As with the `dispense` command, a previous `accept` is mandatory. Before `close` a `dispense` is possible but not mandatory.

The `close` command has the same interface as the `dispense` command. The only difference here is that the e-prescription is not closed with the `dispense` and therefore remains in the `in-progress` state.

Abort prescriptions

erpione

```
erpione abort -t [TID]
```

curl

```
curl -X DELETE \
  -H "apikey: $ERPIONE_API_KEY" \
  https://erpps-
test.dev.gematik.solutions/ru/pharm/[PHARMACY_ID]/abort?taskId=[TID]&ac=[AC]?secret=[SECRET
] | jq
```

The AccessCode `[AC]` and Secret `[SECRET]` are only required for *externally* generated e-prescriptions, as with `accept`.

Create charge items

Das Erstellen bzw. das Ändern der PKV-Abrechnungsinformationen wird über das Sub-Kommando `chargeitem` abgewickelt. Der Inhalt der PKV-Abrechnungsinformationen wird dabei in einem JSON-File definiert und an das Sub-Kommando übergeben. Sowohl für `create` als auch `change` ist die Struktur des JSON-Files identisch und entspricht der Struktur die bereits im Szenario-Player verwendet wird:

The `chargeitem` subcommand is used to create or modify the charge item information. The content of the charge item information is defined in a JSON file and passed to the subcommand. The structure of the JSON file is identical for both create and modify, and is the same as the structure already used in the Scenario Player:

```
{
  "currency": "EUR",
  "vatRate": 12.9,
  "priceComponents": [
    {
      "category": "ZUZAHLUNG",
      "type": "informational",
      "insurantCost": 12.3,
      "totalCost": 75.5,
      "pzn": "12345678"
    },
    {
      "category": "ZUZAHLUNG",
      "type": "informational"
    }
  ]
}
```

To create a ChargeItem, use the following command:

```
erpione chargeitem create --taskid [TASK_ID] [path/to/payload.json]

erpione chargeitem create -t 200.000.002.175.581.70 ./resources/my_examples/chargeitem_payload.json
```

Change charge items

Charge items that have been set initially can be changed later at the request of the insured person. To change an existing charge item, use the following command:

```
erpione chargeitem change --taskid [TID] --accesscode [AC] [path/to/payload.json]

erpione chargeitem change -t 200.000.002.175.581.70 \
      -a 9ea09db15e04fe9b1613cd0a338b812c728b355d672d9af8e54b7596e9098fa8 \
      ./resources/my_examples/chargeitem_payload.json
```

In this use case, the AccessCode has the same structure and the same name as the AccessCode of a prescription. However, the two values are different and should not be confused.

The AccessCode for the ChargeItem is regenerated by the FD when `create` and `change` and must be called up again by the FdV after each change.

Service information

To get information about the service, you can use the following call:

erpione

```
erpione info
```

curl

```
curl -H "apikey: $ERPIONE_API_KEY" https://erpps-test.dev.gematik.solutions/ru/info | jq
```

In addition, the active **Actors** can be accessed with the following call:

erpione

```
erpione actors
```

curl

```
curl -H "apikey: $ERPIONE_API_KEY" https://erpps-test.dev.gematik.solutions/actors | jq
```

Appendix A: Release Notes

Release 0.9.0

Initial public release