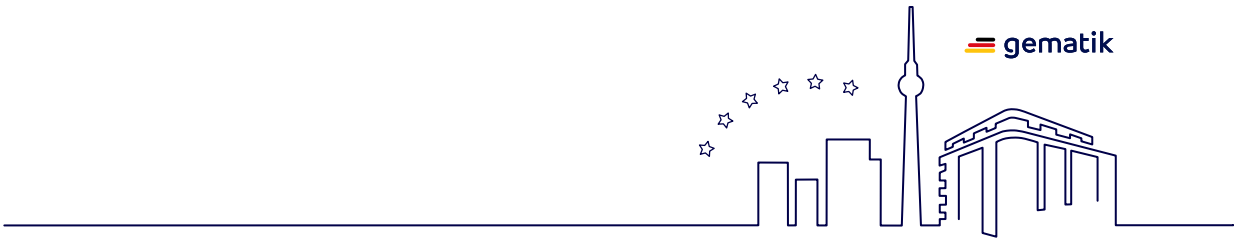
# Erpione 0.3.0
## User Manual

Gematik



# Inhaltsverzeichnis

# 1. Einleitung

Das Kommandozeilen-Werkzeug (CLI) *ERPione* vereinfacht die Kommunikation und Interaktion mit dem *PrimSys-REST*. Die Interaktion mit dem *PrimSys-REST* basiert ausschließlich auf REST und kann auch mit API-Werkzeugen wie cURL und Postman genutzt werden. Diese sind jedoch sehr komplex, haben daher in der Regel eine steile Lernkurve und erfordern darüber hinaus ein technisches Verständnis über die REST-API

Neben cURL und Postman kann man auch den Swagger Editor verwenden, um mit der API zu interagieren. Hierfür benötigt der jeweilige Browser allerdings bereits einen API-Key. Am einfachsten kann dieser über das Browser-Plugin ModHeader (z.B. für Chrome oder Firefox) realisiert werden.

Im Gegensatz zu den generischen API-Werkzeugen bietet *ERPione* den wesentlichen Vorteil, dass sämtliche technische Details abstrahiert werden und den Anwender rein auf die fachliche Anwendungsdomäne fokussiert. Der Anwender muss sich somit nicht mehr um die HTTP-Semantiken kümmern und benötigt nicht einmal genaues Wissen über die REST-Endpunkte. Anwendungsfälle wie z.B. *"E-Rezepte ausstellen oder dispensieren"* stehen ganz klar im Vordergrund.

## 1.1. Motivation

Sowohl *ERPione* als auch *PrimSys-REST* sind interne Werkzeuge - basierend auf der E-Rezept E2E Testsuite - für die Unterstützung der internen Testaktivitäten innerhalb der Gematik.

Wir sehen allerdings auch den Bedarf der Hersteller nach Test-Rezepten und wir wollen die Erprobung des E-Rezeptes durch die Hersteller fördern!

## 1.2. Eponym


Der Name *ERPione* leitet sich als Wortkreuzung von ERP (Abk. E-Rezept) und der griechischen Göttin Epione ab. Als Göttin *für das Lindern des Schmerzes* und Mutter von Hygieia, der Schutzpatronin der Apotheker, ist sie eine ideale Namensgeberin.

# 2. Inbetriebnahme

Die *ERPione* kann in drei Varianten eingesetzt werden. Im einfachsten Fall kann *ERPione* direkt als native und ausführbare Anwendung für das jeweilige Betriebssystem verwendet werden.

Die Container-Variante verwendet für die Ausführung Docker und bietet darüber hinaus zusätzliche Features wie z.B. eine Manpage, die oh-my-zsh und die Integration von erp-cli-fhir.

Die letzte Variante zur Ausführung direkt über die Python Source-files ist möglich, wird jedoch nur für Entwickler und Enthusiasten empfohlen.

Solltest du bereits Docker installiert haben, sollte die Container-Variante deine bevorzugte Wahl sein.

## 2.1. Nativ

Grundsätzlich ist keine besondere Inbetriebnahme notwendig. *ERPione* kann als ausführbare Anwendung direkt und ohne jegliche Installation eingesetzt werden. Aus Gründen der Bequemlichkeit für den Anwender empfiehlt es sich jedoch initial den API-Key in den Umgebungsvariablen einzurichten.

Die ausführbaren Anwendungen müssen für jedes Betriebssystem separat generiert werden. Aktuell wird *ERPione* nativ ausführbar für die folgenden Betriebssysteme bereitgestellt:

- Windows
- Linux
- macOS

## 2.2. Docker

Die *ERPione* kann direkt als Docker-Image aus der Gematik-Registry mit docker pull geladen werden:

```
docker pull gematik1/erpione
```

Nun kann das Image mit dem folgenden Kommando mit docker run gestartet werden:

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

- Mit `-it` (abgekürzt für `--interactive` und `--tty`) wird der Container in einer interaktiven TTY-Session gestartet und der Benutzer landet direkt im Container.
- Der Containername `ERPione` ist frei wählbar
- Mit `--env ERPIONE_API_KEY=$ERPIONE_API_KEY` wird der API-Key im Container als Umgebungsvariable hinterlegt. Hier muss `$ERPIONE_API_KEY` mit dem

tatsächlichen API-Key ersetzt werden, sofern die Umgebungsvariable auf dem Host-Rechner **nicht** hinterlegt ist

- Mit `-v $FULL_HOST_PATH:/home/gematik` wird ein Volume im Container (quasi ein gemeinsamer Ordner zwischen dem Container und dem Host-Rechner) eingebunden um den Austausch von Dateien zwischen dem Container und dem Host-Rechner zu ermöglichen. Falls das nicht benötigt wird, kann dieses Argument auch weggelassen werden.

Durch den `-it`-Schalter wird der Container auch beim Verlassen **nicht** heruntergefahren. Solange der Container läuft, kann man sich immer wieder mit docker exec in den Container einwählen.

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

- Mit `-it` (abgekürzt für `--interactive` und `--tty`) wird der Benutzer in einer interaktiven TTY-Session auf dem Container eingewählt.
- Der Containername `ERPione` der vorher bei `docker run` gewählt wurde
- Das letzte Argument `/bin/zsh` gibt an, welche Shell verwendet werden soll.

## 2.3. Python

Diese Variante wird nur für Entwickler und Python-Enthusiasten empfohlen und benötigt für die Ausführung die Python-Toolchain

- Python 3.11
- pip
- idealerweise venv oder virtualenv

Ein *virtual environment* kann mit dem folgenden Aufruf aus dem Hauptverzeichnis von *ERPione* erstellt und aktiviert werden:

```
python -m venv ERPione_venv

source ERPione_venv/bin/activate
```

Anschließend können innerhalb des *virtual environment* die Abhängigkeiten mit pip installiert werden:

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

## 2.4. API-Key einrichten

Der API-Key wird für jeden Request mit *PrimSys-REST* benötigt. Um diesen nicht bei jeder einzelnen Operation mitgeben zu müssen, gibt es die Möglichkeit den API-Key permanent als Umgebungsvariable abzulegen.

Hierzu muss der API-Key unter der Umgebungsvariable `ERPIONE_API_KEY` im jeweiligen Betriebssystem hinterlegt werden. Dieses Prozedere unterscheidet sich allerdings zwischen den jeweiligen Betriebssystemen. Hierzu existieren unzählige

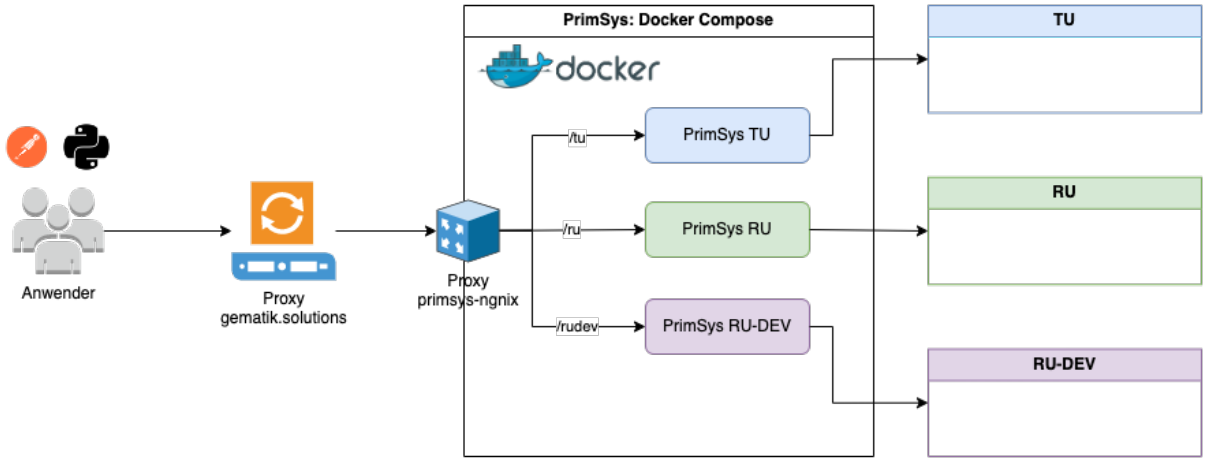
Anleitung im Internet z.B. für Windows und Linux.

## 2.5. Standardumgebung einrichten

Die jeweilige Umgebung (TU, RU, RU-DEV) kann bei jedem Request separat über den Schalter `--env` angegeben werden. Um diesen Schalter nicht bei jedem einzelnen Aufruf mit angeben zu müssen, kann man über die Umgebungsvariable `ERPIONE_ENV` eine Standardumgebung definieren. Die Standardumgebung lässt sich dann weiterhin über `--env` überschreiben.

# 3. Architektur

Während der *PrimSys-REST* als eigenständiges Produkt existieren kann, kann *ERPione* nur in Kombination mit selbigem betrieben werden. Nach dem Client-Server-Modell stellt *PrimSys-REST* das Backend und *ERPione* das Frontend dar.




## 3.1. Backend

Das Backend übernimmt sämtliche Aufgaben wie z.B. die Kommunikation mit der TI, die Verarbeitung von FHIR oder die Erstellung von kryptografischen Signaturen.

## 3.2. Frontend

Das Frontend übernimmt, wie bereits eingangs erwähnt, die Abstraktion der technischen Details, die Aufbereitung der Ergebnisse und weitere Hilfsfunktionalitäten um die Benutzung zu vereinfachen.

Das Frontend selbst ist dabei relativ schlank (Thin Client). Daher enthält es selbst kaum bis kein fachliches Domänenwissen wie z.B. über FHIR oder das E-Rezept selbst, sondern dient lediglich als *Vermittler* zwischen dem Anwender und dem *PrimSys-REST*.

## 3.3. Proxy

Der *PrimSys-Proxy* ist für den Anwender nahezu transparent übernimmt aber zwei essenzielle Aufgaben: nämlich das Routing in die jeweiligen Umgebungen und die Zugangskontrolle zu dem eigentlichen Service über API-Keys.

# 4. Verwendung

Nachfolgend werden die möglichen Anwendungsfälle von *ERPione* skizziert und die CLI Argumente erläutert

In den nachfolgenden Beispielen wird die ausführbare Anwendung in der Kommandozeile als `erpione` benannt. Das kann jedoch abhängig vom verwendeten Betriebssystem variieren.

## 4.1. Hilfe

Der `-h` (bzw. `--help`) Schalter ist die erste Anlaufstelle um eine erste Hilfe über die CLI Argumente zu erhalten.

So kann man sich z.B. über die folgende Hilfe einen Überblick über die vorhanden Sub-Kommandos verschaffen:

```
erpione --help
```

Die jeweilige Hilfe zu einem bestimmten Sub-Kommando erhält dann folgendermaßen:

```
erpione SUB-COMMAND --help
```

Konkret für das `scenario` Sub-Kommando würde der Aufruf dann z.B. die folgende Hilfestellung zur Verwendung liefern:

```
erpione scenario --help

usage: erpione scenario [-h] [-e {tu,ru,rudev}] [-s SERVER]
                        [--api-key API_KEY | --api-key-file API_FILE]
                        scenario
...
```

## 4.2. Allgemeine Schalter

Die meisten Sub-Kommandos haben eine gemeinsame Basis an Schaltern für den *PrimSys-REST*:

Table 1. common arguments

| short | long | Beschreibung |
|---|---|---|
| -e | --env | Auswahl der jeweiligen TI-Umgebung. Dieser Schalter überschreibt die Standardumgebung. Wird die Umgebung weder über `-e` noch über die Standardumgebung angegeben, dann wird auf den Default auf dem *PrimSys-Proxy* (aktuell TU) zurückgegriffen. |
| -s | --server | Der Default entspricht dem internen Hosting innerhalb der gematik und braucht in der Regel nicht verändert zu werden |
| | --api-key | Kann verwendet werden, um die Umgebungsvariable zu überschreiben bzw. einen setzen, falls keine Umgebungsvariable gesetzt wurde |
| | --api-key-file | Kann verwendet werden, um die Umgebungsvariable zu mit einem API-Key aus einer Datei zu überschreiben bzw. einen setzen, falls keine Umgebungsvariable gesetzt wurde |

## 4.3. Meta-Informationen abrufen

Um Informationen über den Service abzurufen, kann der folgende Aufruf verwendet werden:

```
erpione info
```

Zusätzlich können die aktiven Akteure über den folgenden Aufruf abgerufen werden:

```
erpione actors
```

## 4.4. Verordnungsdatensätze generieren

Es kann nötig sein, FHIR-Verordnungsdatensätze (KBV-Bundles) zu generieren. Hierfür ist in dem Container auch erp-cli-fhir installiert. Für das nachfolgende Beispiel zum Ausstellen von mehreren E-Rezepten können die FHIR-Verordnungsdatensätze folgendermaßen generieren werden:

```
erpf generate kbvbundle -n5 /home/gematik/examples/kbv_bundles
```

Damit werden unter /home/gematik/examples/kbv_bundles 5 (`-n5`) zufällige KBV-Bundles generiert. Diese werden lediglich lokal angelegt und noch **nicht** an *PrimSys-REST* oder den Fachdienst gesendet.

## 4.5. E-Rezept ausstellen

Um E-Rezepte direkt über KBV-Bundles z.B. in der TU auszustellen, kann das folgende Kommando verwendet werden:

```
erpione prescribe -e TU /home/gematik/examples/kbv_bundles
```

Wenn ein komplettes Verzeichnis (wie im vorherigen Beispiel) angegeben, dann werden alle KBV-Bundles innerhalb dieses Verzeichnisses hintereinander ausgestellt.

Man kann aber auch nur eine einzelne Verordnung (hier exemplarisch in der RU) ausstellen, indem man das jeweilige KBV-Bundle direkt adressiert:

```
erpione prescribe -e RU /home/gematik/examples/gkv_bundle_01.xml
```

Wenn für die ausgestellten E-Rezepte der dazugehörende Data Matrix Code (DMC) benötigt wird, dann kann der `--dmc` Schalter verwendet werden. Damit wird für jedes E-Rezept zusätzlich der DMC heruntergeladen und als PNG-Bilddatei im Output-Verzeichnis mit dem Namensschema `dmc_{TASK_ID}.png` abgelegt.

```
erpione prescribe --dmc /home/gematik/examples/gkv_bundle_01.xml
```

Im letzten Beispiel wurde der optionale Schalter `-e` aus Gründen der Einfachheit weggelassen. Das E-Rezept wird dann in der *default Umgebung* ausgestellt, die sich in der Zukunft auch ändern kann. Für den automatisierten Einsatz (z.B. in Scripten) wird empfohlen die Umgebung immer explizit anzugeben.

## 4.6. E-Rezepte akzeptieren

E-Rezepte, die über *PrimSys-REST* ausgestellt wurden, können auf der Apotheken-Seite mit dem folgenden Kommando akzeptiert werden:

```
erpione accept --taskid [TASK_ID]
```

Der Parameter `--accesscode` ist in diesem Fall optional, weil *PrimSys-REST* diesen bereits kennt. Anders sieht das für E-Rezepte aus, die nicht über *PrimSys-REST* ausgestellt wurden bzw. ältere E-Rezepte die nicht mehr im internen Speicher von *PrimSys-REST* liegen. Hier muss der AccessCode dann explizit angegeben werden:

```
erpione accept -t [TASK_ID] --accesscode [ACCESS_CODE]
```

Akzeptierte E-Rezepte werden im Fachdienst in den Status `in-progress` geschoben und sind dort für die jeweilige Apotheke *reserviert*, um genau zu sein, für alle *reserviert* die das generierte **Secret** kennen.

Vergessene Secrets können ein Problem darstellen, da solche E-Rezepte nicht weiter bearbeitet werden können und erst nach 90 Tagen automatisch vom Fachdienst entfernt werden

Um dem vorzubeugen, sollten die E-Rezepte nach dem `accept` möglichst weiterverarbeitet werden, (z.B. dispensiert oder gelöscht werden)

```
erpione abort -t [TASK_ID] -a [ACCESS_CODE] --secret [SECRET]
```

Analog zum `accept` werden der AccessCode und das Secret hier nur für *extern* generierte E-Rezepte benötigt.

## 4.7. E-Rezepte dispensieren

Um ein E-Rezept dispensieren zu können, muss vorher zwingend `accept` ausgeführt werden, damit der Fachdienst das E-Rezept in den Status `in-progress` schieben und ein Secret generieren kann. Anschließend kann das E-Rezept mittels folgendem Kommando dispensiert werden:

```
erpione dispense -t [TASK_ID]
```

Analog zum `accept` wird das Secret hier nur für *extern* generierte bzw. akzeptierte E-Rezepte benötigt.

Zusätzlich besteht die Möglichkeit bei der Dispensierung ein Ersatzmedikament bzw. mehrere Ersatzmedikamente anzugeben. Dafür kann die Option `--body-file` verwendet werden, um ein JSON-File mit den Ersatzmedikamenten zu übergeben.

```
erpione dispense -t [TASK_ID] --body-file /home/gematik/examples/dispense_body.json
```

Der Inhalt der `dispense_body.json` könnte dabei dann ungefähr folgenden Inhalt haben, bei dem mehr als nur ein Ersatzmedikament für dieses Rezept ausgegeben wird:

```
[
  {
    "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"
    }
  }
]
```

## 4.8. E-Rezept-Informationen abrufen

Für das Abrufen von E-Rezept-Informationen steht das Kommando `fetch` zur Verfügung.

Mit diesem Kommando werden **nicht** die tatsächlichen E-Rezepte vom Fachdienst abgerufen. Im Grunde kann der *PrimSys-REST* das auch nicht leisten. Stattdessen werden über `fetch` nur E-Rezepte abgerufen, die über den *PrimSys-REST* abgewickelt wurden.

Um E-Rezept-Informationen in der jeweiligen Umgebung abzurufen, können die folgenden Kommandos verwendet werden:

```
erpione fetch open -e RU
erpione fetch accepted -e TU
erpione fetch accepted --pretty
```

Der Schalter `--pretty` ist für alle Sub-Kommandos von `fetch` verfügbar. In der Ausgabe auf dem Terminal kann das allerdings dazu führen, dass Scrollback-Buffer des Terminals nicht ausreicht und die führenden Zeilen abgeschnitten werden. Als möglicher Workaround (neben den nachfolgenden Optionen) kann der Output an `less` weitergereicht werden `erpione fetch accepted`

`--pretty | less`

Darüber hinaus können über `fetch` auch E-Rezept anhand der KVNR über `--kvnr <KVNR>` werden:

```
erpione fetch open --kvnr T027348190 --pretty
erpione fetch accepted --kvnr K010140999 --pretty
erpione fetch accepted --kvnr A010940096 --pretty
```

Die Ausgabe der E-Rezept informationen kann auch in eine Datei geschrieben werden:

```
erpione fetch open --env TU --kvnr T027348190 --pretty /home/gematik//tu/open_T027348190.json
erpione fetch accepted --env RU --kvnr K010140999 --pretty ./ru
erpione fetch accepted --kvnr A010940096 --pretty .
```

Als Ausgabe-Parameter kann hierbei sowohl eine konkrete Datei oder einfach nur ein relativer oder absoluter Pfad zu einem Verzeichnis angegeben werden. Wird also z.B. nur `./ru` als Ausgabe-Verzeichnis angegeben, dann wird die Ausgabe unter `./ru/{STATUS}_{TIMESTAMP}.json` abgelegt.

## 4.9. Communications

Aus der Sicht des AVS wird auf dem *PrimSys-REST* aktuell nur das Versenden der CommunicationReply unterstützt.

Das Versenden einer Antwort-Nachricht, ohne dass vorher ein Versicherter eine Anfrage (z.B. CommunicationInfoReq oder CommunicationDispReq) gestellt hat, ist in der Praxis nicht direkt möglich oder zumindest unüblich.

Mit der *ERPione* ist das nur möglich weil Test-Apotheken im *PrimSys-REST* sowohl auf die Task-IDs als auch die Besitzer (KVNR) aller E-Rezepte Zugriff haben (siehe E-Rezept-Informationen abrufen.

### 4.9.1. Antwort-Nachricht verschicken

Eine Antwort-Nachricht an einen Versicherten zu einem E-Rezept kann mit dem folgenden Kommando verschickt werden:

```
erpione communication reply --taskid [TASK_ID] --kvnr [KVNR] [path/to/payload.json]

erpione communication reply --taskid 160.000.006.460.306.30 --kvnr K220635158
/home/gematik/examples/communication_reply_payload_01.json
```

Wurde das E-Rezept ebenfalls über *PrimSys-REST* ausgestellt (und ist noch im Zwischenspeicher siehe E-Rezept-Informationen abrufen), so kann die *ERPione* die KVNR des Empfängers auch selbst aus den E-Rezept-Informationen ableiten. In diesem Fall kann die Angabe der KVNR auch ausgelassen werden:

```
erpione communication reply --taskid [TASK_ID] [path/to/payload.json]

erpione communication reply --taskid 160.000.006.460.306.30
/home/gematik/examples/communication_reply_payload_01.json
```

Das positionale Argument `[path/to/payload.json]` gibt den Pfad zu dem JSON-Payload gemäß der Anforderung **A_23879** aus **gemSpec_FD_eRp** an. Ein `payload.json` könnte dabei dann ungefähr folgenden Inhalt haben:

```
{
  "version": 1,
  "supplyOptionsType": "onPremise",
  "info_text": "Die Abholung ist ab sofort vor Ort möglich",
  "url": "https://www.adler-apotheke-berlin-tegel.de/",
  "pickUpCodeHR": "12345",
  "pickUpCodeDMC": "6a3acd69-a01d-4780-885e-ea970b6aacdb"
}
```

### 4.9.2. Nachrichten suchen

Um Nachrichten vom E-Rezept Fachdienst abzurufen, kann das Kommando `communication search` mit unterschiedlichen Suchparametern verwendet werden.

So ist es z.B. möglich Nachrichten für eine bestimmte Apotheke ohne Filter abzurufen:

```
erpione communication search --actor [ACTOR_ID | ACTOR_NAME]

erpione communication search --actor "am flughafen" --pretty
```

Damit werden **nicht** zwangsläufig alle vorhanden Nachrichten auf dem Fachdienst abgerufen, weil hier ein Paging-Mechanismus (siehe A_19534-01) zum Einsatz kommt. Das bedeutet, dass für diesen Aufruf maximal 50 Nachrichten zurückgeliefert werden.

Um die Nachrichten an einen bestimmten Empfänger abzurufen, kann der Parameter `--receiver` für die Filterung verwendet werden. Im nachfolgenden Aufruf sucht die Apotheke "Am Flughafen" (hier referenziert über die Akteur-ID) nach allen Nachrichten welche die KVNR `X110498565` als Empfänger haben.

```
erpione communication search --actor [ACTOR_ID | ACTOR_NAME] --receiver [KVNR]

erpione communication search --actor "886c6eda7dd5a1c6b1d112907f544d3" --receiver X110498565
--pretty
```

Der Suchfilter `--receiver` in Kombination mit einer KVNR impliziert, dass mit diesem Aufruf nur `REPLY`-Nachrichten gefunden werden können. Um die Nachrichten in die entgegengesetzte Kommunikations-Richtung (Versicherter an Apotheke) abrufen möchte, muss der Suchfilter `--sender` verwendet werden:

```
erpione communication search --actor [ACTOR_ID | ACTOR_NAME] --receiver [KVNR]
```

```
erpione communication search --actor "886c6eda7dd5a1c6b1d112907f544d3" --sender X110498565 --pretty
```

Theoretisch ist es auch möglich sowohl `--sender` als auch `--receiver` in einem Aufruf zu kombinieren. In diesem Fall muss dann aber einer der beiden Suchfilter zwangsläufig die Telematik-ID der Apotheke aus dem `--actor` entsprechen, anderenfalls wird der E-Rezept Fachdienst (impliziert durch A_19520) eine leere Liste an Nachrichten zurückliefern.

## 4.10. ChargeItems

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:

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

### 4.10.1. ChargeItem erstellen

Um ein ChargeItem zu erstellen, kann das folgende Kommando verwendet werden:

```
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
```

### 4.10.2. ChargeItem ändern

Um ein bereits bestehendes ChargeItem zu ändern, kann das folgende Kommando verwendet werden:

```
erpione chargeitem change --taskid [TASK_ID] --accesscode [ACCESS_CODE] [path/to/payload.json]

erpione chargeitem change -t 200.000.002.175.581.70 -a
9ea09db15e04fe9b1613cd0a338b812c728b355d672d9af8e54b7596e9098fa8
```

```
./resources/my_examples/chargeitem_payload.json
```

Der AccessCode hat hier in diesem Anwendungsfall die gleiche Struktur und denselben Namen wie der AccessCode einer Prescription. Die beiden Werte sind jedoch unterschiedlich und sollten nicht miteinander verwechselt werden.

Der AccessCode für das ChargeItem wird vom Fachdienst jeweils beim `create` und `change` neu generiert und muss vom FdV nach jeder Änderung neu abgerufen werden.

# 5. Scenario Player

Der *Scenario Player* ermöglicht es mit der *ERPione* ganze Abläufe wiederholbar zu beschreiben. Anstatt also ein bestimmtes Szenario immer und immer wieder über mehrere Kommando-Aufrufe manuell durchzuführen, empfiehlt es sich hierfür ein *"Drehbuch"* wie in dem folgenden Beispiel anzulegen:

```
{
  "meta": {
    "description": "Ein einfaches Szenario für Demo-Zwecke",
    "name": "Test Szenario"
  },
  "steps": [
    {
      "type": "create",
      "description": "create an initial prescription",
      "body": {
        "patient": {
          "kvnr": "X123456789"
        },
        "medication": {
          "category": "00"
        }
      }
    },
    {
      "type": "create",
      "description": "create an initial prescription",
      "file": "kbv_bundle.xml"
    },
    {
      "type": "bool_question",
      "description": "Warten bis der Task in der App angekommen ist",
      "instruction": "Ist der Task ${prescriptions[0].task_id} mit ${prescriptions[0].access_code}
in der App angekommen?"
    },
    {
      "type": "accept",
      "description": "Akzeptiere Task ${prescriptions[0].task_id}",
      "idx": 0
    }
  ]
}
```

Um das Szenario z.B. in der RU abzuspielen, kann das folgende Kommando verwendet werden:

```
erpione scenario -e RU /home/gematik/examples/gkv_scenario.json
```

## 5.1. Meta

Jedes Szenario definiert Meta-Informationen wie eine kurze Beschreibung und einen Namen für das Szenario.

Im Beispiel `"meta": {…}`

## 5.2. Steps

Jedes Szenario besteht aus einer Liste von einzelnen Schritten, die

hintereinander durchgeführt werden.

Im Beispiel `"steps": […]`

Aktuell sind folgende Typen von Steps implementiert: - `bool_question` die eine Frage stellt und pausiert bis der Anwender reagiert - `create` um E-Rezepte zu erstellen - `accept` um E-Rezepte zu akzeptieren - `dispense` um E-Rezepte zu dispensieren - `create_charge_item` um für ein dispensiertes PKV-Rezept einen Abrechnungsdatensatz zu erstellen - `change_charge_item` um ausgestellt Abrechnungsdatensätze zu ändern

# 6. Changelog

## 0.5.0

- Implement `chargeitem` subcommands as requested by GitHub Issue 9

## 0.4.2

- Bugfix ANFERP-2680: on `reject`-command falsely an `abort` was performed which moved the prescription into the wrong state

## 0.4.1

- Increase default timeout to 60 seconds
- Make timeout configurable via environment variable `ERPIONE_REQ_TIMEOUT`

## 0.4.0

- Update Python to `3.12`
- Implement `reject`-command which allows giving back already accepted prescriptions

## 0.3.0

- Kommandos für `communication search` und `communication delete` implementiert
- CLI Parameter `--actor` implementiert um explizit Akteure referenzieren zu können

## 0.2.3

- Bugfix: Der Body für `dispense` wird als JSON-String eingelesen (GitHub Issue 4)
- Bugfix: Beim `dispense`-Schritt im Scenario-Player wird das Ersatzmedikament korrekt gesetzt (GitHub Issue 4)

## 0.2.2

- Umgebungsvariable `ERPIONE_ENV` eingeführt um eine Standardumgebung

vorgeben zu können

- Subkommando `communication reply` implementiert, um Antwort-Nachrichten als Apotheke zu verschicken
- Bugfix: Darstellung der Notiz aus der MedicationRequest in der .xlsx-Zusammenfassung
- Bugfix: KVNR-Filter bei `fetch accepted` (GitHub Issue 2)

## 0.2.1

- Initiale öffentliche Version mit Basis-Features

# Glossar

**API**
Application Programming Interface

**CLI**
Command Line Interface

**DMC**
Data Matrix Code

**FHIR**
Fast Healthcare Interoperability Resource

**PNG**
Portable Network Graphics

**REST**
Representational State Transfer

**TI**
Telematik Infrastruktur