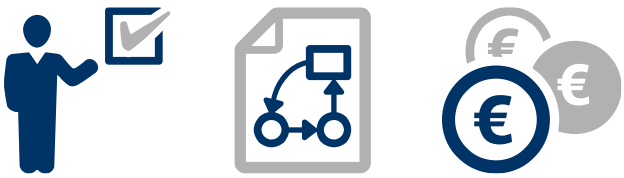
Integrationshandbuch

ePayBL-Konnektor

Version 1.36



DRESEARCH
DResearch Digital Media Systems GmbH

Änderungshistorie

Version	Datum, Autor	Status, Änderung
1.0	29.08.2016, Klaus Peters	initial
1.1	13.9.2016, Klaus Peters, Michael Thiele, Susanne Hirth	Einarbeitung von Review-Kommentaren vom SID
1.2	6.10.2016, Klaus Peters	Einarbeitung der Review-Kommentare der EG

1.3	6.10.2016, Klaus Peters	Anpassung des Datenmodells, optionale Parameter:
		• Buchungsliste
		  ○ Kassenzeichen
		  ○ Transaktionsnummer
		• Buchung
		  ○ id
		• Kunde
		  ○ name
		  ○ emailadresse

1.4	21.12.2016, Klaus Peters, Sigrid Taeschner, Michael Thiele, Erik Schwarzenecker	Anpassungen Konnektor-API, Testframework, Client-Bibliothek.
1.5	13.12.2016 ,Klaus Peters	Zusammenfassung der Dokumente
1.6	3.2.2017, Klaus Peters	Aktualisierung, Client-Bibliotheken PHP, .NET
1.7	9.2.2017, Klaus Peters	Installationshinweis ergänzt
1.8	16.2.2017, Klaus Peters	Administrationshinweise ergänzt, Datenmodell korrigiert (ZV_FM-1059, ZV_FM-1065)
1.9	20.2.2017, Klaus Peters	Anpassungen bzgl. ZV_FM-1065, ZV_FM-1079, ZV_FM-1080, ZV_FM-1084
1.10	21.4.2017, Klaus Peters	Anpassungen bzgl. PayDirekt
1.11	28.4.2017, Klaus Peters	Beschreibung der Architektur angepasst

Version	Datum, Autor	Status, Änderung
1.12	19.5.2017, Klaus Peters, Erik Schwarzenecker	Ergänzungen zu Pflichtparametern, Zertifikatshandling
1.13	07.06.2017, Michael Thiele	Ergänzung zu Pflichtparametern
1.14	28.06.2017, Michael Thiele	Ergänzung zu Pflichtparametern
1.15	05.07.2017, Erik Schwarzenecker	Löschen von (Bestands-) Kunden
1.16	24.8.2017, Klaus Peters	Nummerierung der Methoden
1.17	1.9.2017, Klaus Peters	Update der Beschreibung der Bibliotheken bzgl. Löschen von Kunde
1.18	11.9. 2017, Klaus Peters	Struktur des SEPA-Mandats
1.19	18.10. 2017, Klaus Peters	Überarbeitung der Client-Libs für PHP und C#
1.20	29.11.2017, Michael Thiele	Zahlverfahren LastschriftOhne
1.21	20.12.2017, Michael Thiele	Zahlverfahren LastschriftOhne
1.22	30.1.2018, Klaus Peters	Beispielbuchungsliste korrigiert (ZV_FM-1380) 17050030, ZV_FM-1372: BIC-Only bei Giropay über Giro-Solution
1.23	23.7.2018, Klaus Peters	ZV_FM-1495, Nachname des Kunden ist Pflicht, sobald Adresse oder BV angegeben sind.
1.24	24.8.2018, Klaus Peters	Paypage 4.0 (18050004) (Änderung des Datenmodells: Kapitel 2.2.3.9; Kapitel 2.2.4.2)
1.25	4.10.2018, Klaus Peters	Kapitel 2.2.3.2 Buchungstext
1.26	12.12.2018, Klaus Peters	Zertifikate in den Bibliotheken
1.27	8.1.2019, Klaus Peters	Einarbeitung von Review Kommentaren
1.28	2.7.2019, Klaus Peters	PHP-Konnektor: Übergabe von Keystores

Version	Datum, Autor	Status, Änderung
1.29	16.7.2019, Klaus Peters	Proxy-Konfiguration für die Client-Bibliotheken (Java, .NET, PHP)
1.30	21.8.2019, Klaus Peters	WADL-Generierung.
1.31	15.10.2019, Klaus Peters	Abfrage der Zusatzdaten bei Lesen der Kassenzeicheninfo. Beschreibung des Ergebnisses von Kassenzeicheninfo Java 11 Umgebung für die Bibliothek
1.32	5.11.2019, Klaus Peters	Abfrage der Zusatzdaten bei Lesen der Kassenzeicheninfo, Zugriff auf saldo (über NumericNode)
1.33	21.11.2019, Klaus Peters	ZV_AM-1232 - Konnektor - Zahlungsmethode abfragen
1.34	17.3.2020, Klaus Peters	ZV_AM-1228 – Anzeige Saldos des Kassenzeichens ZV_AM-1230 – Separierung Anlegen und Aktivieren von Kassenzeichen ZV_AM-1329 – Anzeige Transaktionsnummer und Zahlungsvorgangs IDs
1.35	7.4.2020, Klaus Peters, Ines Peters	ZV_AM-1384: Separierung der REST-Bibliotheken
1.36	13.5.2020, Klaus Peters, Ines Peters	Umgang mit alphanumerischen Kassenzeichen

Tabelle 1: Änderungshistorie

Inhaltsverzeichnis

1 Einleitung

1.1 Fachliche Einordnung

Für die Abwicklung von Zahlungen mittels Kreditkarte und giropay bedient sich die ePayBL eines externen Zahlungsverkehrsproviders (ZV-Provider). ZV-Provider-Schnittstellen wie Payplace, Saferpay oder GiroCheckout können

- durch ein bestehendes Fachverfahren entweder in einer Direktintegration selbst implementiert oder
- über die ePayBL genutzt werden.

Für die Zahlungspflichtigen unterscheiden sich die beiden Varianten in der Auswahl oder Bestätigung eines Zahlverfahrens sowie in der Anzeige des Ergebnisses der Zahlung. Je nach gewählter Variante geschieht dies im Fachverfahren selbst oder auf der ePayBL-Paypage. Das Aussehen der Paypage ist anpassbar, gleicht optisch aber in der Regel nicht vollständig dem Fachverfahrens, so dass Zahlungspflichtige hier einen Website-Sprung erleben: Fachverfahren → Paypage → ZV-Provider → Fachverfahren. Fachverfahren erleben dagegen weit größere Unterschiede:

- Direktintegration: Neben der Kommunikation mit der ePayBL muss das Fachverfahren hier auch die Aufrufe zum ZV-Provider selbst abbilden. Das betrifft die Auswertung und Behandlung von Antworten der ZV-Provider-Schnittstelle sowie der ePayBL. So entsteht – nicht zuletzt aufgrund der komplexen Zahlprozesse – ein hoher Entwicklungsaufwand, der sich zusätzlich mit jeder Änderung der ZV-Provider-Schnittstelle weiter erhöht. Da aber bei dieser Nutzungsart während der Zahlung keine zusätzliche Seite zwischen Fachverfahren und ZV-Provider angezeigt wird, stellt sich der Zahlprozess für die Zahlungspflichtigen weiterhin als "vertrautes" Verfahren dar. Ein weiterer Vorteil der Direktintegration ist die direkte Übermittlung der Zahlungsbestätigung an das Fachverfahren, wodurch das Abfragen des Ergebnisses der Zahlung bei der ePayBL entfällt.

- ePayBL-Paypage: Bei Nutzung der ePayBL-Paypage hingegen muss das Fachverfahren lediglich die URLs der Paypage aufrufen und behandeln sowie das Ergebnis der Zahlung bei der ePayBL abfragen. Die Kommunikation mit dem ZV-Provider erledigt in diesem Fall die Paypage der ePayBL. Im Vergleich zur Direktintegration ist der Aufwand für Entwicklung und Wartung bei dieser Nutzungsart geringer. Dafür müssen Zahlungspflichtige jedoch den zusätzlichen Website-Sprung während der

Bezahlung in Kauf nehmen. Da die Paypage nach außen sichtbar ist und unter Umständen Ziel von Angriffen werden kann, müssen die verwendeten Technologien stets dem aktuellen Stand der Technik entsprechen. Da entsprechendes Monitoring und Aktualisierungen sehr aufwendig sind, ist der ePayBL-Konnektor entwickelt worden, welcher die Kommunikation zur ePayBL sowie zum ZV-Provider für das Fachverfahren kapselt, ohne für die Zahlungspflichtigen (oder Angreifer) sichtbar zu sein.

Ziel des ePayBL-Konnektors ist die Bündelung der Aufrufe der ePayBL sowie der ZV-Provider-Schnittstelle, um die Implementierung der Zahlungsprozesse in ein bestehendes Fachverfahren zu vereinheitlichen und deutlich zu vereinfachen. Der ePayBL-Konnektor wird damit als unsichtbares Bindeglied zwischen ePayBL und Fachverfahren alle Schritte, die für die Abwicklung einer Online-Zahlung notwendig sind, übernehmen und das Fachverfahren mit Informationen zur Zahlung versorgen. Dazu gehören detaillierte Informationen, wie z.B. Zeitpunkt und Ergebnis der Zahlung, ePayBL-Kassenzeichen und Betrag. Da der ePayBL-Konnektor Bibliotheken in verschiedenen Entwicklungssprachen wie JAVA, .NET oder PHP zur Verfügung stellt, kann die Schnittstelle einfach und mit sehr geringem Aufwand in bestehende Fachverfahren eingebunden werden.

Damit lässt sich der Aufwand für Implementierung und Wartung der ePayBL-Schnittstelle wirksam verringern. Aspekte wie Layout oder Unterstützung verschiedener Sprachen müssen zudem nicht betrachtet werden, da der Konnektor keine sichtbare Oberfläche bietet. Mit dieser "versteckten" Anbindung an die ePayBL lassen sich außerdem zusätzliche Website-Sprünge während der Zahlung vermeiden und das Risiko für Angriffe von außen verringern. Der Zahlprozess folgt darüber hinaus dem Design des Fachverfahrens, da die Auswahl des Zahlverfahrens und die Aufbereitung der Bezahlinformationen für die Zahlungspflichtigen aufgrund der fehlenden Oberfläche des ePayBL-Konnektors durch das Fachverfahren realisiert werden. Alle anderen Schritte des Zahlprozesses übernimmt der Konnektor.

Der ePayBL-Konnektor ermöglicht durch seine Unabhängigkeit von der verwendeten ePayBL-Version den Umstieg auf höhere Versionen ohne Auswirkungen auf den Zahlprozess und ohne Anpassungen im Fachverfahren.

Die folgenden beiden Abbildungen zeigen die bisherige Modulstruktur inklusive der Hauptkommunikationswege und die neue Struktur unter Einbeziehung des Konnektors.

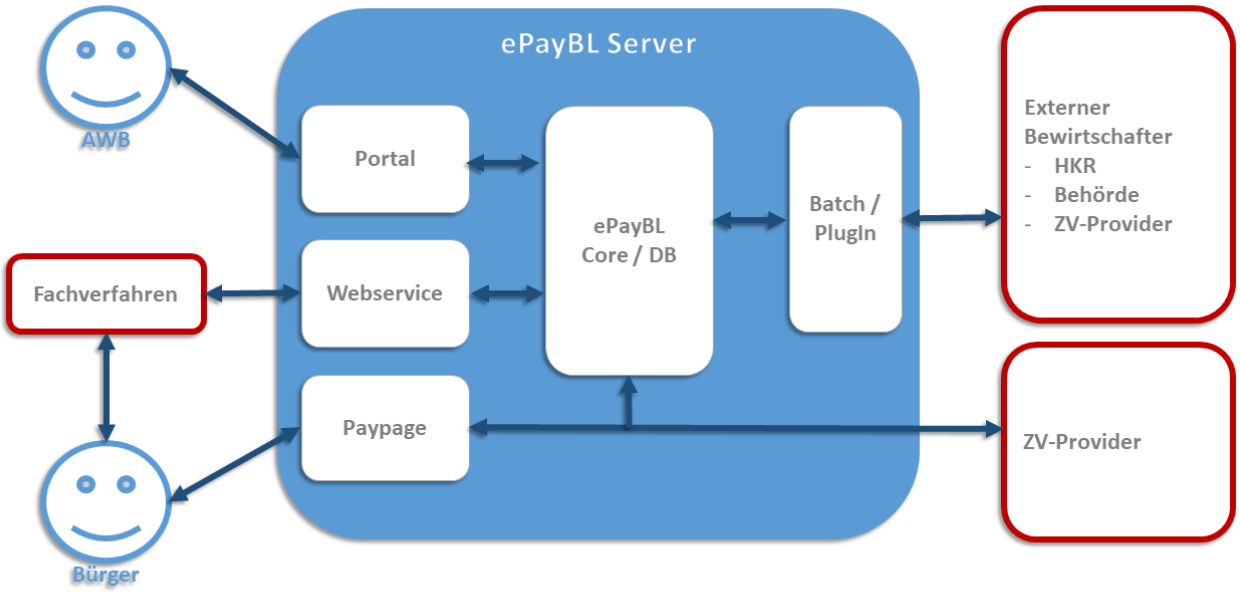


Abbildung 1: Bisheriger Aufbau der ePayBL

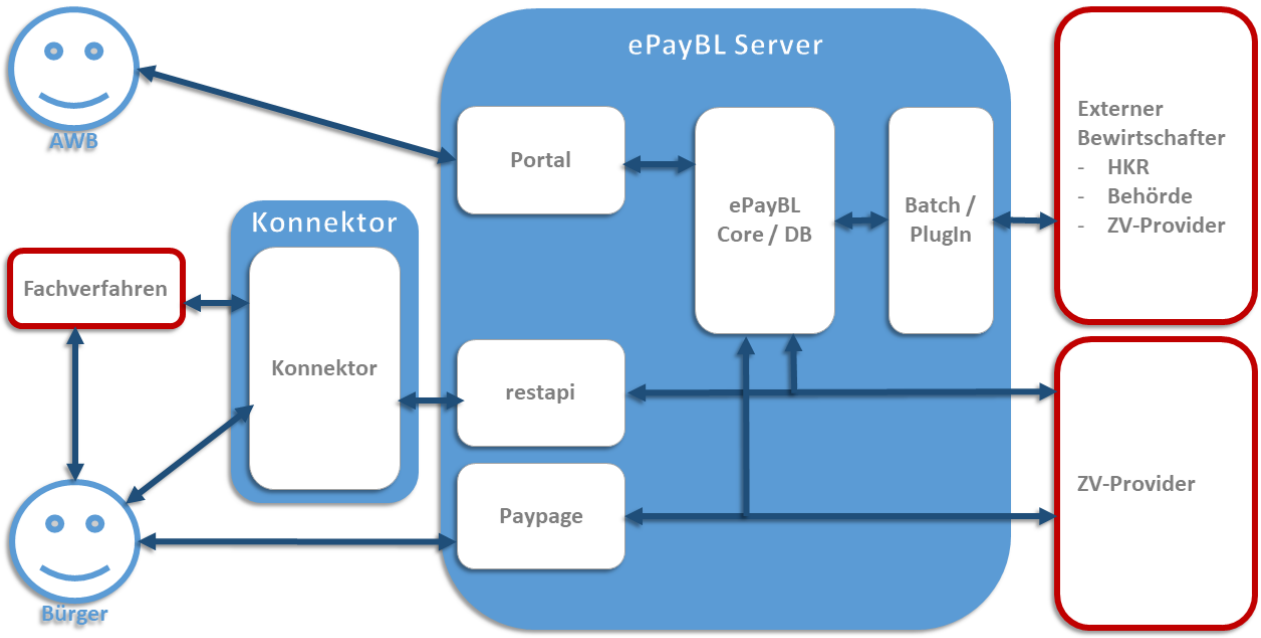


Abbildung 2: Neuer Aufbau der ePayBL mit Konnektor

1.2 Systemarchitektur

Die folgende Abbildung zeigt die Anbindung des ePayBL-Konnektors als Teil der ePayBL an das Fachverfahren, an den ePayBL-Kern und an die ZV-Provider.

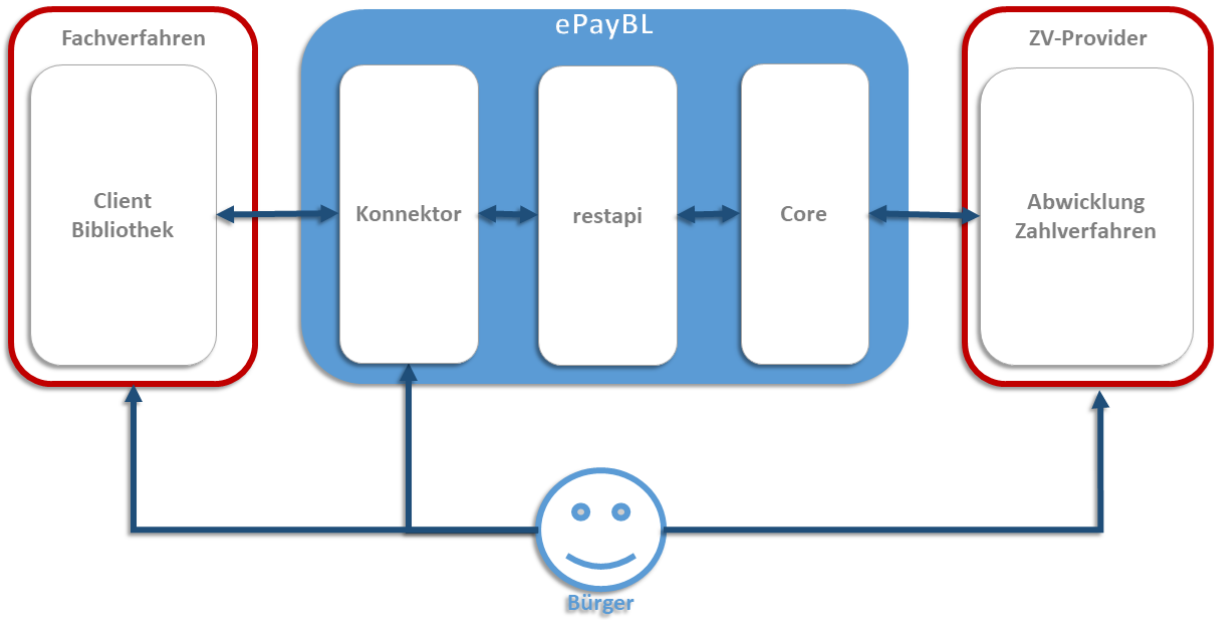


Abbildung 3: Kommunikation des ePayBL-Konnektors

Die komplette Kommunikation der Fachanwendung mit der ePayBL erfolgt über den ePayBL-Konnektor. Der ePayBL-Konnektor ist eine eigenständige Web-Applikation, die im Gegensatz zu anderen ePayBL-Modulen auf einem eigenen Server installiert werden kann. Um Anpassungen der ePayBL zu kapseln, kommuniziert der ePayBL-Konnektor über das restapi-Modul mit dem Kern der ePayBL. Dieses Modul enthält die notwendigen Anpassungen der ePayBL für die Kommunikation mit dem ePayBL-Konnektor. Diese Separierung erleichtert die Anbindung des Konnektors an verschiedene Linien der ePayBL. Die Kommunikation zwischen Fachanwendung und ePayBL-Konnektor sowie zwischen ePayBL-Konnektor und ePayBL (restapi-Modul) erfolgt über REST-Services. Um möglichst versionsunabhängig zu sein, leitet der ePayBL-Konnektor die Requests der Fachanwendung in generischer Weise direkt zur ePayBL weiter. Auf diese Weise kann die Implementierung des Konnektors selbst sehr schmal gehalten werden. Die komplette Verarbeitung und Verifizierung des Requests erfolgt im restapi-Modul der ePayBL. Änderungen und Erweiterungen der Konnektor-Funktionalität müssen so nur in der ePayBL (restapi-Modul) gemacht werden. Die Einfachheit des Konnektors ermöglicht es auch, dass diese Web-Applikation auf einem eigenen Server installiert werden kann. Dies wiederum erhöht die Sicherheit der Anbindung der ePayBL.

Wenn für die Kommunikation mit der ePayBL Client-Zertifikate notwendig sind, muss die Fachanwendung so konfiguriert werden, dass sie in den Requests an den Konnektor SSL-Zertifikate mitschickt. Der Konnektor leitet diese Requests an die ePayBL weiter, wobei das SSL-Zertifikat der Fachanwendung als separates Objekt im Request des ePayBL-Konnektors an die ePayBL mitgeschickt wird. Die Prüfung, ob dieses Zertifikat zum Mandanten passt, erfolgt dann in der ePayBL. Für die Kommunikation zwischen ePayBL-Konnektor und ePayBL verwendet der Konnektor ein eigenes festes Zertifikat, da der Konnektor auf einem eigenen Server außerhalb des Servers des ePayBL-Kerns laufen kann.

Wenn das Fachverfahren eine Zahlung abwickelt möchte, überträgt es die Bezahldaten über den Konnektor an das restapi-Modul. Dieser führt die notwendige Kommunikation mit dem ZV-Provider durch und sendet die URLs des ZV-Providers, über die der Kunde seine Daten eingibt und die Zahlung durchführt, an das Fachverfahren zurück. Hat der Kunde am ZV-Provider seine Zahlung durchgeführt, ruft der ZV-Provider URLs am restapi-Modul auf, um diesem das Ergebnis der Zahlungen mitzuteilen. Dieser wiederum setzt den Status des Kassenzeichens und ruft die Success-URL des Fachverfahrens auf. Die Benachrichtigung über den Ausgang der Zahlung erfolgt durch Aufruf der vom Fachverfahren an den Konnektor übergebenen URLs – successUrl, cancelUrl oder errorUrl. Das Fachverfahren hat darüber hinaus stets die Möglichkeit, den Status am Konnektor abzufragen.

Mit dem Übertragen der Buchungsliste über die REST-Schnittstelle kann das Fachverfahren die Zahlung auf drei Arten abwickeln: direkt, über Links auf einer Rechnung oder über die Paypage. Bei der direkten Variante leitet das Fachverfahren nach dem Anlegen der Buchungsliste den Bürger direkt zum ZV-Provider weiter, so dass dieser dort seine Zahlungsdaten eingeben kann. Der Bürger muss hierbei vor dem Übertragen der Buchungsliste ein Zahlverfahren auswählt haben.

Bei der Zahlung über Rechnungslinks wird die Buchungsliste mit einer Liste von erlaubten Zahlverfahren angelegt und der Bürger erhält eine Liste von Links auf den Konnektor über die jeweils für eines der verfügbaren Zahlverfahren die Zahlung über den Konnektor abgewickelt wird. Die Zahlverfahrensauswahl erfolgt durch Auswahl des entsprechenden Links. Die Links können dem Bürger als Liste in einem PDF-Dokument übermittelt worden sein.

Schließlich kann das Fachverfahren die Paypage als Zahlseite auswählen. Beim Übertragen der Buchungsliste erhält das Fachverfahren als Antwort einen Link auf die Paypage mit einer Zahlungsvorgangs-ID. Ruft der Bürger diesen Link auf, kann er auf der Paypage das gewünschte Zahlverfahren auswählen und bezahlen.

Es wird davon ausgegangen, dass zwischen ePayBL und Konnektor eine 1:1-Beziehung besteht, d.h. am Konnektor ist festgelegt, mit welcher ePayBL dieser kommuniziert.

2 Geschäftsprozesse und Zahlverfahren

Im Folgenden sind die Abläufe der verschiedenen Zahlverfahren und Geschäftsprozesse dargestellt. Die Zahlen in den gelben Kreisen beziehen sich auf die REST-Methoden Tabelle 3 und Tabelle 4.

Achtung: Je nach Ausgang der Zahlung beim ZV-Provider oder im Fall von SEPA-Zahlungen bei der Prüfung der SEPA-Daten wird der Bürger zum Fachverfahren zurückgeleitet. Hierfür übergibt das Fachverfahren eine Success-URL, eine Cancel-URL und eine Error-URL als Pflichtparameter zusammen mit der Buchungsliste. Hierbei ist es wichtig, dass der Bürger beim ZV-Provider nicht einfach den Browser schließt, sondern dort korrekt die Zahlung abschließt.

2.1 Kreditkarte

Dieser Ablauf beschreibt die Zahlung über Kreditkarte. Dabei wird der Kunde aus der Fachanwendung direkt zum Bezahlen auf die Seiten des ZV-Providers geleitet.

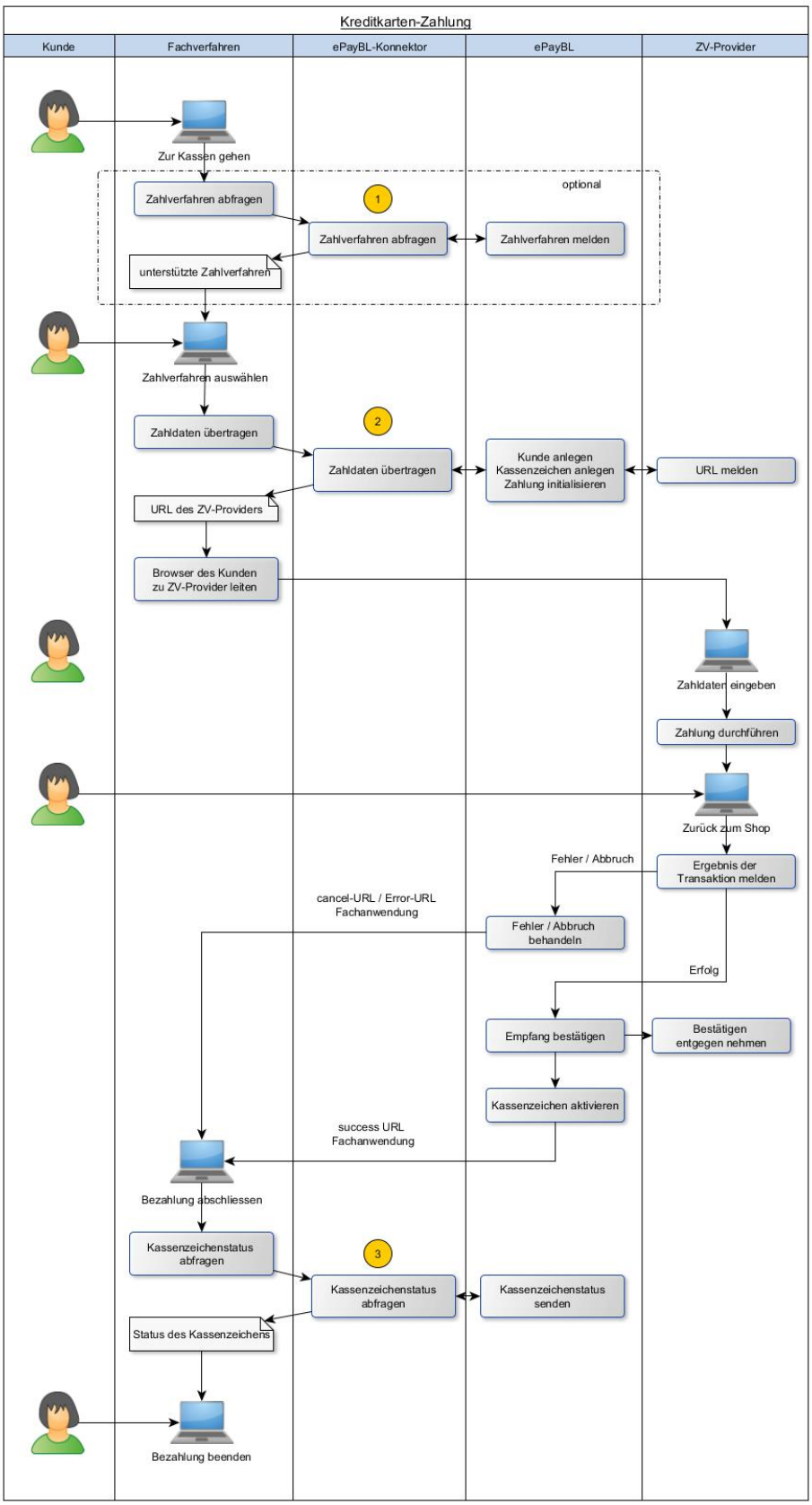


Abbildung 4: Kreditkarten-Zahlung

1. Der Kunde will in der Fachanwendung bezahlen.

2. Die Fachanwendung fragt am Konnektor die für den konfigurierten Mandanten verfügbaren Zahlverfahren ab, inklusive Minimal- und Maximalbeträge. Das Fachverfahren prüft die zu zahlende Rechnung gegen die verfügbaren Zahlverfahren und bietet dem Kunden eine Liste von möglichen Zahlverfahren an.

3. Der Kunde wählt ein Zahlverfahren aus, in diesem Fall die Zahlung über Kreditkarte.

4. Das Fachverfahren übermittelt dem Konnektor die Zahlungsdaten, inklusive Buchungsliste und Kundendaten. [1]Außerdem teilt das Fachverfahren dem Konnektor mit, dass die Zahlung direkt erfolgen soll.

5. Die ePayBL legt den Kunden und die Sollstellung an. In Erwartung der zügigen Bezahlung initiiert die ePayBL die Zahlung am ZV-Provider. Bei dieser Initiierung bekommt die ePayBL einen zeitlich befristeten Authentifizierungscode vom ZV-Provider. Der Kunde wird zum Löschen markiert, wenn es sich um einen temporären Kunden handelt.

6. Der Konnektor gibt die URL des ZV-Providers an das Fachverfahren zurück (inklusive Kassenzeichen und Kundenummer).

7. Das Fachverfahren leitet den Browser des Kunden zur Bezahlseite des ZV-Providers. Hier kann der Kunde seine Kreditkartendaten eingeben und die Bezahlung bestätigen.

8. Der ZV-Provider informiert die ePayBL über das Ergebnis der Bezahlung. Die ePayBL aktiviert daraufhin das Kassenzeichen.

9. Der Kunde schließt inzwischen am Provider die Zahlung ab. Der ZV-Provider ruft die Success-URL der ePayBL auf. Die ePayBL ruft ihrerseits die Success-URL des Fachverfahrens auf. Das Fachverfahren erfährt so, dass die Zahlung beendet ist. Abhängig vom verwendeten ZV-Provider werden die Schritte 8 und 9 in einer Aktion durchgeführt. Die Success-URL ist ein Pflichtparameter beim Überragen der Buchungsliste. Schon beim Übertragen der Buchungsliste kommt es dann zu einem Fehler.

10. Das Fachverfahren fragt am Konnektor und damit an der ePayBL den Status des Kassenzeichens ab.

2.2 giropay

Dieser Ablauf beschreibt die Zahlung über giropay. Dabei wird der Kunde direkt zum Bezahlen auf die Seiten des ZV-Providers geleitet.

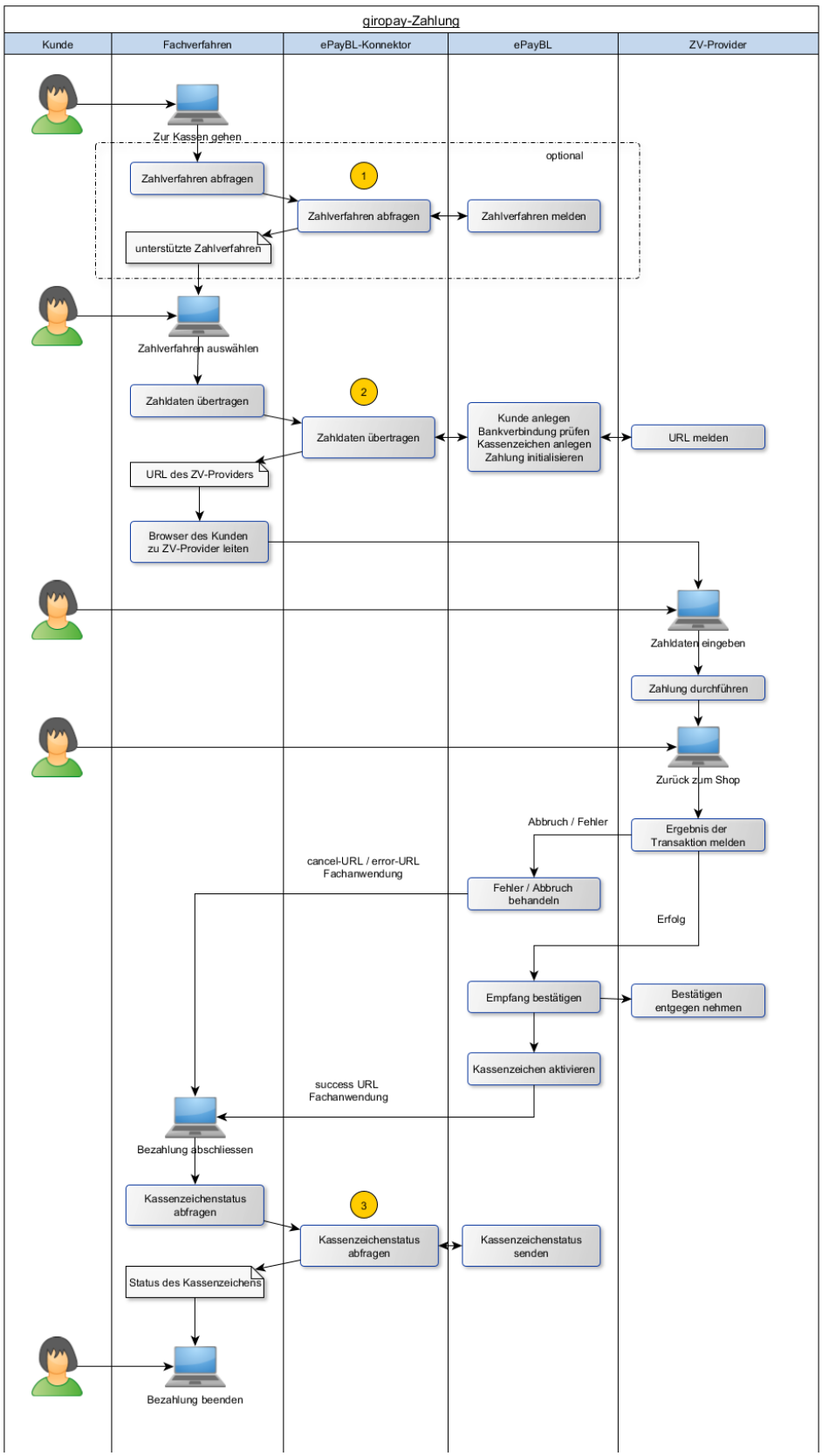


Abbildung 5: giropay-Zahlung

1. Der Kunde will in der Fachanwendung bezahlen.

2. Die Fachanwendung fragt am Konnektor die für den konfigurierten Mandanten verfügbaren Zahlverfahren ab, inklusive Minimal- und Maximalbeträge. Das Fachverfahren prüft die zu zahlende Rechnung gegen die verfügbaren Zahlverfahren und bietet dem Kunden eine Liste von möglichen Zahlverfahren an.

3. Der Kunde wählt ein Zahlverfahren aus, in diesem Fall die Zahlung über giropay.

4. Das Fachverfahren übermittelt dem Konnektor die Zahlungsdaten, inklusive Buchungsliste und Kundendaten. Außerdem teilt das Fachverfahren dem Konnektor mit, dass die Zahlung direkt erfolgen soll.

5. Die ePayBL legt den Kunden und die Sollstellung an. In Erwartung der zügigen Bezahlung initiiert die ePayBL die Zahlung am ZV-Provider. Bei dieser Initiierung bekommt die ePayBL einen zeitlich befristeten Authentifizierungscode. Der Kunde wird zum Löschen markiert, wenn es sich um einen temporären Kunden handelt.

6. Der Konnektor gibt die URL des ZV-Providers an das Fachverfahren zurück (inklusive Kassenzeichen und Kundenummer).

7. Das Fachverfahren leitet den Browser des Kunden zur Bezahlseite des ZV-Providers. Hier kann der Kunde seine Bankdaten eingeben und die Bezahlung bestätigen.

8. Der ZV-Provider informiert die ePayBL über das Ergebnis der Bezahlung. Die ePayBL aktiviert daraufhin das Kassenzeichen.

9. Der Kunde schließt inzwischen am Provider die Zahlung ab. Der ZV-Provider ruft die Success-URL der ePayBL auf. Die ePayBL ruft ihrerseits die Success-URL des Fachverfahrens auf. Das Fachverfahren erfährt so, dass die Zahlung beendet ist. Abhängig vom verwendeten ZV-Provider werden die Schritte 8 und 9 in einer Aktion durchgeführt. Die Success-URL ist ein Pflichtparameter beim Überragen der Buchungsliste. Schon beim Übertragen der Buchungsliste kommt es dann zu einem Fehler.

10. Das Fachverfahren fragt am Konnektor und damit an der ePayBL den Status des Kassenzeichens ab.

2.3 PayPal

[2]Dieser Ablauf beschreibt die Zahlung mittels PayPal direkt über PayPal (PayPal als Zahlverfahren kann auch über die Zahlverkehrsprovider GiroSolution oder PayPlace abgewickelt werden). Dabei wird der Kunde direkt zum Bezahlen auf die Seiten des ZV-Providers geleitet. Dabei ist es gleich, ob die Bezahlung direkt über PayPal oder über einen ZV-Provider abgewickelt wird.

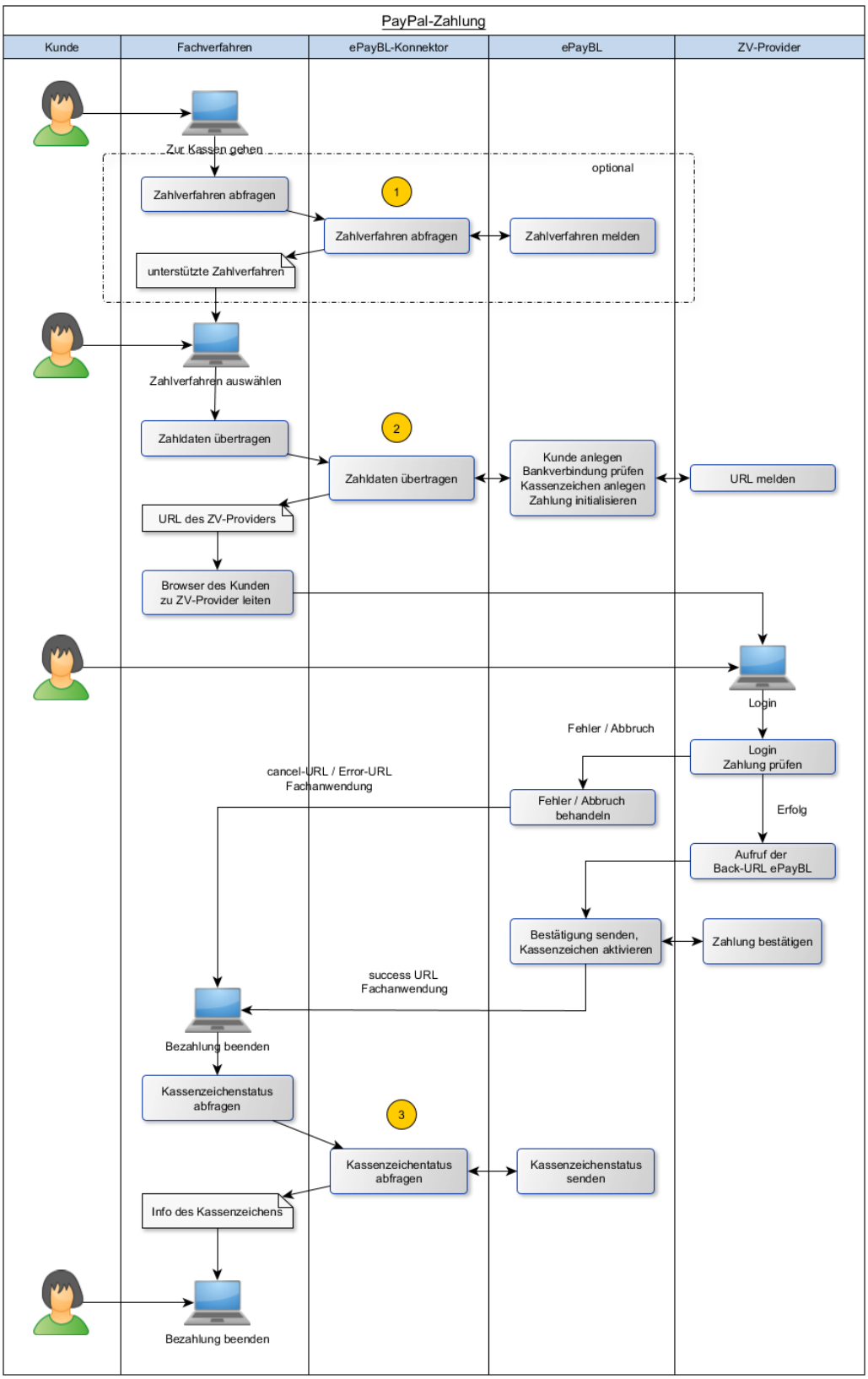


Abbildung 6: PayPal-Zahlung

1. Der Kunde will in der Fachanwendung bezahlen.

2. Die Fachanwendung fragt am Konnektor die für den konfigurierten Mandanten verfügbaren Zahlverfahren ab, inklusive Minimal- und Maximalbeträge. Das Fachverfahren prüft die zu zahlende Rechnung gegen die verfügbaren Zahlverfahren und bietet dem Kunden eine Liste von möglichen Zahlverfahren an.

3. Der Kunde wählt ein Zahlverfahren aus, in diesem Fall die Zahlung über PayPal.

4. Das Fachverfahren übermittelt dem Konnektor die Zahlungsdaten, inklusive Buchungsliste und Kundendaten. Außerdem teilt das Fachverfahren dem Konnektor mit, dass die Zahlung direkt erfolgen soll.

5. Die ePayBL legt den Kunden und die Sollstellung an. In Erwartung der zügigen Bezahlung initiiert die ePayBL die Zahlung am ZV-Provider. Der ZV-Provider ist hier entweder PayPal selbst oder ein ZV-Provider (GiroSolution, PayPlace). Bei dieser Initiierung bekommt die ePayBL einen zeitlich befristeten Authentifizierungscode. Der Kunde wird zum Löschen markiert, wenn es sich um einen temporären Kunden handelt.

6. Der Konnektor gibt die URL des ZV-Providers an das Fachverfahren zurück (inklusive Kassenzeichen und Kundenummer).

7. Das Fachverfahren leitet dem Browser des Kunden zur Bezahlseite des ZV-Providers. Hier kann der Kunde die Bezahlung bestätigen.

8. Der ZV-Provider informiert die ePayBL über das Ergebnis der Bezahlung. Die ePayBL aktiviert daraufhin das Kassenzeichen.

9. Der Kunde schließt inzwischen am Provider die Zahlung ab. Der ZV-Provider ruft die Success-URL der ePayBL auf. Die ePayBL ruft ihrerseits die Success-URL des Fachverfahrens auf. Das Fachverfahren erfährt so, dass die Zahlung beendet ist. Abhängig vom verwendeten ZV-Provider werden die Schritte 8 und 9 in einer Aktion durchgeführt. Die Success-URL ist ein Pflichtparameter beim Überragen der Buchungsliste. Schon beim Übertragen der Buchungsliste kommt es dann zu einem Fehler.

10. Das Fachverfahren fragt am Konnektor und damit an der ePayBL den Status des Kassenzeichens ab.

2.4 PayDirekt

Dieser Ablauf beschreibt die Zahlung mittels PayDirekt direkt über den ZV-Provider PayDirekt (PayDirekt als Zahlverfahren kann auch über den Zahlverkehrsprovider GiroSolution abgewickelt werden). Dabei wird der Kunde direkt zum Bezahlen auf die Seiten des ZV-Providers geleitet.

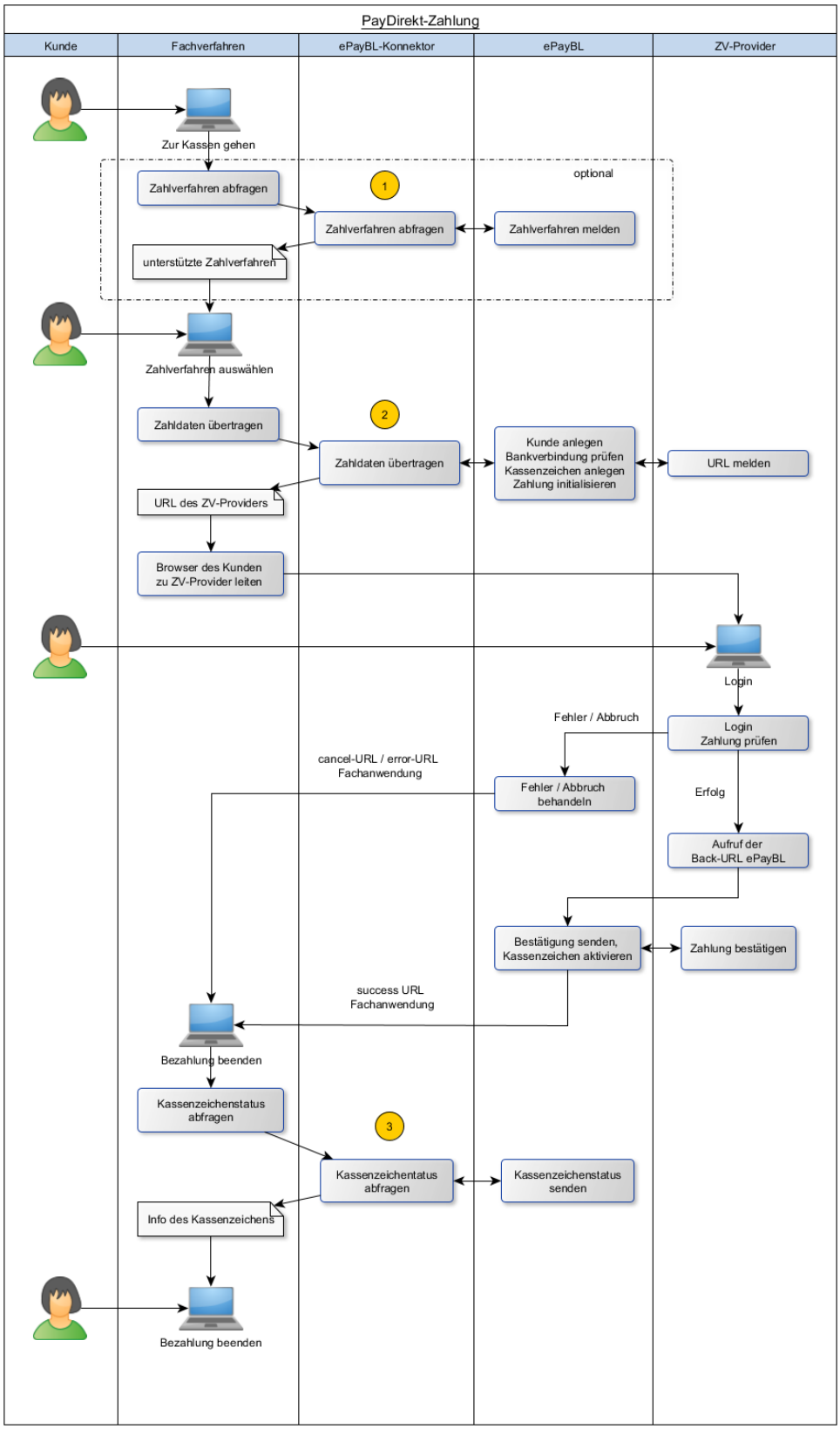


Abbildung 7: PayDirekt-Zahlung

11. Der Kunde will in der Fachanwendung bezahlen.

12. Die Fachanwendung fragt am Konnektor die für den konfigurierten Mandanten verfügbaren Zahlverfahren ab, inklusive Minimal- und Maximalbeträge. Das Fachverfahren prüft die zu zahlende Rechnung gegen die verfügbaren Zahlverfahren und bietet dem Kunden eine Liste von möglichen Zahlverfahren an.

13. Der Kunde wählt ein Zahlverfahren aus, in diesem Fall die Zahlung über PayDirekt.

14. Das Fachverfahren übermittelt dem Konnektor die Zahlungsdaten, inklusive Buchungsliste und Kundendaten. Außerdem teilt das Fachverfahren dem Konnektor mit, dass die Zahlung direkt erfolgen soll.

15. Die ePayBL legt den Kunden und die Sollstellung an. In Erwartung der zügigen Bezahlung initiiert die ePayBL die Zahlung am ZV-Provider. Der ZV-Provider ist hier entweder PayDirekt selbst oder ein ZV-Provider (GiroSolution). Bei dieser Initiierung bekommt die ePayBL einen zeitlich befristeten Authentifizierungscode. Der Kunde wird zum Löschen markiert, wenn es sich um einen temporären Kunden handelt.

16. Der Konnektor gibt die URL des ZV-Providers an das Fachverfahren zurück (inklusive Kassenzeichen und Kundenummer).

17. Das Fachverfahren leitet dem Browser des Kunden zur Bezahlseite des ZV-Providers. Hier kann der Kunde die Bezahlung bestätigen.

18. Der ZV-Provider informiert die ePayBL über das Ergebnis der Bezahlung. Die ePayBL aktiviert daraufhin das Kassenzeichen.

19. Der Kunde schließt inzwischen am Provider die Zahlung ab. Der ZV-Provider ruft die Success-URL der ePayBL auf. Die ePayBL ruft ihrerseits die Success-URL des Fachverfahrens auf. Das Fachverfahren erfährt so, dass die Zahlung beendet ist. Abhängig vom verwendeten ZV-Provider werden die Schritte 8 und 9 in einer Aktion durchgeführt. Die Success-URL ist ein Pflichtparameter beim Überragen der Buchungsliste. Schon beim Übertragen der Buchungsliste kommt es dann zu einem Fehler.

20. Das Fachverfahren fragt am Konnektor und damit an der ePayBL den Status des Kassenzeichens ab.

2.5 SEPA-Lastschrift

Bei der SEPA-Lastschrift wird unterschieden, ob eine externe (außerhalb der ePayBL) oder eine interne Mandatsverwaltung verwendet wird. Bei einer internen Mandatsverwaltung kann man mit Mandatsreferenzen arbeiten und die ePayBL kann sich die Mandatsdaten selbst heraussuchen.

Wenn das Fachverfahren Buchungsdaten an die ePayBL übermittelt und als Zahlverfahren SEPA-Lastschrift angibt, dann prüft die ePayBL zunächst ob es für den Kunden ein SEPA-Mandat gibt oder nicht. Wenn es kein SEPA-Mandat gibt, und auch aus den mitgelieferten Kundendaten keins angelegt werden kann, antwortet die Methode „Buchungslisten übertragen“ mit einer entsprechenden Fehlermeldung. Die Fachanwendung kann dann die Buchungsdaten erneut übertragen mit den notwendigen Daten für ein SEPA-Mandat.

In den folgenden zwei Unterabschnitten gehen wir davon aus, dass die Fachanwendung, z.B. über die oben genannte Abfrage, weiß, ob der Kunde bereits ein SEPA-Mandat hat oder nicht.

2.5.1 Interne Mandatsverwaltung und vorhandenes Mandat

Hier nehmen wir an, dass es sich um eine interne Mandatsverwaltung handelt und dass der Kunde bereits ein SEPA-Mandat hat.

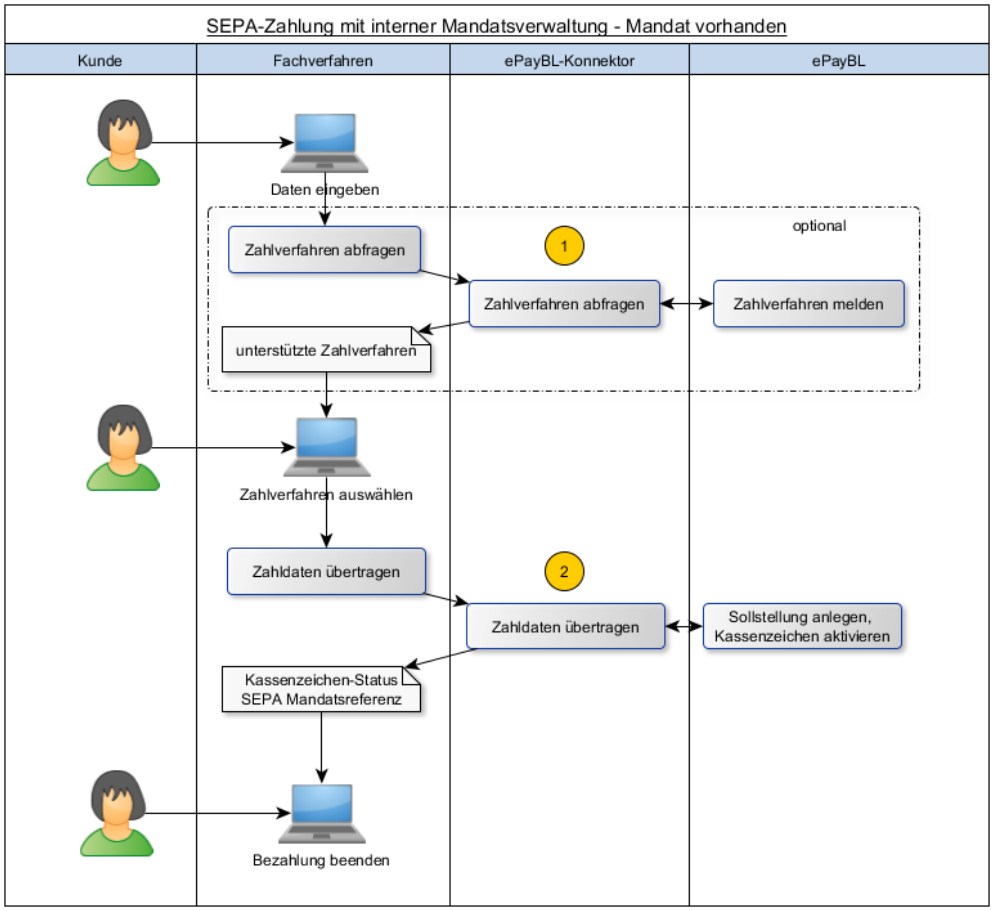


Abbildung 8: SEPA-Zahlung bei interner Mandatsverwaltung, Mandat vorhanden

1. Der Kunde will in der Fachanwendung bezahlen.

2. Die Fachanwendung fragt am Konnektor die für den konfigurierten Mandanten verfügbaren Zahlverfahren ab, inklusive Minimal- und Maximalbeträge. Das Fachverfahren prüft die zu zahlende Rechnung gegen die verfügbaren Zahlverfahren und bietet dem Kunden eine Liste von möglichen Zahlverfahren an.

3. Der Kunde wählt ein Zahlverfahren aus, in diesem Fall SEPA-Lastschrift.

4. Das Fachverfahren übermittelt dem Konnektor die Zahlungsdaten, inklusive Buchungsliste und Kundendaten.

5. Die ePayBL prüft, ob der Kunde vorhanden ist und legt die Sollstellung an. Für den Kunden überprüft die ePayBL, ob ein SEPA-Mandat vorhanden ist. Wir nehmen hier an, dass der Kunde ein SEPA-Mandat hat. Der Kunde kann in dieser Situation kein temporärer Kunde sein, der am Ende des Prozesses wieder gelöscht wird, sondern muss ein Bestandskunde sein. Die ePayBL aktiviert das Kassenzeichen und gibt den Status des Kassenzeichens inklusive verwendeter Mandatsreferenz zurück.

6. Das Fachverfahren informiert den Kunden über die bevorstehende Lastschrift unter Verwendung des angegebenen Mandats.

7. Es ist keine aktive Information des Fachverfahrens über die Aktivierung des Kassenzeichens nötig, da der Kassenzeichenstatus als Antwort auf das Übermitteln der Buchungsdaten geschickt wird. Das Fachverfahren muss daher auch nicht den Status an der ePayBL abfragen.

2.5.2 Interne Mandatsverwaltung ohne vorhandenes Mandat

Hier nehmen wir an, dass es sich um eine interne Mandatsverwaltung handelt und dass der Kunde noch kein SEPA-Mandat hat. In diesem Fall muss das SEPA-Mandat zunächst angelegt werden.

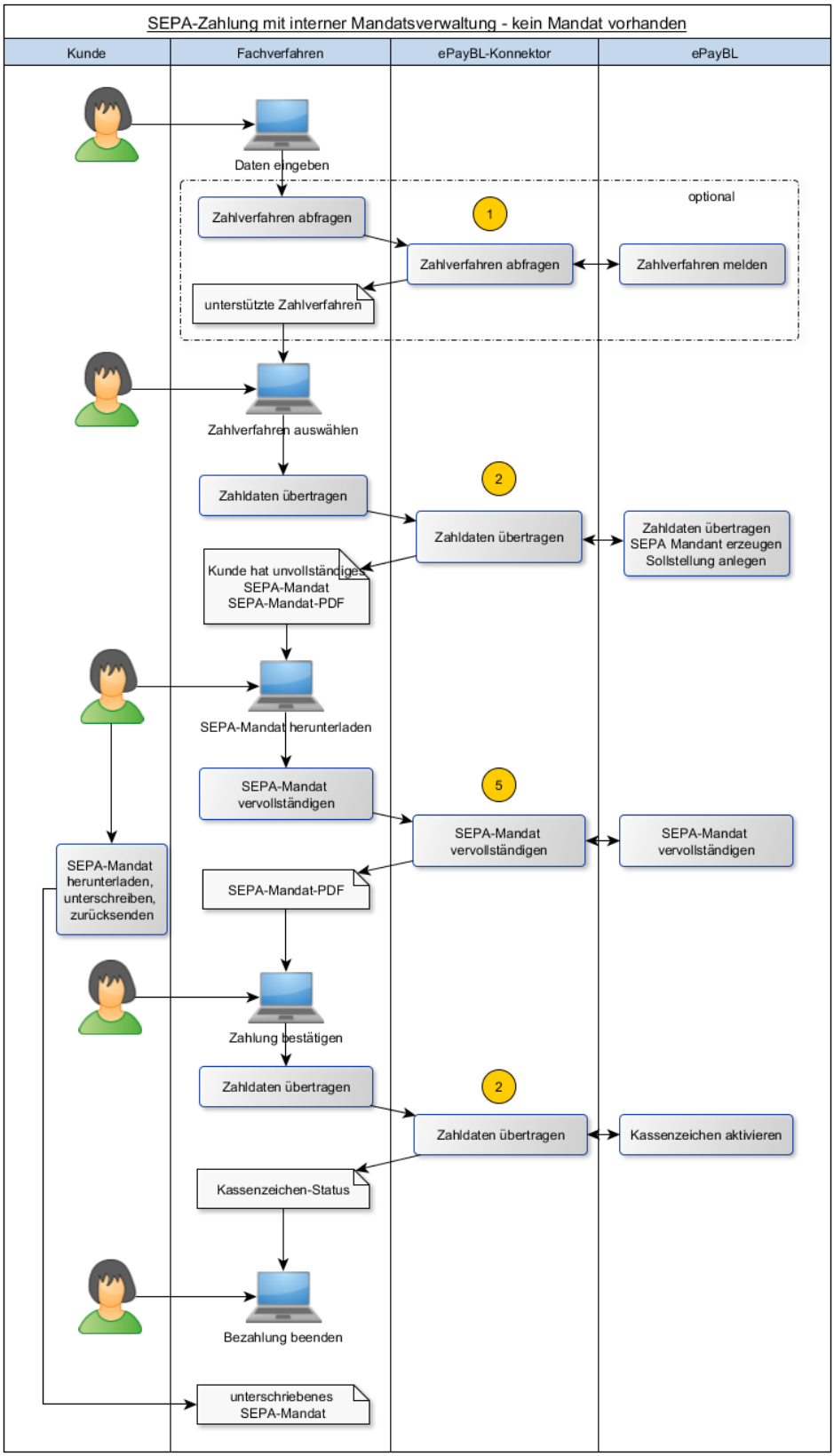


Abbildung 9: SEPA-Zahlung bei interner Mandatsverwaltung, kein Mandat vorhanden

1. Der Kunde will in der Fachanwendung bezahlen.

2. Die Fachanwendung fragt am Konnektor die für den konfigurierten Mandanten verfügbaren Zahlverfahren ab, inklusive Minimal- und Maximalbeträge. Das Fachverfahren prüft die zu zahlende Rechnung gegen die verfügbaren Zahlverfahren und bietet dem Kunden eine Liste von möglichen Zahlverfahren an.

3. Der Kunde wählt ein Zahlverfahren aus, in diesem Fall SEPA-Lastschrift.

4. Das Fachverfahren übermittelt dem Konnektor die Zahlungsdaten, inklusive Buchungsliste und Kundendaten.

5. Die ePayBL prüft, ob der Kunde vorhanden ist und legt den Kunden und die Sollstellung an. Für den Kunden überprüft die ePayBL, ob ein SEPA-Mandat vorhanden ist. Wir nehmen hier an, dass der Kunde noch kein SEPA-Mandat hat. Die ePayBL erstellt aus den übermittelten Kundendaten ein (unvollständiges) SEPA-Mandat (es fehlt das Datum der Unterschrift). Fehlen beim Anlegen des SEPA-Mandats Pflicht-Parameter, liefert die Methode entsprechende Fehlercodes zurück. Das Fachverfahren muss in diesem Fall die Methode nochmal aufrufen mit den fehlenden Parametern.
   Konnte das SEPA-Mandat erfolgreich angelegt werden, wird auch die Sollstellung angelegt und ein SEPA-Mandats-PDF erzeugt. Dieses PDF wird zusammen mit dem Kassenzeichen und der Kundennummer an das Fachverfahren zurückgesendet.

6. Das Fachverfahren bietet das SEPA-Mandats-PDF dem Kunden zum Ausdrucken an, mit der Bitte es unterschrieben zurückzusenden. Das Fachverfahren kann über einen Button die Bestätigung des Kunden hierzu einholen und das aktuelle Datum als Datum der Unterschrift an die ePayBL senden. Diese vervollständigt mit diesem Datum das SEPA-Mandat und aktiviert es hierdurch.

7. Das Fachverfahren lässt den Kunden nun die Zahlung mit dem SEPA-Mandat bestätigen. Die ePayBL aktiviert daraufhin das Kassenzeichen und sendet den Kassenzeichen-Status zurück.

8. Es ist keine aktive Information des Fachverfahrens über die Aktivierung des Kassenzeichens nötig, da der Kassenzeichenstatus als Antwort auf das Übermitteln der Buchungsdaten geschickt wird. Das Fachverfahren muss daher auch nicht den Status an der ePayBL abfragen.

2.5.3 SEPA-Mandat Anlegen bei interner Mandatsverwaltung

Unterstützt das Fachverfahren SEPA-Lastschrift, so sind SEPA-Mandate für die Kunden erforderlich. Als Mandatsverwaltung kann die ePayBL selber genutzt werden (interne Mandatsverwaltung). Das Fachverfahren kann das benötigte SEPA-Mandat im Rahmen der

entsprechenden Zahlung anlegen, es ist aber auch möglich, ein SEPA-Mandat unabhängig von einer Zahlung anzulegen. Das ist insbesondere wichtig für wiederkehrende Zahlungen.

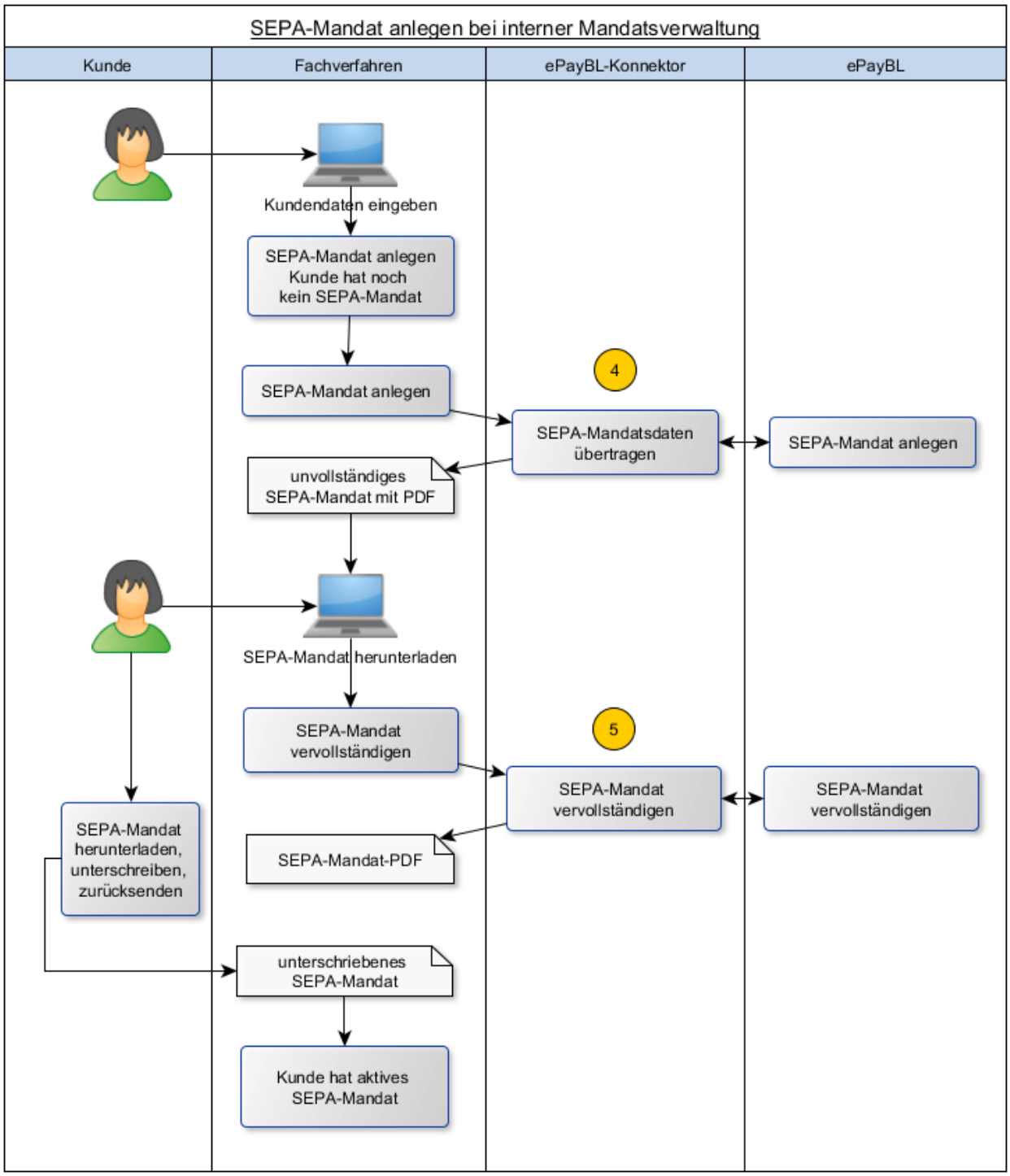


Abbildung 10: SEPA-Mandant anlegen

Beim Anlegen des SEPA-Mandats werden die notwendigen Mandatsdaten übertragen. Das Facherfahren erhält als Antwort die kompletten Mandatsdaten (insbesondere die Mandatsreferenz) und das PDF, das der Kunde unterschrieben zurückschicken muss. Dieses SEPA-Mandat muss noch vervollständigt werden (Datum der Unterschrift). Das kann gemacht werden, wenn das unterschriebene Mandat vorliegt oder unabhängig davon.

2.5.4 SEPA-Lastschrift bei externer Mandatsverwaltung

Bei einer externen Mandatsverwaltung muss das Fachverfahren zunächst bei dieser das SEPA-Mandat erfragen bzw. anlegen. Wir gehen im Ablauf davon aus, dass ein (externes) SEPA-Mandat für den Kunden vorhanden ist und dass die SEPA-Mandatsdaten vorliegen.

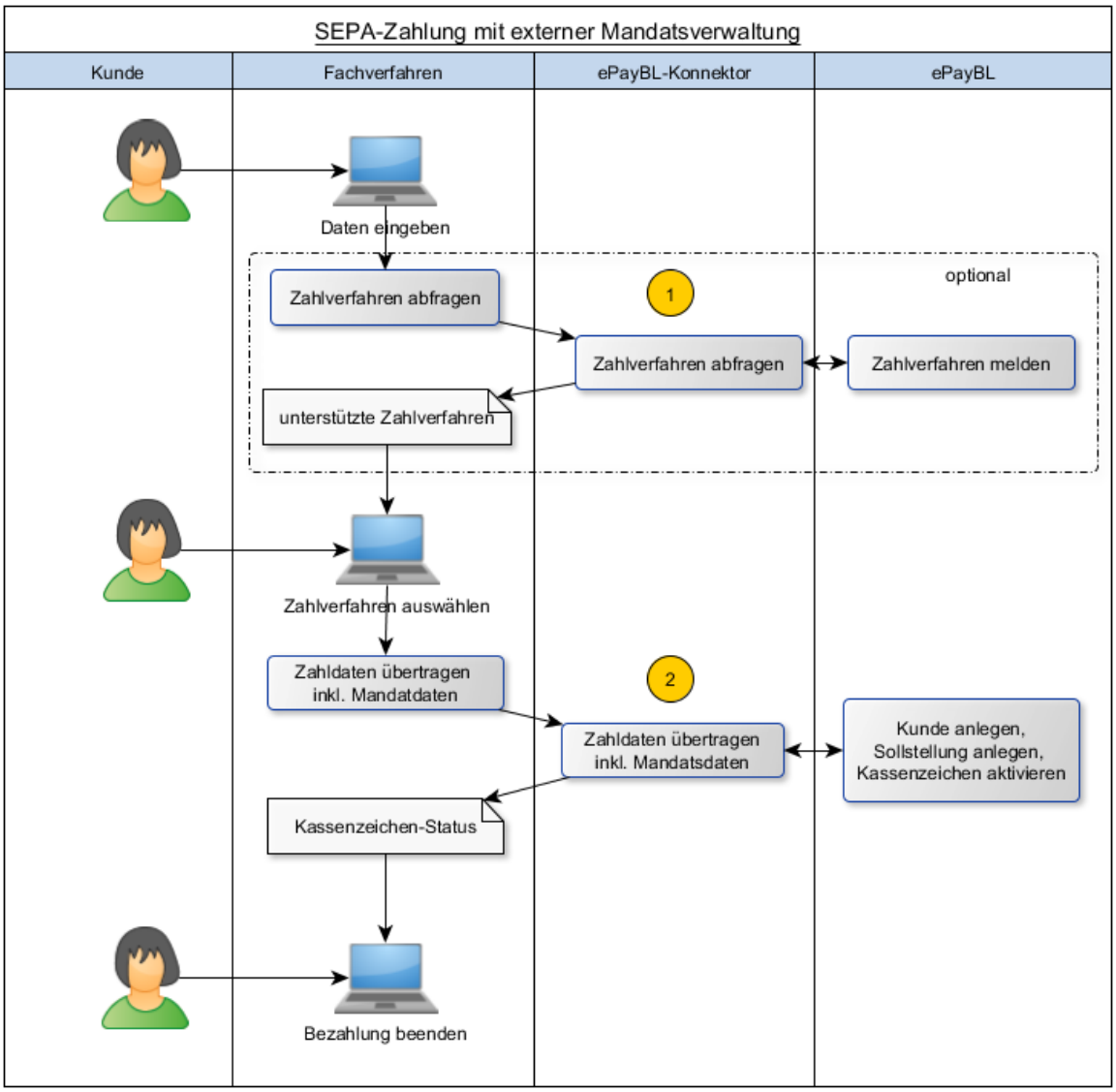


Abbildung 11: SEPA-Zahlung bei externer Mandatsverwaltung

1. Der Kunde will in der Fachanwendung bezahlen.

2. Die Fachanwendung fragt am Konnektor die für den konfigurierten Mandanten verfügbaren Zahlverfahren ab, inklusive Minimal- und Maximalbeträge. Das Fachverfahren prüft die zu zahlende Rechnung gegen die verfügbaren Zahlverfahren und bietet dem Kunden eine Liste von möglichen Zahlverfahren an.

3. Der Kunde wählt ein Zahlverfahren aus, in diesem Fall SEPA-Lastschrift.

4. Das Fachverfahren übermittelt dem Konnektor die Zahlungsdaten, inklusive Buchungsliste, Kundendaten und SEPA-Mandat.

5. Die ePayBL prüft, ob der Kunde vorhanden ist und legt den Kunden und die Sollstellung an. Da das mitgelieferte SEPA-Mandat gültig ist, wird die Sollstellung angelegt und das Kassenzeichen aktiviert.

6. Es ist keine aktive Information des Fachverfahrens über die Aktivierung des Kassenzeichens nötig, da der Kassenzeichenstatus als Antwort auf das Übermitteln der Buchungsdaten geschickt wird. Das Fachverfahren muss daher auch nicht den Status an der ePayBL abfragen.

2.6 Überweisung

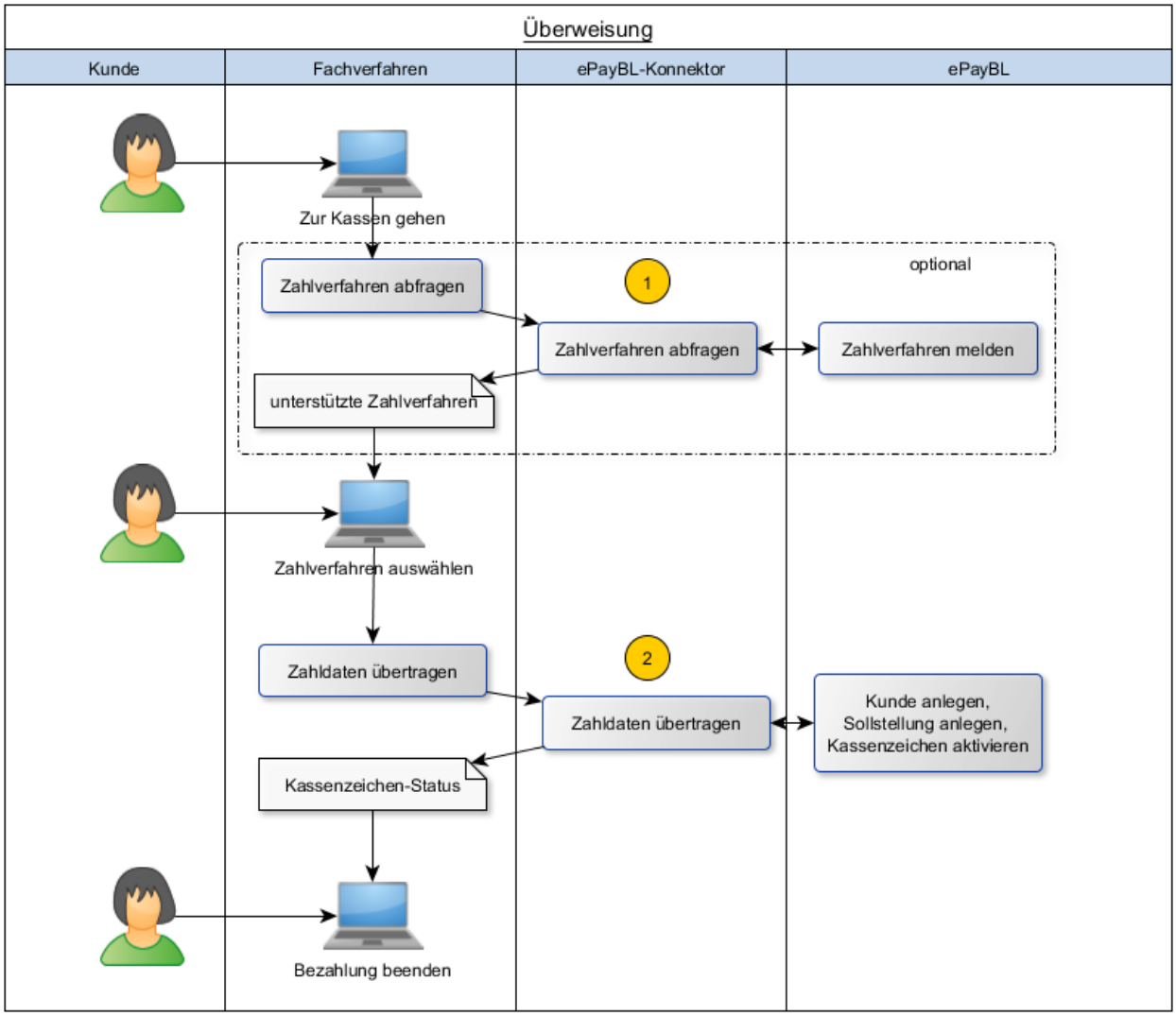


Abbildung 12: Überweisung

1. Der Kunde will in der Fachanwendung bezahlen.
2. Die Fachanwendung fragt am Konnektor die für den konfigurierten Mandanten verfügbaren Zahlverfahren ab, inklusive Minimal- und Maximalbeträge. Das Fachverfahren prüft die zu zahlende Rechnung gegen die verfügbaren Zahlverfahren und bietet dem Kunden eine Liste von möglichen Zahlverfahren an.
3. Der Kunde wählt ein Zahlverfahren aus, in diesem Fall Überweisung.
4. Das Fachverfahren übermittelt dem Konnektor die Zahlungsdaten, inklusive Buchungsliste und Kundendaten. Außerdem teilt das Fachverfahren dem Konnektor mit, dass die Zahlung direkt erfolgen soll.

5. Die ePayBL prüft, ob der Kunde vorhanden ist und legt die Sollstellung an und aktiviert das Kassenzeichen.

6. Das Fachverfahren informiert den Kunden eine Überweisung des Rechnungsbetrages zu veranlassen.

7. Es ist keine aktive Information des Fachverfahrens über die Aktivierung des Kassenzeichens nötig, da der Kassenzeichenstatus als Antwort auf das Übermitteln der Buchungsdaten geschickt wird. Das Fachverfahren muss daher auch nicht den Status an der ePayBL abfragen.

2.7 Lastschrift ohne Einzugsermächtigung

Dieser Ablauf beschreibt die Zahlung über Lastschrift ohne Einzugsermächtigung. Dabei wird der Kunde aus der Fachanwendung direkt zum Bezahlen auf die Seiten des ZV-Providers geleitet. Das Zahlverfahren kann nur über den ZV-Provider GiroSolution abgewickelt werden. Der ZV-Provider wird in der Mandantenkonfiguration der ePayBL eingestellt.

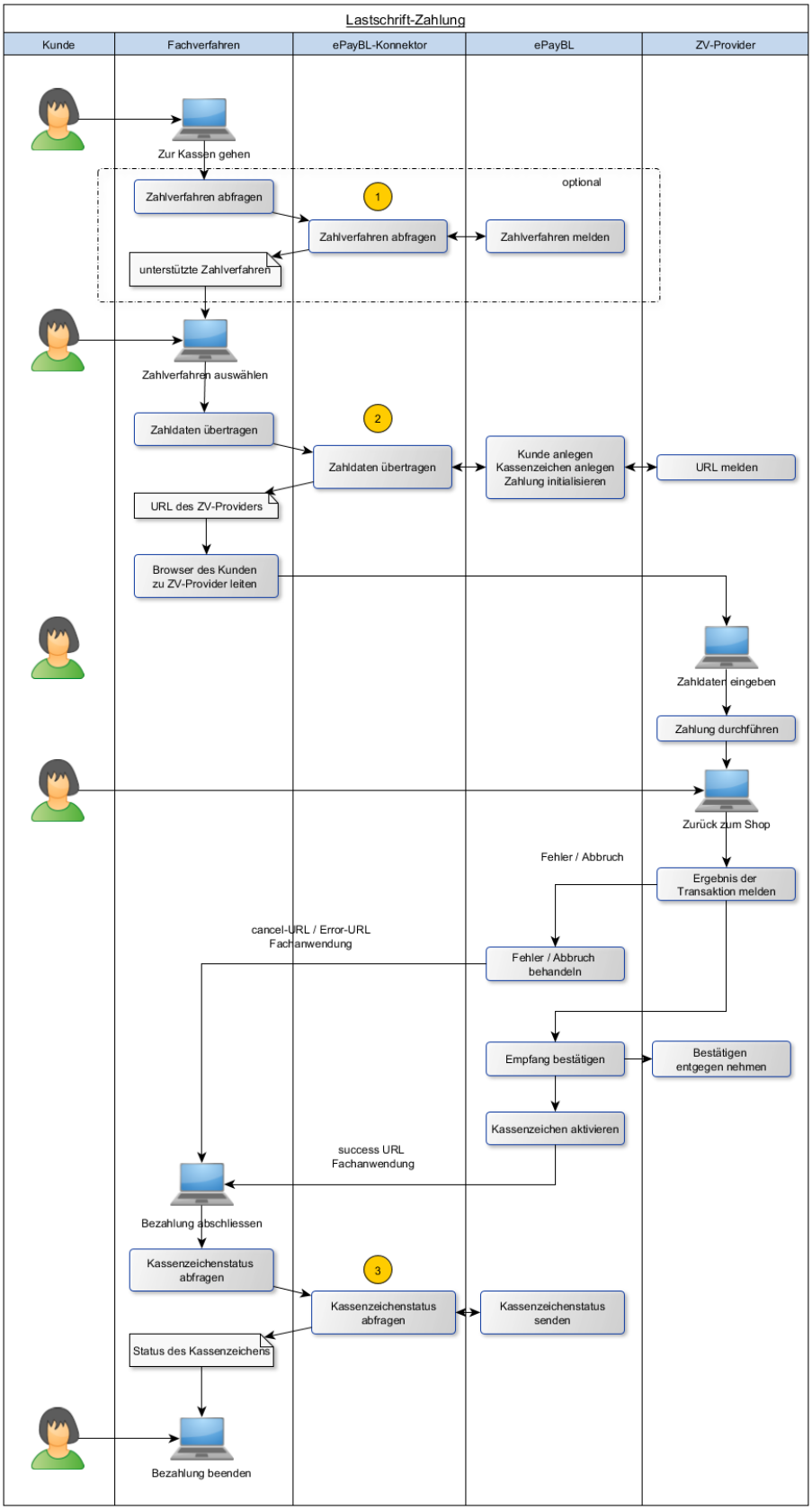


Abbildung 13: Lastschrift-Zahlung

1. Der Kunde will in der Fachanwendung bezahlen.

2. Die Fachanwendung fragt am Konnektor die für den konfigurierten Mandanten verfügbaren Zahlverfahren ab, inklusive Minimal- und Maximalbeträge. Das Fachverfahren prüft die zu zahlende Rechnung gegen die verfügbaren Zahlverfahren und bietet dem Kunden eine Liste von möglichen Zahlverfahren an.

3. Der Kunde wählt ein Zahlverfahren aus, in diesem Fall die Zahlung über „LastschriftOhne“.

4. Das Fachverfahren übermittelt dem Konnektor die Zahlungsdaten, inklusive Buchungsliste und Kundendaten. [3]Außerdem teilt das Fachverfahren dem Konnektor mit, dass die Zahlung direkt erfolgen soll.

5. Die ePayBL legt den Kunden und die Sollstellung an. In Erwartung der zügigen Bezahlung initiiert die ePayBL die Zahlung am ZV-Provider. Bei dieser Initiierung bekommt die ePayBL einen zeitlich befristeten Authentifizierungscode vom ZV-Provider. Der Kunde wird zum Löschen markiert, wenn es sich um einen temporären Kunden handelt.

6. Der Konnektor gibt die URL des ZV-Providers an das Fachverfahren zurück (inklusive Kassenzeichen und Kundenummer).

7. Das Fachverfahren leitet den Browser des Kunden zur Bezahlseite des ZV-Providers. Hier kann der Kunde seine Daten eingeben und die Bezahlung bestätigen.

8. Der ZV-Provider informiert die ePayBL über das Ergebnis der Bezahlung. Die ePayBL aktiviert daraufhin das Kassenzeichen.

9. Der Kunde schließt inzwischen am Provider die Zahlung ab. Der ZV-Provider ruft die Success-URL der ePayBL auf. Die ePayBL ruft ihrerseits die Success-URL des Fachverfahrens auf. Das Fachverfahren erfährt so, dass die Zahlung beendet ist. Abhängig vom verwendeten ZV-Provider werden die Schritte 8 und 9 in einer Aktion durchgeführt.

10. Das Fachverfahren fragt am Konnektor und damit an der ePayBL den Status des Kassenzeichens ab.

2.8 Bezahlen über Rechnungslink mit ZV-Provider

Will das Fachverfahren die Bezahlung nicht selber abwickeln, dann erstellt das Fachverfahren nur eine Rechnung, leitet diese dem Kunden zu (per E-Mail oder per Post) und bittet ihn, diese Rechnung über einen auf der Rechnung angebrachten Link zum ZV-Provider zu bezahlen. Hierbei ist zu beachten, dass es, im Gegensatz zur Zahlungsabwicklung über die Paypage, keine Möglichkeit gibt, nachträglich über eine Oberfläche ein Zahlverfahren auszuwählen. Das Zahlverfahren muss bei Erstellen des Links auf der Rechnung festgelegt werden. Um trotzdem dem Kunden die Möglichkeit zu bieten, erst direkt beim Bezahlen das Zahlverfahren auszuwählen, kann das Fachverfahren beim Anlegen der Buchungen in der ePayBL in diesem Fall eine Liste von Zahlverfahren mitgeben. Der Konnektor erzeugt dann für jedes dieser Zahlverfahren ein entsprechenden Link zum ZV-Provider. Wenn dann alle diese Links auf der Rechnung erscheinen, dann besteht die Zahlverfahrensauswahl für den Kunden darin, dass er den zum gewünschten Zahlverfahren korrespondierenden Link auswählt. Bricht er die Bezahlung über einen Link ab, kann er einen anderen Link aufrufen. Ist die Rechnung über einen Link bezahlt, werden alle anderen Links dadurch ungültig.

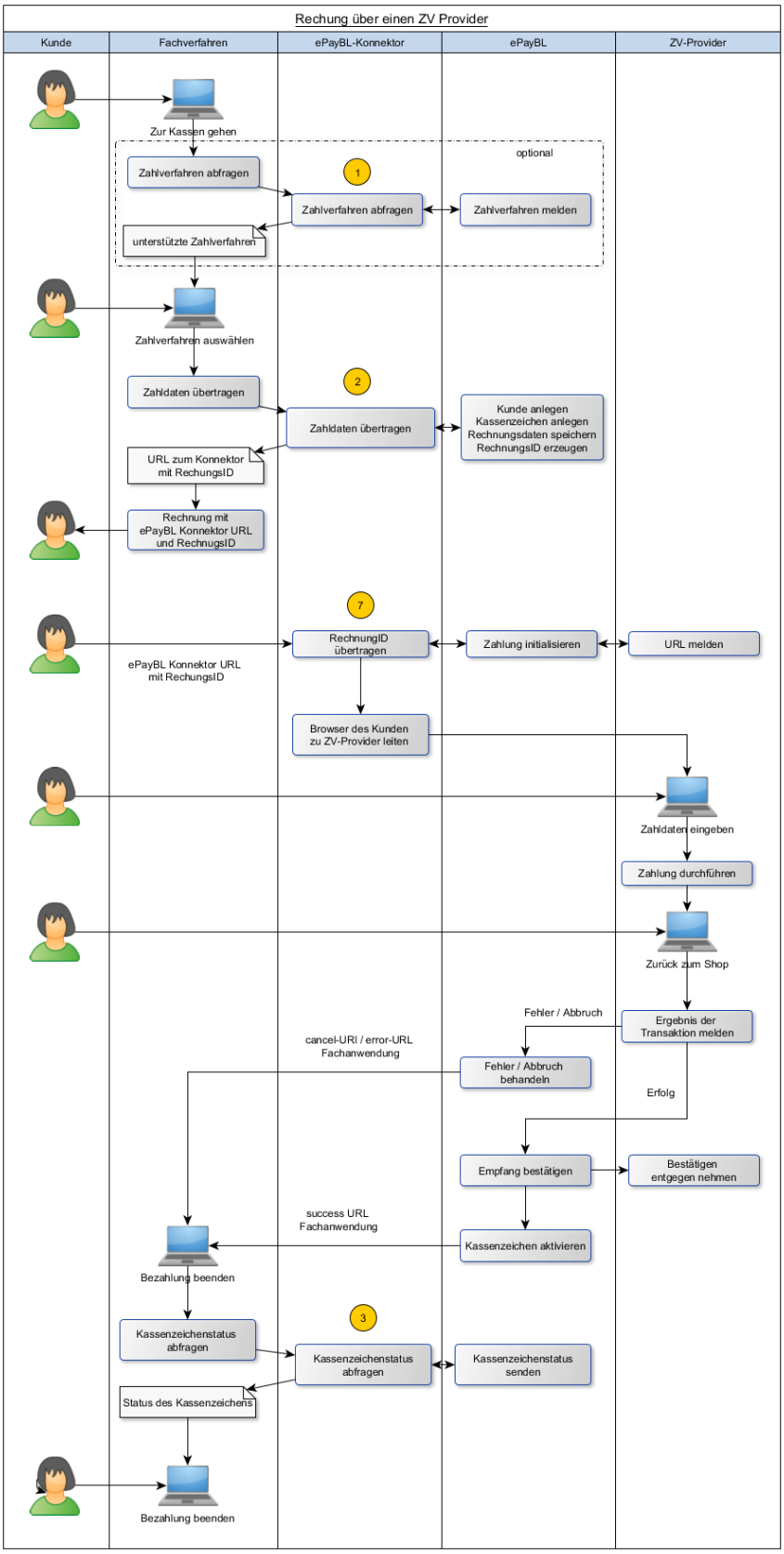


Abbildung 14: Bezahlen über Rechnungslink mit ZV-Provider

1. Der Kunde will in der Fachanwendung bezahlen

2. Die Fachanwendung fragt bei Konnektor die für den konfigurierten Mandanten verfügbaren Zahlverfahren ab, inklusive Minimal- und Maximalbeträge. Das Fachverfahren prüft die zu zahlenden Rechnung gegen die verfügbaren Zahlverfahren und bietet dem Kunden eine Liste von möglichen Zahlverfahren an.

3. Der Kunde wählt die gewünschten Zahlverfahren aus.

4. Das Fachverfahren übermittelt dem Konnektor die Zahlungsdaten, inklusive Buchungsliste und Kundendaten. Außerdem teilt das Fachverfahren der Konnektor mit, dass die Zahlung über eine Rechnung (später) erfolgen soll.

5. Die ePayBL legt den Kunden und die Sollstellung an. Da die Bezahlung später erfolgt, wird die Zahlung am Provider noch nicht initialisiert. Die ePayBL speichert die Zahlungsdaten und vergibt für jedes Zahlverfahren eine Rechnungs-ID.

6. Der Konnektor gibt die Liste der Rechnungs-IDs und die URL des Konnektors an das Fachverfahren zurück (inklusive Kassenzeichen und Kundenummer),

7. Das Fachverfahren erstellt ein Rechnungsdokument mit einer Liste von Bezahl-Links zum Konnektor. Jeder dieser Links ist einem der verfügbaren Zahlverfahren zugeordnet und enthält eine Rechnungs-ID als Parameter.

8. Der Kunde bekommt die Rechnung und ruft in seinem Browser den zu dem von ihm gewünschten Zahlverfahren den entsprechenden Link auf. Der Konnektor leitet diesen Request an die ePayBL weiter. Mit Hilfe der Rechnungs-ID ermittelt die ePayBL die zugehörigen Rechnungsdaten und initiiert die Bezahlung beim ZV-Provider mit dem entsprechenden Zahlverfahren. Als Ergebnis gibt die ePayBL die URL des ZV-Providers zurück.

9. Der Konnektor leitet den Kunden zum ZV-Provider weiter, wo der Kunde seine Zahlung abwickeln kann. Der ZV-Provider informiert die ePayBL über das Ergebnis der Bezahlung. Die ePayBL aktiviert daraufhin das Kassenzeichen.

10. Der Kunde schließt inzwischen am Provider die Zahlung ab. Der ZV-Provider ruft die Success-URL der ePayBL auf. Die ePayBL ruft ihrerseits die Success-URL des Fachverfahrens auf. Das Fachverfahren erfährt so, dass die Zahlung beendet ist.

2.9 Bezahlen über Rechnungslink mit SEPA-Lastschrift oder Überweisung

Die REST-Schnittstelle bietet die Möglichkeit einer Art Zahlverfahrensauswahl für den Bürger ohne dabei eine Paypage-Webseite zur Verfügung zu stellen (Zahltyp=RECHNUNG). Beim Anlegen einer Buchung kann das Fachverfahren eine Liste von Zahlverfahren mitgeben und bekommt für jedes Zahlverfahren eine URL auf den Konnektor mit einem entsprechenden REST-Methodenaufruf zurück (Rechnungslinks). Dem Bürger werden alle diese URLs übergeben, mit dem Hinweis die Buchungsliste über das gewünschte Zahlverfahren auszuwählen, indem er die entsprechenden URL aufruft. Nach erfolgreicher Bezahlung über eine dieser URLs sind die anderen URL natürlich deaktiviert, so dass verhindert wird, dass der Bürger die gleiche Rechnung über zwei verschiedene Zahlverfahren bezahlt.

Im Fall der Bezahlung eines Rechnungslinks über SEPA-Lastschrift oder Überweisung besteht das Problem darin, dass es keine Web-Seite gibt, über die der Kunde über das SEPA-Mandat bzw. über das Ziel-Konto informiert werden kann.

Daher können über den Rechnungslink nur Zahlverfahren angeboten werden, bei denen die Zahlung über einen ZV-Provider abgewickelt wird.

2.10 Bezahlen über die Paypage 4.0

Mit der Paypage steht eine Bezahlseite für den Kunden zur Verfügung. Der Kunde kann auf dieser Seite für seinen Warenkorb ein Zahlverfahren auswählen und die bezahlen.

Wenn man Kassenzeichen für die Bezahlung über die Paypage 4.0 anlegen will, dann muss man beim Übertragen der Buchungsliste den Zahltyp auf „PAYPAGE“ setzen.

Da die Paypage alle Zahlverfahren unterstützt und auch das Anlegen von SEPA-Mandaten ermöglicht (bei einer internen SEPA-Mandatsverwaltung), können beim Übertragen der Buchungsliste beliebige Listen von Zahlverfahren übergeben werden.

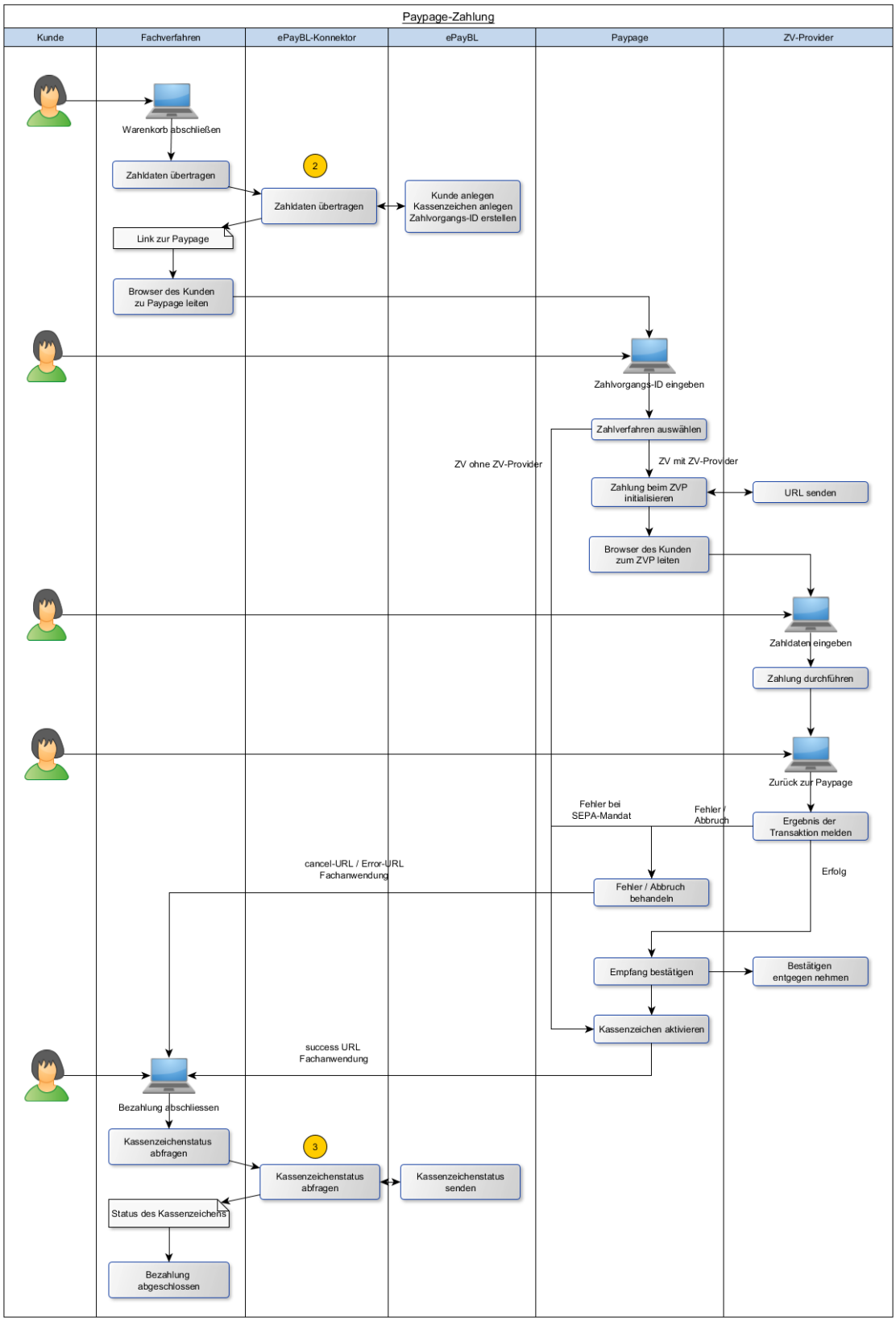


Abbildung 15: Zahlungen über die Paypage

1. Der Kunde schließt seinen Warenkorb / Einkauf beim Fachverfahren ab.

2. Das Fachverfahren überträgt die Buchungsliste mit dem Zahltyp „PAYPAGE“ an die ePayBL über den Konnektor und erhält als Antwort zum einen das Kassenzeichen und zum anderen einen Link auf die Paypage mit einer Zahlvorgangs-ID.

3. Das Fachverfahren kann den Browser des Kunden direkt auf diese URL leiten oder es kann dem Kunden über die Rechnung diesen Link zukommen lassen, so dass der Kunde separat diesen Paypage-Link aufruft.

4. Hat der Kunde den Link zur Paypage aufgerufen oder wurde er dorthin geleitet, hat er dort eine Ansicht mit den Zahlungsdaten und eine Auswahl der zur Verfügung stehenden Zahlverfahren. Wird nur die Basis-URL der Paypage aufgerufen, erscheint eine Seite mit einer Eingabemöglichkeit für die Zahlvorgangs-ID. Nach Eingabe der Zahlvorgangs-ID gelangt der Nutzer wieder zur Zahlverfahrensauswahl.

5. Je nach ausgewähltem Zahlverfahren wird die Zahlungsdurchgeführt.

6. Bei Zahlverfahren, die über ZV-Provider abgewickelt werden (Kreditkarte, giropay, PayPal, paydirekt) wird die Zahlung durch die Paypage am Provider initialisiert. Der Kunde wird dann weitergeleitet zum ZV-Provider, wo er seine Zahlungsdaten (Kreditkartennummer, Anmeldedaten) eingibt und die Zahlung durchgeführt wird. Bei erfolgreicher Zahlung ruft der ZV-Provider eine entsprechende Success-URL an der Paypage auf. Die Paypage aktiviert daraufhin das Kassenzeichen und leitet den Browser des Kunden zur Success-URL des Fachverfahrens. Wenn Fehler bei der Zahlung passieren oder der Kunde die Zahlung beim ZV-Provider abbricht, ruft dieser die Error- bzw. Cancel-URL der Paypage auf, die daraufhin die Error- oder Cancel-URL des Fachverfahrens aufruft. So gelangt der Kunde zur Error- bzw. Cancel-Seite des Fachverfahrens

7. Bei Zahlverfahren, die nicht über ZV-Provider abgewickelt werden (SEPA-Lastschrift, Überweisung), wird zunächst bei SEPA geprüft, ob ein SEPA-Mandat vorhanden ist. Bei einer internen Mandatsverwaltung wird für den Kunden dann eventuell ein Mandat angelegt. Im Erfolgsfall wird das Kassenzeichen aktiviert und der Kunde zur Success-Seite der Fachanwendung zurückgeleitet. Im Fehlerfall wird die Error-URL der Fachanwendung aufgerufen.

8. Das Fachverfahren kann zusätzlich über den Konnektor den Status des Kassenzeichens abfragen.

2.11 Bestandskunden

Die ePayBL unterstützt Bestandskunden von Fachverfahren. Während temporäre Kunden nach dem Übertragen der Buchungsliste in der ePayBL gelöscht werden, können Bestandskunden mehrere Buchungslisten zugeordnet werden.

Neue Bestandskunden können mittels der REST-Schnittstelle angelegt werden. Das Anlegen erfolgt implizit über die Kundeninformation beim Übergeben von Buchungslisten. Ändern sich bei einem bereits vorhandenen Kunden die Daten, wird der Kunde entsprechend modifiziert. Auf diese Weise können auch Adress- oder Bankdaten des Kunden geändert werden.

Wird der Kunde im Fachverfahren nicht mehr benötigt, so wird empfohlen, diesen auch innerhalb der ePayBL über die Kunden-Löschen-Methode zu entfernen.

Achtung: ein gelöschter Kunde kann anschließend weder verwendet noch erneut angelegt werden.

2.12 Zahltypen

Die verschiedenen Varianten zum Bezahlen werden über den Zahltyp abgewickelt. Es gibt folgende Zahltypen:

Zahltyp	Erlaubte Zahlverfahren	Beschreibung
DIREKT	"UEBERWEISUNGVOR", "UEBERWEISUNGNACH", "KREDITKARTE", "GIROPAY","PAYPAL", "SEPASDD", „PAYDIREKT“, „BARZAHLUNG“, „TERMINALZAHLUNG“, „LASTSCHRIFTOHNE“	In der Buchungsliste kann genau eines diese Zahlverfahren angegeben werden. Bei Zahlverfahren, die über ZV-Provider abgewickelt werden, erhält das Fachverfahren einen Link zum Provider. Bei den anderen Zahlverfahren wird das Kassenzeichen gleich aktiviert
RECHNUNG	"KREDITKARTE", "GIROPAY","PAYPAL", „PAYDIREKT“	Die Buchungsliste kann eine Liste der angegebene Zahlverfahren enthalten. Für jedes Zahlverfahren erhält man eine URL über die der Bürger mit diesem Zahlverfahren bezahlen kann.
PAYPAGE	"UEBERWEISUNGVOR", "UEBERWEISUNGNACH", "KREDITKARTE", "GIRO-	Die Buchungsliste kann eine beliebige Auswahl aus diesen Zahlverfahren enthalten. Über die Paypage kann der Bürger dann

	PAY","PAYPAL", "SEPASDD", „PAYDIREKT“, „BARZAHLUNG“, „TERMINALZAHLUNG“, „LASTSCHRIFTOHNE“	daraus ein Zahlverfahren auswählen und die Bezahlung durchführen.
TCONNECT	„TERMINALZAHLUNG“	Es ist nur diese Zahlverfahren möglich. Im Unterschied zum Zahltyp direkt wird das Kassenzeichen nicht aktiviert. Das kann separat erfolgen

Tabelle 2: Zahltypen strukturieren jeweils erlaubte Zahlverfahren

3 Kommunikation mit dem Konnektor

3.1 Durch das Fachverfahren

Fachanwendung und Konnektor sowie Konnektor und ePayBL kommunizieren über REST-Methoden. Der Konnektor leitet die Requests der Fachanwendung im Prinzip 1:1 an die ePayBL weiterleiten. Daher gibt es zu jeder Methode, die die Fachanwendung am Konnektor aufrufen kann, eine entsprechende Methode an der ePayBL, die vom Konnektor aufgerufen wird. Die gelben Nummern in der Tabelle bezeichnen also zum einen die Methoden, die die Fachanwendung am Konnektor aufruft, also auch die Methoden, die der Konnektor an der ePayBL aufruft. Achtung: Die hier angegebenen Namen der Methoden stellen nicht die gültigen URLs für die REST-API dar.

Nr	Methode	Beschreibung	Parameter
1	Zahlverfahren abfragen	gibt die Liste der am Mandanten aktiven Zahlverfahren mit Min-/Max-Werten, Durchführung über Provider zurück GET-Methode	Eingabe: • Mandantennummer, • Bewirtschafternummer Ausgabe: • Ergebnis (Code, Text) • Liste mit Zahlverfahren (Zahlverfahrenscode, Minimumbetrag, Maximalbetrag, ob die Abarbeitung über den Zahlverkehrsprovider erfolgt oder nicht)
2	Buchungslisten übertragen	Kunden- und Zahldaten übermitteln und Bezahlung initialisieren • abhängig von den Eingabeparametern wird der mitgelieferte Kunde gesucht und angelegt, wenn er nicht gefunden wird, bzw. es wird ein Fehler geworfen • anlegen einer Sollstellung entsprechend den Buchungsdaten (bei SEPA: falls Mandat vorhanden) • Aktivierung des Kassenzeichens bei Zahlverfahren SEPA (SEPA-Mandat muss vorhanden sein) und Überweisung • falls die Zahlung über einen ZV-Provider abgewickelt wird: Initialisierung der Zahlung über den ZV-Provider • Kunden zum Löschen markieren, bei temporären Kunden	Eingabe: • Mandantennummer, • Bewirtschafternummer, • Buchungsliste (Kassenzeichenstring (optional), Fälligkeitsdatum, Währungskennzeichen, Transaktionsnummer, Zahlverfahrenscode, Liste von Buchungen, Kunde, Listen von Buchungslistenparameter, Liste von Fachverfahrensparameter, Zahltyp) Ausgabe: • Ergebnis (Code, Text) • Provider-URL • Rechnungs-ID • SEPA-Mandat • Kassenzeichen • Kundennummer • Transaktionsnummer

		• Speicherung der Zahlung in der Datenbank (Kassenzeichen, Back-URL des Fachverfahrens, Aufruf-URL des ZV-Providers) POST-Methode	
3	Kassenzeichenstatus abfragen	Gibt den Status einer Liste von Kassenzeichen zurück: Paypage-Status, Aktivierungszeitpunkt, Saldo GET-Methode	Eingabe: • Mandantennummer, • Bewirtschafternummer • Kassenzeichenstring Ausgabe: • Ergebnis (Code, Text) • Kassenzeichen (Kassenzeichenstring, Zusatzdaten)
4	Sepamandate anlegen	Legt ein SEPA-Mandat an POST-Methode	Eingabe: • Mandantennummer, • Bewirtschafternummer • Mandatsdaten Ausgabe: • Ergebnis (Code, Text) • SEPA-Mandat • Pdf-Mandatsdatei, binär
5	Sepamandate vervollständigen	Vervollständigt ein SEPA-Mandat POST-Methode	Eingabe: • Mandantennummer, • Bewirtschafternummer • Mandatsreferenz, • Datum-der-Unterschrift Ausgabe: • Ergebnis (Code, Text) • SEPA-Mandat • Pdf-Mandatsdatei, binär
6	Kunden löschen	Löscht einen (Bestands-) Kunden DELETE-Methode	Eingabe: • Mandantennummer, • Bewirtschafternummer, • Kundennummer Ausgabe: • Ergebnis (Code, Text)
8	Kasenzeichen aktivieren	Aktiviert ein Kassenzeichen (setzt den Paypage Status auf „INAKTIV“ POST-Methode	Eingabe: • Mandantennummer, • Bewirtschafternummer • Kassenzeichenstring Ausgabe: • Ergebnis (Code, Text) • Kassenzeichen (Kassenzeichenstring, Zusatzdaten)

Tabelle 3: Die Fachanwendung kommuniziert über den Konnektor mit der epayBL

3.2 Durch den Kunden

Wenn der Kunde Dienstleistung über das Fachverfahren in Anspruch nimmt und unmittelbar bezahlt, kommt folgende Methode zur Anwendung.

Nr	Methode	Beschreibung	Parameter
7	bezahlen	Leitet den Kunden zum Durchführen der Bezahlung auf den ZV-Provider; Voraussetzung ist, dass durch das Fachverfahren die Zahlung am Provider initiiert wurde GET-Methode	Eingabe: • Rechnungs-ID Ausgabe: 1. Ergebnis-Seite (eine der vom Fachverfahren beim Übertragen der Buchungslisten angegebenen)

Tabelle 4: Kommunikation des Kunden mit dem ePayBL-Konnektor

4 Datenmodell

4.1 Buchungsliste

Parameter	Felder	Pflicht / Default	Type	Beschreibung
buchungsliste			Buchungsliste	Buchungsliste
	kassenzeichennummer	x, falls am Mandanten konfiguriert ist, dass das Fachverfahren das Kassenzeichen liefert	String	Kassenzeichen Achtung: Wenn alphanumerische Kassenzeichen und EPA-Import von Girosolution verwendet werden, dann ist darauf zu achten, dass bei den Kassenzeichen keine Kleinbuchstaben verwendet werden. Die von Girosolution erzeugten EPA-Dateien wandeln in den Kassenzeichen Kleinbuchstaben in Großbuchstaben um. Damit werden sie durch den EPA-Import nicht mehr gefunden.
	faelligkeitsdatum	x	Date (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss	Fälligkeitsdatum
	waehrungskennzeichen	EUR	String ("EUR")	Währungskennzeichen
	kennzeichenMahnverfahren	x	String	Kennzeichen fürs Mahnverfahren
	transaktionsnummer		String, falls angegeben, muss sie pro Mandant eindeutig sein	Transaktionsnummer, kann zur Identifizierung von Buchungslisten verwendet werden
	zahlverfahrencodes	x	List<String> ("UEBERWEISUNGVOR", "UEBERWEISUNGNACH", "KREDITKARTE", "GIRO PAY","PAYPAL", "SEPA SDD", „PAYDIREKT“, „BARZAHLUNG“, „TERMINALZAHLUNG“, „LASTSCHRIFTOHNE“)	Liste von Zahlverfahrenscodes. Bei Bezahlung direkt über die Fachanwendung (Zahltyp=“DIREKT“) darf die Liste nur aus genau einem Element bestehen. Bei Zahltyp „RECHNUNG“ sind nur die Zahlverfahren: KREDITKARTE, GIROPAY, PAYPAL und PAYDIREKT erlaubt. Bei Zahltyp „PAYPAGE“ kann eine beliebige Liste aus den angegebenen Zahlverfahren verwendet werden.

				Diese Liste schränkt dann die auf der Paypage verfügbare Zahlverfahrensliste ein. Bei Zahltyp „TCONNECT“ ist nur das Zahlverfahren „TERMINALZAHLUNG“ erlaubt.
	buchungen	x	List<Buchung>	Liste der Buchungen
	beschreibung	x	String	Beschreibung der Buchungsliste
	kunde	x	Kunde	Kundendaten
	buchungslistenparameter		Map<String, String>	Buchungslistenparameter (betrifft Fachverfahren und HKR)
	fachverfahrendaten	x	Map<String, String> Pflichteinträge zu Keys: "successUrl", "cancelUrl", "errorUrl"	Map von Daten des Fachverfahrens, insbesondere Rücksprungadressen (Pflicht): • Error-URL für Abort/Error, • Cancel-URL für Cancel/Back und, • Success-URL für Success
	zahltyp	x	enum Zahltyp (DIREKT, RECHNUNG, PAYPAGE, TCONNECT)	DIREKT: das Fachverfahren erhält einen Link zum ZV-Provider über den der Kunde die Rechung bezahlen kann RECHNUNG: das Fachverfahren erhält für jedes angegeben Zahlverfahren eine URL zum Konnektor. Der Kunde kann entsprechend dem gewünschten Zahlverfahren eine URL aufrufen und darüber die Rechnung bezahlen. PAYPAGE: das Fahcverfahren erhält einen Link zur Paypage (mit ZahlungsvorgangsID) Über diesen Link kann der Kunde die Rechnung mittels Paypage bezahlen. TCONNECT: Das Kassenzeichen wird nicht automatisch aktiviert. Der Kunde kann zunächst die Rechnung über ein Bezahlterminal bezahlen und das Fachverfashren kann dann separat das Kassenzeichen aktivieren.
	betrag	x	BigDecimal	Gesamtbetrag der Buchungsliste

Tabelle 5: Struktur der Buchungsliste

4.2 Buchung

Parameter	Komponenten	Pflicht	Type	Beschreibung
buchung			Buchung	Buchung
	bruttobetrag	x	BigDecimal	Bruttobetrag der Buchung
	nettobetrag		BigDecimal	Nettobetrag der Buchung
	id		String	Identifizierung der Buchung (z.B. Belegnummer)
	steuersatz		BigDecimal	Steuersatz der Buchung
	steuerbetrag		BigDecimal	Steuerbetrag der Buchung
	buchungstext	x	String	Buchungstext der Buchung
	kontierung	x	Map<String, String> Keys: "haushaltstelle", "objektnummer", "href"	Kontierungsdaten der Buchung (z.B. Haushaltsstelle, Objektnummer, HREF) (Die Werte für Haushaltsstelle und Objektnummer müssen mit denen am Bewirtschafter übereinstimmen)

Tabelle 6: Struktur einer Buchung

4.3 Kunde

Parameter	Komponenten	Pflicht / Default	Type	Beschreibung
kunde			Kunde	Kundendaten
	name	x, falls „adresse“ oder „bankverbindung“ angegeben ist, insbesondere, wenn Die Zahlverfahren SEPA-Lastschrift oder giropay verwendet werden	String (max. 27 Zeichen)	Nachname des Kunden
	vorname		String (max. 27 Zeichen)	Vorname des Kunden

	typ	x	enum KundeTyp (TEM PORAER, BE-STAND)	Beschreibt, ob der Kunde vorhanden sein muss, ob der Kunde angelegt wird, und ob er zum Löschen markiert werden soll
	firmenkunde	false	boolean	Gibt an, ob der Kund eine Firma ist
	firmenname		String	Name der Firma, falls es sich um eine Firma handelt
	kundennummer	x	String (max. 100 Zeichen) beim Zahlverfahren Paydirekt darf die Kundennummer nur maximal 20 Zeichen lang sein	Nummer des Kunden
	anrede		enum Anrede (AN, EHE LEUTE, FIRMA, FRAU, FRAU_UND_HERRN, FRAUEN, FRAEULEIN, HERR, HERREN, HERRN_UND_FRAU)	Anrede
	emailadresse		String	E-Mail-Adresse des Kunden
	sepamandat		Sepamandat	SEPA-Mandatsdaten des Kunden oder SEPA-Mandatsreferenz
	adresse		Adresse	Adresse
	bankverbindung	x, falls Zahlverfahren SEPA-Lastschrift verwendet wird	Bankverbindung	Bankverbindung

Tabelle 7: Struktur eines Kunden

4.4 Adresse

Parameter	Komponenten	Pflicht	Type	Beschreibung
adresse			Adresse	Adresse
	plz	x	String	Postleitzahl
	ort	x	String	Ort
	strasse	x, falls kein Postfach angegeben	String	Straße
	hausnummer	x, falls kein Postfach angegeben	String	Hausnummer
	land	x	String: ISO 3166-1, zwei Großbuchstaben	Land

	postfach	x, falls keine Straße angegeben	String	Postfach

Tabelle 8: Struktur einer Adresse

4.5 Bankverbindung

Parameter	Komponenten	Pflicht	Type	Beschreibung
bankverbindung			Bankverbindung	Bankverbindung
	kontoinhaber	Nur beim Zahlverfahren giropay über den ZV-Provider GiroSolution kann der Konto-inhaber weggelassen werden	String	Name des Kontoinhabers
	iban	Nur beim Zahlverfahren giropay über den ZV-Provider GiroSolution kann die IBAN weggelassen werden	String	IBAN
	bic		String	BIC

Tabelle 9: Struktur einer Bankverbindung

4.6 Zahlverfahren

Parameter	Komponenten	Pflicht / Default	Type	Beschreibung
zahlverfahren			Zahlverfahren	Beschreibt ein konfiguriertes Zahlverfahren
	zahlverfahrencode	x	String ("UEBERWEISUNGVOR", "UEBER WEI-SUNGNACH", "KRE DITKARTE", "GIROPAY", "PAYPAL", „PAYDIREKT“, "SEPASDD", „BARZAH-LUNG“, „TERMINALZAHLUNG“)	Code des Zahlverfahrens
	minimalbetrag		BigDecimal	Minimal möglicher Betrag
	maximalbetrag		BigDecimal	Maximal möglicher Betrag
	zvp	false	boolean	Gibt an, ob das Zahlverfahren über den ZV-Provi-der abgewickelt wird oder nicht
	pruefungGesperrterKunde	false	boolean	Beim Zahlverfahren wird geprüft, ob der Kunde gesperrt ist

Tabelle 10: Struktur eines Zahlverfahrens

4.7 Kassenzeichen

Parameter	Komponenten	Pflicht	Type	Beschreibung
kassenzeichen			Kassenzeichen	Kassenzeichen
	kassenzeichennummer	x	String	Kassenzeichen-String
	zusatzdaten	x	Map<String, Object> Keys [Typ]: "paypagestatus" [com.fasterxml.jackson.databind.node.TextNode], "aktivierungszeit" [com.fasterxml.jackson.databind.node.TextNode (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss] „bezahlverfahren“ (com.fasterxml.jackson.databind.node.TextNode) "transaktionsnummer" [com.fasterxml.jackson.databind.node.TextNode], "zahlungsvorgangsID" [com.fasterxml.jackson.databind.node.TextNode], "saldo" [com.fasterxml.jackson.databind.node.NumericNode], "betragStorno" [com.fasterxml.jackson.databind.node.NumericNode]	Paypage-Status, Aktivierungszeitpunkt, Bezahlverfahren, Transaktionsnummer (siehe Buchungsliste), ZahlungsvorgangsIDs Saldo, BetragStorno, BetragSonstiges, BetragHauptforderungen, BetragZahlungseingang, BetragLastschrift BetragRücklastschriften Als Bezahlverfahren wird der Zahlverfahrenscode des verwendeten Zahlverfahrens ausgegeben. Wenn das Kassenzeichen noch nicht bezahlt wurde und daher noch kein Zahlverfahren verwendet wurde wird hier „KEINZAHLVERFAHREN“ ausgegeben. Ansonsten sind die möglichen Werte: "UEBERWEISUNGVOR"; "UEBERWEISUNGNACH"; "LASTSCHRIFTMIT", "LASTSCHRIFTOHNE", "KREDITKARTE"; "GIROPAY", "SEPASDD", "PAYPAL", "PAYDIREKT", "BARZAHLUNG" oder „TERMINALZAHLUNG“

			"betragSonstiges" [com.fasterxml.jackson.databind.node.NumericNode] "betragHauptforderungen" [com.fasterxml.jackson.databind.node.NumericNode] "betragRLS" [com.fasterxml.jackson.databind.node.NumericNode] "betragZahlungseingang" [com.fasterxml.jackson.databind.node.NumericNode] "betragLastschrift" [com.fasterxml.jackson.databind.node.NumericNode]	Beim Zahltyp DIREKT werden alle ZahlungsvorgangsIDs ausgegeben zusammen mit dem jeweiligen Zahleverfahren: „zvid1:Zahlverfahren1, zvid2: Zahlverfahren2, …“. Beim Zahltyp PAYPAGE wird die ZahlungsvorgangsID zusammen mit dem Zahltyp ausgegeben: „zvid1:PAYPAGE“

Tabelle 11: Struktur eines Kassenzeichens

Auf die Zusatzdaten kann (in Java) z.B. auf folgende Weise zugegriffen werden:

```
Map<String,Object> mapZusatzdaten =
       abfragenKassenzeichenstatusErgebnis.getKassenzeichen().getZusatzda-
ten();
com.fasterxml.jackson.databind.node.TextNode paypagestatus =
       (com.fasterxml.jackson.databind.node.TextNode)
       mapZusatzdaten.get("paypagestatus");

com.fasterxml.jackson.databind.node.TextNode bezahlverfahren =
        (com.fasterxml.jackson.databind.node.TextNode)
         mapZusatzdaten.get("bezahlverfahren");

com.fasterxml.jackson.databind.node. NumericNode saldo =
       (com.fasterxml.jackson.databind.node. NumericNode)
        mapZusatzdaten.get("saldo");

com.fasterxml.jackson.databind.node.TextNode aktivierungszeit =
        (com.fasterxml.jackson.databind.node.TextNode)
         mapZusatzdaten.get("aktivierungszeit");

System.out.println("paypagestatus = " + paypagestatus.asText());
System.out.println("bezahlverfahren = " + bezahlverfahren.asText());
System.out.println("saldo = " + saldo.asText());

if (aktivierungszeit == null) {
         System.out.println("aktivierungszeit = null");
}
else {
         System.out.println("aktivierungszeit = " +
                        aktivierungszeit.asText());
}
```

4.8 SEPA-Mandat

Je nach Anwendungszweck besteht das SEPA-Mandats-Objekt aus unterschiedlichen Daten. Bei interner Mandatsverwaltung muss nur die Referenz-Nummer übergeben werden, um das Mandat zu identifizieren. Bei externer Mandatsverwaltung muss das Mandat vollständig übergeben werden.

Die folgende Tabelle gibt einen Überblick darüber, welche Parameter in welcher Situation Pflicht (X) oder optional (O) sind und welche Default-Werte angenommen werden.

Parameter	abstrakter Basistyp	SepamandatIntern-Referenz	SepaMandat-InternAnlege-daten	SepamandatIn-ternVervollsta-endigungsdaten	Sepa-manda-tExtern	Sepamandat-Ergebnisdaten
xxtypetagxx	X	X	X	X	X	X
mandatreferenz	-	X	O	-	X	X
einmalmandat	-	-	false	-	-	-
type	-	-	"B"	-	X	X
kundennummer	-	-	X	-	X	X
glaeubigerId	-	-	X	-	X	X
zeichnungsberechtig-ter	-	-	O	-	O	O
datumUnterschrift	-	-	-	X	X	X
ortUnterschrift	-	-	-	O	O	O
sequenceType	-	-	-	-	X	X
datumLetzteNutzung	-	-	-	-	X	X
active	-	-	-	-	-	X
sepamandatdatei	-	-	-	-	-	O
kontoinhaberWeich-tAb	-	-	false	-	false	false
kontoinhaberNach-name	-	-	O	-	O	O
kontoinhaberVorname	-	-	O	-	O	O
kontoinhaberStrasse	-	-	O	-	O	O
kontoinhaberHaus-nummer	-	-	O	-	O	O
kontoinhaberPLZ	-	-	O	-	O	O
kontoinhaberStadt	-	-	O	-	O	O
kontoinhaberLand	-	-	O	-	O	O
kontoinhaberBank-name	-	-	O	-	O	O
kontoinhaberIban	-	-	O	-	O	O
kontoinhaberBic	-	-	O	-	O	O

Tabelle 12: Überblick über die Pflichtparameter beim SEPA-Mandat

4.8.1 Abstrakter Basistyp

Parameter	Komponenten	Pflicht	Type	Beschreibung
sepamandat			Sepamandat	SEPA-Mandat – Daten je nach Usecase
	xxtypetagxx	x	enum Typetag (SepamandatInternReferenz, SepaMandatInternAnlegedaten, Sepaman datInternVervollstaendigungsdaten, Sepa mandatExtern, SepamandatErgebnisdaten)	Angabe des Usecases

Tabelle 13: Abstrakte SEPA-Mandat-Grundform

4.8.2 SepamandatInternReferenz

Parameter	Komponenten	Pflicht	Type	Beschreibung
sepamandat			Sepamandat	SEPA-Mandat – Daten je nach Usecase
	xxtypetagxx	x	Fixwert: "SepamandatInternReferenz"	Angabe des Usecases
	mandatreferenz	x	String	Mandatreferenz eines existierenden intern verwalteten Mandats

Tabelle 14: Struktur eines SEPA-Mandats mit alleiniger Referenzangabe

4.8.3 SepaMandatInternAnlegedaten

Parameter	Komponenten	Pflicht / Default	Type	Beschreibung
sepamandat			Sepamandat	SEPA-Mandat – Daten je nach Usecase
	xxtypetagxx	x	Fixwert: "SepaMandatIn ternAnlegedaten"	Angabe des Usecases
	einmalmandat	false	boolean	Verwendung einmalig oder mehrfach
Zusatzdaten B				
	type	"B"	enum SepaMandatType (B, F, V)	Typ des Mandats
	mandatreferenz	Abhängig von Systemeinstellungen	String	ID des Mandats
	kundennummer	x	String	Kundennummer
	glaeubigerId	Wert des Mandanten	String	Gläubiger-ID
	zeichnungsberechtigter		String	Zeichnungsberechtigter
	kontoinhaberWeichtAb	false	boolean	Kontoinhaber vom Kunden abweichend?

	kontoinhaberNachname	x, falls konto inhaber-WeichtAb==true	String	Angaben zum abweichenden Kontoinhaber
	kontoinhaberVorname	dito	String	dito
	kontoinhaberStrasse	dito	String	dito
	kontoinhaberHausnummer	dito	String	dito
	kontoinhaberPLZ	dito	String	dito
	kontoinhaberStadt	dito	String	dito
	kontoinhaberLand	dito	String	dito
	kontoinhaberBankname	dito	String	dito
	kontoinhaberIban	dito	String	dito
	kontoinhaberBic	dito, und abhängig von Systemeinstellungen	String	dito

Tabelle 15: Struktur eines SEPA-Mandats zum Anlegen eines internen Mandats

4.8.4 SepamandatInternVervollstaendigungsdaten

Parameter	Komponenten	Pflicht	Type	Beschreibung
sepamandat			Sepamandat	SEPA-Mandat – Daten je nach Usecase
	xxtypetagxx	x	Fixwert: "SepamandatInternVervollstaendigungsdaten"	Angabe des Usecases
	datumUnterschrift	x	Date (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss	Datum der Unterschrift
	ortUnterschrift		String	Ort der Unterschrift

Tabelle 16: Struktur eines SEPA-Mandats zum Vervollständigen eines internen Mandats

4.8.5 SepamandatExtern

Parameter	Komponenten	Pflicht / Default	Type	Beschreibung
sepamandat			Sepamandat	SEPA-Mandat – Daten je nach Usecase
	xxtypetagxx	x	Fixwert: "SepamandatExtern"	Angabe des Usecases
Zusatzdaten A				
	sequenceType	x	enum SepaMandatSequenceType: FRST (Erstlastschrift), RCUR (Folgelastschrift), OOFF (Einmallastschrift), FNAL (letzte Lastschrift)	Sequenztyp des Mandats

	datumUnterschrift	x	Date (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss	Datum der Unterschrift
	ortUnterschrift		String	Ort der Unterschrift
	datumLetzteNutzung	x	Date (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss	Datum der letzten Nutzung
Zusatzdaten B				
	type	x	enum SepaMandatType (B, F, V)	Typ des Mandats
	mandatreferenz	x	String	ID des Mandats
	kundennummer	x	String	Kundennummer
	glaeubigerId	x	String	Gläubiger-ID
	zeichnungsberechtigter		String	Zeichnungsberechtigter
	kontoinhaberWeichtAb	false	boolean	Kontoinhaber vom Kunden abweichend?
	kontoinhaberNachname		String	Angaben zum abweichenden Kontoinhaber
	kontoinhaberVorname		String	dito
	kontoinhaberStrasse		String	dito
	kontoinhaberHausnummer		String	dito
	kontoinhaberPLZ		String	dito
	kontoinhaberStadt		String	dito
	kontoinhaberLand		String	dito
	kontoinhaberBankname		String	dito
	kontoinhaberIban		String	dito
	kontoinhaberBic		String	dito

Tabelle 17: Struktur eines SEPA-Mandats bei externer Mandatsverwaltung

4.8.6 SepamandatErgebnisdaten

Parameter	Komponenten	Pflicht	Type	Beschreibung
sepamandat			Sepamandat	SEPA-Mandat – Ergebnisdaten
	xxtypetagxx	x	Fixwert: "SepamandatErgebnisdaten"	Angabe des Usecases
	active	x	boolean	Aktiv?
	sepamandatdatei	x, bei Anlegen, ansonsten, falls bereits erzeugt	byte[]	PDF-Datei mit dem Mandat
Zusatzdaten A				
	sequenceType	x	enum SepaMandatSequenceType: FRST (Erstlastschrift),	Sequenztyp des Mandats

			RCUR (Folgelastschrift), OOFF (Einmallastschrift), FNAL (letzte Lastschrift)	
	datumUnterschrift	x	Date (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss	Datum der Unterschrift
	ortUnterschrift		String	Ort der Unterschrift
	datumLetzteNutzung	x	Date (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss	Datum der letzten Nutzung
Zusatzdaten B				
	type	x	enum SepaMandatType (B, F, V)	Typ des Mandats
	mandatreferenz	x	String	ID des Mandats
	kundennummer	x	String	Kundennummer
	glaeubigerId	x	String	Gläubiger-ID
	zeichnungsberechtigter		String	Zeichnungsberechtigter
	kontoinhaberWeichtAb	x	boolean	Kontoinhaber vom Kunden abweichend?
	kontoinhaberNachname		String	Angaben zum abweichenden Kontoinhaber
	kontoinhaberVorname		String	dito
	kontoinhaberStrasse		String	dito
	kontoinhaberHausnummer		String	dito
	kontoinhaberPLZ		String	dito
	kontoinhaberStadt		String	dito
	kontoinhaberLand		String	dito
	kontoinhaberBankname		String	dito
	kontoinhaberIban		String	dito
	kontoinhaberBic		String	dito

Tabelle 18: Struktur der SEPA-Mandat-Ergebnisdaten

4.9 Zahlvorgangsinformation

Parameter	Komponenten	Pflicht	Type	Beschreibung
zahlvorgangsInfo			ZahlvorgangsInfo	Zahlvorgangs-Information nach Übertragen einer Buchungsliste
	rechnungUrls		Map<String, String>	Liste von Key-Value-Paaren. Im Fall Zahltyp="RECHNUNG", sind die Keys die jeweiligen Zahlverfahrenscodes und die Values die jeweilige URL mit der RechnungsID; im Fall Zahltyp="DIREKT" ist der Key „ZVP" und der Value der Link zum ZV-Provider; im Fall Zahltyp="PAYPAGE" ist der Key „PAYPAGE" und der Value der Link zu Paypage mit der entsprechenden Zahlungsvorgangs-ID.
	sepamandat		SepamandatErgebnisdaten	SEPA-Mandats-Informationen
	kassenzeichen		Kassenzeichen	Kassenzeichen-Informationen
	kundennummer		String	Kundennummer
	transaktionsnummer		String	Transaktionsnummer

Tabelle 19: Struktur der Zahlvorgangsinformation

4.10 Ergebnisse

4.10.1 Ergebnis

Parameter	Komponenten	Pflicht	Type	Beschreibung
ergebnis			Ergebnis	Ergebnis-Code eines Requests
	rc	x	String	Return-Code, alle positiven Werte bedeuten Erfolg.
	ergebnistext		String	Fehlertext

Tabelle 20: Struktur des Ergebnis-Codes

4.10.2 ErgebnisErgebnis

Parameter	Komponenten	Pflicht	Type	Beschreibung
basisErgebnis			ErgebnisErgebnis	Basis-Klasse für ein komplexes Ergebnis-objekt mit spezifischen Daten
	ergebnis	x	Ergebnis	Ergebnis-Code
	date	x	Date (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss	Datum und Uhrzeit der Ergebnislieferung

Tabelle 21: Struktur des Basis-Ergebnisses für anfragespezifische Ergebnisobjekte

4.10.3 ZahlverfahrenListErgebnis

Parameter	Komponenten	Pflicht	Type	Beschreibung
zahlverfahrenListErgebnis			ZahlverfahrenListErgebnis	Ergebnis bei Abfrage der Zahlverfahren
	ergebnis	x	Ergebnis	Ergebnis-Code
	date	x	Date (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss	Datum und Uhrzeit der Ergebnislieferung
	zahlverfahren		List<Zahlverfahren>	Liste verfügbarer Zahlverfahren

Tabelle 22: Struktur des Ergebnisses für Abfrage der Zahlverfahren

4.10.4 KassenzeichenstatusErgebnis

Parameter	Komponenten	Pflicht	Type	Beschreibung
kassenzeichensta-tusErgebnis			KassenzeichenstatusErgebnis	Ergebnis bei Abfrage des Kas-senzeichenstatus
	ergebnis	x	Ergebnis	Ergebnis-Code
	date	x	Date (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss	Datum und Uhrzeit der Ergeb-nislieferung
	kassenzeichen		Kassenzeichen	Kassenzeicheninformation

Tabelle 23: Struktur des Ergebnisses für Abfrage des Kassenzeichenstatus

4.10.5 ZahlvorgangsErgebnis

Parameter	Komponenten	Pflicht	Type	Beschreibung
zahlvorgangsErgebnis			ZahlvorgangsErgebnis	Ergebnis bei Übertragen einer Buchungsliste
	ergebnis	x	Ergebnis	Ergebnis-Code
	date	x	Date (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss	Datum und Uhrzeit der Ergeb-nislieferung
	zahlvorgangsInfo		ZahlvorgangsInfo	Informationen über den Zahl-vorgang

Tabelle 24: Struktur des Ergebnisses für das Übertragen einer Buchungsliste

4.10.6 SepamandatErgebnis

Parameter	Komponenten	Pflicht	Type	Beschreibung
sepamandat-Ergebnis			SepamandatErgebnis	Ergebnis beim Anlegen oder vervollständigen eines SEPA-Mandats
	ergebnis	x	Ergebnis	Ergebnis-Code
	date	x	Date (Datum + Uhrzeit) dd.MM.yyyy HH:mm:ss	Datum und Uhrzeit der Er-gebnislieferung
	sepamandatErgebnisdaten		SepamandatErgebnisdaten	SEPA-Mandatsdaten

Tabelle 25: Struktur des Ergebnisses für das Anlegen oder Vervollständigen eines SEPA-Mandats

5 Methoden

5.1 Zahlverfahren abfragen

Pfad	GET /epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirtschafternr}/zahl-verfahren

Eingabe: Keine weiteren Eingabeparameter im Body

Ausgabe: ZahlverfahrenListErgebnis – Json-codiert.

Beispiel:


```
{
	"ergebnis" : {
		"rc" : "+0000",
		"ergebnistext" : "ok"
	},
	"date" : "17.11.2017 17:05:14",
	"zahlverfahren" : [
		{
			"zahlverfahrencode" : "KREDITKARTE",
			"minimalbetrag" : 0.00,
			"maximalbetrag" : 1000.00,
			"zvp" : true
		},
		{
			"zahlverfahrencode" : "PAYPAL",
			"minimalbetrag" : 0.00,
			"maximalbetrag" : 1000.00,
			"zvp" : false
		},
		{
			"zahlverfahrencode" : "GIROPAY",
			"minimalbetrag" : 0.00,
			"maximalbetrag" : 1000.00,
			"zvp" : true
		}
		{
			"zahlverfahrencode" : "PAYDIREKT",
			"minimalbetrag" : 0.00,
			"maximalbetrag" : 1000.00,
			"zvp" : true
		}
	]
}
```

5.2 Buchungsliste übergeben

Pfad	POST /epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirtschafternr}/buchungslisten

Eingabe: Buchungsliste – Json-codiert im Body.

Beispiel:


```
{
	"kassenzeichennummer":null,
	"faelligkeitsdatum":"31.12.2018 09:09:35",
	"waehrungskennzeichen":"EUR",
	"kennzeichenMahnverfahren":"16",
	"transaktionsnummer":null,
	"zahlverfahrencodes":[
		"KREDITKARTE"
	],
	"buchungen":[
		{
			"bruttobetrag":10,
			"nettobetrag":9,
			"id":null,
			"steuerbetrag":1,
			"buchungstext":"Buchung 1",
			"kontierung":{
				"objektnummer":"129001",
				"haushaltstelle":"4421030
			}
		},
		{
			"bruttobetrag":5,
			"nettobetrag":4,
			"id":null,
			"steuerbetrag":1,
			"buchungstext":"Buchung 2",
			"kontierung":{
				"objektnummer":"129001",
				"haushaltstelle":"4421030"
			}
		},
		{
			"bruttobetrag":5,
			"nettobetrag":4,
			"id":null,
			"steuerbetrag":1,
			"buchungstext":"Buchung 3",
```

```
                        "kontierung":{
                                "haushaltstelle":"4421030",
                                "objektnummer":"129001"
                        }
                }
        ],
        "beschreibung":"Buchungsliste mit Buchungen: 3",
        "kunde":{
                "name":"Cocnito",
                "vorname":null,
                "typ":"TEMPORAER",
                "firmenname":null,
                "kundennummer":"2018011801",
                "anrede":null,
                "emailadresse":null,
                "adresse":null,
                "sepamandat":null,
                "bankverbindung":{
                        "kontoinhaber":null,
                        "iban":null,
                        "bic":"TESTDETT421"
                },
                "firmenkunde":false
        },
        "buchungslistenparameter":{
                "irgendwas":"0815",
                "stsl":"19"
        },
        "fachverfahrendaten":{
                "successUrl" : "http://fachverfahren/paypment?result=success",
                "errorUrl" : "http://fachverfahren/paypment?result=error",
                "cancelUrl" : "http://fachverfahren/paypment?result=cancel"
        },
        "zahltyp":"DIREKT",
        "betrag":20
}
```

Bei der Nutzung von SEPA kann das Mandat spezifiziert werden als: Externes, bestehendes internes – via Referenz -, oder aber neues internes – via Anlegedaten.

Das sieht beispielsweise so aus:
1. Verwendung einer internen Mandatsverwaltung:

```
"sepamandat":{
        "xxtypetagxx" : "SepamandatInternReferenz",
        "mandatreferenz" : "KLAUSIRESTTESTSEPA_I11012017a5"
    },
```

2. Anlegen eines internen Mandats:

```
"sepamandat":{
        "xxtypetagxx" : "SepaMandatInternAnlegedaten",
        "einmalmandat" : true,
        "type" : "V",
        "mandatreferenz" : null,
        "kundennummer" : "k242010",
        "glaeubigerId" : "DE98ABC09999999997",
        "kontoinhaberWeichtAb" : false
    },
```

3. Übertragen eines externen Mandats:

```
"sepamandat":{
        "xxtypetagxx" : "SepamandatExtern",
        "sequenceType" : "OOFF",
        "datumUnterschrift" : "01.12.2016 00:00:00",
        "datumLetzteNutzung" : "01.12.2016 00:00:00",
        "type" : "V",
        "mandatreferenz" : "KLAUSIRE00001130120173e",
        "kundennummer" : "k242010",
        "glaeubigerId" : "DE98ABC09999999998",
        "kontoinhaberWeichtAb" : false
    },
```

Zum Vervollständigen eines SEPA-Mandats gibt es extra Methoden siehe Kapitel 5.5.

Wenn beim Übertragen einer Buchungsliste ein neues (internes) Mandat angelegt wird, werden die Buchungsdaten selbst noch nicht gespeichert, weil die ePayBL noch auf die Vervollständigung des Mandats wartet. Die Daten des angelegten unvollständigen Mandats werden als Ergebnis der Übertragung der Buchungsliste zurückgegeben. Das Fachverfahren muss dann Datum und Ort der Unterschrift des Mandats ermitteln und über die Methode „SEPA-Mandat vervollständigen“ an die ePayBL übermitteln. Beim neuerlichen Übertragen der Buchungsliste mit diesem Mandat wird dann die Buchung gespeichert und die Zahlung wird initiiert.

Achtung: Obwohl ein Einmalmandat bereits ohne Vervollständigung verwendbar ist, wird hier nicht implizit so verfahren! Das die API nutzende System muss diese Entscheidung explizit treffen, indem die Buchungsliste erneut übergeben wird – nun jedoch mit der neu erzeugten Mandatsreferenz und optional bevor die Vervollständigung erfolgte.

Ausgabe: ZahlvorgangsErgebnis – Json-codiert.

Beispiel:


```
{
	"ergebnis" : {
		"rc" : "+0000",
		"ergebnistext" : "Die Aktion wurde fehlerfrei durchgeführt"
	},
	"date" : "17.12.2018 17:05:14",
	"zahlvorgangsInfo" : {
		"sepamandat" : {
		},
		"kassenzeichen" : {
			"kassenzeichennummer" : "2016020000347",
			"zusatzdaten" : {
				"paypagestatus" : "inaktiv",
				"saldo" : "0.0",
				"aktivierungszeit" : "12.08.2018 16:13:34"
			}
		},
		"kundennummer" : "214df54bc",
		"transaktionsnummer" : "RE14030056782011187",
	" rechnungUrls " : {“ZVP”, https://test.saferpay.de/paypment?ORDE-
RID=ReB000/2016020000347?abortURl=http/server/restapi//paypment/abort}

	}
}
```

Umsetzung im restapi-Modul: Eine der wichtigsten Methoden ist das Übertragen der Buchungslisten. Im Konnektor wird der Request nur durchgereicht an die ePayBL, wo die eigentliche Logik erfolgt.

1. Zunächst wird der Kunde anhand der mitgelieferten Kundendaten in der ePayBL gesucht. Es kann konfiguriert werden, ob der Kunde bereits existieren soll, oder ob er bisher nicht vorhanden sein darf. Dem entsprechend kann es zum Fehler kommen, wenn der Kunde gefunden wird oder nicht. Wird der Kunde nicht gefunden und soll er in diesem Fall angelegt werden, dann wird er mit der mitgelieferten Kundennummer in der ePayBL angelegt.

2. Die Methode verzweigt entsprechend des mitgelieferten Zahlverfahrens.

3. Im Fall von SEPA-Lastschrift wird geprüft, ob der Kunde ein gültiges Mandat (bei einer externen Mandatsverwaltung) bzw. eine gültige Mandatsreferenz (bei einer internen Mandatsverwaltung) hat.

4. Hat der Kunde ein gültiges Mandat, so wird eine Sollstellung angelegt und das Kassenzeichen aktiviert. Handelt es sich um einen temporären Kunden, so wird er zum Löschen markiert. Der Status des Kassenzeichens wird zurückgegeben.

5. Hat der Kunde kein gültiges Mandat, so wird im Fall einer externen Mandatsverwaltung mit einem Fehler abgebrochen. Im Fall einer internen Mandatsverwaltung und im Fall, dass SEPA-Mandatsdaten übergeben wurden, wird ein Mandat angelegt. Dieses Mandat wird der Fachanwendung zurückgegeben, damit es vervollständigt werden kann.

6. Im Fall einer Bezahlung über Kreditkarte, giropay, Paypal oder PayDirekt wird zunächst die Sollstellung angelegt und das Kassenzeichen generiert, falls es nicht übergeben wurde.

7. Soll die Rechnung am Fachverfahren bezahlt werden, so wird die Zahlung am ZV-Provider initialisiert. Der Kunde wird zum Löschen markiert und die URL des ZV-Providers zusammen mit dem Kassenzeichen wird an das Fachverfahren zurückgegeben.

8. Soll der Kunde über eine Link separat bezahlen, werden die Zahlungsdaten in der Datenbank gespeichert und eine Identifikationsnummer (Rechnungs-ID) für diese Daten zusammen mit dem Kassenzeichen wird an das Fachverfahren zurückgegeben.

9. Bei einer Überweisung wird die Sollstellung angelegt, das Kassenzeichen aktiviert und der Kassenzeichenstatus zurückgegeben.

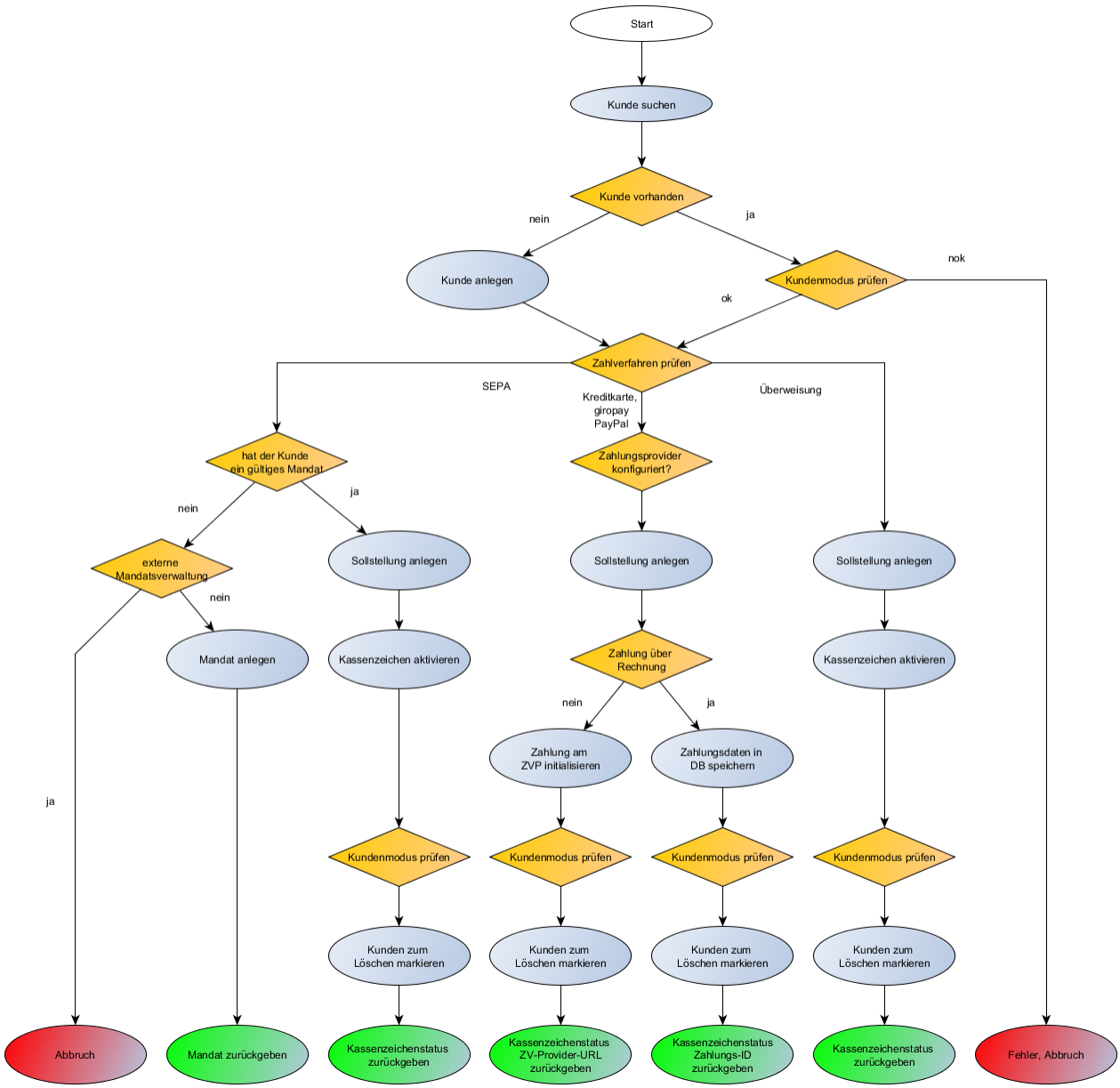


Abbildung 16: Ablauf Buchungsliste übertragen

5.3 Kassenzeichenstatus abfragen

Pfad	GET /epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirtschafternr}/kassen-zeichen/{kassenzeichennummer}

Eingabe: Keine weiteren Parameter im Body,

Ausgabe: ZahlverfahrenListErgebnis – Json-codiert im Body.

Beispiel:


```
{
  "ergebnis":   {
    "rc": "+0000",
    "ergebnistext": "Die Aktion wurde fehlerfrei durchgeführt"
  },
  "date": "17.03.2020 14:04:45",
  "kassenzeichen":   {
    "kassenzeichennummer": "242030000001",
    "zusatzdaten":      {
      "bezahlverfahren": "KREDITKARTE",
      "betragStorno": 0,
      "betragSonstiges": 0,
      "transaktionsnummer": null,
      "betragHauptforderungen": 10,
      "betragRLS": 0,
      "zahlungsvorgangsID": "",
      "saldo": 10,
      "aktivierungszeit": null,
      "betragZahlungseingang": 0,
      "betragLastschrift": 0,
      "paypagestatus": "AKTIV"
    }
  }
}
```


5.4 SEPA-Mandat anlegen

Pfad	POST /epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirtschafternr}/sepa-mandate/anlegen

Siehe auch 5.2.

Achtung: ein SEPA-Mandat kann zwar auch implizit über das Anlegen einer Buchungsliste angelegt werden, aber mit dieser Methode wird es explizit angelegt und zwar ohne eine Buchungsliste haben zu müssen.

Es gelten die üblichen Bedingungen für das Anlegen von Mandaten wie z.B.: Der Kunde darf kein aktives oder unvollständiges Mandat besitzen.

Dieser Restriktion kann umgangen werden, wenn mittels der Buchungslisten-Übertragen-Methode ein Mandat angelegt wird. Dazu ist es erforderlich, eine Bankverbindung am Kunden zu spezifizieren, die vom System implizit als neue Bankverbindung betrachtet wird. Dadurch werden bestehende Mandate am Kunden entfernt, und ein neues kann angelegt werden.

Eingabe: SepaMandatInternAnlegedaten – Json-codiert im Body

Beispiel:


```
{
	"xxtypetagxx" : "SepaMandatInternAnlegedaten",
	"type" : null,
	"mandatreferenz" : null,
	"kundennummer" : "TDG_ESHOP1",
	"glaeubigerId" : null,
	"zeichnungsberechtigter" : null,
	"kontoinhaberWeichtAb" : false,
	"kontoinhaberNachname" : null,
	"kontoinhaberVorname" : null,
	"kontoinhaberStrasse" : null,
	"kontoinhaberHausnummer" : null,
	"kontoinhaberPLZ" : null,
	"kontoinhaberStadt" : null,
	"kontoinhaberLand" : null,
	"kontoinhaberBankname" : null,
	"kontoinhaberIban" : null,
	"kontoinhaberBic" : null,
}
```


Ausgabe: SepamandatErgebnis – Json-codiert im Body

Beispiel:


```
{
	"ergebnis" : {
		"rc" : "+0000",
		"ergebnistext" : "Die Aktion wurde fehlerfrei durchgeführt"
	},
	"date":"22.12.2016 15:06:20",
	"sepamandatErgebnisdaten" : {
```

```
                "xxtypetagxx" : "SepamandatErgebnisdaten",
                "type" : "B",
                "mandatreferenz" : "KLAUSIB9999992212201694",
                "kundennummer" : "TDG_ESHOP1",
                "glaeubigerId" : "DE98ABC09999999999",
                "zeichnungsberechtigter" : null,
                "kontoinhaberWeichtAb" : false,
                "kontoinhaberNachname" : "",
                "kontoinhaberVorname" : "",
                "kontoinhaberStrasse" : "",
                "kontoinhaberHausnummer" : "",
                "kontoinhaberPLZ" : "",
                "kontoinhaberStadt" : "",
                "kontoinhaberLand" : "",
                "kontoinhaberBankname" : "",
                "kontoinhaberIban" : "",
                "kontoinhaberBic" : "",
                "sequenceType" : "OOFF",
                "datumUnterschrift" : null,
                "ortUnterschrift" : null,
                "datumLetzteNutzung" : null,
                "active" : false,
                "sepamandatdatei" : "…“
}
```


5.5 SEPA-Mandat vervollständigen

Pfad	POST /epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirtschafternr}/sepa-mandate/{sepamandatsreferenz}/vervollstaendigen

Einmalmandate können, wie sonst auch üblich, bereits vor der Vervollständigung genutzt werden.

Eingabe: SepamandatInternVervollstaendigungsdaten – Json-codiert im Body

Beispiel:


```
{
        "xxtypetagxx" : "SepamandatInternVervollstaendigungsdaten",
        "datumUnterschrift" : "22.12.2016 15:26:22",
        "ortUnterschrift" : "Ort Unterschrift"
}
```

Ausgabe: SepamandatErgebnis – Json-codiert im Body

Beispiel:


```
{
	"ergebnis" : {
		"rc" : "+0000",
		"ergebnistext" : "Die Aktion wurde fehlerfrei durchgeführt"
	},
	"date":"22.12.2016 15:06:20",
	"sepamandatErgebnisdaten" : {
		"xxtypetagxx" : "SepamandatErgebnisdaten",
		"type" : "B",
		"mandatreferenz" : "KLAUSIB9999992212201694",
		"kundennummer" : "TDG_ESHOP1",
		"glaeubigerId" : "DE98ABC09999999999",
		"zeichnungsberechtigter" : null,
		"kontoinhaberWeichtAb" : false,
		"kontoinhaberNachname" : "",
		"kontoinhaberVorname" : "",
		"kontoinhaberStrasse" : "",
		"kontoinhaberHausnummer" : "",
		"kontoinhaberPLZ" : "",
		"kontoinhaberStadt" : "",
		"kontoinhaberLand" : "",
		"kontoinhaberBankname" : "",
		"kontoinhaberIban" : "",
		"kontoinhaberBic" : "",
		"sequenceType" : "OOFF",
		"datumUnterschrift" : "22.12.2016 00:00:00",
		"ortUnterschrift" : "Ort Unterschrift",
		"datumLetzteNutzung" : "22.12.2016 15:26:23",
		"active" : true,
		"sepamandatdatei" : "…“
}
```


5.6 Löschen Kunde

Pfad	DELETE /epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirtschafternr}/kunden/{kundennummer}

Eingabe: Keine weiteren Parameter im Body.

Ausgabe: ErgebnisErgebnis – Json-codiert im Body.

Beispiel:


```
{
        "ergebnis" : {
                "rc" : "+0000",
                "ergebnistext" : "Die Aktion wurde fehlerfrei durchgeführt"
        },
        "date" : "22.12.2016 15:06:20
}
```


5.7 Bezahlen

Pfad	GET /epayment/kunden/v1_0/rechnungen/<rechnungsid>/bezahlen

Eingabe: Keine weiteren Parameter im Body

Ausgabe: Der Kunde wird umgeleitet.

5.8 Kassenzeichen aktivieren

Pfad	POST /epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirtschafternr}/kassenzeichen/{kassenzeichennummer}

Eingabe: Keine weiteren Parameter im Body

Ausgabe: ZahlverfahrenListErgebnis – Json-codiert im Body.

Beispiel:


```
{
  "ergebnis":    {
    "rc": "+0000",
    "ergebnistext": "Die Aktion wurde fehlerfrei durchgeführt"
  },
  "date": "17.03.2020 14:30:11",
  "kassenzeichen":    {
    "kassenzeichennummer": "242030000002",
    "zusatzdaten":       {
      "bezahlverfahren": "UEBERWEISUNGNACH",
      "betragStorno": 0,
      "betragSonstiges": 0,
      "transaktionsnummer": null,
```

```
        "betragHauptforderungen": 10,
        "betragRLS": 0,
        "zahlungsvorgangsID": "",
        "saldo": 10,
        "aktivierungszeit": "17.03.2020 14:29:46",
        "betragZahlungseingang": 0,
        "betragLastschrift": 0,
        "paypagestatus": "INAKTIV"
    }
  }
}
```


Die Methode aktiviert nur Kassenzeichen, die mit den Zahlverfahren

- "UEBERWEISUNGVOR",
- "UEBERWEISUNGNACH",
- „BARZAHLUNG“,
- „TERMINALZAH-LUNG“

angelegt wurden. Ansonsten erscheint der Fehler
„-0507: Das Zahlverfahren dieser Buchungsliste ist ungültig oder wird nicht unterstützt.“

Ist das Kassenzeichen bereits aktiviert, erscheint die Fehlermeldung
„-0599: Das gewählte Kassenzeichen ist bereits aktiviert."

5.9 WADL des restapi-Moduls

Die WADL für das restapi-Modul kann aufgerufen werden über

`https://<server>/restapi/epayment/application.wadl"`


```
 <application>
<doc jersey:generatedBy="Jersey: 2.24 2016-10-27 14:35:27"/>
<doc jersey:hint="This is simplified WADL with user and core resources only.
                  To get full WADL with extended resources use the query parameter detail.
                  Link: http://epayment242:8090/restapi/epayment/application.wadl?detail=true"/>
<grammars/>
<resources base="http://epayment242:8090/restapi/epayment/">
<resource path="/kunden/v1_0/">
<resource path="/rechnungen/{rechnungsid}/bezahlen">
<param name="rechnungsid" style="template" type="xs:string"/>
<method id="bezahlenRechnung" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
```

```
</resource>
<resource path="/zvpreplys/v1_0/">
<resource path="success/sfp/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="successSaferpay" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/cancel/sfp/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="cancelSaferpay" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/failure/sfp/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="failureSaferpay" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/notify/sfp/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="notifySaferpayGet" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
<method id="notifySaferpayPost" name="POST">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/success/ppc/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="successPayplace" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/notifyfailed/ppc/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="notificationFailedPayplaceGet" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
```

```
<method id="notificationFailedPayplacePost" name="POST">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/failure/ppc/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="failurePayplace" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/cancel/gso/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="cancelGirosolution" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/notify/gso/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="notifyGirosolutionGet" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
<method id="notifyGirosolutionPost" name="POST">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/success/ppl/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="successPaypal" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/cancel/ppl/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="cancelPaypal" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/success/pdk/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="successPaydirekt" name="GET">
```

```
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/cancel/pdk/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="cancelPaydirekt" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/failure/pdk/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="rejectionPaydirekt" name="GET">
<response>
<representation mediaType="text/html"/>
</response>
</method>
</resource>
<resource path="/notify/pdk/mandanten/{mandantnummer}/kassenzeichen/{kassenzeichennummer}">
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<method id="notifyPaydirekt" name="POST">
<request>
<representation mediaType="application/hal+json"/>
<representation mediaType="text/json"/>
</request>
<response>
<representation mediaType="application/hal+json"/>
</response>
</method>
</resource>
</resource>
<resource path="/fachverfahren/v1_0/">
<resource path="/mandanten/{mandantnummer}/bewirtschafter/{bewirtschafternummer}/zahlverfahren">
<param name="mandantnummer" style="template" type="xs:string"/>
<param name="bewirtschafternummer" style="template" type="xs:string"/>
<method id="abfragenZahlverfahren" name="GET">
<response>
<representation mediaType="application/json;charset=utf-8"/>
</response>
</method>
</resource>
<resource path="/mandanten/{mandantnummer}/bewirtschafter/{bewirtschafternummer}/kunden/{kunden-
nummer}">
<param name="kundennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<param name="bewirtschafternummer" style="template" type="xs:string"/>
<method id="loeschenKunde" name="DELETE">
<response>
<representation mediaType="application/json;charset=utf-8"/>
</response>
</method>
</resource>
<resource path="/mandanten/{mandantnummer}/bewirtschafter/{bewirtschafternummer}/kassen-
zeichen/{kassenzeichennummer}">
```

```
<param name="kassenzeichennummer" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<param name="bewirtschafternummer" style="template" type="xs:string"/>
<method id="abfragenKassenzeichenstatus" name="GET">
<response>
<representation mediaType="application/json;charset=utf-8"/>
</response>
</method>
<method id="aktiviereKassenzeichen" name="POST">
<response>
<representation mediaType="application/json;charset=utf-8"/>
</response>
</method>
</resource>
<resource path="/mandanten/{mandantnummer}/bewirtschafter/{bewirtschafternummer}/buchungslisten">
<param name="mandantnummer" style="template" type="xs:string"/>
<param name="bewirtschafternummer" style="template" type="xs:string"/>
<method id="uebertragenBuchungsliste" name="POST">
<request>
<representation mediaType="application/json;charset=utf-8"/>
</request>
<response>
<representation mediaType="application/json;charset=utf-8"/>
</response>
</method>
</resource>
<resource path="/mandanten/{mandantnummer}/bewirtschafter/{bewirtschafternummer}/sepaman-
date/anlegen">
<param name="mandantnummer" style="template" type="xs:string"/>
<param name="bewirtschafternummer" style="template" type="xs:string"/>
<method id="anlegenSepaMandat" name="POST">
<request>
<representation mediaType="application/json;charset=utf-8"/>
</request>
<response>
<representation mediaType="application/json;charset=utf-8"/>
</response>
</method>
</resource>
<resource path="/mandanten/{mandantnummer}/bewirtschafter/{bewirtschafternummer}/sepaman-
date/{sepamandatsreferenz}/vervollstaendigen">
<param name="sepamandatsreferenz" style="template" type="xs:string"/>
<param name="mandantnummer" style="template" type="xs:string"/>
<param name="bewirtschafternummer" style="template" type="xs:string"/>
<method id="vervollstaendigenSepamandat" name="POST">
<request>
<representation mediaType="application/json;charset=utf-8"/>
</request>
<response>
<representation mediaType="application/json;charset=utf-8"/>
</response>
</method>
</resource>
</resource>
</resources>
</application>
```

5.10 Returncodes

Der Konnektor verwendet neben den in der ePayBL üblichen weitere Returncodes. Sie werden zum einem vom Server (S) und zum anderen von der ePayBL-KonnektorClient-Bibliothek (C) geliefert.

Returncode	Beschreibung	Returntext
"0000" - C	- Unbekannt, ob der Request erfolgreich war - HTTP-Response-Status: 1xx - (*): - Response-Entity (Body) kann NICHT als String gelesen werden ODER dessen Inhalt ist LEER ODER es handelt sich NICHT um application/json - ODER der Json-Inhalt ist leer - ODER Json kann nicht auf den gewünschten Zieltyp deserialisiert werden.	- "Ausgang unbekannt“ + - (**): - HTTP-Status “:” HTTP-Reason - Eine gegebenenfalls bei der Json-Deserialisierung aufgetretene Exception mit Message und Stacktrace - Entity (Body) als String, insofern als String lesbar und nicht leer.
"+0010" - C	- HTTP-Response-Status: 2xx - S.o. (*)	- "Ok, jedoch unerwartete Antwort“ + - S.o. (**)
"-9999" - C	- HTTP-Response-Status: 5xx - S.o. (*)	- S.o. (**)
"-9999" - S	- Unbestimmter Server-Fehler.	- "Interner nicht dokumentierter Fehler im ePayment-System."
"-9999" - S	- In einem **Jersey-WebApplicationException-Handler**. Hier wird eine automatische Framework-Response in eine eigene gewandelt. - HTTP-Response-Status: 5xx	- HTTP-Reason “:” HTTP-Status.
"-9001" - C	- HTTP-Response-Status: 4xx, AUSSER: 401, 403, 404, 405,406, 415 – für die spezielle Codes existieren (s.u.)	- S.o. (**)

	-	S.o. (*)		
"-9001" - S	-	In einem **Jersey-WebApplicationException-Handler**. Hier wird eine automatische Framework-Response in eine eigene gewandelt. - HTTP-Response-Status: 4xx, AUSSER: 401, 403, 404, 405,406, 415 – für die spezielle Codes existieren (s.u.)	-	HTTP-Reason ":" HTTP-Status.
"-9002" – C,S	-	HTTP-Response-Status: 403 (Forbidden) - S.o. "-9001"	-	S.o. "-9001"
"-9003" – C,S	-	HTTP-Response-Status: 406 (Not Acceptable) - S.o. "-9001"	-	S.o. "-9001"
"-9004" – C,S	-	HTTP-Response-Status: 405 (Method Not Allowed) - S.o. "-9001"	-	S.o. "-9001"
"-9005" – C,S	-	HTTP-Response-Status: 401 (Unauthorized) - S.o. "-9001"	-	S.o. "-9001"
"-9006" – C,S	-	HTTP-Response-Status: 404 (Not Found) - S.o. "-9001"	-	S.o. "-9001"
"-9007" – C,S	-	HTTP-Response-Status: 415 (Unsupported Media Type) - S.o. "-9001"	-	S.o. "-9001"
"-9000" - S	-	Request-Validierungs-Fehler - Unter annotations-getriebener Validierung und **ConstraintViolationException**	-	Die einzelnen **Violations** werden **zeilenweise** aufgeführt. Die bislang verwendeten Texte sind unter Ersetzung der Platzhalter mit Werten aus der Constraint-Annotation bzw. des voll-qualifizierten Property-Namens: - Für javax.validation.constraints.NotNull.message=**Der Wert für {%s} darf nicht leer sein.** - Für javax.validation.constraints.Size.message=**Die Größe von {%s} muss zwischen {min} und {max} liegen.**

"-9000" - S	-	Request-Validierungs-Fehler - **JsonProcessingException**	-	Je nach konkreter Exception und sonstiger Bedingungen gibt es folgende **FIXE** Texte unter Ersetzung der Platzhalter: (Dabei ist der „originale Fehler“ der die JsonProcessingException auslösende Fehler - JsonProcessingException.getOriginalMessage().) - InvalidFormatException: **"Das Format für {%s} ist ungültig. Json: Zeile <%d>, Spalte <%d>."** - UnrecognizedPropertyException: **"Die Property {%s} ist unbekannt. Json: Zeile <%d>, Spalte <%d>."** - Voll-qualifizierter Property-Name nicht verfügbar: **"Datenfehler. Json: Zeile <%d>, Spalte <%d>. Json: Originaler Fehler: <%s>."** - Sonst: **"Fehler bei Property {%s}. Json: Zeile <%d>, Spalte <%d>. Json: Originaler Fehler: <%s>."**
"+0009" - S	-	Sammlung verschiedener OK-Ergebnisse mit Einschränkungen, d.h. Code > "+0000". - Wenigstens 2 solcher Ergebnisse. - Bislang verwendet im Kontext ZahlvorgangsErgebnis - Da diese Methode aus mehreren Schritten besteht (Anlegen Kunde, Anlegen Kassenzeichen, Initiieren der Zahlung am ZV-Provider) können einige Schritte erfolgreich durchgeführt worden sein, aber eventuell mit Einschränkungen (z.B. beim Anlegen des Kunden wurde festgestellt, dass er schon existiert). Diese Schritte liefern dann ein Returncode größer als Null. - Aktuell sind Einschränkungen möglich beim Kunden-Handling (KundenErgebnisImpl) sowie beim Buchungslisten-Handling (BuchungsListeErgebnis).	-	Die Texte werden mit je einer Zeile pro Ergebnis concateniert.
"-0180" - S	-	Für SEPA darf der Kunde nicht temporär sein.	-	"Für einen temporären Kunden ist das Zahlverfahren SEPA nicht erlaubt."
"+0500" - S	-	Das Übertragen einer Buchungsliste wurde nur teilweise durchgeführt:	-	"Die Buchungsliste wurde nicht verarbeitet. Es wurde nur das noch zu vervollständigende SEPA-Mandat angelegt."

	-	Es wurde nur ein SEPA-mandat angelegt, das noch zu vervollständigen ist.		
"-0550" - S	-	Übertragen Buchungsliste: Fachverfahrensdaten sind unvollständig oder ungültig.	-	"Fachverfahrensdaten der Buchungsliste fehlen oder sind ungültig."
"-0551" - S	-	Übertragen Buchungsliste: Eine Mandatspezifikation über Referenz bezog sich auf ein EXTERNES Mandat.	-	"Ein externes SEPA-Mandat muss komplett und nicht nur als Referenz geliefert werden."
"-0552" - S	-	Übertragen Buchungsliste: Die SEPA-Mandatsreferenz matcht nicht mit der am Kunden hinterlegten.	-	"Die SEPA-Mandatreferenz muss mit der am Kunden übereinstimmen."
"-0553" - S	-	Übertragen Buchungsliste: Es fehlen SEPA-Mandatsdaten	-	"Es fehlen SEPA-Mandats-Daten."
"-9100" - S	-	Übertragen Buchungsliste und Bezahlen auf Rechnung: Fehler beim Speichern des ZahlvorgangsIds.	-	"Fehler beim Speichern der ZahlungsvorgangsId."

Tabelle 26: Zusätzliche Returncodes

6 Konfiguration

6.1 Zertifikatshandling

Kommuniziert eine Webapplikation mit einer anderen Webapplikation auf einem anderen Server, so sind im Allgemeinen Zertifikate nötig, damit der Request auf den anderen Server gelangt. Diese **Zertifikate werden durch die Firewall des Zielservers geprüft**. Das gilt auch für die Kommunikation mit der ePayBL.

Zusätzlich kann hier am Mandanten der ePayBL konfiguriert werden, ob bei der Kommunikation der Fachanwendung mit der ePayBL zusätzlich Client-Zertifikate geprüft werden. Werden Client-Zertifikate geprüft, dann bedeutet das, dass die ePayBL bei jedem Request kontrolliert, ob der DN des mitgelieferten Zertifikates mit den am Mandanten hinterlegten DN übereinstimmt. Damit wird sichergestellt, dass die Fachanwendung nicht mit dem falschen Mandanten kommuniziert. Damit ist sichergestellt, dass die Fachanwendung zum einen mit dem richtigen Server kommuniziert und zum anderen mit dem richtigen Mandanten.

Unter der Voraussetzung, dass der Konnektor auf einem separaten Server läuft, sind im Allgemeinen **Zertifikate sowohl für die Kommunikation zwischen Fachanwendung und Konnektor als auch zwischen Konnektor und ePayBL nötig**.

Um nun auch über den Konnektor eine Client-Zertifikatsprüfung zu realisieren, wird der Konnektor bei allen Requests, die er von der Fachanwendung bekommt, prüfen, ob ein Zertifikat mitgeliefert wurde. Bei der Weiterleitung des Request an die ePayBL wird dann dieses mitgelieferte Zertifikat als separates Objekt mitgeschickt.

Der Konnektor identifiziert sich selbst gegenüber dem Server der ePayBL mit einem eigenen festen Zertifikat.

Ist an der ePayBL ein Clientzertifikat erforderlich, dann prüfen die Methoden des restapi-Moduls der ePayBL, ob die Requests ein Zertifikat als separates Objekt mitliefern. Von diesem Zertifikat wird der DN bestimmt und abgeglichen mit den am Mandanten hinterlegten DN-Strings. Dadurch ist wieder sichergestellt, dass die Fachanwendung mit dem richtigen Mandanten kommuniziert.

Es kommt zu Fehlern, wenn im Request kein Zertifikat gefunden wird, obwohl ein Client-Zertifikat erforderlich ist, oder wenn der DN des gefundenen Zertifikates nicht korrekt ist.

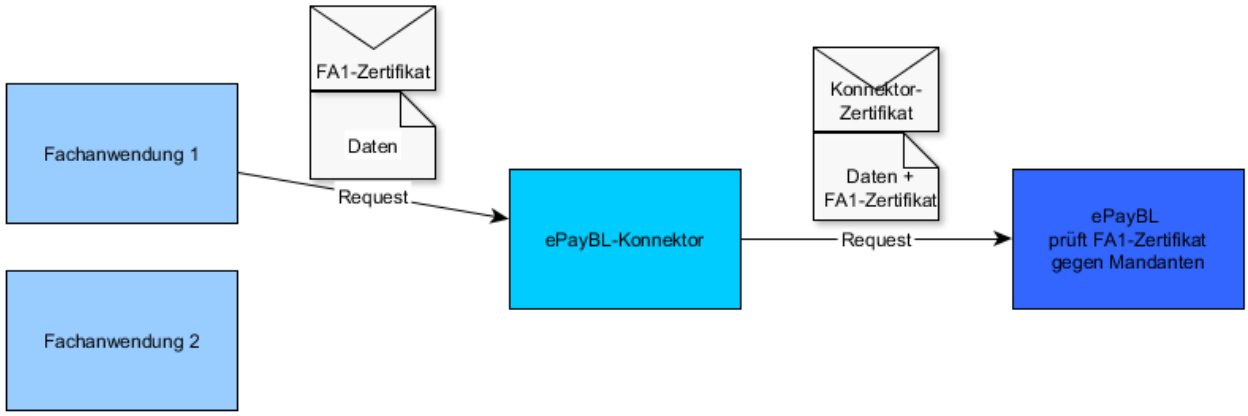


Abbildung 17: Zertifikatsweiterleitung zur ePayBL

6.2 Konfiguration des ePayBL-Konnektors

Der Konnektor wird die Requests, die er vom Fachverfahren, der ePayBL oder dem Kundenbrowser bekommt, prinzipiell 1:1 weiterleiten. Die eigentliche Validierung und Verarbeitung der Requests des Fachverfahrens erfolgt in der ePayBL. Daher benötigt der Konnektor keinerlei umfangreiche Konfiguration. Die Parameter, die der **Konnektor** benötigt sind:

- die **URL der ePayBL**,
- Verzeichnispfad zum **Zertifikats-Keystore** des eigenen Zertifikats. (Als **Truststore** wird implizit der des Application Servers verwendet.)
- ob der Konnektor für die Kommunikation mit der ePayBL ein **Zertifikat benötigt**
- die **Zugangsdaten zum Zertifikats-Keystore** für die Kommunikation mit ePayBL und Fachanwendung.

Diese Parameter werden dem Konnektor bei der Installation über Property-Parameter mitgegeben.

6.3 Konfiguration an der ePayBL für den Konnektor

Über das Portal der ePayBL kann die Kommunikation der ePayBL mit dem Konnektor konfiguriert werden. Wir gehen davon aus, dass die ePayBL nur mit höchstens einem Konnektor kommuniziert.

Am Mandanten kann konfiguriert werden, ob dieser mit der Fachanwendung über den Konnektor kommuniziert oder nicht. (Dieses ist für die Erzeugung der ZV-Provider-Backlinks erforderlich.) Da kein Bedarf bestand, dass die ePayBL bzw. der Konnektor das Fachverfahren aktiv über den Status des Kassenzeichens informieren (das Fachverfahren hätte dafür einen Service implementieren müssen) muss die ePayBL die URL des Konnektors nicht wissen.

7 Bibliotheken zum Anbinden von FV

Die vom Fachverfahren aufzurufenden Methoden sowie Methoden zur Generierung der Parameter-Objekte für diese Methoden werden in einer Bibliothek gekapselt. Diese Bibliothek wird in Java, .NET und php bereitgestellt. Es wird eine Dokumentation für diese Bibliothek geliefert.

7.1 Konnektor-Client-Bibliothek (Java)

Die Bibliothek bietet verschiedene Methoden zum Aufruf der REST-Schnittstelle an. Die Konnektor-Client-Bibliothek stellt die folgenden Methoden bereit:

Methode	Parameter	Return-Typ
abfragenZahlverfahren		ZahlverfahrenListErgebnis
abfragenKassenzeichenstatus	String kassenzeichennummer	KassenzeichenstatusErgebnis
aktivierenKassenzeichen	String kassenzeichennummer	KassenzeichenstatusErgebnis
uebertragenBuchungsliste	Buchungsliste buchungsliste	ZahlvorgangsErgebnis
anlegenSepaMandat	SepaMandatInternAnlegedaten sepaMandatInternAnlegedaten	SepamandatErgebnis
vervollstaendigenSepamandat	String sepamandatsreferenz	SepamandatErgebnis
sepamandatInternVervollstaendigungsdaten	String sepamandatsreferenz, SepamandatInternVervollstaendigungsdaten sepamandatInternVervollstaendigungsdaten	SepamandatInternVervollstaendigungsdaten
loeschenKunde	String kundennummer	ErgebnisErgebnis

Die Beschreibung der API liegt als JavaDoc-Dokument vor.

Bei programmatischer Nutzung wird empfohlen, mit je einer selbst erzeugten Client-Konstante (static) pro Konfiguration zu arbeiten. Sollen diverse Mandanten und Bewirtschafter bearbeitet werden, so kann alternativ zur Verwendung zahlreicher Client-Instanzen auch

mit dynamischer Erzeugung von Clients – jeweils spezifisch für eine Mandant-Bewirtschafter-Kombination – gearbeitet werden; diese werden aus einem Parent-Client heraus erzeugt, wodurch teure Ressourcen geteilt werden können.

Vom Nutzer ist wenigstens die Endpoint-URI zum Server, auf dem der Konnektor läuft, anzugeben. Der Client setzt intern den nötigen Pfad für die korrekte Kommunikation zusammen. Es ist zu beachten, dass die Client-Bibliothek stets mit dem Konnektor (Webapplikation „konnektor“) und nicht mit der ePayBL direkt (restapi-Modul) kommuniziert. Man muss also auch den Konnektor installiert haben.

Exceptions, die während der Request-Stellung auftreten werden gekapselt weitergereicht. Liegt eine Response vor, so wird auf jeden Fall ein Ergebnis-Objekt des entsprechenden Typs geliefert, unabhängig davon, ob bei der Verarbeitung der Response Exceptions auftreten.

Der Nutzer hat den Ergebniscode zu testen. Eine Beschreibung der möglichen Codes befindet sich in der FSpec des ePayBL Konnektors.

Will das Fachverfahren an den Konnektor ein Zertifikat mitgeben, kann hierfür die Klasse **`EpayBLConnectorClientConfiguration`** aus der Bibliothek verwendet werden**.**

Der folgende Code zeigt eine Beispiel-Applikation, die die Client-Bibliothek verwendet.

```
package de.eg.epaybl.restclient.client;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import de.eg.epaybl.restclient.beans.Buchung;
import de.eg.epaybl.restclient.beans.Buchungsliste;
import de.eg.epaybl.restclient.beans.Kunde;
import de.eg.epaybl.restclient.beans.Zahlverfahren;
import de.eg.epaybl.restclient.beans.ZahlvorgangsInfo;
import de.eg.epaybl.restclient.beans.enums.KundeTyp;
import de.eg.epaybl.restclient.beans.enums.Zahltyp;
import de.eg.epaybl.restclient.beans.ergebnisse.KassenzeichenstatusErgebnis;
import de.eg.epaybl.restclient.beans.ergebnisse.ZahlverfahrenListErgebnis;
import de.eg.epaybl.restclient.beans.ergebnisse.ZahlvorgangsErgebnis;
import de.eg.epaybl.restclient.beans.konstanten.BuchungKontierungKeys;
import de.eg.epaybl.restclient.beans.konstanten.BuchungslisteFvDatenKeys;
import de.eg.epaybl.restclient.beans.konstanten.ZahlverfahrenCodes;
import de.eg.epaybl.restclient.client.EpayBLConnectorClientConfiguration;
import de.eg.epaybl.restclient.client.EpayBLKonnektorClient;
import de.eg.epaybl.restclient.client.EpayBLKonnektorClientImpl;

/**
 * kleine Beispielimplementierung für die Einbindung der Clientbibliothek
 * ePayBL-KonnektorClient-1.0.0.jar
 */
public class BeispielImpl {

    public static Buchungsliste fuelleBuchungslistenDaten() {
        final Buchungsliste bListe = new Buchungsliste();
        bListe.setFaelligkeitsdatum(new Date(new Date().getTime() + 200000));
        bListe.setKennzeichenMahnverfahren("23");
        bListe.setZahltyp(Zahltyp.DIREKT);
        bListe.setBetrag(new BigDecimal("10"));
        bListe.setBeschreibung("Buchungsliste mit Buchungen: 1");

        final Kunde kunde = new Kunde();
        kunde.setKundennummer("k242010");
        kunde.setTyp(KundeTyp.BESTAND);
        bListe.setKunde(kunde);

        final List<String> zahlverfahrenCodes = new ArrayList<>();
        zahlverfahrenCodes.add(ZahlverfahrenCodes.UEBERWEISUNGVOR);
        bListe.setZahlverfahrencodes(zahlverfahrenCodes);

        final Map<String, String> fachverfahrendaten = new HashMap<>();
        fachverfahrendaten.put(BuchungslisteFvDatenKeys.SUCCESSURL,
             "http://www.heise.de");
```

```
        fachverfahrendaten.put(BuchungslisteFvDatenKeys.CANCELURL,
             "http://www.dresearch.de");
        fachverfahrendaten.put(BuchungslisteFvDatenKeys.ERRORURL,
             "http://www.amazon.de");
        bListe.setFachverfahrendaten(fachverfahrendaten);
        bListe.setBuchungen(fuelleBuchungsDaten());
        return bListe;
    }

    private static List<Buchung> fuelleBuchungsDaten() {
        final List<Buchung> buchungen = new ArrayList<>();

        final Buchung buchung = new Buchung();
        buchung.setBruttobetrag(new BigDecimal("10"));
        buchung.setNettobetrag(new BigDecimal("9"));
        buchung.setSteuerbetrag(new BigDecimal("1"));
        buchung.setBuchungstext("Buchung 1");

        final Map<String, String> kontierung = new HashMap<>();
        kontierung.put(BuchungKontierungKeys.HAUSHALTSTELLE, "4421030");
        kontierung.put(BuchungKontierungKeys.HREF, "HREF");
        kontierung.put(BuchungKontierungKeys.OBJEKTNUMMER, "129001");
        buchung.setKontierung(kontierung);

        buchungen.add(buchung);
        return buchungen;
    }

    public static void main(final String[] args) {
        final String url = "http://localhost:8080/"; // url for REST mock-service

        final String mandant = "RE00001"; // Mandant Nummer
        final String bewirtschafter = "ReB000"; // Bewirtschafter Nummer

        final String RC_OK = "+0000"; // OK Return Code

        String kassenzeichen = null;
        final EpayBLKonnektorClient restClient = new EpayBLKonnektorClientImpl(
                  new EpayBLConnectorClientConfiguration(url, mandant,
bewirtschafter));
        //restClient.setKeystorePassword(keystorePassword);
        //restClient.setKeystoreFilename(keystoreFilename);
        try {
            final ZahlverfahrenListErgebnis zErgebnis = restClient.abfragenZahlverfah-
ren();
            if (zErgebnis.getErgebnis().getRc().equals(RC_OK)) {
                final List<Zahlverfahren> zvList = zErgebnis.getZahlverfahren();
                for (final Zahlverfahren zv : zvList) {
                    System.out.println(zv.toString());
                }
            } else {
                System.out.println("Fehler: " + zErgebnis.getErgebnis().getErgeb-
nistext());
```

```
            }
            final ZahlvorgangsErgebnis bErgebnis = restClient.uebertragenBuchungs-
liste(fuelleBuchungslistenDaten());
            if (bErgebnis.getErgebnis().getRc().equals(RC_OK)) {
                final ZahlvorgangsInfo zVorgangsInfo = bErgebnis.getZahlvorgangsInfo();
                System.out.println(zVorgangsInfo.toString());
                kassenzeichen = zVorgangsInfo.getKassenzeichen().getKassenzeichennum-
mer();
                System.out.println("Buchungsliste erfolgreich uebertragen - Kassenzei-
chen = " + kassenzeichen);
            } else {
                System.out.println("Fehler: " + bErgebnis.getErgebnis().getErgeb-
nistext());
            }
            if (kassenzeichen != null) {
                final KassenzeichenstatusErgebnis kErgebnis = restClient.abfragenKas-
senzeichenstatus(kassenzeichen);
                if (kErgebnis.getErgebnis().getRc().equals(RC_OK)) {
                    System.out.println(kErgebnis.getKassenzeichen().toString());
                } else {
                    System.out.println("Fehler: " + kErgebnis.getErgebnis().getErgeb-
nistext());
                }
            }
        } catch (final Exception pe) {
            System.out.println("Exception: " + pe.getMessage());
        }
    }
}
```

Abbildung 18: TestImpl.java

Die Bibliothek benötigt die folgenden zusätzlichen Bibliotheken:

Bibliothek	Version
commons-logging-api	1.1
hk2-api	2.6.1
hk2-locator	2.6.1
hk2-utils	2.6.1
httpclient	4.5.12
httpcore	4.4.13
jackson-annotations	2.10.0
jackson-core	2.10.2
jackson-databind	2.10.2
jackson-jaxrs-base	2.10.2
jackson-jaxrs-json-provider	2.10.2
javax.annotation-api	1.3.2
javax.inject	2.5.0-b05
javax.ws.rs-api	2.1.1
jersey-apache-connector	2.30.1
jersey-client	2.30.1
jersey-common	2.30.1
jersey-entity-filtering	2.30.1
jersey-guava	2.25.1
jersey-media-jaxb	2.30.1
jersey-media-json-jackson	2.30.1

Tabelle 27: Die Bibliothek nutzt weitere externe Bibliotheken

Die Bibliothek bietet die Möglichkeit Zertifikate für die Requests mitzugeben. Dazu gibt es die Methoden

- `setKeystorePassword();`
- `setKeystoreFilename();`
- `setTruststorePassword();`
- `setTruststoreFilename();`

am EpayBLConnectorClientConfiguration –Objekt.

Der Keystore enthält das in der ePayBL am Mandanten hinterlegte Zertifikat. Der Keystore wird in der Regel der über die ePayBL-Webapplikation “paycert” heruntergeladene Keystore sein. Der Truststore wird benötigt, wenn der Client (die Fachanwendung) eine Server-Authentifizierung machen will, indem sie das vom Server gesendete Zertifikat gegen die im Truststore hinterlegten prüft.

Läuft das Fachverfahren im Tomcat kann es notwendig sein, den Keystore auch in der Tomcat-Konfiguration anzugeben. Dazu wird Tomcat mit den Options

- `-Djavax.net.ssl.keyStore=<Keystore>`
- `-Djavax.net.ssl.keyStorePassword=<Passwort>`

gestartet.

Es können auch Proxies konfiguriert werden:

- `void setProxyHost(java.lang.String proxyHost)`
- `void setProxyPassword(java.lang.String proxyPassword)`
- `void setProxyPort(int proxyPort)`
- `void setProxyUsername(java.lang.String proxyUsername)`
- `void setUseProxy(boolean useProxy) Default: false`
- `boolean useProxyCredentials()`
- `EpayBLConnectorClientConfiguration withUseWithProxy(java.lang.String proxyHost, int proxyPort)`
- `EpayBLConnectorClientConfiguration withUseWithProxyHost(java.lang.String proxyHost)`

Die Bibliothek kann in einer Javav8 oder auch in einer Java 11 Umgebung verwendet werden. In einer Java 11 Umgebung ist darauf zu achten, dass die Bibliotheken

- javax.activation-api.jar und
- jaxb-api.jar

im Classpath ergänzt werden.

Die Client-Bibliothek wird als Zip-File:

ePayBL-KonnektorClientLib release-<Version>.zip

ausgeliefert.

Das Zip-File enthält:

1. **abstract-ePayBL-KonnektorClientLib-withoutExtJars-<Version>.jar**: enthält die Implementierung der REST-Methoden, der Objekte für die REST-Methoden (Kunde, Buchungsliste, Ergebnis, …) und eine abstrakte Klasse für die Einbindung eine REST-Bibliothek zur Erzeugung von REST-Requests und zum Parsen der JSON-Objekte (z.B. Jackson und Jersey). Diese Jar-File enthält keine externen Bibliotheken.
2. **abstract-ePayBL-KonnektorClientLib-withExtJars-<Version>.jar**: enthält die Implementierung der REST-Methoden, der Objekte für die REST-Methoden (Kunde, Buchungsliste, Ergebnis, …) und eine abstrakte Klasse für die Einbindung eine REST-Bibliothek zur Erzeugung von REST-Requests und zum Parsen der JSON-Objekte (z.B. Jackson und Jersey). Diese Jar-File bringt die benötigten externen Bibliotheken (javax.ws-rd-api.jar, Version 2.1.1) selber mit.
3. **abstract-ePayBL-KonnektorClientLib-javadoc-<Version>.zip**: JavaDoc-Dokumentation für den Basisteil der Bibliothek
4. **ePayBL-KonnektorClientLib-withoutExtJars-<Version>.jar**: Ergänzung des Basisteils der Bibliothek um eine Einbindung konkreter REST-Bibliotheken (Jackson und Jersey). Diese Jar-File enthält aber keine externen Bibliotheken. Diese müssen hierbei separat dazugeladen werden.
5. **ePayBL-KonnektorClientLib-withExtJars-<Version>.jar**: Ergänzung des Basisteils der Bibliothek um eine Einbindung konkreter REST-Bibliotheken (Jackson

und Jersey). Diese Jar-File bringt die benötigten externen Bibliotheken (insbesondere jackson und jersey) selber mit. Dieses Jar-File kann als Standalone-Bibliothek verwendet werden.

6. **ePayBL-KonnektorClientLib-javadoc-<Version>.zip**: JavaDoc-Dokumentation für den Ergänzungsteil der Bibliothek.

Der Ergänzungsteil besteht aus den folgenden Klassen:

1. de.eg.epaybl.restclient.client.EpayBLKonnektorClientImpl: Diese Klasse implementiert die abstrakten Methoden zur Anbindung von Jackson und Jersey, sowie einen Konstruktor:

```
package de.eg.epaybl.restclient.client;

import org.glassfish.jersey.apache.connector.ApacheConnectorProvider;
import org.glassfish.jersey.client.ClientConfig;
import org.glassfish.jersey.client.ClientProperties;
import com.fasterxml.jackson.databind.ObjectMapper;
import de.eg.epaybl.restclient.jsonserialization.JacksonObjectMapperPro-
vider;

/**
 * A THREADSAFE client implementation with configuration options.
 */
public class EpayBLKonnektorClientImpl extends AbstractEpayBL-
KonnektorClient {

    public EpayBLKonnektorClientImpl(
            EpayBLConnectorClientConfiguration config) {
        super(config);
    }

    @Override
    protected  Class<?> getJsonComponent() {
        return JacksonObjectMapperProvider.class;
    }

    @Override
    protected <R> R readJson(String maybeJson, Class<R> clazz)
            throws Exception {
        ObjectMapper om = new JacksonObjectMapperProvider(getConfig().
            getPropertyTypeMappingExtension()).getContext(null);
        return om.readValue(maybeJson, clazz);
    }
```

```
    @Override
    protected ClientConfig clientConfigBase() {
        return new ClientConfig()
                        .connectorProvider(new ApacheConnectorProvider())
                        .property(ClientProperties.CONNECT_TIMEOUT,
                    Intger.valueOf(getConfig().getConnectTimeoutMsec()))
                           .property(ClientProperties.READ_TIMEOUT,
                       Integer.valueOf(getConfig().getReadTimeoutMsec()));
    }

    @Override
    protected ClientConfig  clientConfigWithProxy() {
        ClientConfig clientConfig = clientConfigBase()
          .property(ClientProperties.PROXY_URI, getConfig().getProxy());
        if (getConfig().useProxyCredentials()) {
            clientConfig = clientConfig
                                .property(ClientProperties.PROXY_USERNAME,
                                          getConfig().getProxyUsername())
                                .property(ClientProperties.PROXY_PASSWORD,
                                         getConfig().getProxyPassword());
        }
        return clientConfig;
    }


}
```

2. de.eg.epaybl.restclient.jsonserialization.JacksonDeserUtil: Klasse zur Deserialisierung eines Json-Strings

```
package de.eg.epaybl.restclient.jsonserialization;

import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.core.JsonStreamContext;

class JacksonDeserUtil {

    private JacksonDeserUtil() {
    }

    public static String currentPath(final JsonParser jp) {
        JsonStreamContext ctxt = jp.getParsingContext();
        JsonStreamContext parent;
        String path = ctxt.getCurrentName();
```

```
        String name;

        while (((parent = ctxt.getParent()) != null) &&
                    ((name = parent.getCurrentName()) != null)) {
            path = name + "." + path;
            ctxt = parent;
        }

        return path;
    }
}
```

3. de.eg.epaybl.restclient.jsonserialization.JacksonObjectMapperProvider:

```
package de.eg.epaybl.restclient.jsonserialization;

import java.text.SimpleDateFormat;
import java.util.Map;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.ext.ContextResolver;
import javax.ws.rs.ext.Provider;

import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.module.SimpleModule;

import de.eg.epaybl.restclient.client.Resources;

@Provider
@Produces(MediaType.APPLICATION_JSON)
public class JacksonObjectMapperProvider implements ContextResolver<Ob-
jectMapper> {

    private static final String DATEFORMAT = Resources.DATEFORMAT_WIRE;

    private ObjectMapper customObjectMapper = null;

    public JacksonObjectMapperProvider() {
        this(null);
    }

    public JacksonObjectMapperProvider(
            final Map<String, Class<?>> propertyTypeMappingExtension) {
        customObjectMapper =
```

```
                createCustomObjectMapper(propertyTypeMappingExtension);
    }

    @Override
    public ObjectMapper getContext(final Class<?> arg0) {
        return customObjectMapper;
    }

    private ObjectMapper createCustomObjectMapper(
             final Map<String, Class<?>> propertyTypeMappingExtension) {
        return new ObjectMapper()
               .setDateFormat(new SimpleDateFormat(DATEFORMAT))
               .enable(DeserializationFeature.USE_BIG_DECMAL_FOR_FLOATS)
               .registerModule(new SimpleModule()
                        .addDeserializer(Object.class,
                   new ObjectDeserializer(propertyTypeMappingExtension)))
               .configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPER-
TIES, false)
               .configure(DeserializationFeature.FAIL_ON_MISSING_CREA-
TOR_PROPERTIES, false);

    }
}
```

4. de.eg.epaybl.restclient.jsonserialization. ObjectDeserializer:

```
package de.eg.epaybl.restclient.jsonserialization;

import java.io.IOException;
import java.util.Map;
import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.core.ObjectCodec;
import com.fasterxml.jackson.databind.DeserializationContext;
import com.fasterxml.jackson.databind.deser.std.StdDeserializer;

import de.eg.epaybl.restclient.beans.generic.PropertyTypeMapper;

/**
 * Resolves target types with the help of {@link PropertyTypeMapper}.
 */
class ObjectDeserializer extends StdDeserializer<Object> {

    private static final long serialVersionUID = -1498953585523550486L;

    private Map<String, Class<?>> propertyTypeMappingExtension;
```

```
    public ObjectDeserializer() {
        super(Object.class);
    }

    public ObjectDeserializer(
              final Map<String, Class<?>> propertyTypeMappingExtension) {
        super(Object.class);
        this.propertyTypeMappingExtension =
                                              propertyTypeMappingExtension;
    }

    @Override
    public Object deserialize(
                 final JsonParser jp, final DeserializationContext ctxt)
                            throws IOException, JsonProcessingException {
        Object[] singleResult = new Object[1];

        if (tryPathMapping(jp, singleResult)) {
            return singleResult[0];
        }

        return jp.getCodec().readTree(jp);
    }

    private boolean tryPathMapping(final JsonParser jp,
                        final Object[] singleResult) throws IOException {
        final ObjectCodec codec = jp.getCodec();

        final String path = JacksonDeserUtil.currentPath(jp);
        final Class<?> clazz = PropertyTypeMapper
  .pathToClassWithMappingExtension(path, propertyTypeMappingExtension);

        if (clazz != null) {
            singleResult[0] = codec.readValue(jp, clazz);
            return true;
        }

        return false;
    }
}
```

7.2 Konnektor-Client-Bibliothek (PHP)

Die PHP-Client-Bibliothek wird in Form eines Phar-Files („ePayBLKonnektorClientLib.phar“) bereitgestellt. Diese Bibliothek bindet ihrerseits eine Fremdbibliothek ‚httpful.phar’ ein, die ebenfalls mit ausgeliefert wird, sowie eine weitere Bibliothek „ePayBLKonnektorClientLibObjects.phar“, die Definitionen von Objekten und Konstanten enthält.

Die Bibliothek stellt zu nächste eine Klasse „EpayBLKonnektorClient“ die Konfiguration für die REST-Kommunikation, sowie die Methoden der REST-Schnittstelle selbst bereitstellt.

Die PHP-Client-Bbibliothek „ePayBLKonnektorClientLib.phar“ enthält die folgenden Methoden:

new EpayBL-KonnektorCli-ent()	Konstruktor der Klasse `EpayBLKonnektorClient`		
	Parame-ter	`$serverAddress`	Adresse der Servers (String)
		`$mandant`	Mandantnummer (String)
		`$bewirtschafter`	Bewirtschafternummer (String)
		`$useProxy`	Ob ein Proxy verwendet werden soll; optional (default false) (true / false)
		`$proxyConfig`	Array mit der Proxy-Konfiguration. Keys: • „proxy_host": Hostname • „proxy_port": Port • „auth_type": "CURLAUTH_BASIC" für Proxy-Authentifizierung • „auth_username": Proxy-Username • „auth_password": Proxy-Passwort • „proxy_type": Prxy-Typ: "HTTP" / "SOCKS4" / "SOCKS5"
		`$application`	Name der Applikation (String); optional, default „konnektor“, alternativ „restapi“ möglich
	Return	Klasse `EpayBLKonnektorClient` Serveradresse, Mandant, und Bewirtschafter sind in der Klasse gespeichert	
getResource	Gibt die Liste der möglichen REST-Ressourcen und deren HTTP-Methoden (GET, POST, DELTE) zurück, bzw., falls ein Key angegeben wurde, die Ressource zu diesem Key		

	Parameter	`$key`	Optional, mögliche Keys sind `"zahlverfahren", "buchungslisten", "kassenzeichen", "sepamandatAnlegen", "sepamandatVervollstaendigen", "kundeLoeschen"`
	Return	Die komplette Liste oder die zum Key gehörige Ressource	
getMethod	Gibt die zu einer REST-Ressourcen gehörige HTTP-Methode (GET, POST, DELTE) zurück, die Ressource wird über einen Key ausgewählt		
	Parameter	`$key`	Optional, mögliche Keys sind `"zahlverfahren", "buchungslisten", "kassenzeichen", "sepamandatAnlegen", "sepamandatVervollstaendigen", "kundeLoeschen"`
	Return	die zum Key gehörige http-Methode	
setMandantAndBewirtschafter	Setzt an der Klasse `EpayBLKonnektorClient` den Mandanten und den Bewirtschafter		
	Parameter	`$mandant`	Mandantnummer (String)
		`$bewirtschafter`	Bewirtschafternummer (String)
setSSL	Setzt SSL Parameter für die Kommunikation über https und für die Client-Authentifizierung bei der ePayBL. Der Keystore wird im PEM-Format übergeben. Dabei werden die Keys und die Zertifikate getrennt übergeben. Man kann die benötigtes Dateien aus den von der ePayBL-erzeugten Keystores konvertieren durch: • `openssl pkcs12 -in zertifikat_from_epaybl-ZV.p12 -out keystore.pem –nocerts` • `openssl pkcs12 -in zertifikat_from_epaybl-ZV.p12 -out clientstore.pem -clcerts –nokeys` Beim Konvertieren ist darauf zu achten, dass die PEM-Dateien beide das gleiche Passwort erhalten. Dieses Passwort ist in dieser Methode anzugeben. Die Methode kann wie folgt benutzt werden:		

<table>
<tr><td></td><td colspan="3">

```
…
$epayBLKonnektorClient =
      new EpayBLKonnektorClient( $serverAddress,
          $mandant, $bewirtschafter, $application);
if (/* use SSL*/) {
     $epayBLKonnektorClient->setSSL( $key, $crt,
          $keypassphrase, true);
}
$zahlverfahrenListeErgebnis =
        $epayBLKonnektorClient->abfragenZahlverfahren();
…
```

</td></tr>
<tr><td></td><td>Parameter</td><td>$keystore</td><td>Keystore im PEM-Format</td></tr>
<tr><td></td><td></td><td>$clientstore</td><td>Clientstore im PEM-Format</td></tr>
<tr><td></td><td></td><td>$keypass</td><td>Passwort</td></tr>
<tr><td></td><td></td><td>$useSSL</td><td>true / false</td></tr>
<tr><td>setKassenzeichennummer</td><td colspan="3">Setzt an der Klasse <code>EpayBLKonnektorClient</code> die Kassenzeichennummer zur Verwendung in der Ressource beim Abfragen der Kassenzeichen-Information</td></tr>
<tr><td></td><td>Aufrufparameter</td><td>$kassenzeichennummer</td><td>Kassenzeichennummer (String)</td></tr>
</table>

setSepamandatsreferenz	Setzt an der Klasse `EpayBLKonnektorClient` die SEPA-Mandatsreferenz zur Verwendung in der Ressource zum Vervollständigen eines SEPA-Mandats		
	Parameter	`$sepamandatsreferenz`	SEPA-Mandatsreferenz (String)
setKundennummer	Setzt an der Klasse `EpayBLKonnektorClient` die Kundennummer zur Verwendung in der Ressource zum Löschen eines Kunden		
	Parameter	`$kundennummer`	Kundennummer (String)
abfragenZahlverfahren	Ermittelt für einen Mandanten die aktiven Zahlverfahren		
	Return	Gibt ein Objekt vom Typ `ZahlverfahrenListErgebnis` zurück	
uebertragenBuchungsliste	Übertragt eine Buchungsliste an die ePayBL		
	Parameter	`$buchungsliste`	Buchungsliste (Objekt vom Typ `Buchungsliste`)
	Return	Gibt ein Objekt vom Typ ZahlvorgangsErgebnis zurück	
abfragenKassenzeichenstatus	Fragt den Status eines Kassenzeichens ab		
	Parameter	`$kassenzeichennummer`	Kassenzeichen (String)
	Return	Gibt ein Objekt vom Typ `KassenzeichenstatusErgebnis` zurück	
	Aktiviert ein Kassenzeichen (d.h. setzt den Paypage-Status auf INAKTIV		

aktivierenKassenzeichen	Parameter	`$kassenzeichennummer`	Kassenzeichen (String)
	Return	Gibt ein Objekt vom Typ `KassenzeichenstatusErgebnis` zurück	
anlegenSepaMandat	Legt im Fall einer internen Mandatsverwaltung ein SEPA-Mandat an		
	Parameter	`$sepaMandatInternAnlegedaten`	SEPA-Mandatsdaten (Objekt vom Typ `SepaMandatInternAnlegedaten`)
	Return	Gibt ein Objekt vom Typ `SepamandatErgebnis` zurück	
vervollstaendigenSepamandat	Vervollständigt im Fall einer internen Mandatsverwaltung ein SEPA-Mandat		
	Parameter	`$sepamandatsreferenz`	SEPA-Mandatsreferenz (String)
		`$sepamandatInternVervollstaendigungsdaten`	Daten zum Vervollständigen eines SEPA-Mandats (Objekt vom Typ `SepamandatInternVervollstaendigungsdaten`)
	Return	Gibt ein Objekt vom Typ `SepamandatErgebnis` zurück	
loeschenKunde	Löscht einen Kunden in der ePayBL		
	Parameter	`$kundennummer`	Kundennummer (String)
	Return	Gibt ein Objekt vom Typ `ErgebnisErgebnis` zurück	
getReturnCode	Ermittelt den Retruncode einer Response		
	Parameter	`$response`	Response als JSON-String

	Return	Returncode

Tabelle 28: Verfügbare Funktionen an der Konnektor-Client-Bibliothek für PHP

```
<?php

      require_once 'httpful.phar';
      require_once('ePayBLKonnektorClientLibObjects.phar');

class EpayBLKonnektorClient {

      const RES_ZAHLVERFAHREN = "zahlverfahren";
    const RES_BUCHUNGSLISTEN = "buchungslisten";
      const RES_KASSENZEICHEN = "kassenzeichen";
      const RES_SEPAMANDATANLEGEN = "sepamandatAnlegen";
      const RES_SEPAMANDATVERVOLLSTAENDIGEN = "sepamandatVervollstaendigen";
      const RES_KUNDELOESCHEN = "kundeLoeschen";
      const RES_AKTIVIERENKASSENZEICHEN = "aktivierenKassenzeichen";

      const PARAM_SERVERADRESS = "serverAddress";
      const PARAM_APPLICATION = "application";
      const PARAM_MANDANTENNR = "mandantennr";
      const PARAM_BEWIRTSCHAFTERNR = "bewirtschafternr";

      const PARAM_USESSL = "useSSL";
      const PARAM_KEYSTORE = "keystore";
      const PARAM_CLIENTSTORE = "clientstore";
      const PARAM_KEYPASS = "keypass";

    const PARAM_USE_PROXY = "useProxy";

    // proxyConfig["proxy_host"]
    // proxyConfig["proxy_port"]
    // proxyConfig["auth_type"]
    // proxyConfig["auth_username"]
    // proxyConfig["auth_password"]
    // proxyConfig["proxy_type"] allowed {"HTTP", "SOCKS4", "SOCKS5"}
    const PARAM_PROXY_CONFIG = "proxyConfig";

      const PARAM_KASSENZEICHENNUMMER = "kassenzeichennummer";
      const PARAM_SEPAMANDATSREFERENZ = "sepamandatsreferenz";
      const PARAM_KUNDENNUMMER = "kundennummer";

      private $epayBLKonnektorConfig;
      private $resources;

    public function __construct($serverAddress, $mandant, $bewirtschafter, $useProxy
= false, $proxyConfig = null, $application = "konnektor"){
        $this->init($serverAddress, $mandant, $bewirtschafter, $useProxy, $proxyCon-
fig, $application);
    }

      private function init($serverAddress, $mandant, $bewirtschafter, $useProxy,
$proxyConfig, $application) {

            $this->resources[EpayBLKonnektorClient::RES_ZAHLVERFAHREN] = ar-
ray("/epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirt-
schafternr}/zahlverfahren", "GET");
```

```
            $this->resources[EpayBLKonnektorClient::RES_BUCHUNGSLISTEN] = ar-
ray("/epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirt-
schafternr}/buchungslisten", "POST");
            $this->resources[EpayBLKonnektorClient::RES_KASSENZEICHEN] = ar-
ray("/epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirt-
schafternr}/kassenzeichen/{kassenzeichennummer}", "GET");
            $this->resources[EpayBLKonnektorClient::RES_SEPAMANDATANLEGEN] = ar-
ray("/epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirt-
schafternr}/sepamandate/anlegen", "POST");
            $this->resources[EpayBLKonnektorClient::RES_SEPAMANDATVERVOLLSTAENDIGEN]
= array("/epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirt-
schafternr}/sepamandate/{sepamandatsreferenz}/vervollstaendigen", "POST");
            $this->resources[EpayBLKonnektorClient::RES_KUNDELOESCHEN] = ar-
ray("/epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirt-
schafternr}/kunden/{kundennummer}", "DELETE");
            $this->resources[EpayBLKonnektorClient::RES_AKTIVIERENKASSENZEICHEN] =
array("/epayment/fachverfahren/v1_0/mandanten/{mandantennr}/bewirtschafter/{bewirt-
schafternr}/kassenzeichen/{kassenzeichennummer}", "POST");

            $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_SERVERADRESS]
= $serverAddress;
            $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_APPLICATION] =
$application;
            $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_MANDANTENNR] =
$mandant;
            $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_BEWIRTSCHAF-
TERNR] = $bewirtschafter;
            $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_USE_PROXY] =
$useProxy;
        $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_PROXY_CONFIG] =
$proxyConfig;

            $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_USESSL] =
false;
            $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_KEYSTORE] =
"";
            $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_CLIENTSTORE] =
"";
            $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_KEYPASS] = "";

      }
      /**
      * Sends off the request, and return the response
      * In case of Exceptions an error message including the exception will be re-
turned
      *
      * @param $method : method of the request ("GET" or "POST" or "DELETE")
      * @param $requestUrl : the complete URL http://<server>/konnektor/<resource>
      * @param $requestBody : the body of the request in case of POST-requests
      * @return Response of the request
      */
      private function getResponse($method, $requestUrl, $requestBody) {
            $response = "";

            //print("getResponse: $method, $requestUrl, $requestBody");
```

```
        switch ($method) {
            case "GET":
                $requestObject = \Httpful\Request::get($requestUrl)->expectsJson();
                break;
            case "DELETE":
                $requestObject = \Httpful\Request::delete($requestUrl)->expects-
Json();
                break;
            case "POST":
                $requestObject = \Httpful\Request::post($requestUrl)->body($request-
Body)->sendsJson();
                break;
            default :
                $response = "Wrong method $method, not supported for ePayBL-Konnek-
tor";
                break;
        }

             $paramproxyconf = $this->epayBLKonnektorConfig[EpayBLKonnektorCli-
ent::PARAM_PROXY_CONFIG];

        if ($this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_USE_PROXY] &&
isset($paramproxyconf)){

                $proxy_type = null;

                switch ($paramproxyconf['proxy_type']){
                    case "SOCKS4":
                        $proxy_type = \Httpful\Proxy::SOCKS4;
                        break;
                    case "SOCKS5":
                        $proxy_type = \Httpful\Proxy::SOCKS5;
                        break;
                    case "HTTP":
                    default:
                        $proxy_type = \Httpful\Proxy::HTTP;
                        break;
                }

                $requestObject = $requestObject->useProxy($param-
proxyconf['proxy_host'], $paramproxyconf['proxy_port'], $paramproxyconf['auth_type'],
$paramproxyconf['auth_username'], $paramproxyconf['auth_password'], $proxy_type);

        }

        if ($this->useSSL()) {

                   $requestObject = $requestObject->authenticateWithCert($this-
>epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_CLIENTSTORE], $this->epayBL-
KonnektorConfig[EpayBLKonnektorClient::PARAM_KEYSTORE], $this->epayBLKonnektorCon-
fig[EpayBLKonnektorClient::PARAM_KEYPASS], 'PEM');

             }

             try {
            $response = $requestObject->send();
```

```
        } catch (Exception $exc) {
            $response = "Exception in " . $exc->getFile() . " at line " . $exc->get-
Line() . "\nMessage: " . $exc->getMessage();
        }

        return $response;

      }

      private function createRequestUrl($resource) {
            $requestURL = $this->epayBLKonnektorConfig[ EpayBLKonnektorClient::PA-
RAM_SERVERADRESS ];
            if ( $requestURL[ strlen($requestURL)-1 ] != "/") {
                  $requestURL = $requestURL . "/";
            }

            $application = $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PA-
RAM_APPLICATION];

            $requestURL = $requestURL . $application . $resource;

            return $requestURL;
      }

      private function fillResource($resource) {

            foreach ($this->epayBLKonnektorConfig as $key=>$value) {
                  if (!is_array($value)) {
                        $resource = str_replace("{".$key."}", $value, $resource);
                  }
            }

            return $resource;
      }

      private function parseResponse($response, $paramArray) {
            $obj = json_decode($response, true);
            foreach($paramArray as $param) {
                  $responseParam[$param] = (is_array($obj) && array_key_exists($pa-
ram, $obj) ? $obj[$param] : null);
            }
            return $responseParam;
      }

      private function useSSL(){

            return $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PA-
RAM_USESSL];
      }

      ////////////////////////////////////////////////////////////////////////////
////////////////////////////////

      /**
      * gives the aktual available resources of the REST interface to the ePayBL
connector assigned to a key
```

```
	*
	* allowed keys: "zahlverfahren", "buchungslisten", "kassenzeichen", "sepaman-
datAnlegen", "sepamandatVervollstaendigen", "kundeLoeschen", "aktivierenKassenzei-
chen"
	*
	* @param $key : key for identiying the resource from an array, if not set the
complete array will be returned
	* @return array of available resources including assigned method
	*/
	public function getResource($key = null) {

		if ($key != null && array_key_exists($key, $this->resources)) {
			$resource = $this->resources[$key][0];
			$resource = $this->fillResource($resource);
			return $resource;
		}
		else {
			return $this->resources;
		}
	}

	/**
	* gives the aktual method for a specified resources
	*
	* allowed keys: "zahlverfahren", "buchungslisten", "kassenzeichen", "sepaman-
datAnlegen", "sepamandatVervollstaendigen","kundeLoeschen","aktivierenKassenzeichen"
	*
	* @param $key : key for identiying the resource from an array
	* @return array of available resources including assigned method
	*/
	public function getMethod($key) {

		if ($key != null && array_key_exists($key, $this->resources)) {
			$method = $this->resources[$key][1];
			return $method;
		}
		else {
			return "GET";
		}
	}

	/**
	* sets the mandantNr und bewirtschafternr for the request-URL
	* updates the variable $epayBLKonnektorConfig
	*
	* @return
	*/
	public function setMandantAndBewirtschafter($mandant, $bewirtschafter) {
		$this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_MANDANTENNR] =
$mandant;
		$this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_BEWIRTSCHAF-
TERNR] = $bewirtschafter;
	}

	/**
	* sets the keystore and passwords for client authentication
```

```
	* updates the variable $epayBLKonnektorConfig
	*
	* keystore and clientstore have to be from type PEM
	* create keystore and clientstore from the same PKC12-Keystore via:
	*
	*	openssl pkcs12 -in zertifikat_from_epaybl-ZV.p12 -out keystore.pem -
nocerts
	*	openssl pkcs12 -in zertifikat_from_epaybl-ZV.p12 -out clientstore.pem -
clcerts -nokeys
	*
	* When converting the keystores make sure to set the same password in both con-
versions corretly and to use this passwords here
	*
	* @return
	*/
	public function setSSL($keystore, $clientstore, $keypass, $useSSL = true) {
		$this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_USESSL] =
$useSSL;
		$this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_KEYSTORE] =
$keystore;
		$this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_CLIENTSTORE] =
$clientstore;
		$this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_KEYPASS] =
$keypass;
	}

	/**
	* sets the kassenzeichennummer for the request-URL
	* updates the variable $epayBLKonnektorConfig
	*
	* @return
	*/
	public function setKassenzeichennummer($kassenzeichennummer) {
		$this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_KASSENZEICHEN-
NUMMER] = $kassenzeichennummer;
	}

	/**
	* sets the sepamandatsreferenz for the request-URL
	* updates the variable $epayBLKonnektorConfig
	*
	* @return
	*/
	public function setSepamandatsreferenz($sepamandatsreferenz) {
		$this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_SEPAMANDATSRE-
FERENZ] = $sepamandatsreferenz;
	}

	/**
	* sets the kundennummer for the request-URL
	* updates the variable $epayBLKonnektorConfig
	*
	* @return
	*/
	public function setKundennummer($kundennummer) {
```

```
            $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PARAM_KUNDENNUMMER]
= $kundennummer;
      }

      /**
      * returns all active zahlverfahren for a mandant
      *
      * @return zahlverfahrenListErgebnis of type ZahlverfahrenListErgebnis
      */
      public function abfragenZahlverfahren() {

            $resource = $this->getResource(EpayBLKonnektorClient::RES_ZAHLVERFAH-
REN);
            $method = $this->getMethod(EpayBLKonnektorClient::RES_ZAHLVERFAHREN);
            $requestUrl = $this->createRequestUrl($resource);

            $response = $this->getResponse($method, $requestUrl, null);

            $responseParam = $this->parseResponse($response, array("ergeb-
nis","date","zahlverfahren"));
            $zahlverfahrenListErgebnis = new ZahlverfahrenListErgebnis($responsePa-
ram["ergebnis"], $responseParam["date"], $responseParam["zahlverfahren"]);

            return $zahlverfahrenListErgebnis;
      }

      /**
      * transfers a buchungsliste
      *
      * @param $buchungsliste : buchungsliste of type Buchungsliste
      * @return ZahlvorgangsErgebnis of type ZahlverfahrenListErgebnis
      */
      public function uebertragenBuchungsliste($buchungsliste) {

            $resource = $this->getResource(EpayBLKonnektorClient::RES_BUCHUNGSLIS-
TEN);
            $method = $this->getMethod(EpayBLKonnektorClient::RES_BUCHUNGSLISTEN);
            $requestUrl = $this->createRequestUrl($resource);
            $requestBody = $buchungsliste->toJson();

            $response = $this->getResponse($method, $requestUrl, $requestBody);

            $responseParam = $this->parseResponse($response, array("ergeb-
nis","date", "zahlvorgangsInfo"));
            $zahlvorgangsErgebnis = new ZahlvorgangsErgebnis($responseParam["ergeb-
nis"], $responseParam["date"], $responseParam["zahlvorgangsInfo"]);

            return $zahlvorgangsErgebnis;
      }

      /**
      * gets information about a certain kassenzeichen
      *
      * @param $kassenzeichennummer : kassenzeichennummer
      * @return ZahlvorgangsErgebnis of type ZahlverfahrenListErgebnis
      */
```

```
	public function abfragenKassenzeichenstatus($kassenzeichennummer) {

		$this->setKassenzeichennummer($kassenzeichennummer);

		$resource = $this->getResource(EpayBLKonnektorClient::RES_KASSENZEI-
CHEN);
		$method = $this->getMethod(EpayBLKonnektorClient::RES_KASSENZEICHEN);
		$requestUrl = $this->createRequestUrl($resource);

		$response = $this->getResponse($method, $requestUrl, null);

		$responseParam = $this->parseResponse($response, array("ergeb-
nis","date","kassenzeichen"));
		$kassenzeichenstatusErgebnis = new KassenzeichenstatusErgebnis($respon-
seParam["ergebnis"], $responseParam["date"], $responseParam["kassenzeichen"]);

		return $kassenzeichenstatusErgebnis;
	}

	/**
	* activates a certain kassenzeichen, i.e. sets the paypage state to "INAKTIV"
	* only allowed for kassenzeichen with zahlverfahren TERMINALZAHLUNG
	*
	* @param $kassenzeichennummer : kassenzeichennummer
	* @return ZahlvorgangsErgebnis of type ZahlverfahrenListErgebnis
	*/
	public function aktivierenKassenzeichen($kassenzeichennummer) {

		$this->setKassenzeichennummer($kassenzeichennummer);

		$resource = $this->getResource(EpayBLKonnektorClient::RES_AKTIVIERENKAS-
SENZEICHEN);
		$method = $this->getMethod(EpayBLKonnektorClient::RES_AKTIVIERENKASSEN-
ZEICHEN);
		$requestUrl = $this->createRequestUrl($resource);

		$response = $this->getResponse($method, $requestUrl, null);

		$responseParam = $this->parseResponse($response, array("ergeb-
nis","date","kassenzeichen"));
		$kassenzeichenstatusErgebnis = new KassenzeichenstatusErgebnis($respon-
seParam["ergebnis"], $responseParam["date"], $responseParam["kassenzeichen"]);

		return $kassenzeichenstatusErgebnis;
	}

	/**
	* set a SEPA mandat
	*
	* @param $sepaMandatInternAnlegedaten : Sepamandat of type SepaMandatInternAn-
legedaten
	* @return ZahlvorgangsErgebnis of type ZahlverfahrenListErgebnis
	*/
	public function anlegenSepaMandat($sepaMandatInternAnlegedaten) {
		$this->setKassenzeichennummer($kassenzeichennummer);
```

```
            $resource = $this->getResource(EpayBLKonnektorClient::RES_SEPAMANDATAN-
LEGEN);
            $method = $this->getMethod(EpayBLKonnektorClient::RES_SEPAMANDATANLE-
GEN);
            $requestUrl = $this->createRequestUrl($resource);
            $requestBody = $sepaMandatInternAnlegedaten->toJson();

            $response = $this->getResponse($method, $requestUrl, $requestBody);

            $responseParam = $this->parseResponse($response, array("ergeb-
nis","date", "sepamandatErgebnisdaten"));
            $sepamandatErgebnis = new SepamandatErgebnis($responseParam["ergebnis"],
$responseParam["date"], $responseParam["sepamandatErgebnisdaten"]);

            return $sepamandatErgebnis;

      }

      /**
      * set a SEPA mandat
      *
      * @param $sepamandatsreferenz : sepamandatsreferenz
      * @param $sepamandatInternVervollstaendigungsdaten : Sepamandat of type Sepa-
mandatInternVervollstaendigungsdaten
      * @return ZahlvorgangsErgebnis of type ZahlverfahrenListErgebnis
      */
      public function vervollstaendigenSepamandat($sepamandatsreferenz, $sepamandat-
InternVervollstaendigungsdaten) {
            $this->setSepamandatsreferenz($sepamandatsreferenz);

            $resource = $this->getResource(EpayBLKonnektorClient::RES_SEPAMANDATVER-
VOLLSTAENDIGEN);
            $method = $this->getMethod(EpayBLKonnektorClient::RES_SEPAMANDATVER-
VOLLSTAENDIGEN);
            $requestUrl = $this->createRequestUrl($resource);
            $requestBody = $sepamandatInternVervollstaendigungsdaten->toJson();

            $response = $this->getResponse($method, $requestUrl, $requestBody);

            $responseParam = $this->parseResponse($response, array("ergeb-
nis","date", "sepamandatErgebnisdaten"));
            $sepamandatErgebnis = new SepamandatErgebnis($responseParam["ergebnis"],
$responseParam["date"], $responseParam["sepamandatErgebnisdaten"]);

            return $sepamandatErgebnis;
      }

      /**
      * delets a customer
      *
      * @param $kundennummer : kundennummer
      * @return ZahlvorgangsErgebnis of type ZahlverfahrenListErgebnis
      */
      public function loeschenKunde($kundennummer) {

            $this->setKundennummer($kundennummer);
```

```
            $resource = $this->getResource(EpayBLKonnektorClient::RES_KUNDELO-
ESCHEN);
            $method = $this->getMethod(EpayBLKonnektorClient::RES_KUNDELOESCHEN);
            $requestUrl = $this->createRequestUrl($resource);

            $response = $this->getResponse($method, $requestUrl, null);

            $responseParam = $this->parseResponse($response, array("ergeb-
nis","date"));
            $loeschenKundeErgebnis = new ErgebnisErgebnis($responseParam["ergeb-
nis"], $responseParam["date"]);

            return $loeschenKundeErgebnis;
      }

      /**
      * Extracts the return code of the ePayBL-Konnektor-response
      *
      * @param $response : $response (JSON)
      * @return return code
      */
      public function getReturnCode($response) {
            $obj = json_decode($response, true);
            return $obj["ergebnis"]["rc"];
      }

      public function printConfiguration() {

            print("serveradress     : " . $this->epayBLKonnektorConfig[EpayBL-
KonnektorClient::PARAM_SERVERADRESS] ."\n");
            print("application      : " . $this->epayBLKonnektorConfig[EpayBL-
KonnektorClient::PARAM_APPLICATION] ."\n");
            print("mandnatnr        : " . $this->epayBLKonnektorConfig[EpayBL-
KonnektorClient::PARAM_MANDANTENNR] ."\n");
            print("bewirtschafternr : " . $this->epayBLKonnektorConfig[EpayBL-
KonnektorClient::PARAM_BEWIRTSCHAFTERNR] ."\n");
            print("use_proxy        : " . ($this->epayBLKonnektorConfig[EpayBL-
KonnektorClient::PARAM_USE_PROXY] ? "true" : "false") ."\n");
        print("proxy_config     : \n");

        $paramproxyconf = $this->epayBLKonnektorConfig[EpayBLKonnektorClient::PA-
RAM_PROXY_CONFIG];

        print("\tproxy_host     : " . $paramproxyconf['proxy_host'] ."\n");
            print("\tproxy_port     : " . $paramproxyconf['proxy_port'] ."\n");
            print("\tauth_type      : " . $paramproxyconf['auth_type'] ."\n");
            print("\tauth_username  : " . $paramproxyconf['auth_username'] ."\n");
            print("\tauth_password  : " . $paramproxyconf['auth_password'] ."\n");
            print("\tproxy_type     : " . $paramproxyconf['proxy_type'] ."\n");

            print("useSSL           : " . ($this->epayBLKonnektorConfig[EpayBL-
KonnektorClient::PARAM_USESSL] ? "true" : "false") . "\n");
            print("keystore         : " . $this->epayBLKonnektorConfig[EpayBL-
KonnektorClient::PARAM_KEYSTORE] . "\n");
```

```
            print("clientstore      : " . $this->epayBLKonnektorConfig[EpayBL-
KonnektorClient::PARAM_CLIENTSTORE] . "\n");
            print("keypass          : " . $this->epayBLKonnektorConfig[EpayBL-
KonnektorClient::PARAM_KEYPASS] . "\n");

      }
}

?>
```

Abbildung 19: Quellcode von ePayBLKonnektorClientLib.phar

Der folgende Code zeigt eine Beispielapplikation für die Nutzung der Bibliothek ePayBL-KonnektorClientLib.phar

```
<?php
require_once('ePayBLKonnektorClientLib.phar');
require_once('ePayBLKonnektorClientLibObjects.phar');

$serverAddress = "http://epayment244:8090";
$mandant = "RE00001";
$bewirtschafter = "ReB000";
$kassenzeichennummer = "asdc00000504";
$kundennummer = "k244013";
$application = "restapi";
$requestKey = EpayBLKonnektorClient::RES_ZAHLVERFAHREN

$response = "";
//Initialisierung
$epayBLKonnektorClient = new EpayBLKonnektorClient($serverAddress, $mandant, $bewirt-
schafter, $application);

if ($requestKey === EpayBLKonnektorClient::RES_ZAHLVERFAHREN) {
$zahlverfahrenListeErgebnis = $epayBLKonnektorClient->abfragenZahlverfahren();
      $response = $zahlverfahrenListeErgebnis->toString();
}

if ($requestKey === EpayBLKonnektorClient::RES_KASSENZEICHEN) {
      $kassenzeichenstatusErgebnis = $epayBLKonnektorClient-> abfragenKassenzeichen-
status($kassenzeichennummer);
      $response = $kassenzeichenstatusErgebnis->toString();
}
if ($requestKey === EpayBLKonnektorClient::RES_KUNDELOESCHEN) {
      $loeschenKundeErgebnis = $epayBLKonnektorClient->loeschenKunde($kundennummer);
      $response = $loeschenKundeErgebnis->toString();
}

if ($requestKey === EpayBLKonnektorClient::RES_BUCHUNGSLISTEN) {
      $buchungsliste = new Buchungsliste();
      $buchungsliste->setFaelligkeitsdatum("25.12.2018 16:13:34");
      $buchungsliste->setKennzeichenMahnverfahren("23");
      $buchungsliste->setZahltyp(Zahltyp::DIREKT);
```

```
	$buchungsliste->setBetrag(10.00);
	$buchungsliste->setBeschreibung("Buchungsliste mit Buchungen: 1");
	//$buchungsliste->setZahlverfahrencodes(array(ZahlverfahrenCodes::KREDIT-
KARTE));
	$buchungsliste->setZahlverfahrencodes(array(ZahlverfahrenCodes::SEPASDD));

	$fachverfahrendaten = array();
	$fachverfahrendaten[BuchungslisteFvDatenKeys::SUCCESSURL] =
		"http://www.heise.de";
	$fachverfahrendaten[BuchungslisteFvDatenKeys::CANCELURL] =
		"http://www.google.de";
	$fachverfahrendaten[BuchungslisteFvDatenKeys::ERRORURL] =
		"http://www.amazon.de";
	$buchungsliste->setFachverfahrendaten($fachverfahrendaten);

	$buchungen = array();
	$buchung = new Buchung();
	$buchung->setBruttobetrag(10.00);
	$buchung->setNettobetrag(9.00);
	$buchung->setSteuerbetrag(1.00);
	$buchung->setBuchungstext("Buchung 1");

	$kontierung = array();
	$kontierung[BuchungKontierungKeys::HAUSHALTSTELLE] = "1100";
	$kontierung[BuchungKontierungKeys::OBJEKTNUMMER] = "10000";
	$kontierung[BuchungKontierungKeys::HREF] = "HREF";

	$buchung->setKontierung($kontierung);

	$buchungen[] = $buchung;

	$buchungsliste->setBuchungen($buchungen);
	$kunde = new Kunde();
	$kunde->setKundennummer("k242017");
	$kunde->setTyp(KundeTyp::BESTAND);
	$kunde->setName("Peters");

	$sepamandatExtern = new SepamandatExtern();
	$sepamandatExtern->setSequenceType(SepaMandatSequenceType::OOFF);
	$sepamandatExtern->setDatumUnterschrift("16.05.2017 12:00:00");
	$sepamandatExtern->setDatumLetzteNutzung("28.08.2017 14:21:00");
	$sepamandatExtern->setType(SepaMandatType::V);
	$sepamandatExtern->setMandatreferenz("MR20170911165701");
	$sepamandatExtern->setKundennummer("k242017");
	$sepamandatExtern->setGlaeubigerId("DE17HKS00000032546");
	$kunde->setSepamandat($sepamandatExtern);

	$bankverbindung = new Bankverbindung();
	$bankverbindung->setBic("TESTDETT421");
	$bankverbindung->setIban("DE62370205000000102030");
	$bankverbindung->setKontoinhaber("Gerd Tester");
	$kunde->setBankverbindung($bankverbindung);

	$buchungsliste->setKunde($kunde);
```

```
	$loeschenKundeErgebnis = $epayBLKonnektorClient-> uebertragenBuchungsliste($bu-
chungsliste);
	$response = $loeschenKundeErgebnis->toString();
}

print(“Response = $response\n“);

?>
```

Abbildung 20: Beispielapplikation zur ePayBL-Konnektor-Client-Bibliothek

Die verwendeten Objekte („RestApiClientTestHelper.phar“) orientieren sich nach dem in diesem Dokument beschriebene Datenmodell. Die einzelnen Objekte haben Konstruktoren und Setter. Die Ergebnis-Objekte haben eine toString-Methode zum Umwandeln in einen Json-String. Auf die Unterobjekte kann über die entsprechenden Parameternamen zugegriffen werden:

Um beispielsweise auf den Returncode eines Ergebnisobjektes zuzugreifen kann

```
$obj["ergebnis"]["rc"]
```

aufgerufen werden.

Für die im Datenmodell angegegebenen Enums stehen entsprechende Konstantenklassen bereit.

```
<?php


////////////////////////////////////////////////////////////////////////////////
// Objects
////////////////////////////////////////////////////////////////////////////////

class ZahlverfahrenListErgebnis {
	private $ergebnis;
	private $date;
	private $zahlverfahren;

    public function __construct($ergebnis, $date, $zahlverfahren){
        $this->init($ergebnis, $date, $zahlverfahren);
    }

	private function init($ergebnis, $date, $zahlverfahren) {
		$this->ergebnis = $ergebnis;
		$this->date = $date;
		$this->zahlverfahren = $zahlverfahren;
	}

	public function getErgebnis() {
		return $this->ergebnis;
	}

	public function getDate() {
		return $this->date;
	}

	public function getZahlverfahren() {
		return $this->zahlverfahren;
	}

	public function toString() {
		$out = "{\"ergebnis\":" . json_encode($this->ergebnis) . ",\"date\":" .
json_encode($this->date) . ",\"zahlverfahren\":" . json_encode($this->zahlverfahren)
."}";
		return $out;
	}
}

class KassenzeichenstatusErgebnis {
	private $ergebnis;
	private $date;
	private $kassenzeichen;

    public function __construct($ergebnis, $date, $kassenzeichen){
        $this->init($ergebnis, $date, $kassenzeichen);
    }

	private function init($ergebnis, $date, $kassenzeichen) {
```

```
		$this->ergebnis = $ergebnis;
		$this->date = $date;
		$this->kassenzeichen = $kassenzeichen;
	}

	public function getErgebnis() {
		return $this->ergebnis;
	}

	public function getDate() {
		return $this->date;
	}

	public function getKassenzeichen() {
		return $this->kassenzeichen;
	}

	public function toString() {
		$out = "{\"ergebnis\":" . json_encode($this->ergebnis) . ",\"date\":" .
json_encode($this->date) . ",\"kassenzeichen\":" . json_encode($this->kassenzeichen)
."}";
		return $out;
	}
}

class ErgebnisErgebnis {
	private $ergebnis;
	private $date;

    public function __construct($ergebnis, $date){
        $this->init($ergebnis, $date);
    }

	private function init($ergebnis, $date) {
		$this->ergebnis = $ergebnis;
		$this->date = $date;
	}

	public function getErgebnis() {
		return $this->ergebnis;
	}

	public function getDate() {
		return $this->date;
	}

	public function toString() {
		$out = "{\"ergebnis\":" . json_encode($this->ergebnis) . ",\"date\":" .
json_encode($this->date) ."}";
		return $out;
	}
}

class ZahlvorgangsErgebnis {
	private $ergebnis;
	private $date;
```

```
      private $zahlvorgangsInfo;

    public function __construct($ergebnis, $date, $zahlvorgangsInfo){
        $this->init($ergebnis, $date, $zahlvorgangsInfo);
    }

      private function init($ergebnis, $date, $zahlvorgangsInfo) {
            $this->ergebnis = $ergebnis;
            $this->date = $date;
            $this->zahlvorgangsInfo = $zahlvorgangsInfo;
      }

      public function getErgebnis() {
            return $this->ergebnis;
      }

      public function getDate() {
            return $this->date;
      }

      public function getZahlvorgangsInfo() {
            return $this->zahlvorgangsInfo;
      }

      public function toString() {
            $out = "{\"ergebnis\":" . json_encode($this->ergebnis) . ",\"date\":" .
json_encode($this->date) . ",\"zahlvorgangsInfo\":" . json_encode($this->zahlvor-
gangsInfo) ."}";
            return $out;
      }
}

class SepamandatErgebnis {
      private $ergebnis;
      private $date;
      private $sepamandatErgebnisdaten;

    public function __construct($ergebnis, $date, $sepamandatErgebnisdaten){
        $this->init($ergebnis, $date, $sepamandatErgebnisdaten);
    }

      private function init($ergebnis, $date, $sepamandatErgebnisdaten) {
            $this->ergebnis = $ergebnis;
            $this->date = $date;
            $this->sepamandatErgebnisdaten = $sepamandatErgebnisdaten;
      }

      public function getErgebnis() {
            return $this->ergebnis;
      }

      public function getDate() {
            return $this->date;
      }

      public function getSepamandatErgebnisdaten() {
```

```
            return $this->sepamandatErgebnisdaten;
      }

      public function toString() {
            $out = "{\"ergebnis\":" . json_encode($this->ergebnis) . ",\"date\":" .
json_encode($this->date) . ",\"sepamandatErgebnisdaten\":" . json_encode($this-
>sepamandatErgebnisdaten) ."}";
            return $out;
      }
}

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////
/// classes for the request body
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////

class SepamandatInternReferenz {
      private $xxtypetagxx;
      private $mandatreferenz;

    public function __construct(){
        $this->init();
    }

      private function init() {
            $this->xxtypetagxx = "SepamandatInternReferenz";
      }

      public function setMandatreferenz($mandatreferenz) {
            $this->mandatreferenz = $mandatreferenz;
      }

      public function toJson() {
            $out = "{\"xxtypetagxx\":" . json_encode($this->xxtypetagxx) . ",\"man-
datreferenz\":" . json_encode($this->mandatreferenz) ."}";
            return $out;
      }

}

class SepaMandatInternAnlegedaten {
      private $xxtypetagxx;
      private $mandatreferenz;
      private $einmalmandat;
      private $type;
      private $kundennummer;
      private $glaeubigerId;
      private $zeichnungsberechtigter;
      private $kontoinhaberWeichtAb;
      private $kontoinhaberNachname;
      private $kontoinhaberVorname;
      private $kontoinhaberStrasse;
      private $kontoinhaberHausnummer;
      private $kontoinhaberPLZ;
      private $kontoinhaberStadt;
```

```
      private $kontoinhaberLand;
      private $kontoinhaberBankname;
      private $kontoinhaberIban;
      private $kontoinhaberBic;

   public function __construct(){
       $this->init();
   }

      private function init() {
            $this->xxtypetagxx = "SepaMandatInternAnlegedaten";
      }

      public function setMandatreferenz($mandatreferenz) {
            $this->mandatreferenz = $mandatreferenz;
      }
      public function setEinmalmandat($einmalmandat) {
            $this->einmalmandat = $einmalmandat;
      }
      public function setType($type) {
            $this->type = $type;
      }
      public function setKundennummer($kundennummer) {
            $this->kundennummer = $kundennummer;
      }
      public function setGlaeubigerId($glaeubigerId) {
            $this->glaeubigerId = $glaeubigerId;
      }
      public function setZeichnungsberechtigter($zeichnungsberechtigter) {
            $this->zeichnungsberechtigter = $zeichnungsberechtigter;
      }
      public function setKontoinhaberWeichtAb($kontoinhaberWeichtAb) {
            $this->kontoinhaberWeichtAb = $kontoinhaberWeichtAb;
      }
      public function setKontoinhaberNachname($kontoinhaberNachname) {
            $this->kontoinhaberNachname = $kontoinhaberNachname;
      }
      public function setKontoinhaberVorname($kontoinhaberVorname) {
            $this->kontoinhaberVorname = $kontoinhaberVorname;
      }
      public function setKontoinhaberStrasse($kontoinhaberStrasse) {
            $this->kontoinhaberStrasse = $kontoinhaberStrasse;
      }
      public function setKontoinhaberHausnummer($kontoinhaberHausnummer) {
            $this->kontoinhaberHausnummer = $kontoinhaberHausnummer;
      }
      public function setKontoinhaberPLZ($kontoinhaberPLZ) {
            $this->kontoinhaberPLZ = $kontoinhaberPLZ;
      }
      public function setkontoinhaberStadt($kontoinhaberStadt) {
            $this->kontoinhaberStadt = $kontoinhaberStadt;
      }
      public function setKontoinhaberLand($kontoinhaberLand) {
            $this->kontoinhaberLand = $kontoinhaberLand;
      }
      public function setKontoinhaberBankname($kontoinhaberBankname) {
```

```
		$this->kontoinhaberBankname = $kontoinhaberBankname;
	}
	public function setKontoinhaberIban($kontoinhaberIban) {
		$this->kontoinhaberIban = $kontoinhaberIban;
	}
	public function setKontoinhaberBic($kontoinhaberBic) {
		$this->kontoinhaberBic = $kontoinhaberBic;
	}

	public function toJson() {
		$out = "{\"xxtypetagxx\":" . json_encode($this->xxtypetagxx) .
			",\"mandatreferenz\":" . json_encode($this->mandatreferenz) .
			",\"einmalmandat\":" . json_encode($this->einmalmandat) .
			",\"type\":" . json_encode($this->type) .
			",\"kundennummer\":" . json_encode($this->kundennummer) .
			",\"glaeubigerId\":" . json_encode($this->glaeubigerId) .
			",\"zeichnungsberechtigter\":" . json_encode($this->zeichnungsbe-
rechtigter) .
			",\"kontoinhaberWeichtAb\":" . json_encode($this->kontoinhaber-
WeichtAb) .
			",\"kontoinhaberNachname\":" . json_encode($this->kontoinhaber-
Nachname) .
			",\"kontoinhaberVorname\":" . json_encode($this->kontoinhaberVor-
name) .
			",\"kontoinhaberStrasse\":" . json_encode($this->kontoinhaber-
Strasse) .
			",\"kontoinhaberHausnummer\":" . json_encode($this->kontoinhaber-
Hausnummer) .
			",\"kontoinhaberPLZ\":" . json_encode($this->kontoinhaberPLZ) .
			",\"kontoinhaberStadt\":" . json_encode($this->kontoinhaberStadt)
.
			",\"kontoinhaberLand\":" . json_encode($this->kontoinhaberLand) .
			",\"kontoinhaberBankname\":" . json_encode($this->kontoinha-
berBankname) .
			",\"kontoinhaberIban\":" . json_encode($this->kontoinhaberIban) .
			",\"kontoinhaberBic\":" . json_encode($this->kontoinhaberBic) .
"}";
		return $out;
	}

}

class SepamandatInternVervollstaendigungsdaten {
	private $xxtypetagxx;
	private $datumUnterschrift;
	private $ortUnterschrift;

	public function __construct(){
	  $this->init();
    }

	private function init() {
		$this->xxtypetagxx = "SepamandatInternVervollstaendigungsdaten";
	}

	public function setDatumUnterschrift($datumUnterschrift) {
```

```
		$this->datumUnterschrift = $datumUnterschrift;
	}
	public function setOrtUnterschrift($ortUnterschrift) {
		$this->ortUnterschrift = $ortUnterschrift;
	}

	public function toJson() {
		$out = "{\"xxtypetagxx\":" . json_encode($this->xxtypetagxx) .
			",\"datumUnterschrift\":" . json_encode($this->datumUnterschrift)
.
			",\"ortUnterschrift\":" . json_encode($this->ortUnterschrift) .
"}";
		return $out;
	}
}

class SepamandatExtern {
	private $xxtypetagxx;
	private $mandatreferenz;
	private $type;
	private $kundennummer;
	private $glaeubigerId;
	private $zeichnungsberechtigter;
	private $datumUnterschrift;
	private $ortUnterschrift;
	private $sequenceType;
	private $datumLetzteNutzung;
	private $kontoinhaberWeichtAb;
	private $kontoinhaberNachname;
	private $kontoinhaberVorname;
	private $kontoinhaberStrasse;
	private $kontoinhaberHausnummer;
	private $kontoinhaberPLZ;
	private $kontoinhaberStadt;
	private $kontoinhaberLand;
	private $kontoinhaberBankname;
	private $kontoinhaberIban;
	private $kontoinhaberBic;

	public function __construct(){
	  $this->init();
    }
	private function init() {
		$this->xxtypetagxx = "SepamandatExtern";
	}

	public function setMandatreferenz($mandatreferenz) {
		$this->mandatreferenz = $mandatreferenz;
	}
	public function setType($type) {
		$this->type = $type;
	}
	public function setKundennummer($kundennummer) {
		$this->kundennummer = $kundennummer;
	}
	public function setGlaeubigerId($glaeubigerId) {
```

```
		$this->glaeubigerId = $glaeubigerId;
	}
	public function setZeichnungsberechtigter($zeichnungsberechtigter) {
		$this->zeichnungsberechtigter = $zeichnungsberechtigter;
	}
	public function setDatumUnterschrift($datumUnterschrift) {
		$this->datumUnterschrift = $datumUnterschrift;
	}
	public function setOrtUnterschrift($ortUnterschrift) {
		$this->ortUnterschrift = $ortUnterschrift;
	}
	public function setSequenceType($sequenceType) {
		$this->sequenceType = $sequenceType;
	}
	public function setDatumLetzteNutzung($datumLetzteNutzung) {
		$this->datumLetzteNutzung = $datumLetzteNutzung;
	}
	public function setKontoinhaberWeichtAb($kontoinhaberWeichtAb) {
		$this->kontoinhaberWeichtAb = $kontoinhaberWeichtAb;
	}
	public function setKontoinhaberNachname($kontoinhaberNachname) {
		$this->kontoinhaberNachname = $kontoinhaberNachname;
	}
	public function setKontoinhaberVorname($kontoinhaberVorname) {
		$this->kontoinhaberVorname = $kontoinhaberVorname;
	}
	public function setKontoinhaberStrasse($kontoinhaberStrasse) {
		$this->kontoinhaberStrasse = $kontoinhaberStrasse;
	}
	public function setKontoinhaberHausnummer($kontoinhaberHausnummer) {
		$this->kontoinhaberHausnummer = $kontoinhaberHausnummer;
	}
	public function setKontoinhaberPLZ($kontoinhaberPLZ) {
		$this->kontoinhaberPLZ = $kontoinhaberPLZ;
	}
	public function setkontoinhaberStadt($kontoinhaberStadt) {
		$this->kontoinhaberStadt = $kontoinhaberStadt;
	}
	public function setKontoinhaberLand($kontoinhaberLand) {
		$this->kontoinhaberLand = $kontoinhaberLand;
	}
	public function setKontoinhaberBankname($kontoinhaberBankname) {
		$this->kontoinhaberBankname = $kontoinhaberBankname;
	}
	public function setKontoinhaberIban($kontoinhaberIban) {
		$this->kontoinhaberIban = $kontoinhaberIban;
	}
	public function setKontoinhaberBic($kontoinhaberBic) {
		$this->kontoinhaberBic = $kontoinhaberBic;
	}

	public function toJson() {
		$out = "{\"xxtypetagxx\":" . json_encode($this->xxtypetagxx) .
			",\"mandatreferenz\":" . json_encode($this->mandatreferenz) .
			",\"type\":" . json_encode($this->type) .
			",\"kundennummer\":" . json_encode($this->kundennummer) .
```

```
                    ",\"glaeubigerId\":" . json_encode($this->glaeubigerId) .
                    ",\"zeichnungsberechtigter\":" . json_encode($this->zeichnungsbe-
rechtigter) .
                    ",\"datumUnterschrift\":" . json_encode($this->datumUnterschrift)
.
                    ",\"ortUnterschrift\":" . json_encode($this->ortUnterschrift) .
                    ",\"sequenceType\":" . json_encode($this->sequenceType) .
                    ",\"datumLetzteNutzung\":" . json_encode($this->datumLetzteNut-
zung) .
                    ",\"kontoinhaberWeichtAb\":" . json_encode($this->kontoinhaber-
WeichtAb) .
                    ",\"kontoinhaberNachname\":" . json_encode($this->kontoinhaber-
Nachname) .
                    ",\"kontoinhaberVorname\":" . json_encode($this->kontoinhaberVor-
name) .
                    ",\"kontoinhaberStrasse\":" . json_encode($this->kontoinhaber-
Strasse) .
                    ",\"kontoinhaberHausnummer\":" . json_encode($this->kontoinhaber-
Hausnummer) .
                    ",\"kontoinhaberPLZ\":" . json_encode($this->kontoinhaberPLZ) .
                    ",\"kontoinhaberStadt\":" . json_encode($this->kontoinhaberStadt)
.
                    ",\"kontoinhaberLand\":" . json_encode($this->kontoinhaberLand) .
                    ",\"kontoinhaberBankname\":" . json_encode($this->kontoinha-
berBankname) .
                    ",\"kontoinhaberIban\":" . json_encode($this->kontoinhaberIban) .
                    ",\"kontoinhaberBic\":" . json_encode($this->kontoinhaberBic) .
"}";
             return $out;
      }
}

class Buchungsliste {
      private $kassenzeichennummer;
      private $faelligkeitsdatum;
      private $waehrungskennzeichen;
      private $kennzeichenMahnverfahren;
      private $transaktionsnummer;
      private $zahlverfahrencodes;
      private $buchungen;
      private $beschreibung;
      private $kunde;
      private $buchungslistenparameter;
      private $fachverfahrendaten;
      private $zahltyp;
      private $betrag;

      public function __construct(){
        $this->init();
    }
      private function init() {
      }

      public function setKassenzeichennummer($kassenzeichennummer) {
            $this->kassenzeichennummer = $kassenzeichennummer;
      }
```

```
	public function setFaelligkeitsdatum($faelligkeitsdatum) {
		$this->faelligkeitsdatum = $faelligkeitsdatum;
	}
	public function setWaehrungskennzeichen($waehrungskennzeichen) {
		$this->waehrungskennzeichen = $waehrungskennzeichen;
	}
	public function setKennzeichenMahnverfahren($kennzeichenMahnverfahren) {
		$this->kennzeichenMahnverfahren = $kennzeichenMahnverfahren;
	}
	public function setTransaktionsnummer($transaktionsnummer) {
		$this->transaktionsnummer = $transaktionsnummer;
	}
	public function setZahlverfahrencodes($zahlverfahrencodes) {
		$this->zahlverfahrencodes = $zahlverfahrencodes;
	}
	public function setBuchungen($buchungen) {
		$this->buchungen = $buchungen;
	}
	public function setBeschreibung($beschreibung) {
		$this->beschreibung = $beschreibung;
	}
	public function setKunde($kunde) {
		$this->kunde = $kunde;
	}
	public function setBuchungslistenparameter($buchungslistenparameter) {
		$this->buchungslistenparameter = $buchungslistenparameter;
	}
	public function setFachverfahrendaten($fachverfahrendaten) {
		$this->fachverfahrendaten = $fachverfahrendaten;
	}
	public function setZahltyp($zahltyp) {
		$this->zahltyp = $zahltyp;
	}
	public function setBetrag($betrag) {
		$this->betrag = $betrag;
	}

	public function toJson() {
		$out = "{\"kassenzeichennummer\":" . json_encode($this->kassenzeichen-
nummer) .
			",\"faelligkeitsdatum\":" . json_encode($this->faelligkeitsdatum)
.
			",\"waehrungskennzeichen\":" . json_encode($this->waehrungskenn-
zeichen) .
			",\"kennzeichenMahnverfahren\":" . json_encode($this->kennzeichen-
Mahnverfahren) .
			",\"transaktionsnummer\":" . json_encode($this->transaktionsnum-
mer) .
			",\"zahlverfahrencodes\":" . json_encode($this->zahlverfahren-
codes) .
			",\"buchungen\":" . $this->buchungenToJson($this->buchungen) .
			",\"beschreibung\":" . json_encode($this->beschreibung) .
			",\"kunde\":" . $this->kunde->toJson() .
			",\"buchungslistenparameter\":" . json_encode($this->buchungslis-
tenparameter) .
```

```
                    ",\"fachverfahrendaten\":" . json_encode($this->fachverfahrenda-
ten) .
                    ",\"zahltyp\":" . json_encode($this->zahltyp) .
                    ",\"betrag\":" . json_encode($this->betrag) . "}";
             return $out;
      }

      private function buchungenToJson($buchungen) {
             $out = "[";
             if ($buchungen == null) {
                    $out = "null";
             }
             else {
                    foreach ($buchungen as $buchung) {
                           if ($out !== "[") {
                                  $out = $out = $out . ",";
                           }
                           $out = $out . $buchung->toJson();
                    }
                    $out = $out . "]";
             }
             return $out;
      }

}

class Buchung {
      private $bruttobetrag;
      private $nettobetrag;
      private $id;
      private $steuersatz;
      private $steuerbetrag;
      private $buchungstext;
      private $kontierung;

      public function __construct(){
        $this->init();
    }
      private function init() {
      }

      public function setBruttobetrag($bruttobetrag) {
             $this->bruttobetrag = $bruttobetrag;
      }
      public function setNettobetrag($nettobetrag) {
             $this->nettobetrag = $nettobetrag;
      }
      public function setId($id) {
             $this->id = $id;
      }
      public function setSteuersatz($steuersatz) {
             $this->steuersatz = $steuersatz;
      }
      public function setSteuerbetrag($steuerbetrag) {
             $this->steuerbetrag = $steuerbetrag;
      }
```

```
	public function setBuchungstext($buchungstext) {
		$this->buchungstext = $buchungstext;
	}
	public function setKontierung($kontierung) {
		$this->kontierung = $kontierung;
	}

	public function toJson() {
		$out = "{\"bruttobetrag\":" . json_encode($this->bruttobetrag) .
			",\"nettobetrag\":" . json_encode($this->nettobetrag) .
			",\"id\":" . json_encode($this->id) .
			",\"steuersatz\":" . json_encode($this->steuersatz) .
			",\"steuerbetrag\":" . json_encode($this->steuerbetrag) .
			",\"buchungstext\":" . json_encode($this->buchungstext) .
			",\"kontierung\":" . json_encode($this->kontierung) . "}";
		return $out;
	}

}

class Kunde {
	private $name;
	private $vorname;
	private $typ;
	private $firmenkunde;
	private $firmenname;
	private $kundennummer;
	private $anrede;
	private $emailadresse;
	private $sepamandat;
	private $adresse;
	private $bankverbindung;

	public function __construct(){
	  $this->init();
    }
	private function init() {
	}

	public function setName($name) {
		$this->name = $name;
	}
	public function setVorname($vorname) {
		$this->vorname = $vorname;
	}
	public function setTyp($typ) {
		$this->typ = $typ;
	}
	public function setFirmenkunde($firmenkunde) {
		$this->firmenkunde = $firmenkunde;
	}
	public function setFirmenname($firmenname) {
		$this->firmenname = $firmenname;
	}
	public function setKundennummer($kundennummer) {
		$this->kundennummer = $kundennummer;
```

```
	}
	public function setAnrede($anrede) {
		$this->anrede = $anrede;
	}
	public function setEmailadresse($emailadresse) {
		$this->emailadresse = $emailadresse;
	}
	public function setSepamandat($sepamandat) {
		$this->sepamandat = $sepamandat;
	}
	public function setAdresse($adresse) {
		$this->adresse = $adresse;
	}
	public function setBankverbindung($bankverbindung) {
		$this->bankverbindung = $bankverbindung;
	}

	public function toJson() {
		$out = "{\"name\":" . json_encode($this->name) .
			",\"vorname\":" . json_encode($this->vorname) .
			",\"typ\":" . json_encode($this->typ) .
			",\"firmenkunde\":" . json_encode($this->firmenkunde) .
			",\"firmenname\":" . json_encode($this->firmenname) .
			",\"kundennummer\":" . json_encode($this->kundennummer) .
			",\"anrede\":" . json_encode($this->anrede) .
			",\"emailadresse\":" . json_encode($this->emailadresse) .
			",\"sepamandat\":" . $this->toJsonSubobject($this->sepamandat) .
			",\"adresse\":" . $this->toJsonSubobject($this->adresse) .
			",\"bankverbindung\":" . $this->toJsonSubobject($this->bankverb-
indung) . "}";
		return $out;
	}
	private function toJsonSubobject($obj) {
		if ($obj == null) {
			return "null";
		}
		else {
			return $obj->toJson();
		}
	}
}

class Adresse {
	private $plz;
	private $ort;
	private $strasse;
	private $hausnummer;
	private $land;
	private $postfach ;

	public function __construct(){
	  $this->init();
    }
	private function init() {
	}
```

```
	public function setPlz($plz) {
		$this->plz = $plz;
	}
	public function setOrt($ort) {
		$this->ort = $ort;
	}
	public function setStrasse($strasse) {
		$this->strasse = $strasse;
	}
	public function setHausnummer($hausnummer) {
		$this->hausnummer = $hausnummer;
	}
	public function setLand($land) {
		$this->land = $land;
	}
	public function setPostfach($postfach) {
		$this->postfach = $postfach;
	}

	public function toJson() {
		$out = "{\"plz\":" . json_encode($this->plz) .
			",\"ort\":" . json_encode($this->ort) .
			",\"strasse\":" . json_encode($this->strasse) .
			",\"hausnummer\":" . json_encode($this->hausnummer) .
			",\"land\":" . json_encode($this->land) .
			",\"postfach\":" . json_encode($this->postfach) . "}";
		return $out;
	}
}

class Bankverbindung {
	private $kontoinhaber;
	private $iban;
	private $bic;

		public function __construct(){
	  $this->init();
    }
	private function init() {
	}

	public function setKontoinhaber($kontoinhaber) {
		$this->kontoinhaber = $kontoinhaber;
	}
	public function setIban($iban) {
		$this->iban = $iban;
	}
	public function setBic($bic) {
		$this->bic = $bic;
	}

	public function toJson() {
		$out = "{\"kontoinhaber\":" . json_encode($this->kontoinhaber) .
			",\"iban\":" . json_encode($this->iban) .
			",\"bic\":" . json_encode($this->bic) . "}";
		return $out;
```

```
      }

}

abstract class Anrede {
    const AN = "AN";
    const EHELEUTE = "EHELEUTE";
    const FIRMA = "FIRMA";
    const FRAU = "FRAU";
    const FRAU_UND_HERR = "FRAU_UND_HERR";
    const FRAUEN = "FRAUEN";
    const FRAEULEIN = "FRAEULEIN";
    const HERR = "HERR";
    const HERREN = "HERREN";
    const HERRN_UND_FRAU = "HERRN_UND_FRAU";
}

abstract class KundeTyp {
    const TEMPORAER = "TEMPORAER";
      const BESTAND = "BESTAND";
}

abstract class SepaMandatSequenceType {
    const FRST = "FRST"; // Erstlastschrift
    const RCUR = "RCUR"; // Folgelastschrift
    const OOFF = "OOFF"; // Einmallastschrift
    const FNAL = "FNAL";  // letzte Lastschrift
}

abstract class SepaMandatType {
      const B = "B"; // Mandatstyp fuer SDD-Core Basis
      const F = "F"; // Mandatstyp fuer B2B-SDD
      const V = "V"; // Mandatstyp fuer SDD-Core-Basis mit verkuerzter Vorlagefrist
}

abstract class Zahltyp{
    const DIREKT = "DIREKT";
    const RECHNUNG = "RECHNUNG";
    const PAYPAGE = "PAYPAGE";

}

abstract class BuchungKontierungKeys {
    const HAUSHALTSTELLE = "haushaltstelle";
    const OBJEKTNUMMER = "objektnummer";
    const HREF = "href";
}

abstract class BuchungslisteFvDatenKeys {
    const SUCCESSURL = "successUrl";
    const CANCELURL = "cancelUrl";
    const ERRORURL = "errorUrl";
}

abstract class KassenzeichenZusatzdatenKeys {
    const PAYPAGESTATUS = "paypagestatus";
```

```
    const SALDO = "saldo";
    const AKTIVIERUNGSZEIT = "aktivierungszeit";
}

abstract class ZahlverfahrenCodes {
    const UEBERWEISUNGVOR = "UEBERWEISUNGVOR";
    const UEBERWEISUNGNACH = "UEBERWEISUNGNACH";
    const KREDITKARTE = "KREDITKARTE";
    const GIROPAY = "GIROPAY";
    const SEPASDD = "SEPASDD";
    const PAYPAL = "PAYPAL";
    const PAYPDIREKT = "PAYDIREKT";
    const BARZAHLUNG = "BARZAHLUNG";
    const TERMINALZAHLUNG = "TERMINALZAHLUNG";
    const LASTSCHRIFTOHNE = "LASTSCHRIFTOHNE";
}


?>
```

7.3 Konnektor-Client-Bibliothek (C#)

Es wird eine .NET-Bibliothek bereitgestellt, die von der Fachanwendung genutzt werden kann, um mit dem Konnektor zu kommunizieren. Die C#-Bibliothek

Die C#.Net-Bibliothek orientiert sich komplett an der Implementation der Java-Bibliothek. Es existieren die gleichen Methoden, mit den gleichen Parametern.

Abweichend zur Java-Implementation wird auch jede Server-Response, die nicht ok ist, also keinen 2xx Status aufweist, per Exception signalisiert. Diese spezielle Exception beinhaltet dann die aufbereitete Response.

Die in im Datenmodell beschriebene Enums sind in C# als Strings dargestellt, wobei es aber entsprechende Klassen mit Konstanten gibt, aus denen die zulässigen Werte entnommen werden können.

Die Bibliothek bietet verschiedene Methoden zum Aufruf der REST-Schnittstelle an.

new EpayBL-RestClient()	Konstruktor der Klasse `EPayBLRestClient`		
	Parameter	`$ePayBLRestClient-Configuration`	Speichert die Konfiguration der Kommunikation mit dem Konnektor (Mandant, Bewirtschafter, Serveradresse, Endpunkt, Keystore)
	Return	Objekt der Klasse `EpayBL-RestClient`	
Abfragen-Zahlverfahren	Ermittelt für einen Mandanten die aktiven Zahlverfahren		
	Return	Gibt ein Objekt vom Typ `ZahlverfahrenListErgebnis` zurück	
Uebertragen-Buchungsliste	Übertragt eine Buchungsliste an die ePayBL		
	Parameter	`$buchungsliste`	Buchungsliste (Objekt vom Typ `Buchungsliste`)
	Return	Gibt ein Objekt vom Typ `ZahlvorgangsErgebnis` zurück	
AbfragenKassenzeichen-status	Fragt den Status eines Kassenzeichens ab		
	Parameter	`$kassenzeichennummer`	Kassenzeichen (String)
	Return	Gibt ein Objekt vom Typ `KassenzeichenstatusErgebnis` zurück	
Aktivieren-Kassenzeichen	Aktiviert ein Kassenzeichen, d.h. setzt den Paypagestatus auf „INAKTIV“		
	Parameter	`$kassenzeichennummer`	Kassenzeichen (String)
	Return	Gibt ein Objekt vom Typ `KassenzeichenstatusErgebnis` zurück	
AnlegenSepaMandat	Legt im Fall einer internen Mandatsverwaltung ein SEPA-Mandat an		
	Parameter	`$sepaMandatInternAnlege-daten`	SEPA-Mandatsdaten (Objekt vom Typ `SepaMandatIn-ternAnlegedaten`)

	Return	Gibt ein Objekt vom Typ `SepamandatErgebnis` zurück	
vervollsta-endigenSe-pamandat	Vervollständigt im Fall einer internen Mandatsverwaltung ein SEPA-Mandat		
	Parameter	`$sepamandatsreferenz`	SEPA-Mandatsreferenz (String)
		`$sepamandatInternVer-vollstaendigungsdaten`	Daten zum Vervollständigen eines SEPA-Mandats (Objekt vom Typ `SepamandatInternVervollstaendigungs-daten`)
	Return	Gibt ein Objekt vom Typ `SepamandatErgebnis` zurück	
Loeschen-Kunde	Löscht einen Kunden in der ePayBL		
	Parameter	`$kundennummer`	Kundennummer (String)
	Return	Gibt ein Objekt vom Typ `ErgebnisErgebnis` zurück	

Tabelle 29: Verfügbare Funktionen an der Konnektor-Client-Bibliothek für C#

Auch hier können Proxies

- `EPayBLRestClientConfiguration WithUseWithProxyHost(string proxyHost)`
- `EPayBLRestClientConfiguration WithUseWithProxy(string proxyHost, int proxyPort)`
- `EPayBLRestClientConfiguration WithUseWithProxyCredentials(string proxyUser, string proxyPwd)`

und Keystores konfiguriert werden

- `EPayBLRestClientConfiguration withKeystoreFilename(string keystoreFilename)`
- `EPayBLRestClientConfiguration WithKeystorePassword(string keystorePassword`

Die Beschreibung der kompletten API liegt als DoxyGen-Dokument vor.

Es wird die externe Bibliothek Newtonsoft.Json.dll verwendet.

Der folgende Code zeigt die Benutzung der Bibliothek.

```
using De.Eg.Epaybl.Restclient.Beans;
using De.Eg.Epaybl.Restclient.Beans.Constants;
using De.Eg.Epaybl.Restclient.Beans.Enums;
using De.Eg.Epaybl.Restclient.Beans.Ergebnisse;
using De.Eg.Epaybl.Restclient.Beans.RequestBodies;
using De.Eg.Epaybl.Restclient.Beans.Sepa;
using De.Eg.Epaybl.Restclient.Beans.Subobjects;
using De.Eg.Epaybl.Restclient.Client;
using De.Eg.Epaybl.Restclient.Client.ClientExceptions;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Globalization;

namespace De.Eg.Epaybl.RestclientTest
{
    class Program
    {

        internal const string ENDPOINT_URI = "http://epayment244:8090";
        internal const string ENDPOINT_APPNAME = "konnektor";

        internal const string MANDANTNUMMER = "RE00001";
        internal const string BEWIRTSCHAFTERNUMMER = "ReB000";

        static void Main(string[] args)
        {
            TestZV();
            TestBLUebertragen();
            TestKundeLoeschen();

        }

        static void TestZV() {
            try {
```

```
                EPayBLRestClientConfiguration ePayBLRestClientConfiguration = new EPay-
BLRestClientConfiguration(ENDPOINT_URI, MANDANTNUMMER, BEWIRTSCHAFTERNUMMER).WithEnd-
pointAppName(ENDPOINT_APPNAME);
                EPayBLRestClient ePayBLRestClient = new EPayBLRestClient(ePayBLRestCli-
entConfiguration);

                ZahlverfahrenListErgebnis zahlverfahrenListErgebnis = ePayBLRestClient.Ab-
fragenZahlverfahren();
                Debug.WriteLine("AbfragenZahlverfahren: Response:" + zahlverfahrenListEr-
gebnis.ToString());
            }
            catch (EPayBLRestClientErrorResponseException e)
            {
                Debug.WriteLine("EPayBLRestClientErrorResponseException: " + e.Message);
                if (e.EPayBLRestClientErgebnis.IsBodyJson) {
                    ZahlverfahrenListErgebnis zahlverfahrenListErgebnis = e.EPayBLRestCli-
entErgebnis.AsObject<ZahlverfahrenListErgebnis>();
                    Debug.WriteLine("AbfragenZahlverfahren: Response:" + zahlverfahrenLis-
tErgebnis.ToString());
                }

                Debug.WriteLine("EPayBLRestClientErrorResponseException: " + e.Message +
", Code:" + e.EPayBLRestClientErgebnis.Body);
                Debug.WriteLine("EPayBLRestClientErrorResponseException: " + e.Message +
", Code:" + e.EPayBLRestClientErgebnis.HttpStatusCodeAsInt);
                throw;
            }
            catch (EPayBLRestClientException e)
            {
                Debug.WriteLine("EPayBLRestClientException: " + e.Message);
                throw;
            }
            catch (Exception e)
            {
                Debug.WriteLine("Exception: " + e.Message);
                throw;
            }
        }

        static void TestBLUebertragen() {
            try {

                Buchungsliste buchungsliste = new Buchungsliste();

                buchungsliste.faelligkeitsdatum =
                DateTime.ParseExact("25.12.2018 16:13:34", "dd.MM.yyyy HH:mm:ss", Cul-
tureInfo.InvariantCulture);

                buchungsliste.kennzeichenMahnverfahren = "23";
                buchungsliste.zahltyp = Zahltyp.DIREKT;
                buchungsliste.betrag = 10;
                buchungsliste.beschreibung = "Buchungsliste mit Buchungen: 1";
                //buchungsliste.zahlverfahrencodes =  new List<string> { Zahlverfahren-
Codes.KREDITKARTE };
                buchungsliste.zahlverfahrencodes = new List<string> { Zahlverfahren-
Codes.SEPASDD };
```

```
                Dictionary<string, string> fachverfahrendaten = new Dictionary<string,
string>();
                fachverfahrendaten[BuchungslisteFvDatenKeys.SUCCESSURL] =
"http://www.heise.de";
                fachverfahrendaten[BuchungslisteFvDatenKeys.CANCELURL] =
"http://www.google.de";
                fachverfahrendaten[BuchungslisteFvDatenKeys.ERRORURL] = "http://www.ama-
zon.de";

                buchungsliste.fachverfahrendaten = fachverfahrendaten;

                List<Buchung> buchungen = new List<Buchung>();

                Buchung buchung = new Buchung();
                buchung.bruttobetrag = 10;
                buchung.nettobetrag = 9;
                buchung.steuerbetrag = 1;
                buchung.buchungstext = "Buchung 1";

                Dictionary<string, string> kontierung = new Dictionary<string, string>();
                kontierung[BuchungKontierungKeys.HAUSHALTSTELLE] = "1100";
                kontierung[BuchungKontierungKeys.OBJEKTNUMMER] = "10000";
                kontierung[BuchungKontierungKeys.HREF] = "HREF";

                buchung.kontierung = kontierung;

                buchungen.Add(buchung);

                buchungsliste.buchungen = buchungen;

                Kunde kunde = new Kunde();
                kunde.kundennummer = "k242018";
                kunde.typ = KundeTyp.BESTAND;
                kunde.name = "Peters";
                kunde.anrede = Anrede.HERR;

                SepamandatExtern sepamandatExtern = new SepamandatExtern();
                sepamandatExtern.sequenceType = SepaMandatSequenceType.OOFF;
                sepamandatExtern.datumUnterschrift = DateTime.ParseExact("16.05.2017
12:00:00", "dd.MM.yyyy HH:mm:ss", CultureInfo.InvariantCulture);
                sepamandatExtern.datumLetzteNutzung = DateTime.ParseExact("28.08.2017
14:21:00", "dd.MM.yyyy HH:mm:ss",
                CultureInfo.InvariantCulture);
                sepamandatExtern.type = SepaMandatType.V;
                sepamandatExtern.mandatreferenz = "MR20170911165701";
                sepamandatExtern.kundennummer = "k242017";
                sepamandatExtern.glaeubigerId = "DE17HKS00000032546";

                kunde.sepamandat = sepamandatExtern;


                Bankverbindung bankverbindung = new Bankverbindung();
                bankverbindung.bic = "TESTDETT421";
                bankverbindung.iban = "DE62370205000000102030";
                bankverbindung.kontoinhaber = "Gerd Tester";
                kunde.bankverbindung = bankverbindung;

                buchungsliste.kunde = kunde;
```

```
                EPayBLRestClientConfiguration ePayBLRestClientConfiguration = new EPay-
BLRestClientConfiguration(ENDPOINT_URI, MANDANTNUMMER, BEWIRTSCHAFTERNUMMER).WithEnd-
pointAppName(ENDPOINT_APPNAME);

                EPayBLRestClient ePayBLRestClient = new EPayBLRestClient(ePayBLRestCli-
entConfiguration);

                ZahlvorgangsErgebnis zahlvorgangsErgebnis = ePayBLRestClient.Uebertragen-
Buchungsliste(buchungsliste);
                Debug.WriteLine("UebertragenBuchungsliste: Code:" + zahlvorgangsErgeb-
nis.ergebnis.rc);
                Debug.WriteLine("UebertragenBuchungsliste: Response:" + zahlvorgangsErgeb-
nis.ToString());
            }
            catch (EPayBLRestClientErrorResponseException e)
            {
                ZahlvorgangsErgebnis zahlvorgangsErgebnis = e.EPayBLRestClientErgebnis.As-
Object<ZahlvorgangsErgebnis>();

                Debug.WriteLine("EPayBLRestClientErrorResponseException: " + e.Message +
", Code:" + e.EPayBLRestClientErgebnis.HttpStatusCodeAsInt);
                throw;
            }
            catch (EPayBLRestClientException e)
            {
                Debug.WriteLine("EPayBLRestClientException: " + e.Message);
                throw;
            }
            catch (Exception e)
            {
                Debug.WriteLine("Exception: " + e.Message);
                throw;
            }
        }

        static void TestKundeLoeschen() {
            try {
                EPayBLRestClientConfiguration ePayBLRestClientConfiguration = new EPay-
BLRestClientConfiguration(ENDPOINT_URI, MANDANTNUMMER, BEWIRTSCHAFTERNUMMER).WithEnd-
pointAppName(ENDPOINT_APPNAME);
                EPayBLRestClient ePayBLRestClient = new EPayBLRestClient(ePayBLRestCli-
entConfiguration);

                ErgebnisErgebnis ergebnisErgebnis = ePayBLRestClient.LoeschenKunde(TD.KUN-
DENNUMMER);
                Debug.WriteLine("LoeschenKunde: Response:" + ergebnisErgebnis.ToString());

            } catch (EPayBLRestClientErrorResponseException e) {
                Debug.WriteLine("EPayBLRestClientErrorResponseException: " + e.Message);
                throw;
            } catch (EPayBLRestClientException e) {
                Debug.WriteLine("EPayBLRestClientException: " + e.Message);
                throw;
            } catch (Exception e) {
                Debug.WriteLine("Exception: " + e.Message);
                throw;
            }
```

```
        }

    }
}
```

Abbildung 21: Beispielapplikation für die Verwendung der C#-Bibliothek

8 Abbildungsverzeichnis

9 Tabellenverzeichnis

NOTIZEN