SOFT-LOGIC

Paylogic

Универсальный шлюз 3-й версии. Программное обеспечение «Процессинговый центр Pay-logic».

Руководство администратора

АННОТАЦИЯ

Описывает процесс настройки универсального шлюза 3-й версии программного обеспечения «Процессинговый центр Pay-logic»

Версия руководства: 3.0

Руководство актуально для кабинета «Процессингового центра Pay-logic» версий 5.9.x

ООО «Софт-Лоджик»

656006, г. Барнаул, Малахова ул., дом 146в

Тел: (3852) 72-27-27

ОГЛАВЛЕНИЕ

ЧАСТЬ 1. ВВЕДЕНИЕ И ПОДГОТОВКА 8
1.1 О ДОКУМЕНТЕ 8
1.1.1 НАЗНАЧЕНИЕ ДОКУМЕНТА 8
1.1.2 ИСТОРИЯ ИЗМЕНЕНИЙ 8
1.2 ДЛЯ КОГО ПРЕДНАЗНАЧЕНО РУКОВОДСТВО 9
1.2.1 ТРЕБОВАНИЯ К СПЕЦИАЛИСТУ 9
1.3 ОБЩАЯ ИНФОРМАЦИЯ О ШЛЮЗЕ 9
1.4 ПОДГОТОВКА РАБОЧЕГО ОКРУЖЕНИЯ 11
1.4.1 НЕОБХОДИМОЕ ПО 11
1.4.2 УСТАНОВКА INTELLIJ IDEA 11
1.4.3 НАСТРОЙКА JSON-СХЕМЫ ДЛЯ ВАЛИДАЦИИ 11
1.4.3.1 ПОШАГОВАЯ НАСТРОЙКА 12

ЧАСТЬ 2. КОНФИГУРАЦИОННЫЙ ФАЙЛ: БАЗОВЫЙ УРОВЕНЬ 14
2.1. СТРУКТУРА КОНФИГУРАЦИОННОГО ФАЙЛА 14
2.1.1. СОСТАВ ФАЙЛА 14
2.1.2. ОБЩАЯ СТРУКТУРА 14
2.2. СЕКЦИЯ SERVERS — ПОДКЛЮЧЕНИЕ К ПРОВАЙДЕРУ 15
2.2.1. НАЗНАЧЕНИЕ 15
2.2.2. ПАРАМЕТРЫ СЕКЦИИ SERVER 16
2.2.3. НАСТРОЙКИ ПРОКСИ (SERVERPROXY) 17
2.2.4. ПРИМЕР С РЕЗЕРВНЫМ СЕРВЕРОМ И ПРОКСИ 17
2.3. СЕКЦИЯ LOGGER — ЖУРНАЛ ШЛЮЗА 18
2.3.1. НАЗНАЧЕНИЕ 18
2.3.2. СИНТАКСИС 18
2.3.3. ПРИМЕР 18
2.4. СЕКЦИЯ CODES — КОДЫ ОТВЕТОВ ПРОВАЙДЕРА 18
2.4.1. НАЗНАЧЕНИЕ 18
2.4.2. УРОВНИ НАСТРОЙКИ 18
2.4.3. ТИПЫ ЗАПРОСОВ ДЛЯ НАСТРОЙКИ КОДОВ ОТВЕТОВ 19
2.4.4. СТАТУСЫ ПЛАТЕЖА 20
2.4.5. БАЗОВЫЙ СИНТАКСИС 20
2.4.6. ОБРАБОТКА НЕЧИСЛОВЫХ КОДОВ (MAPPING) 21
2.4.7. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ (REGEX-MAPPING) 21
2.4.8. КОДЫ ДЛЯ ADVANCED-ЗАПРОСОВ 22

2.5. СЕКЦИЯ REQUESTS — БАЗОВЫЕ НАСТРОЙКИ ЗАПРОСОВ 22
2.5.1. СПИСОК ТИПОВ ЗАПРОСОВ 22
2.5.2. УРОВНИ НАСТРОЙКИ ЗАПРОСОВ 23
2.5.3. СТРУКТУРА СЕКЦИИ REQUESTS 24
2.6. СЕКЦИЯ SETTINGS — ОБЩИЕ НАСТРОЙКИ ШЛЮЗА 24
2.6.1. ПЕРЕЧЕНЬ ПАРАМЕТРОВ 24
2.6.2. БАЗОВЫЕ ПАРАМЕТРЫ 27
2.6.3. ПРИМЕР НАСТРОЕК 28
2.6.4. ПРИМЕР МИНИМАЛЬНОЙ РАБОЧЕЙ КОНФИГУРАЦИИ 29

ЧАСТЬ 3. РАСШИРЕННАЯ НАСТРОЙКА ЗАПРОСОВ 30

3.1. СПРАВОЧНИК ПАРАМЕТРОВ ЗАПРОСОВ 30
3.1.1. ПАРАМЕТРЫ REQUEST 30
3.1.2. ПАРАМЕТРЫ УПРАВЛЕНИЯ ЗАПРОСОМ 32
3.1.3. ПАРАМЕТРЫ РАЗБОРА ОТВЕТА 33
3.1.4. ПАРАМЕТРЫ ПОСТ-ОБРАБОТКИ 34
3.2. ПАРАМЕТРЫ GET/POST-ЗАПРОСОВ (REQUEST-PARAMS) 35
3.2.1. НАЗНАЧЕНИЕ 35
3.2.2. ПЕРЕМЕННЫЕ ПЛАТЕЖА 35
3.2.3. ПРЕОБРАЗОВАНИЯ 37
3.2.4. АТРИБУТЫ БАНКОВСКИХ ОПЕРАЦИЙ 38
3.2.5. ПАРАМЕТРЫ ШЛЮЗА, VELOCITY, JS, ТОКЕН, РЕНДЕРЫ 39

ЧАСТЬ 4. ОБРАБОТКА ОТВЕТОВ ПРОВАЙДЕРА (RESULT-PARAMS) 41

4.1. ОБЩАЯ СХЕМА РАЗБОРА ОТВЕТОВ 41
4.2. СЕКЦИЯ PARAMS — ПРОСТЫЕ ПАРАМЕТРЫ 41
4.2.1. НАЗНАЧЕНИЕ 41
4.2.2. АТРИБУТЫ СЕКЦИИ PARAMS 42
4.2.3. ГРУППЫ ПУТЕЙ (КОНКАТЕНАЦИЯ) 42
4.2.4. СТАТИЧНЫЕ ЗНАЧЕНИЯ ПАРАМЕТРОВ 43
4.2.5. ЛОКАЛИЗАЦИЯ (BUNDLE-TITLE/BUNDLE-VALUE) 43
4.2.6. ПОСТ-ОБРАБОТКА (POST-PROCESS) 43
4.2.7. ЗНАЧЕНИЯ ИЗ HTTP-ЗАГОЛОВКОВ 44
4.2.8. ПРИМЕРЫ 44
4.3. СЕКЦИЯ SELECTOR — ВЫБОР ИЗ СПИСКА 44
4.3.1. НАЗНАЧЕНИЕ 45
4.3.2. АТРИБУТЫ СЕКЦИИ SELECTOR 45
4.3.3. ПРИМЕР: СПИСОК СЧЕТОВ С ФИЛЬТРОМ 46

Web: https://www.pay-logic.ru/
Mail: info@soft-logic.ru

4.3.4. ИЗМЕНЕНИЕ КОНТЕКСТА (CHANGE-CONTEXT-PATH) 47
4.4. СЕКЦИЯ COLLECTIONS — ИНДЕКСАЦИЯ МАССИВОВ 47
4.4.1. НАЗНАЧЕНИЕ 47
4.4.2. АТРИБУТЫ СЕКЦИИ COLLECTIONS 47
4.4.3. ПРИМЕР 48
4.5. СЕКЦИЯ PARAMS-ARRAY — МАССИВ ОПЦИОНАЛЬНЫХ ПАРАМЕТРОВ 48
4.5.1. НАЗНАЧЕНИЕ 49
4.5.2. АТРИБУТЫ СЕКЦИИ PARAMS-ARRAY 49
4.5.3. ПРИМЕР 49
4.6. СЕКЦИЯ PARAMS-FROM-ADVANCED — ВЫЗОВ ADVANCED-ФУНКЦИИ 50
4.6.1. НАЗНАЧЕНИЕ 50
4.6.2. АТРИБУТЫ СЕКЦИИ PARAMS-FROM-ADVANCED 50
4.6.3. ПРИМЕР С ФИЛЬТРОМ И ПРОБРОСОМ ОШИБКИ 50
4.7. СЕКЦИЯ CLIENT-FEE — КОМИССИЯ С КЛИЕНТА 51
4.7.1. НАЗНАЧЕНИЕ 51
4.7.2. ПАРАМЕТРЫ СЕКЦИИ CLIENT-FEE 51
4.7.3. ПРИМЕР 51
4.8. ВСПОМОГАТЕЛЬНЫЕ СЕКЦИИ 52
4.8.1. SAVE-ON-ERROR 52
4.8.2. CLEAR-PARAMS 52
4.8.3. UNKNOWN-PARAMS 52

ЧАСТЬ 5. УЗКОНАПРАВЛЕННЫЕ И СПЕЦИАЛЬНЫЕ НАСТРОЙКИ 53

5.1. ADVANCED-ЗАПРОСЫ 53
5.1.1. НАЗНАЧЕНИЕ 53
5.1.1. БАЗОВАЯ СТРУКТУРА 53
5.1.2. СПЕЦИАЛЬНЫЕ МЕТОДЫ ДЛЯ ADVANCED 54
5.1.3. ПОСЛЕДОВАТЕЛЬНЫЙ ВЫЗОВ (ADVANCED-SEQUENCES) 55
5.1.4. УСЛОВНЫЙ ВЫЗОВ (CONDITIONAL-SEQUENCES) 56
5.1.5. АСИНХРОННЫЕ ADVANCED-ЗАПРОСЫ 57
5.2. СПЕЦИАЛЬНЫЕ НАСТРОЙКИ ШЛЮЗА (SETTINGS) 58
5.2.1. УПРАВЛЕНИЕ ЛОГАМИ В КАБИНЕТЕ (LOGGER-SETTINGS) 58
5.2.2. СТАТУСЫ И ТАЙМАУТЫ 59
5.3. БЕЗОПАСНОСТЬ И КРИПТОГРАФИЯ 60
5.3.1. ЭЛЕКТРОННАЯ ПОДПИСЬ (SIGNER) 60
5.3.2. XML-ПОДПИСЬ (XML-SIGNER) 61
5.3.3. ТОКЕНЫ (TOKEN) 62
5.3.4. ШИФРОВАНИЕ (ENCRYPTOR) 63

5.4. ДИНАМИЧЕСКИЕ ДАННЫЕ В ЗАПРОСАХ 64
5.4.1. РЕНДЕРЫ (RENDER-LOADER) 64
5.4.2. ЛОКАЛИЗАЦИЯ 64
5.4.3. ПАРАМЕТРЫ ШЛЮЗА (PARAMS) 65
5.4.4. VELOCITY-ПАРАМЕТРЫ (VELOCITY-PARAMS) 65
5.4.5. JAVASCRIPT-ПАРАМЕТРЫ (JS-PARAMS) 66
5.5. РЕДАКТИРОВАНИЕ БАЗЫ ДАННЫХ ЧЕРЕЗ ШЛЮЗ (EDIT-DB) 66
5.5.1. НАЗНАЧЕНИЕ 66
5.5.2. АКТИВАЦИЯ ФУНКЦИОНАЛА 67
5.5.3. ИСПОЛЬЗОВАНИЕ SQL-ЗАПРОСОВ 67
5.5.4. ДОСТУПНЫЕ ПЕРЕМЕННЫЕ В SQL 68
5.5.5. БЕЗОПАСНОСТЬ 68
5.6. ВХОДЯЩИЕ CALLBACK-УВЕДОМЛЕНИЯ 68
5.6.1. НАЗНАЧЕНИЕ 68
5.6.2. КОНФИГУРАЦИЯ 69
5.6.3. БЕЗОПАСНОСТЬ: БЕЛЫЙ СПИСОК IP 69
5.6.4. КОДЫ ОТВЕТОВ ДЛЯ CALLBACK 69
5.7. CRON-ЗАДАЧИ 70
5.7.1. НАЗНАЧЕНИЕ 70
5.7.2. СПИСОК ПАРАМЕТРОВ 70
5.7.3. CRON-ВЫРАЖЕНИЕ 71
5.7.4. ПРИМЕРЫ 71
5.7.5. ТИПЫ РЕЗУЛЬТАТОВ 72
5.7.6. СОХРАНЕНИЕ РЕЗУЛЬТАТА 72

ЧАСТЬ 6. ИНСТРУМЕНТЫ И СРЕДА 73

6.1. VELOCITY-ШАБЛОНЫ 73
6.2. НАЗНАЧЕНИЕ 73
6.3. ОБЪЕКТЫ VELOCITY-ШАБЛОНОВ 73
6.3.1. ОБЪЕКТ PAYMENT 74
6.3.2. ПРИМЕР ШАБЛОНА (JSON) 75
6.3.3. ПРИМЕР ШАБЛОНА (XML) 76
6.4. ДОПОЛНИТЕЛЬНЫЕ СЕРВЕРЫ (ADDITIONAL-SERVERS) 76
6.4.1. НАЗНАЧЕНИЕ 76
6.4.2. СИНТАКСИС 76
6.4.3. ПРИМЕР ИСПОЛЬЗОВАНИЯ В ЗАПРОСЕ 77
6.5. ВНЕШНИЕ ПАРАМЕТРЫ (PARAM) 77
6.5.1. НАЗНАЧЕНИЕ 77

6.5.2. ФАЙЛ СВОЙСТВ 77
6.5.3. ПРИМЕР ИСПОЛЬЗОВАНИЯ В КОНФИГЕ 78
6.5.4. ПРИОРИТЕТ ПОДСТАНОВКИ 78

ЧАСТЬ 7. ЭКСПЛУАТАЦИЯ 79

7.1. ЗАПУСК ШЛЮЗА 79
7.1.1. РАЗМЕЩЕНИЕ ФАЙЛОВ 79
7.1.2. РЕГИСТРАЦИЯ В GATES.XML 79
7.1.3. ПЕРЕЗАПУСК 80
7.2. ТЕСТИРОВАНИЕ 80
7.2.1. ТЕСТОВАЯ УТИЛИТА 80
7.2.2. СТРУКТУРА ТЕСТОВОГО КОНФИГА 81
7.2.3. ПАРАМЕТР OFFSET-ID 81
7.2.4. ТИПЫ ШАГОВ 82
7.2.5. ЛОКАЛЬНЫЕ ОТВЕТЫ (LOCAL-RESPONSE) 83

ПРИЛОЖЕНИЯ 84

ПРИЛОЖЕНИЕ A. БЫСТРЫЙ СТАРТ: ЧЕК-ЛИСТ ЗА 5 МИНУТ 84
ПРИЛОЖЕНИЕ B. СПРАВОЧНИК ТИПОВ ЗАПРОСОВ 85
ПРИЛОЖЕНИЕ C. СПРАВОЧНИК МЕТОДОВ ЗАПРОСОВ 86
ПРИЛОЖЕНИЕ D. СПРАВОЧНИК СТАТУСОВ ПЛАТЕЖА 86
ПРИЛОЖЕНИЕ E. КОДЫ ОШИБОК ADVANCED-ЗАПРОСОВ 87
ПРИЛОЖЕНИЕ F. ШПАРГАЛКА: XPATH/JSON-PATH 88
ПРИЛОЖЕНИЕ G. ГЛОССАРИЙ 88

ЧАСТЬ 1. ВВЕДЕНИЕ И ПОДГОТОВКА

1.1 О ДОКУМЕНТЕ

1.1.1 НАЗНАЧЕНИЕ ДОКУМЕНТА

Данное руководство описывает процесс настройки универсального шлюза 3-й версии для программного обеспечения «Процессинговый центр Pay-logic» версии 5.x.x.

Шлюз написан на Java и является модулем ядра системы шлюзов. Он позволяет подключать внешние системы через YAML-конфигурацию без написания кода.

1.1.2 ИСТОРИЯ ИЗМЕНЕНИЙ

Версия	Дата публикации	Изменения
1.0	01.01.2025	Первая версия документа
1.1	22.09.2025	Актуализация под версию кабинета 5.8.x
3.0	17.02.2026	Полное соответствие JSON-схеме. Добавлены разделы: callback, cancel-status, chargeback, subscription, client-fee, edit-db, allowed-callback-ip-addresses. Исправлена структура servers (массив).

1.2 ДЛЯ КОГО ПРЕДНАЗНАЧЕНО РУКОВОДСТВО

1.2.1 ТРЕБОВАНИЯ К СПЕЦИАЛИСТУ

Настройку универсального шлюза выполняет технический специалист (системный администратор, интегратор, разработчик), который:

Обязательно:

- Имеет опыт установки и настройки шлюзов процессинга;
- Понимает форматы данных: YAML, JSON, XML;
- Работал с языками запросов: JSON-path, Xpath;
- Знает основы регулярных выражений.

Желательно:

- Знаком с шаблонизатором Velocity;
- Имеет базовые навыки программирования;
- Имеет опыт работы с Shell и Unix-командами (для Linux-систем).

1.3 ОБЩАЯ ИНФОРМАЦИЯ О ШЛЮЗЕ

Универсальный шлюз позволяет подключить большинство поставщиков услуг к процессингу.

Поддерживаемые операции:

- Онлайн- и оффлайн-платежи;
- Проверка реквизитов (в том числе онлайн с точек);

- Запрос баланса (для авансовых схем);
- Произвольные запросы через усовершенствованный обработчик (advanced);
- Передача параметров на клиентские устройства;
- Обработка входящих callback-уведомлений от провайдера;
- Создание абонентской платы на основе ответа провайдера;
- Выполнение SQL-запросов к базе данных.

Не поддерживается подключение с помощью универсального шлюза:

- Поставщики с нестандартной логикой обработки транзакций;
- Провайдеры с кастомными механизмами подписи, не реализованными в шлюзе;
- Протоколы, отличные от HTTP(S).

1.4 ПОДГОТОВКА РАБОЧЕГО ОКРУЖЕНИЯ

1.4.1 НЕОБХОДИМОЕ ПО

Необходимые компоненты ПО:

Компонент	Рекомендация
IDE	IntelliJ IDEA (Community/Ultimate)
Формат конфига	YAML (.yml, .yaml)
Валидация	JSON Schema (поставляется с модулем)

1.4.2 УСТАНОВКА INTELLIJ IDEA

1. Перейдите на официальный сайт https://www.jetbrains.com/idea/;
2. Выберите версию под вашу ОС (Windows, Linux, macOS);
3. Скачайте и запустите установщик;
4. Следуйте инструкциям установщика.

1.4.3 НАСТРОЙКА JSON-СХЕМЫ ДЛЯ ВАЛИДАЦИИ

Что такое JSON-схема?

Это декларативный документ, который описывает допустимую структуру конфигурационного файла. Схема:

- Проверяет обязательные поля;
- Подсказывает доступные параметры;
- Предупреждает об ошибках.

Где взять схему?

Файл *universal-gate-v3-schema.json* передаётся клиенту вместе с модулем шлюза.

1.4.3.1 ПОШАГОВАЯ НАСТРОЙКА

Шаг 1. Создайте проект

1. Откройте *IntelliJ IDEA*;
2. Нажмите *New Project*;
3. Выберите *Java* → *Next* → *Next*;
4. Укажите название проекта и путь до папки, где будет лежать конфиг;
5. Нажмите *Finish*.

Шаг 2. Добавьте JSON-схему в проект

1. Скопируйте файл схемы в папку проекта;
2. В меню IDEA выберите: *IntelliJ IDEA* → *Settings* (Windows: *File* → *Settings*);
3. Перейдите в раздел: *Languages & Frameworks* → *Schemas and DTDs* → *JSON Schema Mappings*;
4. Нажмите кнопку + (добавить схему);
5. В поле *Schema file or URL* укажите путь к файлу схемы;
6. В поле *Name* задайте название (например, Universal Gate v3);

7. Нажмите *OK*.

Шаг 3. Привяжите схему к конфигурационному файлу

Способ А — при добавлении схемы:

1. В окне *JSON Schema Mappings* нажмите *+* под полем *Schema version*;
2. Выберите файл(ы), которые нужно валидировать.

Способ Б — при работе с файлом:

1. Создайте файл *gate.yml* в проекте;
2. В правом нижнем углу IDEA найдите надпись *No JSON schema*;
3. Нажмите на неё и выберите из списка вашу схему.

Результат:

- При редактировании YAML работают подсказки (по сочетанию клавиш Ctrl+Space);
- Схема подсвечивает ошибки и незаполненные обязательные поля;
- Можно открыть файл схемы и посмотреть допустимые параметры;

ЧАСТЬ 2. КОНФИГУРАЦИОННЫЙ ФАЙЛ: БАЗОВЫЙ УРОВЕНЬ

2.1. СТРУКТУРА КОНФИГУРАЦИОННОГО ФАЙЛА

2.1.1. СОСТАВ ФАЙЛА

Секции, входящие в состав файла:

Секция	Описание	Обязательность
logger	Имя журнала шлюза	Да
servers	Массив серверов провайдера (один или несколько)	Да
codes	Настройки кодов ответов	Да
requests	Настройки запросов	Да
settings	Общие настройки шлюза	Нет
additional-servers	Карта дополнительных серверов (псевдоним → массив)	Нет

2.1.2. ОБЩАЯ СТРУКТУРА

```
servers:
  - url: "https://api.provider.com:443/"
```

```
logger: "gate_name"

codes:
  common:
    success: [0]
    error: [1, 2]

requests:
  common:
    default:
      headers:
        Content-Type: "application/json"
  check:
    default:
      url: "/check"
      request-body: "check.vsl"

settings:
  method: "BODY"
  response-type: "JSON"
```

2.2. СЕКЦИЯ SERVERS — ПОДКЛЮЧЕНИЕ К ПРОВАЙДЕРУ

2.2.1. НАЗНАЧЕНИЕ

В соответствии с JSON-схемой, секция **`servers`** является массивом. Это позволяет указывать несколько серверов для одного провайдера (основной, резервный, балансировка нагрузки).

Правильный синтаксис:

```
servers:
  - url: "https://api.provider.com:443/"
    timeout: 30
    max-conn: 10
  - url: "https://backup.provider.com:443/"
    timeout: 60
    max-conn: 5
```

2.2.2. ПАРАМЕТРЫ СЕКЦИИ SERVER

Доступные параметры секции SERVERS:

Параметр	Описание	Тип	Обяз.	По умолч.
url	Адрес сервера (без пути)	Строка	Да	—
timeout	Таймаут подключения (сек)	Число	Нет	60
protocols	Список протоколов (например, ["TLSv1.3"])	Массив	Нет	Системные
ciphers	Массив шифров	Массив	Нет	Системные
proxy	Настройки прокси	Объект	Нет	—
max-conn	Макс. одновременных подключений	Целое	Нет	5
keystore-path	Файл хранилища ключей (PKCS12/PFX)	Строка	Нет	—
keystore-password	Пароль хранилища	Строка	Нет	—
auth-basic-user	Логин для Basic-авторизации	Строка	Нет	—
auth-basic-password	Пароль для Basic-авторизации	Строка	Нет	—

2.2.3. НАСТРОЙКИ ПРОКСИ (SERVERPROXY)

Параметры настройки:

Параметр	Описание	Тип	Обяз.
host	Адрес прокси-сервера (IP или домен)	Строка	Да
port	Порт прокси-сервера	Целое	Да
user	Имя пользователя для авторизации	Строка	Нет
password	Пароль пользователя	Строка	Нет

2.2.4. ПРИМЕР С РЕЗЕРВНЫМ СЕРВЕРОМ И ПРОКСИ

```
servers:
  - url: "https://primary.provider.com"
    timeout: 30
    auth-basic-user: "api_user"
    auth-basic-password: "securepass"
    max-conn: 10
  - url: "https://backup.provider.com"
    timeout: 60
    auth-basic-user: "api_user"
    auth-basic-password: "securepass"
    proxy:
      host: "proxy.company.local"
      port: 3128
      user: "proxy_user"
      password: "proxy_pass"
```

2.3. СЕКЦИЯ LOGGER — ЖУРНАЛ ШЛЮЗА

2.3.1. НАЗНАЧЕНИЕ

Определяет имя журнала, в который шлюз будет писать события.

2.3.2. СИНТАКСИС

```
logger: "имя_журнала"
```

Путь к лог-файлам: **<gate_folder>/log/gates/<имя_журнала>/**

2.3.3. ПРИМЕР

```
logger: "universal_v3_provider_1690"
```

Рекомендация: используйте уникальное имя для каждого шлюза, чтобы логи не смешивались.

2.4. СЕКЦИЯ CODES — КОДЫ ОТВЕТОВ ПРОВАЙДЕРА

2.4.1. НАЗНАЧЕНИЕ

Секция **codes** устанавливает соответствие между числовыми кодами в ответе провайдера и статусами платежа в процессинге.

2.4.2. УРОВНИ НАСТРОЙКИ

Уровни настройки:

Уровень	Приоритет	Описание
common	Низкий	Действует для всех запросов по умолчанию
check/pay/и т.д.	Высокий	Только для конкретного типа запроса

2.4.3. ТИПЫ ЗАПРОСОВ ДЛЯ НАСТРОЙКИ КОДОВ ОТВЕТОВ

Типы запросов для настройки:

Тип запроса	Описание
common	Коды по умолчанию для всех запросов
check	Проверка реквизитов
pay	Проведение платежа
status	Запрос статуса
confirm	Подтверждение платежа
cancel	Отмена платежа
cancel-status	Статус отмены
advanced	Произвольные функции
balance	Запрос баланса

Тип запроса	Описание
callback	Входящие уведомления от провайдера

2.4.4. СТАТУСЫ ПЛАТЕЖА

Возможные статусы платежа:

Ключ	Статус платежа	Применение
success	Успех	Платеж проведен
error	Ошибка (нефинальная)	Требуется повтор
unknown	Проведение — Неизвестен	Будет уточняться
process	Проводится	В обработке
confirm	Требуется подтверждение	Ожидает confirm
token	Ошибка токена	Токен недействителен
chargeback	Возврат / опровержение	Платеж оспорен
callback	Callback-уведомление	Для входящих запросов

2.4.5. БАЗОВЫЙ СИНТАКСИС

```
codes:
  common:
    success: [0]
```

```
    error: [2, 3, -10001]
    unknown: [-10000]
  pay:
    success: [200]
    error: [400, 500]
```

2.4.6. ОБРАБОТКА НЕЧИСЛОВЫХ КОДОВ (MAPPING)

Если провайдер возвращает строковые коды или диапазоны, используйте **`mapping`**:

```
codes:
  common:
    success: [0]
    mapping:
      "ok": 0
      "TransactionIdAlreadyExists": 1
      "InvalidClientIdOrSign": 2
  pay:
    mapping:
      "0-3": 0
      "4-10": 1
```

Важно! **`mapping`** и **`regex-mapping`** могут быть использованы на любом уровне (common, pay, advanced и т.д.), не только в common.

2.4.7. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ (REGEX-MAPPING)

```
codes:
  common:
    regex-mapping:
      "(?s).*(Token|token|is broken|is closed).*": 401
  advanced:
    regex-mapping:
      ".*timeout.*": 504
      ".*database.*": 500
```

2.4.8. КОДЫ ДЛЯ ADVANCED-ЗАПРОСОВ

Для advanced-запросов в ядре определены собственные коды, которые отправляются на точку. Они не заменяют настройки в **codes**, а транслируются в соответствии с таблицей:

Код ядра	Значение	Условие
0	SUCCESS	Успех
1	NOT_FOUND	Абонент не найден
3	PROVIDER_ANSWER_ERROR	Невозможно разобрать ответ провайдера
5	NO_DEBTS	Задолженность отсутствует
9	TEXT_ERROR	Текстовая ошибка (содержит message)

2.5. СЕКЦИЯ REQUESTS — БАЗОВЫЕ НАСТРОЙКИ ЗАПРОСОВ

2.5.1. СПИСОК ТИПОВ ЗАПРОСОВ

Типы запросов:

Тип запроса	Описание
common	Общие настройки по умолчанию
check	Проверка реквизитов
pay	Платёж

Тип запроса	Описание
status	Статус
confirm	Подтверждение
cancel	Отмена
cancel-status	Статус отмены
balance	Баланс
advanced	Advanced-функции
callback	Обработка входящих callback-уведомлений

2.5.2. УРОВНИ НАСТРОЙКИ ЗАПРОСОВ

Уровни запросов:

Уровень	Где задаётся	Приоритет
Общий (common)	**requests.common.default**	Низкий
Тип запроса	**requests.check.default** и т.д.	Средний
Сервисный	**requests.pay.1234** (ID сервиса)	Высокий

2.5.3. СТРУКТУРА СЕКЦИИ REQUESTS

```
requests:
  common:
    default:
      headers:
        Content-Type: "application/json"
      result-path: "$.code"

  check:
    default:
      url: "/subscriber"
      request-body: "check.vsl"
    1234:
      url: "/subscriber/vip"
      request-body: "check_vip.vsl"

  pay:
    default:
      url: "/payment"
      request-body: "pay.vsl"

  callback:
    default:
      method: "POST"
      response-type: "XML"
      result-path: "/notification/status"
```

2.6. СЕКЦИЯ SETTINGS — ОБЩИЕ НАСТРОЙКИ ШЛЮЗА

2.6.1. ПЕРЕЧЕНЬ ПАРАМЕТРОВ

Параметры настройки:

Параметр	Тип	Описание	По умолч.
method	enum	POST, PUT, GET, DELETE, PATCH, BODY, RMI, BLANK, SQL	BODY

Параметр	Тип	Описание	По умолч.
timeout	integer	Таймаут очереди (сек)	20
params	object	Статичные параметры (**$gateParams**)	—
localization	object	Настройки локализации	—
token	object	Настройки токена	—
editDbEnabled	boolean	Включение поддержки SQL-запросов через шлюз	false
edit-db-password	string	Пароль для активации функционала edit-db	—
response-charset	string	Кодировка ответа	UTF-8
request-charset	string	Кодировка запроса	UTF-8
response-type	enum	XML, JSON, REGEX, HTTPCODE	XML
request-date-format	string	Формат даты в запросе	yyyyMMddHHmmss
response-date-format	string	Формат даты в ответе	yyyy-MM-dd'T'HH:mm:ss
render-loader	boolean	Включить рендеры	false
balance-in-penny	boolean	Баланс в копейках	false

Параметр	Тип	Описание	По умолч.
thread-count	integer	Количество потоков	1
status-using-pay	boolean	Использовать pay для статуса	false
separate-confirm	boolean	Отдельный confirm-запрос	false
only-online-verify	boolean	Проверка только онлайн	false
basic-auth-in-header	boolean	Basic auth в заголовок	false
disable-follow-redirects	boolean	Не следовать редиректам	false
timeout-confirm	integer	Таймаут подтверждения (сек)	10
timeout-process	integer	Таймаут "Проводится" (сек)	60
timeout-unknown	integer	Таймаут "Неизвестен" (сек)	300
timezone	string	Часовой пояс	Время сервера
allow-empty-params	boolean	Разрешить пустые параметры	false
js-params	object	JavaScript-параметры	—

Параметр	Тип	Описание	По умолч.
velocity-params	object	Velocity-параметры	—
signer	object	Электронная подпись	—
xml-signer	object	XML-подпись	—
encryptor	object	Шифрование	—
advanced-sequences	object	Последовательности advanced	—
conditional-sequences	object	Условные последовательности	—
logger-settings	object	Настройки логов в кабинете	—
cron-jobs	array	Cron-задачи	—
allowed-callback-ip-addresses	array	Белый список IP для callback	—

2.6.2. БАЗОВЫЕ ПАРАМЕТРЫ

method

Метод отправки запроса по умолчанию. Может быть переопределён в конкретном запросе.

Доступные методы:

- **BODY** — запрос с телом (Velocity-шаблон);
- **GET, POST, PUT, DELETE, PATCH** — HTTP-методы;
- **RMI** — вызов другого шлюза;
- **BLANK** — без обращения к провайдеру;
- **SQL** — выполнение SQL-запроса к БД.

timeout
Задержка выборки платежей из очереди в секундах. Для высоконагруженных шлюзов рекомендуется уменьшить до 1-5 секунд.

request-date-format / timezone
Формат даты и времени в запросах. Используется при подстановке **#date#**, **#time#** в request-params.
Формат задаётся согласно спецификации SimpleDateFormat (https://docs.oracle.com/javase/6/docs/api/java/text/SimpleDateFormat.html).

response-type
Тип ответа провайдера по умолчанию:

- **XML** — парсинг через Xpath;
- **JSON** — парсинг через JSON-path;
- **REGEX** — извлечение кода через регулярное выражение;
- **HTTPCODE** — использовать HTTP-статус ответа как код результата.

thread-count
Количество потоков для обработки платежей. Увеличивайте при высокой нагрузке.

2.6.3. ПРИМЕР НАСТРОЕК

```
settings:
```

```
  method: "BODY"
  timeout: 5
  request-date-format: "yyyy-MM-dd'T'HH:mm:ss,SSS"
  timezone: "GMT+3"
  response-type: "JSON"
  thread-count: 4
  allow-empty-params: false
  render-loader: true
```

2.6.4. ПРИМЕР МИНИМАЛЬНОЙ РАБОЧЕЙ КОНФИГУРАЦИИ

```
servers:
  - url: "https://test.provider.com/api"
    timeout: 30
    auth-basic-user: "test"
    auth-basic-password: "test123"

logger: "test_gate"

codes:
  common:
    success: [0]
    error: [1, 2, 500]
    unknown: [-10000]

requests:
  common:
    default:
      headers:
        Content-Type: "application/json"
      result-path: "$.code"

  check:
    default:
      url: "/subscriber"
      request-body: "check.vsl"

  pay:
    default:
      url: "/payment"
      request-body: "pay.vsl"
```

```
settings:
  method: "BODY"
  response-type: "JSON"
  timeout: 10
```

ЧАСТЬ 3. РАСШИРЕННАЯ НАСТРОЙКА ЗАПРОСОВ

3.1. СПРАВОЧНИК ПАРАМЕТРОВ ЗАПРОСОВ

3.1.1. ПАРАМЕТРЫ REQUEST

Параметры запроса Request:

Параметр	Тип	Описание	Применимость
server	string	Псевдоним из **additional-servers**	Все запросы
url	string	Относительный URL	Все запросы
headers	object	HTTP-заголовки	Все запросы
method	enum	POST, PUT, GET, DELETE, PATCH, BODY, RMI, BLANK, SQL	Все запросы
subscription	object	Настройки абонентской платы	check, advanced
token-name	string	Имя токена для сброса при ошибке	Все запросы

Параметр	Тип	Описание	Применимость
request-body	string	Имя файла .vsl (для BODY)	BODY
request-params	string	Строка параметров (для GET/POST)	GET, POST
response-type	enum	XML, JSON, REGEX, HTTPCODE	Все запросы
result-path	string	Путь до кода ответа	Все запросы
result-from-param	string	Переопределение кода из атрибута	Все запросы
error-path	string	Путь до текста ошибки	Все запросы
transaction-path	string	Путь до ID транзакции	Все запросы
time-process-path	string	Путь до даты проведения	Все запросы
response-charset	string	Кодировка ответа	Все запросы
request-charset	string	Кодировка запроса	Все запросы
request-encode	boolean	Шифровать GET-запрос	GET
send-if-param-available	string	Отправлять только при наличии параметра	Все запросы
change-context-path	string	Смена контекста (для вложенных данных)	Все запросы

Параметр	Тип	Описание	Применимость
js-post-process	string	JS-скрипт пост-обработки	Все запросы
result-params	object	Параметры ответа	Все запросы
rmi-request	object	RMI-переадресация	RMI
async-advanced	object	Асинхронный advanced	advanced

3.1.2. ПАРАМЕТРЫ УПРАВЛЕНИЯ ЗАПРОСОМ

url
Относительный адрес запроса. Добавляется к базовому URL из секции **servers**. Если на уровне запроса не задан, используется значение из **common**.

method
Метод отправки запроса. Переопределяет метод, заданный в **settings**.

headers
HTTP-заголовки запроса. Формат: ключ: значение.

```
headers:
  Accept: "application/json"
  X-API-Key: "#token#"
```

request-body
Имя файла Velocity-шаблона (*.vsl*), который формирует тело запроса. Файл должен находиться в той же папке, что и конфигурационный YAML-файл.

request-params
Строка параметров для GET/POST-запросов. Параметры разделяются точкой с запятой (Поддерживаются переменные и функции (см. раздел 3.2).

```
request-params: "user_id=#id1#;amount=#realSum#;sign=md5(#id1##id2#)"
```

request-encode
Шифровать ли GET-запрос (true/false). По умолчанию false.

send-if-param-available
Отправлять запрос только при наличии указанного параметра в платеже. Например, **`#provider-trans#`**. Если параметр отсутствует, возвращается ошибка сети (-10000).

3.1.3. ПАРАМЕТРЫ РАЗБОРА ОТВЕТА

result-path
Путь до кода ответа в ответе провайдера. Формат зависит от **`response-type`**:

- Для XML: XPath (например, **`/RESPONSE/CODE`**);
- Для JSON: JSON-path (например, **`$.status.code`**);
- Для REGEX: регулярное выражение (первая группа);
- Спецзначение **`{HTTPCODE}`**: взять HTTP-статус ответа.

error-path
Путь до текстового описания ошибки. Аналогичный синтаксис.

transaction-path
Путь до идентификатора транзакции, присвоенного провайдером.

time-process-path
Путь до даты/времени проведения платежа в ответе провайдера. Формат разбора задаётся в **`settings.response-date-format`**.

change-context-path
Изменяет корневой контекст для последующих запросов XPath/JSON-path.

Используется, когда внутри параметра ответа (или в секции CDATA) находится вложенный XML/JSON.

result-from-param
Переопределяет код ответа значением указанного атрибута.

Алгоритм работы:

1. Шлюз получает код по **`result-path;`**
2. Разбираются атрибуты ответа (result-params);
3. Если среди атрибутов найден параметр, указанный в **`result-from-param`**, его значение становится новым кодом ответа;
4. Код транслируется в статус платежа по таблице из секции **`codes.`**

Пример:

```
pay:
  default:
    result-path: "/RESPONSE/STATUS_CODE"
    result-from-param: "real_status"
    result-params:
      params:
        real_status:
          value: "/RESPONSE/DATA/STATUS_CODE"
```

3.1.4. ПАРАМЕТРЫ ПОСТ-ОБРАБОТКИ

js-post-process
Имя JavaScript-файла для пост-обработки ответа. Файл должен находиться в папке с конфигом.

В контексте скрипта доступны:

- Для advanced-запросов: **`response`** (объект Response), **`payment`** (объект Payment);

- Для остальных запросов: **`result`** (PaymentResult), **`payment`** (Payment).

Пример **`(postprocess.js)`**:

```
var total = 0;
var elements = response.getData().getElements();
for (var i = 0; i < elements.size(); i++) {
    if (elements.get(i).getKey() == 'amount') {
        total += parseInt(elements.get(i).getValue());
    }
}
if (total > 100000) {
    response.setServiceError(5);
```

3.2. ПАРАМЕТРЫ GET/POST-ЗАПРОСОВ (REQUEST-PARAMS)

3.2.1. НАЗНАЧЕНИЕ

Параметр **`request-params`** используется, когда запрос отправляется методом **`GET`** или **`POST`** и данные передаются в строке запроса, а не в теле.

Формат: строка с параметрами, разделёнными **`;`** (точка с запятой).

```
request-params:
"user_id=#id1#;amount=#realSum#;sign=hash(SHA256,UTF8,#id1##id2#)"
```

3.2.2. ПЕРЕМЕННЫЕ ПЛАТЕЖА

Описание переменных:

Переменная	Описание	Пример значения
`#id#`	ID платежа в системе	123456
`#id1#, #id2#`	Данные из формы (реквизиты)	911234567890

Переменная	Описание	Пример значения
#date#	Дата и время (формат из **request-date-format**)	2026-02-12 14:30:00
#time#	Время (HH:mm:ss)	14:30:00
#realSum#	Сумма в рублях (с копейками)	100.50
#sum#	Сумма в копейках	10050
#realSumIn#	Вложенная сумма в рублях	110.00
#sumIn#	Вложенная сумма в копейках	11000
#comm#	Комиссия в копейках	50
#realComm#	Комиссия в рублях	0.50
#point#	Номер точки приёма платежей	101
#check#	Номер чека	998877
#provider-trans#	Номер транзакции провайдера	TXN-2026-02-12-001
#queueId#	ID промежуточной очереди (для advanced)	98765
#date_process#	Дата обработки платежа	2026-02-12
#time_process#	Время обработки платежа	14:30:05

Переменная	Описание	Пример значения
#date_point#	Дата терминала	2026-02-12
#time_point#	Время терминала	14:29:55

3.2.3. ПРЕОБРАЗОВАНИЯ

Функции можно применять к переменным и константам.

Функция	Описание	Пример
lower()	Нижний регистр	`lower(#id1#)`
upper()	Верхний регистр	`upper(#id1#)`
md5()	MD5-хеш (нижний регистр)	`md5(#id1#)`
md5U()	MD5-хеш (верхний регистр)	`md5U(#id1#)`
sha1()	SHA-1 хеш	`sha1(#id1##id2#)`
base64()	Base64 кодирование	`base64(#id1#)`
encode()	URL-кодирование	`encode(#id1#, utf8)`
hash()	Произвольный хеш-алгоритм	`hash(SHA512,UTF8,#id1##sum#)`
hash64()	Хеш с Base64-выходом	`hash64(SHA256,UTF8`

Функция	Описание	Пример
		,#id1#)
signBase64()	Электронная подпись (Base64)	signBase64(#id1#)
signHex()	Электронная подпись (Hex)	signHex(#id1#)

3.2.4. АТРИБУТЫ БАНКОВСКИХ ОПЕРАЦИЙ

Если платёж совершён банковской картой, в **request-params** доступны атрибуты первой банковской операции.

Префикс: **ps_**

Переменная	Описание
#ps_id#	ID операции
#ps_date#	Дата операции
#ps_time#	Время операции
#ps_sum	Сумма платежа (копейки)
#ps_realSum#	Сумма платежа (рубли)
#ps_comm#	Комиссия (копейки)
#ps_realComm#	Комиссия (рубли)

Переменная	Описание
#ps_rrn#	Retrieval Reference Number
#ps_auth_code#	Код авторизации
#ps_pma#	Номер карты/кошелька
#ps_token#	Токен карты
#ps_card_name#	Название карты (Visa, Mastercard)
#ps_card_type#	Тип карты
#ps_имя_атрибута#	Произвольный атрибут операции

3.2.5. ПАРАМЕТРЫ ШЛЮЗА, VELOCITY, JS, ТОКЕН, РЕНДЕРЫ

В **request-params** также доступны:

Источник	Синтаксис	Пример
Параметры шлюза	**#params.имя#**	#params.username#
Velocity-параметры	**#имя_vsl_параметра#**	#signature#
JS-параметры	**#имя_js_параметра#**	#computed_data#
Токен	**#имя_токена#**	#access_token#

Источник	Синтаксис	Пример
Рендеры	**#ключ_рендера#**	#PaymentType#

ЧАСТЬ 4. ОБРАБОТКА ОТВЕТОВ ПРОВАЙДЕРА (RESULT-PARAMS)

4.1. ОБЩАЯ СХЕМА РАЗБОРА ОТВЕТОВ

Секция **result-params** определяет, как извлечь данные из ответа провайдера и в каком виде передать их на клиентское устройство (терминал, кассу, сценарий).

Базовый синтаксис:

```
result-params:
  params:
    ключ_параметра:
      атрибут1: значение1
      атрибут2: значение2
  selector:
    name: "#selector_name"
    path: "..."
    result-params:
      params:
        ...
  collections:
    collection_name:
      path: "..."
      param:
        static-title: "..."
        value: "…"
```

4.2. СЕКЦИЯ PARAMS — ПРОСТЫЕ ПАРАМЕТРЫ

4.2.1. НАЗНАЧЕНИЕ

Извлекает из ответа провайдера отдельные значения (ФИО, баланс, номер договора и т.п.) и передаёт их на точку.

4.2.2. АТРИБУТЫ СЕКЦИИ PARAMS

Доступные атрибуты параметров:

Атрибут	Тип	Описание
title	string	Путь до заголовка параметра в ответе
value	string	Путь до значения параметра. Поддерживает группы **{...}** для конкатенации
flags	integer	Флаги параметра (десятичное число)
static-title	string	Статичный заголовок (константа)
bundle-title	string	Ключ для локализации заголовка
bundle-value	string	Ключ для локализации значения
static-value	string	Статичное значение (константа)
post-process	string	JavaScript-выражение для пост-обработки значения

4.2.3. ГРУППЫ ПУТЕЙ (КОНКАТЕНАЦИЯ)

Используйте фигурные скобки **{...}** для объединения нескольких полей ответа:

```
fio:
  static-title: "ФИО"
  value: "{$.lastName} {$.firstName} {$.middleName}"
```

4.2.4. СТАТИЧНЫЕ ЗНАЧЕНИЯ ПАРАМЕТРОВ

```
params:
  operation_type:
    static-title: "Тип операции"
    static-value: "PAYMENT"
  provider_id:
    static-title: "ID провайдера"
    static-value: "1690"
```

4.2.5. ЛОКАЛИЗАЦИЯ (BUNDLE-TITLE/BUNDLE-VALUE)

Для использования локализации необходимо настроить **`settings.localization`**:

```
settings:
  localization:
    resource-bundle: "messages.properties"
    lang-key: "#lang#"
```

Затем в **`result-params`**:

```
params:
  status_text:
    bundle-title: "payment.status.title"
    bundle-value: "payment.status.value"
```

Система ищет:

1. **`messages_#lang#.properties`** (например, **`messages_az.properties`**);
2. Если не найден — **`messages.properties`.**

4.2.6. ПОСТ-ОБРАБОТКА (POST-PROCESS)

JavaScript-выражение, которое выполняется после извлечения значения. В контексте доступна переменная с именем параметра.

```
params:
  amount:
    value: "/response/amount"
    post-process: "Math.round(parseFloat(amount) * 100) / 100"
```

```
  masked_pan:
    value: "$.cardNumber"
    post-process: "value.substring(0,6) + '******' + value.substring(12)"
```

4.2.7. ЗНАЧЕНИЯ ИЗ HTTP-ЗАГОЛОВКОВ

```
params:
  location:
    value: "HEADER:Location"
  content_type:
    value: "HEADER:Content-Type"
```

4.2.8. ПРИМЕРЫ

Простой параметр:

```
params:
  fio:
    title: "ФИО"
    value: "$.subscriber.fullName"
  balance:
    static-title: "Баланс"
    value: "$.account.balance"
```

С флагами:

```
params:
  account_number:
    static-title: "Номер счета"
    value: "$.account"
    flags: 512   # 0x200 в десятичном виде
```

4.3. СЕКЦИЯ SELECTOR — ВЫБОР ИЗ СПИСКА

4.3.1. НАЗНАЧЕНИЕ

Используется, когда провайдер возвращает набор однотипных элементов (список банков, счета абонента, тарифы). На точке отображается экран типа SELECTOR — клиент выбирает один из вариантов.

4.3.2. АТРИБУТЫ СЕКЦИИ SELECTOR

Атрибуты, доступные в секции SELECTOR:

Атрибут	Тип	Описание	Обязат.
name	string	Название селектора (отображается на точке)	Да
path	string	Путь к элементам селектора в ответе	Да
result-params	object	Правила разбора параметров внутри каждого элемента	Нет
params-filter	object	Фильтр элементов (ключ: регулярное выражение)	Нет
clear-params	array	Список ключей, которые не передавать на точке	Нет
change-context-path	string	Изменяет корневой контекст внутри элемента	Нет
empty-error	integer	Код ошибки, если селектор пустой	Нет
uniq-param-code	string	Ключ параметра, по которому удалять дубликаты	Нет

4.3.3. ПРИМЕР: СПИСОК СЧЕТОВ С ФИЛЬТРОМ

Ответ провайдера (JSON):

```
{
  "accounts": [
    { "number": "40817810099990000001", "currency": "RUB", "balance":
15000 },
    { "number": "40817810099990000002", "currency": "USD", "balance": 500
},
    { "number": "40817810099990000003", "currency": "RUB", "balance": 0 }
  ]
}
```

Конфигурация:

```
selector:
  name: "#accounts"
  path: "$.accounts[*]"
  params-filter:
    currency: "RUB"
    balance: "[1-9][0-9]*"
  clear-params: ["currency"]
  uniq-param-code: "number"
  result-params:
    params:
      number:
        static-title: "Номер счета"
        value: "number"
      balance:
        static-title: "Баланс"
        value: "balance"
```

Результат на точке:

- Передан только RUB-счета с положительным балансом;
- Параметр **`currency`** удалён;
- Дубликаты номеров счетов исключены.

4.3.4. ИЗМЕНЕНИЕ КОНТЕКСТА (CHANGE-CONTEXT-PATH)

Используется, когда внутри параметра ответа находится вложенный JSON/XML.

Пример:

```
selector:
  name: "#costInfos"
  change-context-path: "$.informations[0].costInfo"
  path: "$.[*]"
  result-params:
    params:
      title:
        value: "cost"
      amount:
        static-title: "Сумма"
        value: "amount"
```

4.4. СЕКЦИЯ COLLECTIONS — ИНДЕКСАЦИЯ МАССИВОВ

4.4.1. НАЗНАЧЕНИЕ

Провайдер вернул список значений с одинаковыми ключами. Чтобы все значения доехали до точки, шлюз автоматически добавляет индекс: **cart1**, **cart2**, **cart3**...

4.4.2. АТРИБУТЫ СЕКЦИИ COLLECTIONS

Атрибуты, доступные в секции COLLECTIONS^

Атрибут	Тип	Описание	Обязат.
path	string	Путь к элементам массива	Да
param	object	Правила разбора для каждого элемента	Да

Атрибут	Тип	Описание	Обязат.

4.4.3. ПРИМЕР

```
collections:
  account_number:
    path: "//GetAccListResult/ExtraInfo/M1On_AccExtraStruc"
    param:
      static-title: "Номер счета"
      value: "accountNumber"
  contract_code:
    path: "//GetAccListResult/ExtraInfo/M1On_AccExtraStruc"
    param:
      static-title: "Код договора"
      value: "ContractCode"
```

Ответ провайдера (XML):

```
<M1On_AccExtraStruc>
  <accountNumber>111</accountNumber>
  <ContractCode>qqq</ContractCode>
</M1On_AccExtraStruc>
<M1On_AccExtraStruc>
  <accountNumber>222</accountNumber>
</M1On_AccExtraStruc>
```

Результат на точке:

- **`account_number1=111`**, **`contract_code1=qqq;`**
- **`account_number2=222`**, **`contract_code2`** отсутствует.

4.5. СЕКЦИЯ PARAMS-ARRAY — МАССИВ ОПЦИОНАЛЬНЫХ ПАРАМЕТРОВ

4.5.1. НАЗНАЧЕНИЕ

Используется, когда провайдер возвращает массив параметров с переменным составом. Часто применяется совместно с рендерами.

4.5.2. АТРИБУТЫ СЕКЦИИ PARAMS-ARRAY

Атрибуты, доступные в секции PARAMS-ARRAY:

Атрибут	Тип	Описание	Обязат.
path	string	Путь к массиву параметров	Да
code	string	Путь до ключа параметра	Да
value	string	Путь до значения параметра	Да
title	string	Путь до заголовка параметра	Нет
modify-with-render	boolean	Применять рендеры к каждому параметру	Нет (false)

4.5.3. ПРИМЕР

```
params-array:
  path: "//AWResponse/MessageBody/KindAttribute"
  code: "Name"
  value: "Value"
  title: "Description"
  modify-with-render: true
```

4.6. СЕКЦИЯ PARAMS-FROM-ADVANCED — ВЫЗОВ ADVANCED-ФУНКЦИИ

4.6.1. НАЗНАЧЕНИЕ

Вызывает advanced-функцию и добавляет её результат к текущим параметрам ответа.

4.6.2. АТРИБУТЫ СЕКЦИИ PARAMS-FROM-ADVANCED

Атрибуты, доступные в секции PARAMS-FROM-ADVANCED:

Атрибут	Тип	Описание	Обязат.
function	string	Имя advanced-функции	Да
replace	boolean	Заменить текущие параметры результатом функции	Да
filter	object	Фильтр для вызова функции	Нет
save-params	array	Какие параметры сохранить (все, если не задано)	Нет
pass-error-up	boolean	Пробросить ошибку в родительский метод	Нет

4.6.3. ПРИМЕР С ФИЛЬТРОМ И ПРОБРОСОМ ОШИБКИ

```
params-from-advanced:
  function: "get_contract_detail"
  replace: false
  pass-error-up: true
  filter:
    productCode: "(KAM1|KAM2|KAX3)"
  save-params: ["contract_number", "contract_date"]
```

4.7. СЕКЦИЯ CLIENT-FEE — КОМИССИЯ С КЛИЕНТА

4.7.1. НАЗНАЧЕНИЕ

Динамически добавляет комиссию к параметрам ответа. Комиссия может быть процентом от суммы или фиксированной.

4.7.2. ПАРАМЕТРЫ СЕКЦИИ CLIENT-FEE

Параметры, доступные в секции CLIENT-FEE:

Параметр	Тип	Описание
type	enum	**numeric** (с копейками) или **integer** (целые)
percent	string	Процент комиссии (например, "1.5")
fixed	string	Фиксированная комиссия
min	string	Минимальная сумма комиссии
max	string	Максимальная сумма комиссии

4.7.3. ПРИМЕР

```
result-params:
  client-fee:
    type: numeric
    percent: "2.5"
    min: "10"
    max: "1000"
```

Результат:
К сумме платежа будет добавлена комиссия 2.5%, но не менее 10 руб. и не более 1000 руб.

4.8. ВСПОМОГАТЕЛЬНЫЕ СЕКЦИИ

4.8.1. SAVE-ON-ERROR

Сохранять ли параметры ответа, если вернулся код ошибки.

```
result-params:
  save-on-error: true
```

4.8.2. CLEAR-PARAMS

Удаляет указанные параметры из результата перед отправкой на точку.

```
result-params:
  clear-params: ["internal_id", "session_token", "debug_info"]
```

4.8.3. UNKNOWN-PARAMS

Параметры, которые возвращаются, если запрос не удалось разобрать или произошла неизвестная ошибка.

```
result-params:
  unknown-params:
    error_code:
      static-title: "Код ошибки"
      static-value: "UNKNOWN"
    message:
      static-title: "Сообщение"
      static-value: "Сервис временно недоступен"
```

ЧАСТЬ 5. УЗКОНАПРАВЛЕННЫЕ И СПЕЦИАЛЬНЫЕ НАСТРОЙКИ

5.1. ADVANCED-ЗАПРОСЫ

5.1.1. НАЗНАЧЕНИЕ

Advanced-запросы — это произвольные функции, которые вызываются:

- из сценариев оплаты;
- из других advanced-функций (последовательно или условно);
- по расписанию (cron).

5.1.1. БАЗОВАЯ СТРУКТУРА

```
advanced:
  get_token:
    url: "/api/v1/login"
    method: "BODY"
    request-body: "login.vsl"
    result-path: "$.status"
    error-path: "$.message"
    result-params:
      params:
        access_token:
          value: "$.accessToken"
        expires_in:
          value: "$.expiresIn"

  get_balance:
    url: "/api/v1/balance"
    method: "GET"
    headers:
```

```
      Authorization: "Bearer #access_token#"
    request-params: "account=#id1#"
    result-path: "$.code"
    result-params:
      params:
        balance:
          value: "$.balance"
```

5.1.2. СПЕЦИАЛЬНЫЕ МЕТОДЫ ДЛЯ ADVANCED

Метод **BLANK** — не обращается к провайдеру, просто возвращает статичные параметры.

```
advanced:
  static_data:
    method: "BLANK"
    result-params:
      params:
        status:
          static-value: "OK"
        version:
          static-value: "1.0"
```

Метод **RMI** — перенаправляет запрос в шлюз другого провайдера.

```
advanced:
  get_accounts_rmi:
    method: "RMI"
    rmi-request:
      provider: -10
      service: 123
      function: "get_accounts"
      save-params: ["account", "balance"]
```

Параметры **`rmi-request`**:

Параметр	Тип	Описание	Обязат.
`service`	integer	ID сервиса для запроса	Да

Параметр	Тип	Описание	Обязат.
provider	integer	ID поставщика	Да
function	string	Имя advanced-функции	Нет
save-params	array	Какие параметры сохранить	Нет
use-request-service	boolean	Использовать сервис из исходного запроса	Нет

Метод **SQL** — выполняет запрос к базе данных.

```
advanced:
  get_client_limit:
    method: "SQL"
    request-body: "get_limit.sql"
    result-params:
      params:
        total:
          value: "total"
        currency:
          value: "currency"
```

Файл **get_limit.sql**:

```
with cif as (
  select #cif# as cif
)
select cif.cif, COALESCE(cl.total, 0) as total, cl.currency
from cif
left join accounts.cif_limits cl on cif.cif = cl.cif
```

5.1.3. ПОСЛЕДОВАТЕЛЬНЫЙ ВЫЗОВ (ADVANCED-SEQUENCES)

Где задаётся: в секции **settings**.

```
settings:
  advanced-sequences:
```

```
    get_full_info: ["get_token", "get_balance", "get_transactions"]
    pay_with_check: ["validate", "process_payment", "confirm"]
```

Функция **get_full_info** вызовет:

1. **get_token** — получить токен;
2. **get_balance** — запросить баланс (используя токен из п.1);
3. **get_transactions** — запросить историю (используя токен и ID клиента).

Параметры ответа каждой функции доступны следующим функциям в последовательности.

5.1.4. УСЛОВНЫЙ ВЫЗОВ (CONDITIONAL-SEQUENCES)

Где задаётся: в секции **settings**.

```
settings:
  conditional-sequences:
    process_by_type:
      - filter: "params.get('account').matches('^IZI.*')"
        function: "process_izi"
        params-post-filter: "params.get('sum') < 10000"
        stop-on-error: true
        save-params: ["transaction_id", "commission"]
      - filter: "params.get('account').matches('^AZA.*')"
        function: "process_aza"
        params-post-filter: "params.get('status') == 'ACTIVE'"
```

Атрибуты CONDITIONAl-SEQUENCES:

Атрибут	Тип	Описание
filter	string	JS-выражение (true — выполнить функцию)
function	string	Имя вызываемой функции

Атрибут	Тип	Описание
params-post-filter	string	JS-фильтр для результатов функции
stop-on-error	boolean	Остановить всю последовательность при ошибке
save-params	array	Какие параметры сохранить в ответ

5.1.5. АСИНХРОННЫЕ ADVANCED-ЗАПРОСЫ

Используются, когда провайдер не возвращает результат сразу (например, платёж в обработке).

```
advanced:
  async_payment:
    method: "BODY"
    request-body: "payment.vsl"
    result-path: "$.state"
    async-advanced:
      function: "check_status"
      empty: false
      repeat-timeout: 30
      try-count: 10
      process-codes: [200, 202]
```

Параметры ADVANCED-запросов^

Параметр	Тип	Описание	Обязат.
function	string	Функция для проверки статуса	Да
empty	boolean	Пустой асинхронный запрос	Да
repeat-timeout	integer	Интервал между попытками (сек)	Да

Параметр	Тип	Описание	Обязат.
try-count	integer	Максимальное количество попыток	Да
process-codes	array	Коды, означающие "ещё в процессе"	Нет

empty: true

Используется, когда нужно только проверить статус, без вызова дополнительной функции.

```
async-advanced:
  empty: true
  repeat-timeout: 15
  try-count: 20
  process-codes: [100, 101]
```

5.2. СПЕЦИАЛЬНЫЕ НАСТРОЙКИ ШЛЮЗА (SETTINGS)

5.2.1. УПРАВЛЕНИЕ ЛОГАМИ В КАБИНЕТЕ (LOGGER-SETTINGS)

```
settings:
  logger-settings:
    secured: false
    clear-regex: "password=[^&]+"
    clear-replace: "password=***"
    regex-replace:
      "([0-9]{4})[0-9]{8}([0-9]{4})": "$1********$2"
      "card=[0-9]{16}": "card=****"
```

Параметры LOGGER-SETTINGS:

Параметр	Тип	Описание
secured	boolean	true — логи скрыты, false — доступны админу в кабинете
clear-regex	string	Регулярное выражение для поиска конфиденциальных данных

Параметр	Тип	Описание
clear-replace	string	Строка замены
regex-replace	object	Карта регулярных выражений для маскировки (ключ -> замена)

5.2.2. СТАТУСЫ И ТАЙМАУТЫ

Доступные параметры статусов и таймаутов:

Параметр	Описание	По умолч.
status-using-pay	Использовать **pay** для проверки статуса (если нет **status**)	false
timeout-process	Интервал проверки статуса «Проводится»	60 сек
timeout-unknown	Интервал проверки статуса «Неизвестен»	300 сек
separate-confirm	Отдельный confirm-запрос с сохранением в БД	false
timeout-confirm	Таймаут очереди подтверждений	10 сек
only-online-verify	Использовать verify только для онлайн-проверки	false
balance-in-penny	Баланс приходит в копейках	false
allow-empty-params	Разрешить пустые параметры в ответе	false

Параметр	Описание	По умолч.
disable-follow-redirects	Не следовать HTTP-редиректам	false
basic-auth-in-header	Добавлять Basic Auth в заголовок Authorization	false

5.3. БЕЗОПАСНОСТЬ И КРИПТОГРАФИЯ

5.3.1. ЭЛЕКТРОННАЯ ПОДПИСЬ (SIGNER)

Где задаётся: в секции **settings**.

```
settings:
  signer:
    private-key: "private.pem"
    public-key: "public.pem"
    private-key-password: "12345"
    algorithm: "SHA256withRSA"
    charset: "UTF-8"
```

Параметры SINGER:

Параметр	Тип	Описание	Обязат.
private-key	string	Файл закрытого ключа (PEM)	Да
public-key	string	Файл открытого ключа (PEM)	Да
private-key-password	string	Пароль от закрытого ключа	Нет
algorithm	string	Алгоритм подписи (SHA256withRSA)	Да

Параметр	Тип	Описание	Обязат.
charset	string	Кодировка (UTF-8)	Да

Использование в Velocity:

```
#set($sign = $signer.signBase64($payment.account.identifier))
<SIGN>$sign</SIGN>
```

Использование в request-params:

```
request-params: "sign=signBase64(#id1#)"
```

5.3.2. XML-ПОДПИСЬ (XML-SIGNER)

Только для **method: BODY** и **response-type: XML**.

```
settings:
  xml-signer:
    private-key: "key.pem"
    certificate: "cert.pem"
    private-key-password: "pass"
```

Параметры XML-SINGER:

Параметр	Тип	Описание	Обяз.
private-key	string	Файл закрытого ключа (PEM)	Да
certificate	string	Файл сертификата (PEM)	Да
private-key-password	string	Пароль от закрытого ключа	Нет

Использование в Velocity:

```
#set($body = "<REQUEST>...</REQUEST>")
$xmlSigner.generateXMLSignature($body)
```

5.3.3. ТОКЕНЫ (TOKEN)

Автоматическое получение и обновление токена.

```
settings:
  token:
    function: "login"
    token-name: "access_token"
    expiration-timeout: 3600
    response-expiration: "expires_in"
    response-expiration-format: "yyyy-MM-dd HH:mm:ss"
    additional-tokens:
      - function: "refresh_token"
        token-name: "refresh_token"
        expiration-timeout: 86400
```

Параметры TOKEN:

Параметр	Тип	Описание	Обязат.
function	string	Advanced-функция, возвращающая токен	Нет
token-name	string	Имя переменной токена	Да
expiration-timeout	integer	Срок действия (сек), если не вернул провайдер	Нет
response-expiration	string	Поле в ответе со сроком действия	Нет
response-expiration-format	string	Формат даты в поле expiration	Нет
additional-tokens	array	Дополнительные токены	Нет

Использование:

```
headers:
  Authorization: "Bearer #access_token#"
```

Сброс токена по ошибке:

```
pay:
  default:
    token-name: "access_token"
```

При получении кода из массива **token** в секции **codes** указанный токен будет принудительно обновлён.

5.3.4. ШИФРОВАНИЕ (ENCRYPTOR)

```
settings:
  encryptor:
    public-key: "public.pem"
    algorithm: "RSA/ECB/PKCS1Padding"
    charset: "UTF8"
```

Параметры ENCRYPTOR:

Параметр	Тип	Описание	Обяз.
public-key	string	Файл открытого ключа (PEM)	Да
algorithm	string	Алгоритм шифрования	RSA
charset	string	Кодировка	UTF8

В Velocity:

```
#set($encrypted = $encryptor.encryptBase64($secret))
#set($encryptedHex = $encryptor.encryptHex($secret))
```

В request-params:

```
request-params: "data=encryptBase64(#id1#)"
```

5.4. ДИНАМИЧЕСКИЕ ДАННЫЕ В ЗАПРОСАХ

5.4.1. РЕНДЕРЫ (RENDER-LOADER)

Позволяют переопределять параметры запроса через кабинет, без правки конфига.

Включение:

```
settings:
  render-loader: true
```

Настройка в кабинете (в свойствах сервиса провайдера):

```
render.VendorId=111
render.PaymentType=#type#
render.check.customerCode=#id1#
```

Использование в request-params:

```
request-params: "vendor=#VendorId#;type=#PaymentType#"
```

Использование в Velocity:

```
$renderValues.get('PaymentType')
```

Рендеры результата:

```
result.title.customerCode=CUSTOMER CODE TITLE
result.key.customerCode=code-customer
result.flags.code-customer=2
result.copy.code-customer=id2
```

5.4.2. ЛОКАЛИЗАЦИЯ

```
settings:
  localization:
    resource-bundle: "messages.properties"
    lang-key: "#lang#"
```

Параметры LOCALIZATION:

Параметр	Тип	Описание
resource-bundle	string	Файл ресурсов (.properties)
lang-key	string	Ключ языка в параметрах запроса

5.4.3. ПАРАМЕТРЫ ШЛЮЗА (PARAMS)

Статичные параметры, доступные в **request-params** и Velocity как **$gateParams**.

```
settings:
  params:
    username: "api_user"
    password: "secret"
    endpoint_id: "TERMINAL-01"
```

Использование:

```
request-params: "login=#params.username#&pwd=#params.password#"
```

В Velocity:

```
<login>$gateParams.username</login>
```

5.4.4. VELOCITY-ПАРАМЕТРЫ (VELOCITY-PARAMS)

Предварительно вычисленные Velocity-шаблоны. Файлы *.vsl* находятся в папке конфига.

```
settings:
  velocity-params:
    api_token: "token.vsl"
    signature: "signature.vsl"
    service_id: "service_id.vsl"
```

Использование:

```
request-params: "#signature#"
```

5.4.5. JAVASCRIPT-ПАРАМЕТРЫ (JS-PARAMS)

Выполнение JS-скрипта, результат доступен в запросе.

```
settings:
  js-params:
    computed_data: "script.js"
    masked_account: "mask.js"
```

script.js:

```
var sum = payment.getRealSumOutcome();
var curr = payment.account.attributes.curr;
var result = sum * curr;
"sum=" + result.toFixed(2);
```

Использование:

```
request-params: "data=#computed_data#"
```

В Velocity:

```
$computed_data
```

5.5. РЕДАКТИРОВАНИЕ БАЗЫ ДАННЫХ ЧЕРЕЗ ШЛЮЗ (EDIT-DB)

5.5.1. НАЗНАЧЕНИЕ

Функционал позволяет выполнять SQL-запросы к базе данных процессинга непосредственно из конфигурации шлюза. Используется для:

- Обновления статусов платежей;
- Записи дополнительных атрибутов;
- Выполнения служебных операций.

5.5.2. АКТИВАЦИЯ ФУНКЦИОНАЛА

Функционал является опциональным и активируется **только по согласованию с компанией Soft-Logic**.

Для активации необходимо:

1. Получить пароль активации у службы поддержки
2. Указать его в конфигурации:

```
settings:
  editDbEnabled: true
  edit-db-password: "полученный_пароль"
```

Важно! Без корректного пароля функционал не будет работать, даже если **`editDbEnabled: true`**.

5.5.3. ИСПОЛЬЗОВАНИЕ SQL-ЗАПРОСОВ

SQL-запросы выполняются методом SQL в advanced-функциях.

```
advanced:
  update_payment_status:
    method: "SQL"
    request-body: "update_status.sql"
    result-params:
      params:
        updated:
          static-value: "true"
```

update_status.sql:

```
UPDATE payments
SET status = 1, provider_trans = '#provider-trans#'
WHERE id = #id#
RETURNING id;
```

5.5.4. ДОСТУПНЫЕ ПЕРЕМЕННЫЕ В SQL

В SQL-запросах доступны все переменные платежа в формате **`#переменная#`**:

- **`#id#, #id1#, #id2#;`**
- **`#sum#, #realSum#;`**
- **`#provider-trans#;`**
- **`#date#, #time#;`**
- и другие (см. раздел 3.2.2).

5.5.5. БЕЗОПАСНОСТЬ

Все SQL-запросы выполняются через **`PreparedStatement;`**

- Прямая конкатенация строк запрещена;
- Параметры подставляются безопасно;
- Доступ к БД ограничен правами пользователя процессинга.

5.6. ВХОДЯЩИЕ CALLBACK-УВЕДОМЛЕНИЯ

5.6.1. НАЗНАЧЕНИЕ

Обработка входящих HTTP-запросов от провайдера (статусы платежей, подтверждения, отмены, уведомления).

5.6.2. КОНФИГУРАЦИЯ

```
requests:
  callback:
    default:
      method: "POST"
      response-type: "XML"
      result-path: "/notification/status"
      transaction-path: "/notification/order_id"
      result-params:
        params:
          payment_id:
            value: "/notification/payment_id"
          status:
            value: "/notification/state"
```

5.6.3. БЕЗОПАСНОСТЬ: БЕЛЫЙ СПИСОК IP

Обязательно настроить ограничение по IP-адресам:

```
settings:
  allowed-callback-ip-addresses:
    - "192.168.1.100"
    - "10.10.10.0/24"
    - "203.0.113.5"
```

Поведение:

- Если список задан — запросы с других IP отклоняются с кодом 403;
- Если не задан — все IP разрешены (не рекомендуется для production).

5.6.4. КОДЫ ОТВЕТОВ ДЛЯ CALLBACK

В секции **codes** можно настроить обработку статусов из callback:

```
codes:
  callback:
    success: [0, 200]
```

```
error: [400, 500]
unknown: [300]
```

5.7. CRON-ЗАДАЧИ

5.7.1. НАЗНАЧЕНИЕ

Периодический вызов advanced-функций по расписанию.

5.7.2. СПИСОК ПАРАМЕТРОВ

Параметры, доступные для CRON:

Параметр	Тип	Описание	Обяз.
name	string	Название задачи	Нет
expression	string	Cron-выражение	Да
function	string	Advanced-функция для вызова	Да
params	object	Параметры запроса (ключ-значение)	Нет
email	string	Email(ы) для отправки результата (через запятую)	Нет
clazz	string	Класс-обработчик, реализующий **JobProcessor**	Нет
result-type	enum	**WHOLE_BODY** или **RESULT_PARAM**	Нет
save-path	string	Папка для сохранения результата на сервере	Нет

Параметр	Тип	Описание	Обяз.

5.7.3. CRON-ВЫРАЖЕНИЕ

Стандартное cron-выражение: **`сек мин час день_мес месяц день_нед.`**

Примеры:

- **`0 0 6 * * ?`** — каждый день в 6:00;
- **`0 */15 * * * ?`** — каждые 15 минут;
- **`0 0 0 * * MON`** — каждый понедельник в 0:00.

5.7.4. ПРИМЕРЫ

Базовый пример:

```
settings:
  cron-jobs:
    - name: "daily_rates"
      expression: "0 0 6 * * ?"
      function: "get_currency_rates"
      result-type: "WHOLE_BODY"
      save-path: "/opt/payload/rates/"
```

С параметрами и email:

```
cron-jobs:
  - name: "sync_clients"
    expression: "0 0 */2 * * ?"
    function: "sync_client_data"
    params:
      source: "erp"
```

```
      full_sync: "false"
    result-type: "RESULT_PARAM"
    email: "admin@company.ru, logs@company.ru"
```

С кастомным обработчиком:

```
cron-jobs:
  - name: "process_chargebacks"
    expression: "0 0 3 * * ?"
    function: "get_chargebacks"
    clazz: "com.company.ChargebackProcessor"
    result-type: "WHOLE_BODY"
```

5.7.5. ТИПЫ РЕЗУЛЬТАТОВ

Возможные типы результатов:

Тип	Описание
WHOLE_BODY	Сохраняется/отправляется полное тело ответа провайдера
RESULT_PARAM	Сохраняется/отправляется параметр ответа с ключом 'result'

5.7.6. СОХРАНЕНИЕ РЕЗУЛЬТАТА

При указании **save-path** файл сохраняется с именем: **<id_провайдера>_<имя_задачи>_<дата_время>.txt**

Пример: **1690_daily_rates_20260212_060000.txt**

ЧАСТЬ 6. ИНСТРУМЕНТЫ И СРЕДА

6.1. VELOCITY-ШАБЛОНЫ

6.2. НАЗНАЧЕНИЕ

Velocity (VSL) — язык шаблонов для формирования тела запроса (XML/JSON) при **`method: BODY`**.

Файлы *.vsl* лежат в той же папке, что и конфигурационный YAML-файл.

6.3. ОБЪЕКТЫ VELOCITY-ШАБЛОНОВ

Доступные объекты VELOCITY:

Объект	Описание
`$payment`	Текущий платёж
`$dateFormat`	Форматирование даты: **`$dateFormat.format($date, "pattern")`**
`$numberFormat`	Форматирование суммы
`$now`	Текущее время сервера
`$gateParams`	Параметры из **`settings.params`**
`$hash`	Хеши: **`$hash.md5()`**, **`$hash.sha1()`**, **`$hash.base64()`**

Объект	Описание
`$signer`	ЭЦП: `$signer.signBase64()`, `$signer.signHex()`
`$renderValues`	Рендеры (ключ-значение)
`$xmlSigner`	XML-подпись
`$encryptor`	Шифрование
`$String`, `$Double`, `$Integer`	Java-классы
`velocity-params`	Все параметры, объявленные в `velocity-params`
`js-params`	Все параметры, объявленные в `js-params`

6.3.1. ОБЪЕКТ PAYMENT

Атрибуты объекта PAYMENT:

Выражение	Описание
`$payment.id`	ID платежа
`$payment.realSumOutcome`	Сумма в рублях
`$payment.sumOutcome`	Сумма в копейках
`$payment.date`	Дата платежа

Выражение	Описание
$payment.account.identifier	**id1** из формы
$payment.account.identifier2	**id2** из формы
$payment.account.attributes.название	Любой атрибут из формы
$payment.point.id	ID точки
$payment.check	Номер чека
$payment.providerDate	Дата обработки у провайдера
$payment.providerTrans	Номер транзакции провайдера
$payment.pointTrans	Номер операции на точке

6.3.2. ПРИМЕР ШАБЛОНА (JSON)

Файл: payment.vsl

```
{
  "header": {
    "msgId": "$payment.id",
    "msgTime": "$dateFormat.format($payment.date, "yyyy-MM-
dd'T'HH:mm:ss")",
    "login": "$gateParams.username"
  },
  "body": {
    "account": "$payment.account.identifier",
    "amount": $payment.realSumOutcome,
    "currency": "RUB",
```

```
    "description": "#if($payment.account.attributes.urgent)Срочный
платеж#{else}Обычный платеж#end"
  }
}
```

6.3.3. ПРИМЕР ШАБЛОНА (XML)

Файл: check.vsl

```
<REQUEST>
  <ACTION>CHECK</ACTION>
  <DATA>
    <CLIENT_ID>$payment.account.identifier</CLIENT_ID>
    <TIMESTAMP>$dateFormat.format($now, "yyyyMMddHHmmss")</TIMESTAMP>
    #set($hash =
$hash.md5("$payment.account.identifier$gateParams.salt"))
    <SIGN>$hash</SIGN>
  </DATA>
</REQUEST>
```

6.4. ДОПОЛНИТЕЛЬНЫЕ СЕРВЕРЫ (ADDITIONAL-SERVERS)

6.4.1. НАЗНАЧЕНИЕ

Позволяет использовать разные URL для разных запросов.

6.4.2. СИНТАКСИС

```
additional-servers:
  soap:
    - url: "https://soap.provider.com/endpoint"
      timeout: 30
    - url: "https://soap-backup.provider.com/endpoint"
      timeout: 60
  rest:
    - url: "https://rest.provider.com/api/v2"
```

```
  internal:
    - url: "http://localhost:8080/internal"
```

6.4.3. ПРИМЕР ИСПОЛЬЗОВАНИЯ В ЗАПРОСЕ

```
check:
  server: "soap"
  url: "/check"  # итоговый URL: https://soap.provider.com/endpoint/check

pay:
  server: "rest"
  url: "/payments"  # итоговый URL:
https://rest.provider.com/api/v2/payments
```

6.5. ВНЕШНИЕ ПАРАМЕТРЫ (PARAM)

6.5.1. НАЗНАЧЕНИЕ

Позволяет вынести чувствительные данные (пароли, URL) во внешний файл или переменные окружения.

6.5.2. ФАЙЛ СВОЙСТВ

Создайте файл **<имя_конфига>.properties** в той же папке, что и YAML-файл.

Файл: gate.yml.properties:

```
url=https://prod.provider.com
password=super_secret
username=api_user
timeout=45
```

6.5.3. ПРИМЕР ИСПОЛЬЗОВАНИЯ В КОНФИГЕ

```
servers:
  - url: "{param:url}"
    timeout: "{param:timeout}"
    auth-basic-user: "{param:username}"
    auth-basic-password: "{param:password}"
```

6.5.4. ПРИОРИТЕТ ПОДСТАНОВКИ

1. Значение из **`.properties`** файла;
2. Значение из переменной окружения (если не найдено в файле);
3. Исходный текст **`{param:key}`** (если ничего не найдено).

ЧАСТЬ 7. ЭКСПЛУАТАЦИЯ

7.1. ЗАПУСК ШЛЮЗА

7.1.1. РАЗМЕЩЕНИЕ ФАЙЛОВ

Создайте каталог шлюза: **`/paylogic/gates/configs/название_шлюза/.`**

Поместите туда:

- **`gate.yml`** — конфигурационный файл;
- Все *.vsl* шаблоны;
- Ключи, сертификаты (PEM, PKCS12);
- JS-скрипты (если используются);
- JSON-схему (опционально).

7.1.2. РЕГИСТРАЦИЯ В GATES.XML

Отредактируйте **`/paylogic/gates/config.xml`**:

```
<gates>
  <gate
    enable="true"
    factory="ru.softlogic.process.ing.gates.universal.v3.Factory"
    id-provider="1690"
    config="/paylogic/gates/configs/my_gate/gate.yml"/>
</gates>
```

Где атрибуты принимают следующие значения:

Атрибут	Описание
enable	true — шлюз активен
factory	Всегда **ru.softlogic...universal.v3.Factory**
id-provider	ID провайдера в кабинете
config	Полный путь к YAML-файлу

7.1.3. ПЕРЕЗАПУСК

Выполните команду:

```
service payloadc restart
```

7.2. ТЕСТИРОВАНИЕ

7.2.1. ТЕСТОВАЯ УТИЛИТА

Расположение тестовой утилиты:

/paylogic/gates/universal_factory3/gate/test/univ3-test-tool-1.1.0.jar

Выполните запуск:

```
/paylogic/java/java8/bin/java -jar univ3-test-tool-1.1.0.jar ./test.yml
```

7.2.2. СТРУКТУРА ТЕСТОВОГО КОНФИГА

```
provider-id: 1690
offset-id: -1   # -1 = автоматический сдвиг ID (System.currentTimeMillis)

datasource:
  url: jdbc:postgresql://127.0.0.1/work
  username: postgres
  password: postgres

operations:
  - check: 0
    id: 1
    point: 100
    service: 100
    ext-service: "100"
    sum-outcome: 10000
    attributes:
      id1: "9132223344"
      account: "00000"

steps:
  - type: ADVANCED
    id: 1
    function: "check_subscriber"
    local-response: "check.xml"
  - type: SLEEP
    sleep: 2000
  - type: PAYMENT
    id: 1
  - type: STATUS
    id: 1
```

7.2.3. ПАРАМЕТР OFFSET-ID

Возможные значения OFFSET-ID:

Значение	Поведение
-1	Автоматический сдвиг = **`System.currentTimeMillis()`**

Значение	Поведение
0	Без сдвига
>0	Фиксированный сдвиг (прибавляется к ID операции)

7.2.4. ТИПЫ ШАГОВ

Описание типов секции STEPS:

Тип	Описание
SLEEP	Пауза (мс)
CHECK	Проверка реквизитов
PAYMENT	Платёж
STATUS	Запрос статуса
BALANCE	Запрос баланса
ADVANCED	Вызов advanced-функции
CANCEL	Отмена платежа
CONFIRM	Подтверждение платежа

7.2.5. ЛОКАЛЬНЫЕ ОТВЕТЫ (LOCAL-RESPONSE)

Для отладки можно подставить локальный файл с ответом провайдера:

```
- type: ADVANCED
  id: 1
  function: "check"
  local-response: "check_response.xml"
```

Если в запросе задан **`response-type: HTTPCODE`**, код будет взят из HTTP-статуса в файле.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А. БЫСТРЫЙ СТАРТ: ЧЕК-ЛИСТ ЗА 5 МИНУТ

1. Подготовка

- Получить JSON-схему у поставщика;
- Настроить IDEA (см. раздел 1.4).

2. Минимальный конфиг

- Заполнить **`servers`** (массив, минимум один URL);
- Задать **`logger;`**
- Описать **`codes`** (success/error/unknown);
- Создать **`requests`** с **`check`** и **`pay.`**

3. Velocity-шаблоны

- Создать **`check.vsl`** и **`pay.vsl;`**
- Проверить **`$payment.account.identifier`** и **`$payment.realSumOutcome.`**

4. Запуск

- Разместить файлы на сервере;
- Прописать в **`gates.xml;`**
- **`service payloadc restart.`**

5. Проверка

- Открыть кабинет → Диспетчерская → Поиск платежа;
- Найти тестовый платёж;
- Проверить логи шлюза.

ПРИЛОЖЕНИЕ В. СПРАВОЧНИК ТИПОВ ЗАПРОСОВ

Тип запроса	Где используется	Статус
`check`	Проверка реквизитов	✅
`pay`	Платёж	✅
`status`	Статус	✅
`confirm`	Подтверждение	✅
`cancel`	Отмена	✅
`cancel-status`	Статус отмены	✅
`balance`	Баланс	✅
`advanced`	Произвольные функции	✅
`callback`	Входящие уведомления	✅

ПРИЛОЖЕНИЕ C. СПРАВОЧНИК МЕТОДОВ ЗАПРОСОВ

Метод	Описание	Поддержка
GET	GET-запрос	✅
POST	POST-запрос	✅
PUT	PUT-запрос	✅
DELETE	DELETE-запрос	✅
PATCH	PATCH-запрос	✅
BODY	Запрос с телом (Velocity)	✅
RMI	Вызов другого шлюза	✅
BLANK	Без вызова провайдера	✅
SQL	Запрос к базе данных	✅

ПРИЛОЖЕНИЕ D. СПРАВОЧНИК СТАТУСОВ ПЛАТЕЖА

Статус в codes	Описание
success	Успех

Статус в codes	Описание
`error`	Ошибка (нефинальная)
`unknown`	Проведение — Неизвестен
`process`	Проводится
`confirm`	Требуется подтверждение
`token`	Ошибка токена
`chargeback`	Возврат / опровержение
`callback`	Callback-уведомление

ПРИЛОЖЕНИЕ Е. КОДЫ ОШИБОК ADVANCED-ЗАПРОСОВ

Код	Константа	Описание
0	SUCCESS	Успех
1	NOT_FOUND	Абонент не найден
3	PROVIDER_ANSWER_ERROR	Невозможно разобрать ответ провайдера
5	NO_DEBTS	Задолженность отсутствует

Код	Константа	Описание
9	TEXT_ERROR	Текстовая ошибка (содержит message)

ПРИЛОЖЕНИЕ F. ШПАРГАЛКА: XPATH/JSON-PATH

Язык	Пример	Описание
XPath	`/RESPONSE/CODE`	Абсолютный путь
XPath	`//DATA/ACCOUNT`	Любой вложенный элемент
XPath	`/RESPONSE/DATA[STATUS='OK']`	Условие
JSONPath	`$.code`	Корневой объект, поле code
JSONPath	`$.items[*].name`	Все элементы массива, поле name
JSONPath	`$.items[?(@.type=='RUB')]`	Фильтр

ПРИЛОЖЕНИЕ G. ГЛОССАРИЙ

Термин	Определение
Advanced-функция	Произвольный запрос, вызываемый из сценария или другой функции

Термин	Определение
Velocity (VSL)	Язык шаблонов для формирования тела запроса
Рендер (render)	Динамическая подстановка параметров, настроенная в кабинете
Селектор (selector)	Экран на точке с выбором одного значения из списка
RMI-запрос	Перенаправление запроса в шлюз другого провайдера
Токен (token)	Автоматически обновляемый ключ доступа к API провайдера
Callback	Входящее уведомление от провайдера
Subscription	Автоматическое создание счёта для абонента
Client-fee	Динамическое добавление комиссии к параметрам ответа
Conditional sequence	Условная последовательность вызова advanced-функций