Platius. POS API v2 (simplified)

## История изменений

|  |  |  |
| --- | --- | --- |
| **Дата** | **Автор** | **Изменение** |
|  | Сухоносенко К.В | Исходная версия. Описание сценариев. |
|  |  |  |
|  |  |  |
|  |  |  |

## Содержание

[История изменений](#h.gjdgxs)

[Содержание](#h.30j0zll)

[Преамбула](#h.1fob9te)

[Структура системы](#h.3znysh7)

[Сценарии интеграции](#h.3mp0xrdf5lst)

[Fastfood/retail](#h.tyjcwt)

[Tableservice](#h.3dy6vkm)

[Предусловие](#h.1t3h5sf)

[Сценарий - чекин в чек для накопления бонусов c помощью приложения platius](#h.4d34og8)

[Сценарий - оплата бонусами с помощью приложения platius и разового платежного кода](#h.2s8eyo1)

[Сценарий - оплата бонусами и банковской картой c помощью приложения platius и разового платежного кода](#h.17dp8vu)

[Сценарий - отмена пречека](#h.prdnfjxj9aha)

[Сценарий - нехватка средств на счете (бонусы и/или банковская карта)](#h.3rdcrjn)

[Сценарий - оплата бонусами и банковской картой c помощью приложения platius и промо-кода](#h.26in1rg)

[Сценарий - возврат заказа](#h.lnxbz9)

[Сценарий – перебор платежных кодов](#h.35nkun2)

[Сценарий - чекин в чек для накопления бонусов c помощью пластиковой карты](#h.1ksv4uv)

[Сценарий – оплата+скидки c помощью пластиковой карты](#h.44sinio)

[Сценарий - работа кассы без интернета](#h.2jxsxqh)

[Учет совместимости товарных позиций и скидок и бонусов](#h.jwjv14w7lhg1)

[Прочие требования](#h.z337ya)

[Прочие возможности – не реализовано](#h.byp83mphf3et)

[Методы и структуры](#h.wxffwh9u9rz)

[Пример - основной сценарий retail-fastfood](#h.4i7ojhp)

[Пример - основной сценарий table-service](#h.mziqastbn2w3)

[Базовые интерфейсы](#h.2xcytpi)

[Параметры подключения](#h.1ci93xb)

[Исключения](#h.1wzkvhcio40n)

[Заказ и связанные структуры](#h.3whwml4)

[Данные пользователя](#h.2bn6wsx)

[Чекин](#h.qsh70q)

[Результаты работы акций (скидки)](#h.3as4poj)

[Оплаты и возврат](#h.1pxezwc)

[Закрытие заказа](#h.49x2ik5)

[Приложение 1: Форматирование чеков](#h.v7nou9nivun)

[Описание формата XML](#h.po9yr5ajmd3x)

[Управление шрифтами](#h.576rlq2cqso5)

[Управление выравниванием](#h.7auq46o7kc3e)

[Управление переносом текста](#h.n66f5rxxwsii)

[Прочие теги форматирования](#h.voythgo2hx5d)

[Пример использования](#h.ufcmgaw9akf8)

[Приложение 2: Авторизация гостей](#h.i0cw02q1o1xc)

[Приложение 3: Получение данных о платежах](#h.305hpfdat8f0)

## Преамбула

Документ описывает прикладной программный интерфейс взаимодействия системы лояльности platius и внешних кассовых систем.

Документ описывает:

* структуру системы
* сценарии взаимодействия касса-гость-кассир-platius
* программный интерфейс (API) platius
* примеры использования API

## Структура системы

Система используется:

* Мерчантом – владельцем заведения
* Гостем – человеком, кто пользуется скидками и бонусами в различных заведениях
* Кассиром
* Официантом

Система состоит из:

* Сервера platius
* Локального модуля platius, устанавливаемого на кассу
* Опционального, локального для заведения (точки продаж) модуля-агрегатора (POS-сервер)
* Кассового ПО

Кассовое ПО использует API, описываемый в документе, для взаимодействия с системой platius.

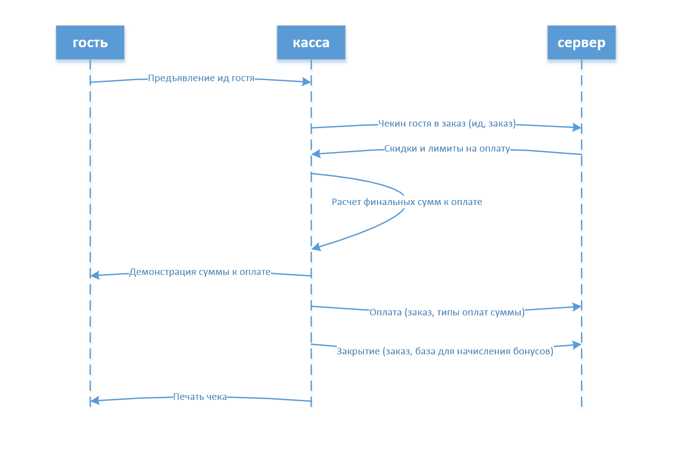
Взаимодействие с системой platius идет по защищенному протоколу https (порт 9010) через сеть Интернет.

POS-сервер осуществляет маршрутизацию вызовов касс к серверу platius и выполняет кеширование данных, которые могут быть получены без обращения к серверу platius (пул промокодов, настройки организации, тексты offline-чеков)

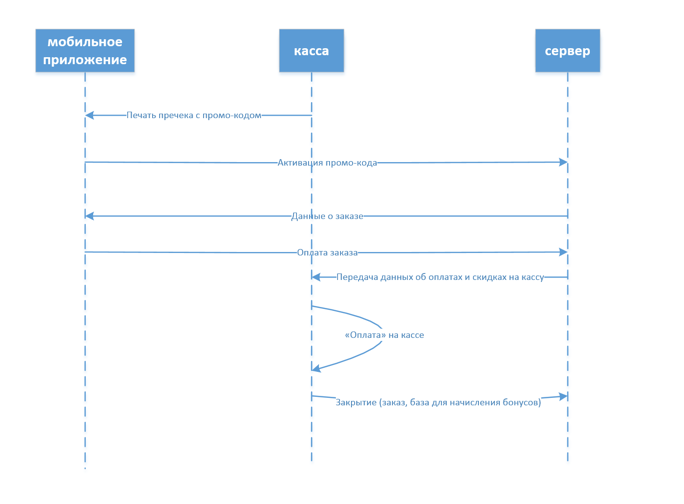
## Сценарии интеграции

На верхнем уровне сценарии относятся к поведению 2-х типов – для заведений с обслуживанием (пречеком) и заведений без обслуживания (фастфуд, ритейл). Смотри диаграммы ниже.

### Fastfood/retail



### Tableservice



### Предусловие

* у гостя есть мобильное приложение [platius](http://www.google.com/url?q=http%3A%2F%2Fiiko.net%2F&sa=D&sntz=1&usg=AFQjCNEZNa-kRHOSq6AMyOoCbrzhzLirgg)
* или у гостя есть пластиковая карта
* к аккаунту гостя может быть привязана банковская карта
* у мерчанта работает сочетание бонусной и скидочной системы. Например, штатно работают бонусы (пороговая схема), но периодически он выдает какие-то подарки.

### Сценарий - чекин в чек для накопления бонусов c помощью приложения platius

* **Применимо: fast-food + table service**
* **Предусловие: гость еще не установил приложение, и POS сообщает гостю о системе platius через данные на чеке.**
* касса закрывает заказ и печатает чек, с информацией о программе лояльности данного мерчанта
* гость видит чек с подвалом platius и инструкциями по присоединению к системе лояльности
* гость скачивает приложение и активирует промо-код с чека
* приложение показывает гостю сумму его чека и кол-во бонусов, которые ему начислены и которыми он может расплатиться в следующем заказе
* **Методы: IBasicWorkflow.Close()**

### Сценарий - оплата бонусами с помощью приложения platius и разового платежного кода

* **Применимо: в основном fast-food, но можно использовать и в table service**
* **Предусловие: у гостя установлено приложение platius, и оплата банковской картой не подключена**
* кассир набрал заказ, спрашивает "как будете платить", гость отвечает "[platius](http://www.google.com/url?q=http%3A%2F%2Fiiko.net%2F&sa=D&sntz=1&usg=AFQjCNEZNa-kRHOSq6AMyOoCbrzhzLirgg)"
* кассир выбирает этот тип оплаты, вводит вручную или сканирует РПК с экрана приложения platius
* касса получает скидки и оплаты бонусами от platius
* кассир нажимает Оплатить
* печатается чек, на нем печатается подвал "спасибо за использование системы лояльности <merchant>", возможно печатаются данные о госте. Эти данные возвращает сервер platius.
* если текст может быть получен без обращения на сервер (все данные есть локально) - то так и будет сделано.
* заказ отправляется на сервер для начисления бонусов
* **Методы: Checkin(), Pay(), Close()**

### Сценарий - оплата бонусами и банковской картой c помощью приложения platius и разового платежного кода

* **Применимо: в основном fast-food, но можно использовать и в table service**
* **Предусловие: у гостя установлено приложение platius, и оплата банковской картой подключена, и гость привязал банковскую карту**
* начало сценария такое же как при оплате бонусами без карты
* после выбора типа оплаты platius на кассу возвращаются скидки и оплаты из platius
* отличие от сценария без карты в том, что на кассу возвращается массив из двух "оплат": оплата бонусами (мин сумма - макс сумма), оплата картой (мин сумма - макс сумма).
* касса может выставить свои суммы к оплате в рамках переданных диапазонов – например отработать совместимость бонусов и блюд в заказе или отработать правила округления.
* кассир нажимает оплатить и заказ закрывается.
* **Методы: Checkin(), Pay(), Close()**

### Сценарий - отмена пречека

TODO - при этом оплата в мобилке уже сделана?

### Сценарий - нехватка средств на счете (бонусы и/или банковская карта)

* при оплате обнаруживается, что средств недостаточно, о чем сообщается кассиру и гостю
* касса показывает кассиру ошибку - недостаточно средств на кассе
* кассир удаляет тип оплаты "[platius](http://www.google.com/url?q=http%3A%2F%2Fiiko.net%2F&sa=D&sntz=1&usg=AFQjCNEZNa-kRHOSq6AMyOoCbrzhzLirgg) - банк карта"
* закрывает заказ за другие типы оплаты
* **Методы: Pay(), Close()**

### Сценарий - оплата бонусами и банковской картой c помощью приложения platius и промо-кода

* **Применимо: table service**
* **Предусловие: у гостя установлено приложение platius, и оплата банковской картой подключена, и гость привязал банковскую карту**
* касса печатает на пречеке подвал platius с промо-кодом
* гость вводит промо-код в приложение и видит на экране состав и сумму своего заказа
* гость нажимает оплатить
  + если он оплачивает заказ целиком – он может уходить
  + если нет – он оставляет недостающую сумму наличными и тоже уходит
* кассир заходит в заказ и видит переданные на кассу скидки и оплаты
* кассир закрывает заказ и печатает чек
* **Методы: Precheque(), Pay(), Close()**

### Сценарий - возврат заказа

* **Применимо: везде**
* Гость решает вернуть заказ
* Кассир выбирает заказ для возврата и удаляет позиции
* Нажимает Возврат
* **Методы: Cancel()**

### Сценарий – перебор платежных кодов

* **Применимо: все сценарии**
* сервер блокирует работу platius для терминала, с которого идет перебор платежных кодов и регистрирует опасную операцию.
* **Методы: Checkin().**

### Сценарий - чекин в чек для накопления бонусов c помощью пластиковой карты

проработать

### Сценарий – оплата+скидки c помощью пластиковой карты

проработать

### Сценарий - работа кассы без интернета

* Общие требования – при отсутствии связи с сервером platius – кассовые операцию не «тормозят»
* связи с интернетом нет
* касса закрывает заказы и печатает чеки с заранее запасенными промо-кодами и анонимным подвалом. Если запас промо-кодов исчерпан, то они не печатаются
* попытка оплаты возвращает ошибку и не блокируется надолго
* попытка отмены заказа возвращает ошибку и не блокируется надолго, запрос на отмену записывается в очередь.
* касса копит закрытые заказы в очереди на отправку на сервер [platius](http://www.google.com/url?q=http%3A%2F%2Fiiko.net%2F&sa=D&sntz=1&usg=AFQjCNEZNa-kRHOSq6AMyOoCbrzhzLirgg)
* **Методы: Precheque(), Pay(), Cancel(), Close()**

### Учет совместимости товарных позиций и скидок и бонусов

* Кассовая система передает в систему platius данные:
  + о возможности платить бонусами за артикул
  + о минимальной продажной цене за артикул

### Прочие требования

* Обработка округлений
* Предложения подарков и купленных акций
* АПИ будет работать в схеме с разовым платежным кодом, NFC-платежем, интеграцией с пинпадом

### Прочие возможности – не реализовано

* Просмотр баланса счета на кассе
* Пополнение бонусного счета на кассе
* Выписка операций на кассе
* Установка реферальной связи между гостями на кассе
* Привязка карты к телефону гостя на кассе
* Локализация

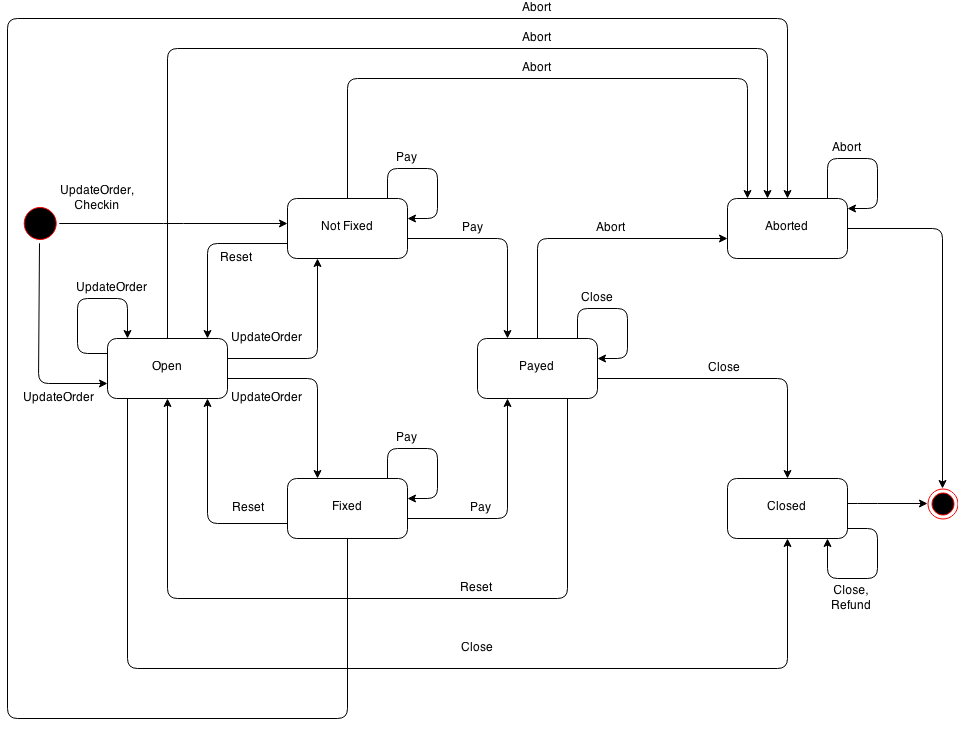
## Методы и структуры

Ниже приведена диаграмма переходов-состояний для заказа с точки зрения клиентской логики.

Выделены следующие состояния:

* Open - заказ создан, но данные о привязке пользователя и оплатах не получены
* NotFixed - данные об оплатах получены в виде лимитов
* Fixed - данные об оплатах получены в виде точных сумм
* Payed - оплаты проведены на стороне кассовой системы
* Closed - заказ без ошибок закрыт на стороне кассовой системы
* Aborted - workflow заказа прервано на стороне кассовой системы

Важно отметить, что в начале работы с заказом нужно вызвать один из инициализирующих методов (UpdateOrder, Checkin), а в конце - один из методов, приводящий заказ в терминальное состояние (Closed или Aborted). Нежелательно оставлять “висящие” заказы.



### Пример - основной сценарий retail-fastfood

|  |
| --- |
| // Задаем параметры, с которыми будет работать клиентская библиотека platius  var startupParams = new StartupParams  {  Host = "platius",  Login = "0001",  Password = "qwerty",  TerminalId = "POS-1",  SyncCallTimeoutSec = 15,  AsyncCallWaitTimeoutSec = 3,  Locale = "ru-Ru",  UseCompression = true,  PosInfo = "POS system, ver.1.0.0 build at 2014.10.14"  };  // Создаем экземпляр клиента, подходящий нам по бизнес-процессу  var api = PlatiusFiiacade.CreateBasicWorkflow(startupParams);  // Создаем заказ с двумя элементами  var item1 = new OrderItem  {  ProductCode = "100100",  ProductName = "Портвейн 777",  Amount = 1,  FixedPrice = 600,  Sum = 600,  SumAfterDiscount = 600  };  var item2 = new OrderItem  {  ProductCode = "200100",  ProductName = "Огурцы свежие",  Amount = 0.53,  FixedPrice = 40,  Sum = 21.2m,  // в магазине действует локальная скидка 50% на овощи  SumAfterDiscount = 10.6m  };  var order = new Order  {  Id = Guid.NewGuid(),  Number = "444",  Sum = item1.Sum + item2.Sum,  SumAfterDiscount = item1.SumAfterDiscount + item2.SumAfterDiscount,  Items = new[] { item1, item2 }  };  // Установим ограничения: спиртные напитки нельзя оплачивать бонусами  // + минимальная цена портвейна 500р  var productLimit = new ProductLimit  {  ProductCode = "100100",  CanBePaidByBonuses = false, // касса учтет это при обработке запроса на доступные лимиты оплат  MinPrice = 500m  };  const string userPaymentToken = "12345";  // выполняем checkin  var checkinResult = api.Checkin(userPaymentToken, UserSearchScope.PaymentToken,  order, new [] { productLimit });  // platius передал скидки, и мы применяем их  AppliedDiscount[] appliedDiscounts = null;  if (checkinResult.LoyaltyResult != null)  {  appliedDiscounts = checkinResult.LoyaltyResult.ProgramResults  .SelectMany(program => program.Operations, (program, operation) => new { program, operation })  .Select(item => new AppliedDiscount  {  ProgramId = item.program.ProgramId,  OperationCode = item.operation.Code,  ProductCode = item.operation.ProductCode,  ProductName = item.operation.ProductName,  Sum = item.operation.DiscountSum  })  .ToArray();  }  // platius передал диапазон возможных оплат  var payments = new List<WalletPayment>();  if (checkinResult.PaymentLimits != null)  {  // Считаем фактические суммы оплат  var sumToPay = GetSumToPay(order);  // Приоритет оплат задается порядком их перечисления  foreach (var limit in checkinResult.PaymentLimits)  {  var payment = new WalletPayment  {  WalletCode = limit.WalletCode,  Sum = Math.Min(sumToPay, limit.MaxSum)  };  // Совершаем оплату только если она больше или равна допустимому лимиту  if (payment.Sum < limit.MinSum)  continue;  AddPaymentToOrder(order, payment.Sum, payment.WalletCode);  payments.Add(payment);  sumToPay -= payment.Sum;  }  }  ASSERT( sumToPay >=0 )  if (sumToPay > 0)  {  // нужна доплата на кассе - не все можно оплатить оплатами с platius  // ….  }  // Выполняем оплату.  // Метод вызывается при выполнении любого из условий:  // - касса проводит оплаты, связанные с platius  // - в заказе есть примененные скидки, связанные с platius  if (payments.Any() || appliedDiscounts != null)  {  // Идентификатор транзакции нужно задавать в клиентском коде,  // т.к. может потребоваться инициировать отмену оплаты  // Например, при возникновении ошибок в оплатах, не связанных с platius  var transactionId = Guid.NewGuid();  // Если в процессе выполнения возникнет ошибка  // - автоматически будет инициирована отмена оплаты  api.Pay(order.Id, transactionId, payments.ToArray(), appliedDiscounts);  }  // Закрытие заказа  // передать его на сервер и указать на какую сумму начислять бонусы  var sumForBonus = GetSumForBonus(order);  var closeResult = api.Close(order, sumForBonus);  // Добавляем в чек информацию, полученную от platius  AppendChequeFooter(order, closeResult.ChequeFooter); |

### Пример - основной сценарий table-service

|  |
| --- |
| // Задаем параметры, с которыми будет работать клиентская библиотека platius  var startupParams = new StartupParams  {  Host = "platius",  Login = "0002",  Password = "qwerty",  TerminalId = "POS-1",  SyncCallTimeoutSec = 15,  AsyncCallWaitTimeoutSec = 3,  Locale = "ru-Ru",  UseCompression = true,  PosInfo = "POS system, ver.1.0.0 build at 2014.10.14"  };  // Создаем экземпляр клиента, подходящий нам по бизнес-процессу  var api = PlatiusFacade.CreateComplexWorkflow(startupParams);  // Создаем заказ с двумя элементами  var item1 = new OrderItem  {  ProductCode = "100100",  ProductName = "Салат",  Amount = 1,  FixedPrice = 600,  Sum = 600,  SumAfterDiscount = 600  };  var item2 = new OrderItem  {  ProductCode = "200100",  ProductName = "Чай",  Amount = 1,  FixedPrice = 50,  Sum = 50,  // в магазине действует локальная скидка 50% на чай  SumAfterDiscount = 25  };  var order = new Order  {  Id = Guid.NewGuid(),  Number = "7",  WaiterName = "Петров Сергей",  Sum = item1.Sum + item2.Sum,  SumAfterDiscount = item1.SumAfterDiscount + item2.SumAfterDiscount,  Items = new[] { item1, item2 }  };  // Установим ограничения: чай нелья оплачивать бонусами  var productLimit = new ProductLimit  {  ProductCode = "200100",  CanBePaidByBonuses = false,  };  // обновляем заказ на сервере (например, при переходе между экранами)  var checkinResult = api.UpdateOrder(order, new[] { productLimit });  // platius передал скидки, и мы применяем их  AppliedDiscount[] appliedDiscounts = null;  if (checkinResult.LoyaltyResult != null)  {  appliedDiscounts = checkinResult.LoyaltyResult.ProgramResults  .SelectMany(program => program.Operations, (program, operation) => new { program, operation })  .Select(item => new AppliedDiscount  {  ProgramId = item.program.ProgramId,  OperationCode = item.operation.Code,  ProductCode = item.operation.ProductCode,  ProductName = item.operation.ProductName,  Sum = item.operation.DiscountSum  })  .ToArray();  }  // может вернуться привязка гостя и оплаты, совершенные через приложение  var payments = new List<WalletPayment>();  if (checkinResult.Payments != null)  {  foreach (var payment in checkinResult.Payments)  {  AddPaymentToOrder(order, payment.Sum, payment.WalletCode);  payments.Add(new WalletPayment { Sum = payment.Sum, WalletCode = payment.WalletCode });  }  }  // пречек  var prechequeTime = DateTimeOffset.Now;  // Выполняем оплату.  // Метод вызывается при выполнении любого из условий:  // - касса проводит оплаты, связанные с platius  // - в заказе есть примененные скидки, связанные с platius  if (payments.Any() || appliedDiscounts != null)  {  // Идентификатор транзакции нужно задавать в клиентском коде,  // т.к. может потребоваться инициировать отмену оплаты  // Например, при возникновении ошибок в оплатах, не связанных с platius  var transactionId = Guid.NewGuid();  // Если в процессе выполнения возникнет ошибка  // - автоматически будет инициирована отмена оплаты  api.Pay(order.Id, transactionId, payments.ToArray(), appliedDiscounts);  }  // Закрытие заказа (все оплаты проведены, но фискальный чек еще не напечатан)  // добавляем поля, появляющиеся при закрытии и указываем на какую сумму  // начислять бонусы  var sumForBonus = GetSumForBonus(order);  order.FiscalChequeNumber = 233;  order.CashierName = "Кудрявцева Ирина";  order.PrechequeTime = prechequeTime;  order.CloseTime = DateTimeOffset.Now;  var closeResult = api.Close(order, sumForBonus);  // Добавляем в чек информацию, полученную от platius  AppendChequeFooter(order, closeResult.ChequeFooter); |

### Пример - сценарий refund

### 

|  |
| --- |
| //Предусловие: На кассе существует Закрытый заказ (был вызван метод API Close). //Гость через сайт оплатил заказ картой и бонусами. А метод UpdateOrder вернул на кассу информацию о том, сколько пользователь платит картой и бонусами. //Касса сама рассчитывает, какой товар получает какую часть этой оплаты, и сохраняет у себя эту информацию, а в будущем передаёт её в метод Refund. #region Prerequisites // Задаем параметры, с которыми будет работать клиентская библиотека var startupParams = new StartupParams { Host = "www4.iiko.net", Login = "1714", Password = "1714", SyncCallTimeoutSec = 15\*60, UseCompression = true, TerminalId = "1", Vendor = "Platius", Product = "C# Test client", ProductVersion = Assembly.GetExecutingAssembly().GetName().Version.ToString(), PluginVersion = Assembly.GetExecutingAssembly().GetName().Version.ToString() }; // Создаем экземпляр клиента, подходящий нам по бизнес-процессу var flow = new ComplexWorkflow(startupParams); // Создаем заказ с двумя элементами var item1 = new OrderItem { ProductCode = "100100", ProductName = "Салат", Amount = 1, Sum = 600, SumAfterDiscount = 600 }; var item2 = new OrderItem { ProductCode = "200100", ProductName = "Чай", Amount = 2, Sum = 50, SumAfterDiscount = 25 }; var order = new Order { Id = Guid.NewGuid(), Number = "1", WaiterName = "Петров Сергей", Sum = item1.Sum + item2.Sum, SumAfterDiscount = item1.SumAfterDiscount + item2.SumAfterDiscount, Items = new[] {item1, item2} }; const string credential = "555444"; flow.Checkin(credential, UserSearchScope.CardNumber, order, null); //Оплачиваем заказ var paymentTransactionId = Guid.NewGuid(); const decimal cardPayment = 200; const decimal bonusPayment = 50; flow.Pay(order.Id, paymentTransactionId, new[] { new WalletPayment {WalletCode = Consts.BonusWalletCode, Sum = bonusPayment}, // new WalletPayment {WalletCode = Consts.PaymentCardWalletCode, Sum = cardPayment} //В данном примере оплата картой закомментирована, потому что нельзя платить картой с таким способом авторизации. }, null); //Закрываем заказ - без этого рефанд работать не будет flow.Close(order, cardPayment); #endregion #region Refunds //В первом рефанде отменяем одну позицию item2. //Касса считает, сколько средств им надо вернуть (вероятно, пропорционально сумме оплаты). const decimal firstBonusRefundSum = 30; //Идентификатор первой операции refund. Для каждой Успешной операции нужен новый идентификатор. Для повтора Неуспешной операции (предыдущий вызов кинул исключение) используется этот же идентификатор. var firstRefundId = Guid.NewGuid(); var firstRefundResult = flow.Refund(order.Id, firstRefundId, new[] { new WalletRefund {Sum = firstBonusRefundSum, WalletCode = Consts.BonusWalletCode}, }, new[] { new CancelledOrderItem {Amount = 1, ProductCode = item2.ProductCode} }); var chequeFooter = firstRefundResult.ChequeFooter; Console.WriteLine(chequeFooter); //Во втором рефанде отменяем все оставшиеся товары (передавая null) const decimal secondBonusRefundSum = bonusPayment - firstBonusRefundSum; //const decimal secondCardRefundSum = cardPayment; //Идентификатор второй операции refund. var secondRefundId = Guid.NewGuid(); var secondRefundResult = flow.Refund(order.Id, secondRefundId, new[] { new WalletRefund {Sum = secondBonusRefundSum, WalletCode = Consts.BonusWalletCode}, // new WalletRefund {Sum = secondCardRefundSum, WalletCode = Consts.PaymentCardWalletCode} //Возврат средств на карту закомментирован по той же причине, что и оплата. }, //null - значит полную отмену всех товаров. Сумму при этом касса всё равно должна передать. null); chequeFooter = secondRefundResult.ChequeFooter; Console.WriteLine(chequeFooter); #endregion |

### Базовые интерфейсы

// Интерфейс работы без пречека (ритейл, фастфуд).

**public interface IBasicWorkflow**

**{**

//Состояние подключения к серверу platius.

// Может использоваться для показа предупреждающих сообщений.

// на основе этого флага нельзя делать вывод как завершился какой-либо иной метод АПИ - синхронно

// или с постановкой запроса в очередь

**bool IsOnline { get; }**

// Чекин гостя в заказ – связывание гостя с заказом

// и одновременное получение результата работы скидочных механик

// и диапазона допустимых оплат.

// Синхронный.

**ICheckinResult Checkin(string credential, UserSearchScope searchScope, Order order, ProductLimit[] limits);**

// Оплата заказа.

// Синхронный.

// При таймауте автоматически ставит запрос на отмену оплаты в очередь.

// Повторный вызов, в случае если предыдущий находится в очереди, вернет ошибку Call\_in\_progress

// TransactionId - уникальный идентификатор, генерируется на стороне кассовой системы

// AppliedDiscount - скидки, применные кассовой системой, из числа предложенных platius

**void Pay(Guid orderId, WalletPayment[] payments, AppliedDiscount[] discounts);**

// Отмена оплаты.

// Асинхронный.

// Может вызываться в случае любых исключений или при возникновении неопределенности при вызове

// метода Pay. Другой вариант использования - несколько раз попытаться вызвать Pay (идемпотентен)

// и если не получилось - CancelPay

**void CancelPay(Guid orderId, Guid transactionId);**

// Полный или частичный возврат заказа, по одному заказу возможно несколько частичных возвратов

// Синхронный.

// Ставится в очередь, если невозможно выполнить за отведенное время.

// Повторный вызов, в случае если предыдущий находится в очереди, вернет ошибку Call\_in\_progress.

// Каждый частичный возврат должен вызываться со своим уникальным transactionId

// Вызовы с разным transactionId считаются различными возвратами

// Касса сама сообщает сколько нужно вернуть на кошельки

**RefundResult Refund(Guid orderId, Guid transactionId, WalletRefund[] refunds, CancelledOrderItem[] cancelledItems);**

// Закрытие чека.

// Синхронно-асинхронный – если не успевает отработать за отведенное время ставиться в очередь

// и выдает оффлайн-чек

// принимает заказ и сумму, на которую надо начислить бонус.

// Обычно, если заказ оплачивается комбинацией типов оплат,

// то бонусная часть оплаты исключается из базы начисления очередных бонусов.

**ICloseResult Close(Order order, decimal SumForBonus);**

// Прерывание обработки флоу (например при аварийном удалении заказа)

// Синхронно-асинхронный.

// Не кидает исключений.

// Отменяет проведенные оплаты, скидки и чекин.

// После отработки этого метода можно проводить повторный чекин.

// Заказ после успешного Close() не может быть Abort()-ed.

**Abort() nothrow()**

// Получить оплаты, совершенные на стороне platius (через мобильное приложение).

// Синхронный, выполняет быстрый запрос по данным, закешированным на клиенте platius.

// 1) Если orderIds == null или пуст - возвращает данные по всем активным оплатам,

// про которые знает клиент. Может использоваться для начальной инициализации кассы

// 2) Если в OrderIds задан один или более идентификаторов заказа - возвращает данные

// только по этим заказам. Может использоваться для начальной инициализации кассы,

// смены вида отображения и т.п., когда известно, информацию по каким заказам нужно получить

**OrderPayments[] GetPayments(Guid[] orderIds)**

**}**

// Интерфейс для процессов, предполагающих наличие пречека (ресторан с обслуживанием).

**public interface IComplexWorkflow : IBasicWorkflow**

**{**

// Получает данные для пречека.

// Обновляет данные о заказе на сервере.

// Принимает данные о привязке гостя к заказу (гость сделал чекин на сервере через приложение).

// Синхронный

**ICheckinResult UpdateOrder(Order order, ProductLimit[] limits);**

**}**

### Параметры подключения

**public sealed class StartupParams**

**{**

// Адрес сервера platius.

**public string Host { get; set; }**

// Логин и пароль для подключения к серверу.

**public string Login { get; set; }**

**public string Password { get; set; }**

// Таймаут (в секундах) на вызов серверного метода.

// Рекомендуется оставить значение по умолчанию.

**public int SyncCallTimeoutSec { get; set; }**

// Таймаут (в cекундах) завершения синхронного вызова метода, который может быть

// выполнен асинхронно.

// Например, если метод Close завершится за установленное время - вызывающему коду вернется

// результат, посчитанный сервером Platius. Если не завершится - результат для offline-режима обработки, но

// выполнение метода будет продолжено асинхронно.

**public int AsyncCallWaitTimeoutSec { get; set; }**

// Использовать ли сжатие тела http-запросов.

// Для production-использования рекомендуется оставлять значение по умолчанию (true).

**public bool UseCompression { get; set; }**

// Язык работы системы (согласно RFC 5646).

**public string Locale { get; set; }**

// Номер терминала в рамках заведения.

**public string TerminalId { get; set; }**

// Ширина кассовой ленты в символах.

**public int TapeWidth { get; set; }**

// true - на кассу будут отправлены чеки в XML виде

// false - чек будет отформатирован на сервере, касса печатает только QR-код

**public bool UseXMLCheques { get; set; }**

// Идентификатор кассововго терминала

**public string TerminalId { get; set; }**

// Разработчик кассового ПО (Например: iiko, UCS).

**public string Vendor { get; set; }**

// Название продукта (Например: R-Keeper 6, iikoFront, Intellect Style).

**public string Product { get; set; }**

// Версия кассового ПО (Например: 3.1.2.3).

**public string ProductVersion { get; set; }**

// Версия библиотеки (плагина) Platius (Например: 3.1.2.3).

**public string PluginVersion { get; set; }**

**}**

### Исключения

Все исключения от сервера Platius приводятся к HttpException, который всегда содержит локализованное сообщение об ошибке (с локалью, которая была указана при создании StartupParameters). Для обработки исключения важным является только это поле.

### Заказ и связанные структуры

// Тип заказа.

**public enum OrderType**

**{**

**Unknown = 0,**

**FastFoodOrder = 1,**

**TableServiceOrder = 2,**

**DeliveryOrder = 3,**

**SelfServiceOrder = 4,**

**PetroleumOrder = 6,**

**BanquetOrder = 7,**

**RetailOrder = 8**

**}**

**public sealed class Order**

**{**

// Уникальный guid заказа в platius – генерируется кассовой системой.

**public Guid Id { get; set; }**

// Тип заказа

**public OrderType OrderType {get; set;}**

// Состав заказа.

**public OrderItem[] Items { get; set; }**

// Сумма заказа после скидок кассовой системы (не включает скидки Platius).

**public decimal SumAfterDiscount { get; }**

// Сумма заказа по прайсу.

**public decimal Sum { get; }**

// Номер стола в table service.

**public string TableNumber { get; set; }**

// Номер заказа в POS-системе (порядковый номер в смене).

**public string Number { get; set; }**

// Номер фискального чека.

**public int? FiscalChequeNumber { get; set; }**

// Число гостей в заказе.

**public int GuestsCount { get; set; }**

// Имя официанта.

**public string WaiterName { get; set; }**

// Имя кассира.

**public string CashierName { get; set; }**

// Дата-время создание заказа.

**public DateTimeOffset OpenTime { get; set; }**

// Время пречека.

**public DateTimeOffset? PrechequeTime { get; set; }**

// Время когда заказ оплачен и напечатан чек.

**public DateTimeOffset? CloseTime { get; set; }**

// Дополнительные параметры заказа - через них можно передавать

// признаки для заказа и настраивать логику реагирования на них через маркетинговые акции

**public Dictionary<string, string> AdditionalData { get; }**

**}**

**public sealed class OrderItem**

**{**

// Код товара в POS-системе (артикул).

**public string ProductCode { get; set; }**

// Название товара в POS-системе.

**public string ProductName { get; set; }**

// Название категории или список категорий одной строкой.

**public string ProductCategory { get; set; }**

// Количество позиций с данным артикулом.

**public decimal Amount { get; set; }**

// Стоимость всех позиций с данным артикулом до дисконта (по прайсу).

**public decimal Sum { get; set; }**

// Стоимость всех позиций с данным артикулом после применения

// локальных скидок POS-системы.

**public decimal SumAfterDiscount { get; set; }** // сумма после дисконта - к оплате

**}**

// Возвращаемая позиция чека при частичном возврате.

**public sealed class CancelledOrderItem**

**{**

// Артикул возвращаемой позиции.

**public string ProductCode { get; set; }**

// Количество возвращаемых позиций.

**public decimal Amount { get; set; }**

**}**

// Ограничения, накладываемые кассовой системой на продажу продукта.

**public sealed class ProductLimit**

**{**

// Артикул продукта.

**public string ProductCode { get; set; }**

// Минимальная цена продукта.

**public decimal MinPrice { get; set; }**

// Можно ли оплатить боунсами.

**public bool CanBePaidByBonuses { get; set; }**

**}**

### Данные пользователя

// Информация о госте.

**public sealed class UserData**

**{**

**public string FullName { get; set; }**

**public DateTime? Birthday { get; set; }**

**public string PhotoUrl { get; set; }**

**}**

// Способы идентификации гостя.

**public enum UserSearchScope**

**{**

**Reserved = 0,**

// по номеру телефона

**Phone = 1,**

// по номеру карты

**CardNumber = 2,**

// по треку карты

**CardTrack = 4,**

// по разовому платежному коду

**PaymentToken = 16,**

**}**

### Чекин

**public interface ICheckinResult**

**{**

// Данные пользователя, который сделал чекин.

**UserData UserData { get; }**

// Скидки по заказу.

**LoyaltyResult LoyaltyResult { get; }**

// Допустимые лимиты оплаты.

// Касса получает лимиты, в рамках которых она сама может устанавливать требуемые суммы оплаты,

// исходя из правил округления, сочетания бонусов и категорий блюд и тп

// касса НЕ может задавать значения платежей, которые находятся вне предложенных лимитов

// лимиты имеют приоритет

// сначала касса должна расходовать суммы с первого из переданных кошельков (обычно это бонусы),

// затем со второго (банковская карта) и тп.

// если массив пуст – гость решил не платить ни с одного кошелька

**PaymentLimit[] PaymentLimits { get; }**

// Уже совершенные гостем оплаты (из мобильного приложения, с сайта platius).

**Payment[] Payments { get; set; }**

// Форматированный под размер подвал для пречека.

**string ChequeFooter { get; }**

**}**

### Результаты работы акций (скидки)

**public sealed class LoyaltyResult**

**{**

// Сумма начисленных бонусов (поле валидно только после закрытия заказа).

**public decimal TotalBonus { get; set; }**

// Список результатов отработки маркетинговых акций.

**public LoyaltyProgramResult[] ProgramResults { get; set; }**

**}**

**public sealed cless LoyatyProgramResult**

**{**

// Идентификатор акции - нужен, чтобы отправить обратно на сервер

// информацию о примененных скидках.

**public Guid ProgramId { get; set; }**

// Наименование акции

**public string Name { get; set; }**

// Операции, применяемые к отдельным элементам заказа

**public LoyaltyOperation[] Operations { get; set; }**

**}**

**public sealed class LoyaltyOperation**

**{**

// Строковый код (тип) операции

**public string Code { get; set; }**

// Код продукта (в кассовой системе, артикул).

**public string ProductCode { get; set; }**

// Наименование продукта

**public string ProductName { get; set; }**

// Сумма скидки для продукта от данной акции.

**public decimal DiscountSum { get; set; }**

// Комментарии к блюду от программы лояльности (происхождение скидки и т.п.)

**public string Comment { get; set; }**

**}**

### Оплаты и возврат

// Коды кошельков – зарезервированные значения

// бонусный кошелек

// и кошелек привязанной банковской карты.

**const string WalletBonus = "bonus"**

**const string WalletPaymentCard = "paymentCard"**

// Ограничения на оплату с кошелька.

**public sealed class PaymentLimit**

**{**

// Код кошелька.

**public string WalletCode { get; set; }**

// Минимально допустимая сумма оплаты с указанного кошелька

// для привязанной банковской карты может быть больше нуля

// из-за минимальной комиссии за эквайринг.

**public decimal MinSum { get; set; }**

// Максимально допустимая сумма к оплате с кошелька

// для бонусного кошелька может фиксироваться максимальный процент суммы чека

// разрешенный для оплаты бонусами.

**public decimal MaxSum { get; set; }**

**}**

// Оплата с кошелька.

**public sealed class WalletPayment**

**{**

// Код кошелька.

**public string WalletCode { get; set; }**

// Сумма оплаты.

**public decimal Sum { get; set; }**

**}**

// Информация по оплатам заказа, совершенным на стороне platius

**public sealed class OrderPayments**

**{**

// Идентификатор зказа

**private Guid OrderId { get; set; }**

// Совершенные оплаты

**private WalletPayment[] Payments { get; set; }**

**}**

// Возврат на кошелек.

**public sealed class WalletRefund**

**{**

// Код кошелька.

**public string WalletCode { get; set; }**

// Сумма возврата.

**public decimal Sum { get; set; }**

**}**

// Результат обработки полного или частичного возврата заказа.

**public sealed class RefundResult**

**{**

// Отформатированный подвал чека..

**public string ChequeFooter { get; set; }**

**}**

// Скидки, примененная кассовой системой

**public sealed class AppliedDiscount**

**{**

// Идентификатор акции

**public Guid ProgramId { get; set; }**

// Строковый код операции

**public string OperationCode { get; set; }**

// Код продукта (в кассовой системе, артикул)

**public string ProductCode { get; set; }**

// Наименование продукта

**public string ProductName { get; set; }**

// Сумма скидки.

**public decimal Sum { get; set; }**

// Примечания/пояснения по скидке

**public string Comment { get; set; }**

**}**

### Закрытие заказа

**public interface ICloseResult**

**{**

// Результат работы программы лояльности

// данные о реальных начисленных бонусах и отработавших акциях.

**LoyaltyResult LoyaltyResult { get; }**

// Отформатированный подвал чека.

**string ChequeFooter { get; }**

**}**

## Приложение 1: Форматирование чеков

В зависимости от параметра **StartupParams.UseXMLCheques** сервер может отдавать 2 вида чеков:

1. XML формат, с управлением размером шрифтов, выравниванием и т.п.
2. Текст, который может содержать в себе XML тега **qrcode** и **barcode.** Если касса не поддерживает qr-коды и баркоды, то эти теги она должна удалить сама.

### Описание формата XML

Документ является валидным xml, с корневым тэгом doc.

**Пример:**

<doc>текст для печати</doc>

**Вывод:**

текст для печати

По умолчанию используется шрифт 0, выравнивание текста по левому краю, способ переноса текста, при котором разбивается в любом месте.

### Управление шрифтами

Имеется 3 различных размера шрифта, нумеруемых цифрами от 0 до 2, в порядке увеличения размера, т.е. шрифт номер 0 — самый маленький, а шрифт номер 2 — самый большой. Все теги, указанные в таблице 2, меняют шрифт для всех вложенных в них элементов.

*Теги для управления шрифтами*

|  |  |
| --- | --- |
| **Название** | **Назначение** |
| <f0></f0> | Форматирование вложенного контента шрифтом размера 0. |
| <f1></f1> | Форматирование вложенного контента шрифтом размера 1. |
| <f2></f2> | Форматирование вложенного контента шрифтом размера 2. |
| <barcode></barcode> | Форматирование вложенного текста как штрихкод. Поддерживает только текст. Если устройство не поддерживает вывод штрихкодов, то данный тег будет проигнорирован. |
| <qrcode></qrcode> | Форматирование вложенного текста как двумерный штрихкод (QR-код). Поддерживает только текст. Опциональны атрибуты размера size QR-кодаи степени помехоустойчивости correction. Пример тега: <qrcode size="normal" correction="medium"> http://www.Platius.ru/ </qrcode> |

Атрибуты тега **<qrcode>**

|  |  |  |
| --- | --- | --- |
| **Название** | **Возможные значения** | **Назначение** |
| size | tiny, small, normal, large, extralarge | Размер QR-кода |
| correction | low, medium, high, ultra | Степень коррекции (помехоустойчивости). Для малого размера не следует указывать значения ultra/high, т.к. это может привести к ошибке распознавания QR-кода. |

### 

### Управление выравниванием

По умолчанию в документе задано выравнивание по левому краю. В таблице представлены теги позволяющие менять способ выравнивания для всех вложенных в них элементов.

*Теги для управления выравниванием*

|  |  |
| --- | --- |
| **Название** | **Назначение** |
| <center></ center> | Выравнивание по центру. |
| <left></left> | Выравнивание по левому краю. |
| <right></right> | Выравнивание по правому краю. |

### Управление переносом текста

Теги указывающие способ переноса текста для всех вложенных в них элементов.

*Теги для управления переносом текста*

|  |  |
| --- | --- |
| **Название** | **Назначение** |
| <split></ split> | Разбить текст на строки, перенося на пробельных символах (пробел, табуляция и др.). |
| <wrap></wrap> | Разбить текст на строки, перенося в любом месте. |
| <br /> | Переход на новую строку. Переход осуществится только если текущая строка пустая. Любое количество подряд-идущих <br /> на выходе дадут одну пустую строку. |

### Прочие теги форматирования

Будем обозначать @attribute значение атрибута attribute из исходного тега.

|  |  |
| --- | --- |
| **Тег** | **Описание** |
| <line /> | Выводит одну строку заполненную символом @symbols. По умолчанию @symbols=’-’, то есть в результате применения тега <line /> без атрибутов, на выходе получится горизонтальная линия на всю ширину документа. |
| <pair /> | Выводит пару значений @left, @right выровненные влево и вправо соответственно в одну строку. |

### Пример использования

В XML формате:

<doc>

<left>Ранг 1</left>

<line symbols=”\*”/>

<f1>Внимание! ПОДАРОК!</f1>

<f0>Сбарро Октябрьcкая дарит Вам</f0>

<f0>15% от суммы этого заказа!</f0>

<f0>А в рублях это 7,50 р.</f0>

<left>Зайдите на</left>

<f1>www.platius</f1>

<left>и введите промо-код:</left>

<f1>5012#5640</f1>

<left>или отправьте sms на номер 3299</left>

<left>"bonus 5012#5640"</left>

<left>ДЕЙСТВУЕТ В ТЕЧЕНИЕ 15 МИНУТ!</left>

<qrcode>http://platius/PromoCode/Activate/5012-5640</qrcode>

<pair left="27.01.2015" right="16:45" />

<br/>

<barcode>1234567890</barcode>

<center>

<f2>Текст по центру</f2>

</center>

<br/>

<right>Текст справа</right>

<br/>

<left>Текст слева</left>

</doc>

В формате только с QR кодами:

Ранг 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Внимание! ПОДАРОК!

Сбарро Октябрьcкая дарит Вам

15% от суммы этого заказа!

А в рублях это 7,50 р.

Зайдите на

www.platius

и введите промо-код:

5012#5640

или отправьте sms на номер 3299

"bonus 5012#5640"

ДЕЙСТВУЕТ В ТЕЧЕНИЕ 15 МИНУТ!

<qrcode>http://platius/PromoCode/Activate/5012-5640</qrcode>

27.01.2015 16:45

<br/>

<barcode>1234567890</barcode>

Текст по центру

Текст справа

Текст слева

## 

## 

## Приложение 2: Авторизация гостей

## 

## 

## Приложение 3: Получение данных о платежах

Данные о платежах должны отображаться кассиру с минимальной задержкой, относительно момента времени, когда гость оплатил заказ в приложении.

Клиентская библиотека сама реализует логику быстрого получения информации о платеже и сохранении его в локальный кеш (“переживает” рестарты библиотеки).

Для чтения локального кеша данных о платежах используется GetPayments(...).