iikoNET. POS API v2 (simplified)

## История изменений

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

## Содержание

История изменений

Содержание

Преамбула

Структура системы

Сценарии интеграции

Fastfood/retail

Tableservice

Предусловие

Сценарий - чекин в чек для накопления бонусов c помощью приложения iiko.net

Сценарий - оплата бонусами с помощью приложения iiko.net и разового платежного кода

Сценарий - оплата бонусами и банковской картой c помощью приложения iiko.net и разового платежного кода

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

Сценарий - оплата бонусами и банковской картой c помощью приложения iiko.net и промо-кода

Сценарий - возврат заказа

Сценарий – перебор платежных кодов

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

Сценарий – оплата+скидки c помощью пластиковой карты

Сценарий - временные сбои в сети

Сценарий - работа кассы без интернета

Сценарий - нет интернета или сервер недоступен и пул оффлайн промо кодов исчерпан

Прочие требования

Методы и структуры

Пример - основной сценарий

Базовые интерфейсы

Параметры подключения

Заказ и связанные структуры

Данные пользователя

Результаты работы акций (скидки)

Оплаты и возврат

Чекин

Закрытие заказа

## Преамбула

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

Документ описывает:

* структуру системы
* сценарии взаимодействия касса-гость-кассир-iiko.net
* программный интерфейс (API) iiko.net
* 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 iiko.net
* Локального модуля iiko.net, 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 iiko.net.

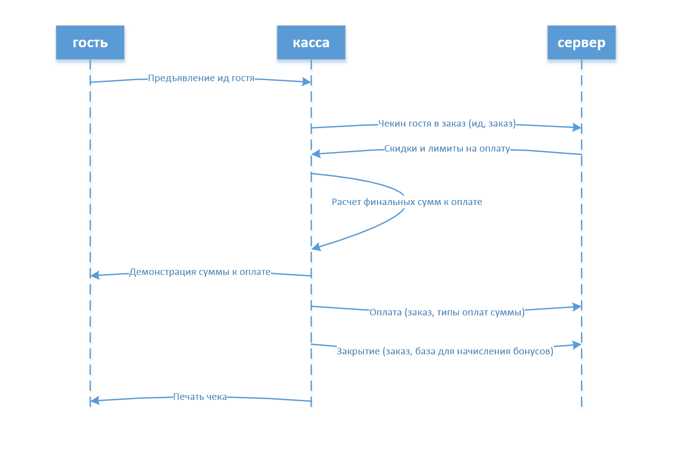
Взаимодействие с системой iiko.net идет по защищенному протоколу https (порт 9010) через сеть Интернет.

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

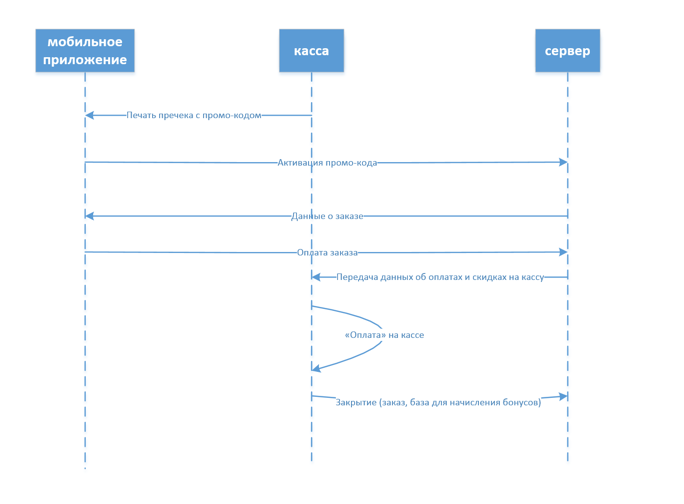
## Сценарии интеграции

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 [iiko.net](http://iiko.net/)
* 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 iiko.net

* **Применимо: fast-food + table service**
* **Предусловие: гость еще не установил приложение, и POS сообщает гостю о системе iiko.net через данные на чеке.**
* касса закрывает заказ и печатает чек, с информацией о программе лояльности данного мерчанта
* гость видит чек с подвалом iiko.net 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 iiko.net и разового платежного кода

* **Применимо: в основном fast-food, но можно использовать и в table service**
* **Предусловие: у гостя установлено приложение iiko.net, и оплата банковской картой не подключена**
* кассир набрал заказ, спрашивает "как будете платить", гость отвечает "[iiko.net](http://iiko.net/)"
* кассир выбирает этот тип оплаты, вводит вручную или сканирует РПК с экрана приложения iiko.net
* касса получает скидки и оплаты бонусами от iiko.net
* кассир нажимает Оплатить
* печатается чек, на нем печатается подвал "спасибо за использование системы лояльности <merchant>", возможно печатаются данные о госте. Эти данные возвращает сервер iiko.net.
* 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 iiko.net и разового платежного кода

* **Применимо: в основном fast-food, но можно использовать и в table service**
* **Предусловие: у гостя установлено приложение iiko.net, и оплата банковской картой подключена, и гость привязал банковскую карту**
* начало сценария такое же как при оплате бонусами без карты
* после выбора типа оплаты iiko.net на кассу возвращаются скидки и оплаты из iiko.net
* 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 "[iiko.net](http://iiko.net/) - банк карта"
* закрывает заказ за другие типы оплаты
* **Методы: Pay(), Close()**

### Сценарий - оплата бонусами и банковской картой c помощью приложения iiko.net и промо-кода

* **Применимо: table service**
* **Предусловие: у гостя установлено приложение iiko.net, и оплата банковской картой подключена, и гость привязал банковскую карту**
* касса печатает на пречеке подвал iiko.net 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 iiko.net 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 iiko.net – 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 [iiko.net](http://iiko.net/)
* **Методы: Precheque(), Pay(), Cancel(), Close()**

### Учет совместимости товарных позиций и скидок и бонусов

* Кассовая система передает в систему iiko.net 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

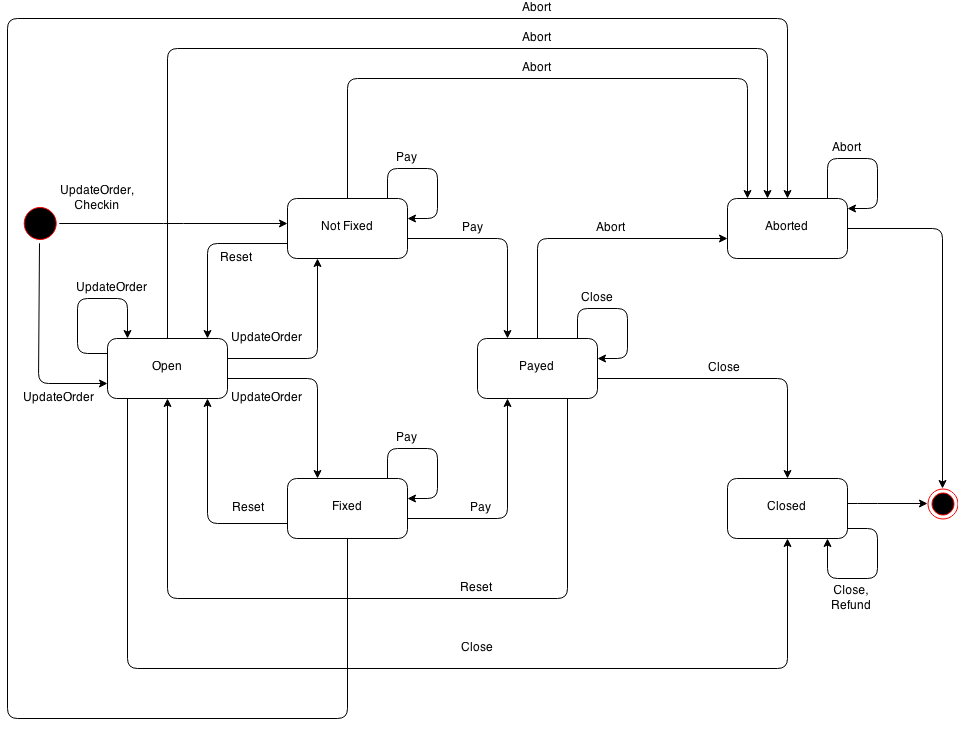
## Методы и структуры

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 iiko.net  var startupParams = new StartupParams  {  Host = "iiko.net",  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 = IikoNetFacade.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 });  // iiko.net передал скидки на элементы заказа  if (checkinResult.LoyaltyResult.Operations != null)  {  foreach (var operation in checkinResult.LoyaltyResult.Operations)  {  var item = GetItemByCode(order, operation.ProductCode);  // запоминаем operationId, чтобы отправить обратно iiko.net информацию о примененных скидках  AddDiscountToOrderItem(order, operation.DiscountSum, item, operation.OperationId);  }  }  // iiko.net передал скидку на весь заказ (в дополнение акциям на артикул)  if (checkinResult.LoyaltyResult.TotalDiscount != decimal.Zero)  {  // запоминаем operationId, чтобы отправить обратно iiko.net информацию о примененных скидках  AddDiscountToOrder(  order, checkinResult.LoyaltyResult.TotalDiscount, checkinResult.LoyaltyResult.OperationId);  }  // iiko.net 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 iiko.net  // ….  }  // Выполняем оплату.  // Метод вызывается при выполнении любого из условий:  // - касса проводит оплаты, связанные с iiko.net  // - в заказе есть примененные скидки, связанные с iiko.net  var appliedDiscounts = GetAppliedIikoNetDiscounts(order);  if (payments.Any() || appliedDiscounts.Any())  {  // Идентификатор транзакции нужно задавать в клиентском коде,  // т.к. может потребоваться инициировать отмену оплаты  // Например, при возникновении ошибок в оплатах, не связанных с iiko.net  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 iiko.net  AppendChequeFooter(order, closeResult.ChequeFooter); |

### Пример - основной сценарий table-service

|  |
| --- |
| // Задаем параметры, с которыми будет работать клиентская библиотека iiko.net  var startupParams = new StartupParams  {  Host = "iiko.net",  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 = IikoNetFacade.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 });  // iiko.net передал скидки на элементы заказа - применяем  if (checkinResult.LoyaltyResult.Operations != null)  {  foreach (var operation in checkinResult.LoyaltyResult.Operations)  {  var item = GetItemByCode(order, operation.ProductCode);  // запоминаем operationId, чтобы отправить обратно iiko.net информацию о примененных скидках  AddDiscountToOrderItem(order, operation.DiscountSum, item, operation.OperationId);  }  }  // iiko.net передал скидку на весь заказ  if (checkinResult.LoyaltyResult.TotalDiscount != decimal.Zero)  {  // запоминаем operationId, чтобы отправить обратно iiko.net информацию о примененных скидках  AddDiscountToOrder(  order, checkinResult.LoyaltyResult.TotalDiscount, checkinResult.LoyaltyResult.OperationId);  }  // 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 iiko.net  // - в заказе есть примененные скидки, связанные с iiko.net  var appliedDiscounts = GetAppliedIikoNetDiscounts(order);  if (payments.Any() || appliedDiscounts.Any())  {  // Идентификатор транзакции нужно задавать в клиентском коде,  // т.к. может потребоваться инициировать отмену оплаты  // Например, при возникновении ошибок в оплатах, не связанных с iiko.net  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 iiko.net  AppendChequeFooter(order, closeResult.ChequeFooter); |

### Базовые интерфейсы

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

**public interface IBasicWorkflow**

**{**

//Состояние подключения к серверу iiko.net.

// 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 iiko.net

**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 iiko.net (через мобильное приложение).

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

// 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 iiko.net.

**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 iikoNet. 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; }**

// Информация о терминале

// Формат: Производитель; Продукт; Версия

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

**}**

### 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 iiko.net – 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 iikoNet).

**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 всех скидок

// (iiko.net + локальные скидки 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 iiko.net).

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

// Форматированный под размер подвал для пречека.

**string ChequeFooter { get; }**

**}**

### Результаты работы акций (скидки)

**public sealed class LoyaltyResult**

**{**

// Полная скидка по всему заказу (не включает поэлементные скидки).

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

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

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

// Список результатов отработки индвидуальных акций.

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

**}**

**public sealed class LoyaltyOperation**

**{**

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

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

// Сумма скидки для продукта от данной акции.

// ВНИМАНИЕ: iiko.net не выполняет округления и если касса ожидает,

// то суммы будут округлены до рублей, десятков рублей и тп

// – она должна выполнить это округление сама.

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

// Комментарии от данной акции.

**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; }**

**}**

// Информацция по оплатам заказа, совершенным на стороне iiko.net

**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**

**{**

// Название скидки.

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

// Артикул блюда, к которому применена скидка. Если скидка на весь заказ - не задается.

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

// Сумма скидки.

**public decimal Sum { 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; }**

**}**