



# Directum RX

---

Описание шаблона решения

«Утилита импорта данных 4.0»

Быстрый старт для разработчика

© Компания Directum, 2023

# Содержание

<b>Содержание .....</b>	<b>2</b>
<b>Введение.....</b>	<b>3</b>
<b>Комплект поставки .....</b>	<b>3</b>
<b>Системные требования .....</b>	<b>4</b>
<b>Запуск утилиты импорта данных .....</b>	<b>4</b>
<b>Модификация утилиты .....</b>	<b>5</b>
<b>Объектная модель .....</b>	<b>5</b>
<b>Описание процесса модификации утилиты импорта</b>	<b>Ошибка! Закладка не определена.</b>
Реализация импорта новых сущностей.....	7
Реализация импорта табличной части .....	11
Доработка существующего правила импорта .....	14
Сборка программы установки.....	15

## Введение

Решение «Утилита импорта данных» предназначено для переноса документов и справочников из сторонней системы в Directum RX. С его помощью можно импортировать новые данные или изменять существующие.

В стандартной поставке реализован импорт документов и справочников следующих типов:

- документы:
  - Договоры;
  - Дополнительные соглашения;
  - Входящие письма;
  - Исходящие письма;
  - Приказы;
- справочники:
  - Организации;
  - Наши организации;
  - Подразделения;
  - Должности;
  - Сотрудники;
  - Персоны.

Решение является шаблоном, который можно адаптировать для задач конкретной организации.

## Комплект поставки

В комплект поставки входят:

- утилита импорта данных;
- исходные коды решения;
- набор XLSX-шаблонов;
- скрипт для сборки программы установки через NSIS;
- документация.

## Системные требования

Решение поддерживает импорт данных в систему Directum RX 4.0 и выше. Системные требования см. в документе «Типовые требования к аппаратному и программному обеспечению», входит в комплект документации Directum RX.

Компьютер, на котором выполняется модификация утилиты импорта данных, должен удовлетворять требованиям:

Компонент	Требование
Процессор (Intel/AMD-совместимый x86/x64)	2 ядра с частотой 2 ГГц
Память (ОЗУ)	2 ГБ
Дополнительные компоненты для модификации утилиты	Microsoft Visual Studio 2019 и выше/Microsoft Visual Studio Code/Microsoft Visual Studio Community .Net Framework 4.8 и выше .Net Core 3.1 SDK – для модификации утилиты, реализованной на .NET Core Nullsoft Scriptable Install System (NSIS) – для сборки программы установки утилиты

## Запуск утилиты импорта данных

1. Убедитесь, что в конфигурационном файле утилиты \_ConfigSettings.xml верно указан адрес сервиса интеграции. Если система установлена в облаке, адрес сервиса интеграции имеет формат:

`https://<Наименование тенанта>-rx.directum24.ru/IntegrationService/odata`

Пример содержимого файла \_ConfigSettings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<settings>
  <var name="INTEGRATION_SERVICE_URL" value="https://company-
rx.directum24.ru/IntegrationService/odata"/>
  <var name="INTEGRATION_SERVICE_REQUEST_TIMEOUT" value="600"/>
</settings>
```

2. Запустите утилиту в командной строке с параметрами:

**-n, --name** – имя пользователя. Если имя пользователя состоит из нескольких слов, то укажите его в кавычках;

**-p, --password** – пароль пользователя;

**-a, --action** – вызываемое действие. Возможные значения:

- **importcompany** – импорт сотрудников, наших организаций, подразделений;
- **importcompanies** – импорт организаций;
- **importpersons** – импорт персон;
- **importcontracts** – импорт договоров;
- **importsupagreements** – импорт дополнительных соглашений;
- **importincomingletters** – импорт входящих писем;
- **importoutgoingletters** – импорт исходящих писем;
- **importorders** – импорт приказов;

**Ошибка! Используйте вкладку "Главная" для применения Заголовков 1;Глава к тексту, который должен здесь отображаться.**

- **importaddendums** – импорт приложений;
- f, --file – путь к заполненному шаблону \*.xlsx;
- dr, --doc\_register\_id – журнал регистрации;
- h, --help – справка по работе с утилитой.

Формат строки запуска:

```
-n <Имя пользователя> -p <Пароль пользователя> -a <действие> -f "<Путь к  
заполненному шаблону *.xlsx>"
```

Пример:

```
-n Administrator -p 11111 -a importcompanies -f  
"D:\import\Template\Example\Организации.xlsx"
```

**СОВЕТ.** Чтобы во время модификации решения быстро запускать утилиту, в Microsoft Visual Studio откройте свойства проекта и в разделе «Отладка» укажите параметры командной строки.

## Модификация утилиты

Для модификации решения доработайте классы объектной модели, а также создайте или измените XLSX-шаблоны. Стандартные XLSX-шаблоны находятся в дистрибутиве в папке Template.

## Объектная модель

Схема наследования классов для импорта справочников:

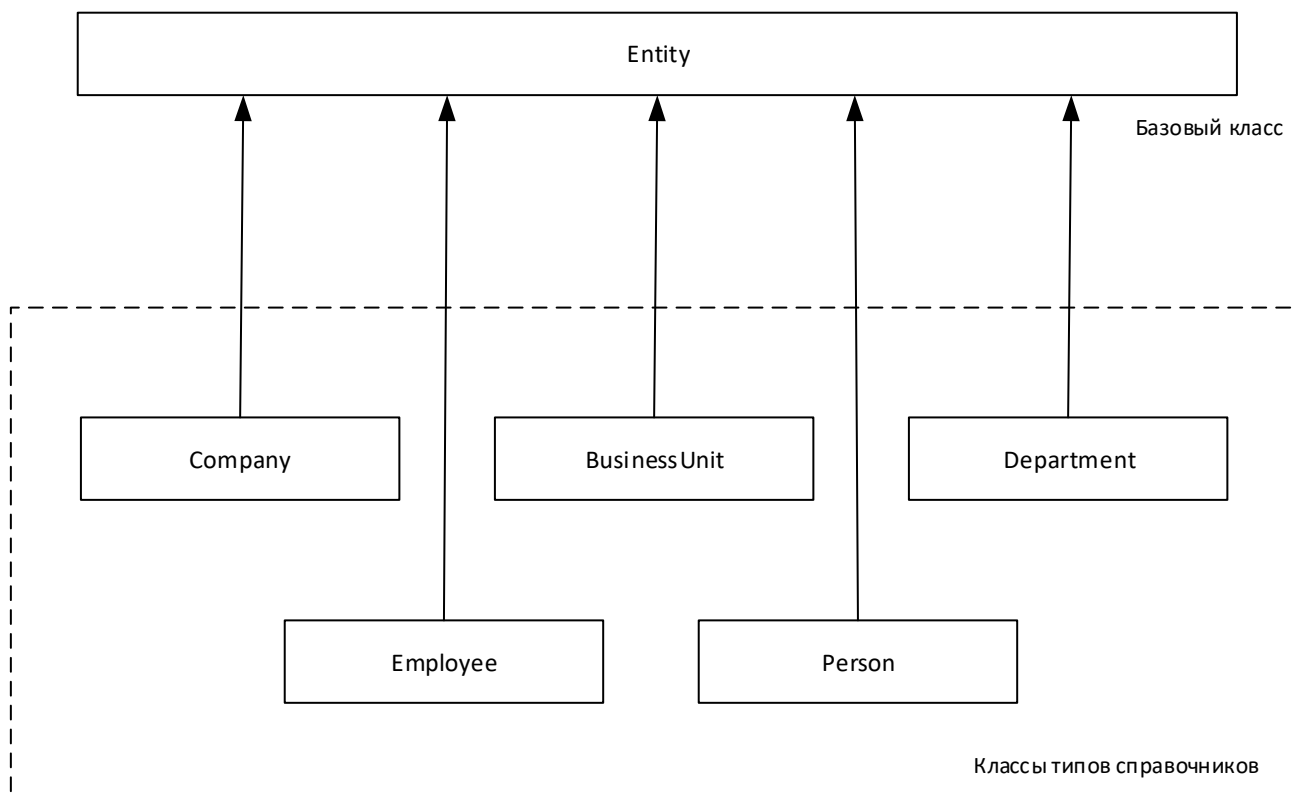
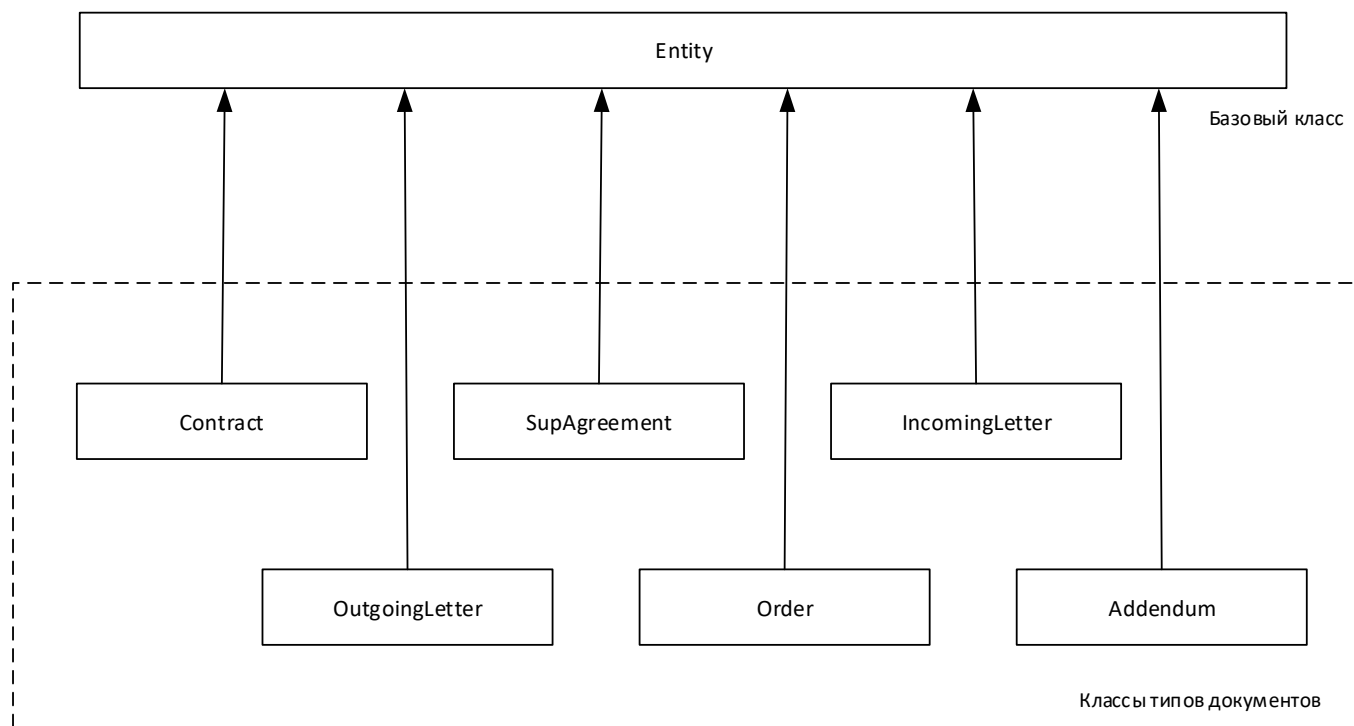


Схема наследования классов для импорта документов:



## Перекрываемые классы

**Entity** – базовый абстрактный класс, от которого наследуются остальные классы. Реализует универсальный механизм импорта сущностей в Directum RX. Правило импорта класса **Entity** реализовано в классах **EntityProcessor** и **EntityWrapper**.

Ключевые перекрываемые методы класса **Entity**:

- **GetPropertiesCount()** – получить число запрашиваемых параметров. Используется для сравнения числа параметров, указанных в классе, и полученных при чтении XLSX-шаблона;
- **SaveToRX()** – сохранить сущности в Directum RX.

**BusinessUnit, Company, Department, Employee, Person** – классы справочников (Databooks). Реализуют процесс импорта исторических данных в справочники системы Directum RX.

**Addendum, Contract, IncomingLetter, Order, OutgoingLetter, SupAgreement** – классы документов (EDocs). Реализуют импорт документов в систему Directum RX как с содержимым, так и без него.

Методы, которые прямо не относятся к классам сущностей, реализуются в классе **BusinessLogic**.

Механизмы работы с XLSX-шаблонами реализованы в классе **ExcelProcessor** с помощью библиотеки OpenXml.

## Порядок модификации утилиты

Модификация утилиты импорта, как правило, выполняется в три этапа:

1. Реализуйте импорт новых сущностей или доработайте существующее правило импорта.
2. Реализуйте импорт табличных частей для необходимых сущностей (при необходимости).
3. Соберите программу установки утилиты.

## Реализация импорта новых сущностей

Реализацию импорта новых сущностей рассмотрим на примере импорта документов с типом «Входящий счет». Чтобы реализовать импорт:

1. Создайте правило импорта новой сущности.
2. Создайте XLSX-шаблон.

## Создание правила импорта новой сущности

1. В Microsoft Visual Studio откройте проект решения **ImportData**.
2. В узле **EDocs** создайте новый класс **IncomingInvoice**.
3. В узле **Reference** подключите библиотеки:  

```
using System.Collections.Generic;  
using System.Globalization;  
using NLog;  
using System.Linq;
```
4. Измените наименование пространства имен.  
Было:  

```
namespace ImportData.Entities.EDocs
```

  
Стало:  

```
namespace ImportData
```
5. Укажите, что класс **IncomingInvoice** является наследником класса **Entity**:  

```
class IncomingInvoice : Entity
```
6. В параметре **PropertiesCount** укажите количество колонок используемого XLSX-шаблона:  

```
public int PropertiesCount = 8;  
/// <summary>  
/// Получить наименование число запрашиваемых параметров.  
/// </summary>  
/// <returns>Число запрашиваемых параметров.</returns>  
public override int GetPropertiesCount()  
{  
    return PropertiesCount;  
}
```
7. В папке проекта `ImportData\IntegrationServicesClient\Models` создайте файл `IncomingInvoices.cs` с описанием модели (класса) для сущности «Входящий счет». Состав нового класса должен соответствовать структуре полей сущности в Directum RX.

Пример:

```
using System;
namespace ImportData.IntegrationServicesClient.Models
{
    [EntityName("Входящий счет")]
    class IIncomingInvoices : IOfficialDocuments
    {
        public DateTimeOffset Dated { get; set; }
        public string InNumber { get; set; }
        public IEmployees Addressee { get; set; }
        public IBusinessUnits BusinessUnit { get; set; }
        public ICounterparties Correspondent { get; set; }
        public IMailDeliveryMethods DeliveryMethod { get; set; }
    }
}
```

8. Перекройте метод **SaveToRX()** и добавьте логику обработки для типа документа «Входящий счет». В метод передается одна строка из XLSX-шаблона. Из каждой колонки строки получается значение свойства.

Пример:

```
var numberInvoice = this.Parameters[shift + 0];
DateTime? dateInvoice = DateTime.MinValue;
var style = NumberStyles.Number | NumberStyles.AllowCurrencySymbol;
var culture = CultureInfo.CreateSpecificCulture("en-GB");
var dateInvoiceDouble = 0.0;
```

9. Если документ с указанными реквизитами уже есть в системе, то повторно импортировать его не нужно. Чтобы импортировались только новые сущности, в код по получению сущности добавьте проверку на наличие дублей в системе.

Для поиска сущностей используется метод **BusinessLogic.GetEntityWithFilter()**. Пример работы с ним описан в разделе «Доработка существующего правила импорта». Реализовывать собственные методы для поиска не нужно.

Пример:

```
// Получение сущности «Входящий счет» по регистрационным данным.
var incomingInvoice = BusinessLogic.GetEntityWithFilter<IIncomingInvoices>(x
=> x.RegistrationNumber == regNumber && x.RegistrationDate.ToString("yyyy-MM-dd'T'HH:mm:ss.fffffff'Z'") == regDate.ToString("yyyy-MM-dd'T'HH:mm:ss.fffffff'Z'"), exceptionList, logger);
```

Процесс создания сущности выполняется помощью метода **BusinessLogic.CreateEntity()**, который отправляет запрос на создание сущности через Интеграционное API.

```
var incomingInvoice = new IIncomingInvoices();
incomingInvoice.Date = dateInvoice;
incomingInvoice.DocumentKind = documentKind;
incomingInvoice.Counterparty = counterparty;
incomingInvoice.TotalAmount = totalAmount;
incomingInvoice.Currency = currency;
incomingInvoice.BusinessUnit = businessUnit;
incomingInvoice.Department = department;
var createdIncomingInvoice = BusinessLogic.CreateEntity<IIncomingInvoices>(incomingInvoice, exceptionList, logger);
```

10. В файле Constants.cs в классе **SheetName** добавьте имя листа, которое используется в XLSX-шаблоне, например, **Вх.Счет**. В классе **Actions** добавьте наименование действия **importincominginvoice**. В справочник **dictActions** добавьте действие.

```
public class SheetNames
```



**Ошибка! Используйте вкладку "Главная" для применения Заголовков 1;Глава к тексту, который должен здесь отображаться.**

```
{
    public const string BusinessUnits = "НашиОрганизации";
    public const string Departments = "Подразделения";
    public const string Employees = "Сотрудники";
    public const string Companies = "Контрагенты";
    public const string Persons = "Персоны";
    public const string Contracts = "Договоры";
    public const string SupAgreements = "Доп.Соглашения";
    public const string IncomingLetters = "ВходящиеПисьма";
    public const string OutgoingLetters = "ИсходящиеПисьма";
    public const string Orders = "Приказы и Распоряжения";
    public const string Addendums = "Приложения";
    public const string IncomingInvoice = "Вх.Счет";
}

public class Actions
{
    public const string ImportCompany = "importcompany";
    public const string ImportCompanies = "importcompanies";
    public const string ImportPersons = "importpersons";
    public const string ImportContracts = "importcontracts";
    public const string ImportSupAgreements = "importsupagreements";
    public const string ImportIncomingLetters = "importincomingletters";
    public const string ImportOutgoingLetters = "importoutgoingletters";
    public const string ImportOrders = "importorders";
    public const string ImportAddendums = "addendums";
    public const string ImportIncomingInvoice = "importincominginvoice";

    public static Dictionary<string, string> dictActions = new
    Dictionary<string, string>
    {
        {ImportCompany, ImportCompany},
        {ImportCompanies, ImportCompanies},
        {ImportPersons, ImportPersons},
        {ImportContracts, ImportContracts},
        {ImportSupAgreements, ImportSupAgreements},
        {ImportIncomingLetters, ImportIncomingLetters},
        {ImportOutgoingLetters, ImportOutgoingLetters},
        {ImportOrders, ImportOrders},
        {ImportAddendums, ImportAddendums},
        {ImportIncomingInvoice, ImportIncomingInvoice}
    };
}
```

11. В файле Program.cs в методе **ProcessByAction()** добавьте вызов обработчика для импорта списка сущностей типа «Входящий счет».

```
/// <summary>
/// Выполнение импорта в соответствии с требуемым действием.
/// </summary>
/// <param name="action">Действие.</param>
/// <param name="xlsxPath">Входной файл.</param>
/// <param name="extraParameters">Дополнительные параметры.</param>
/// <param name="logger">Логировщик.</param>
/// <returns>Соответствующий тип сущности.</returns>
static void ProcessByAction(string action, string xlsxPath,
    Dictionary<string, string> extraParameters, string ignoreDuplicates,
    NLog.Logger logger)
```

```

{
    switch (action)
    {
        case "importcompany":
            logger.Info("Импорт сотрудников");
            logger.Info("-----");
            EntityProcessor.Process(typeof(Employee), xlsXPath,
Constants.SheetNames.Employees, extraParameters, ignoreDuplicates, logger);
            logger.Info("Импорт НОП");
            logger.Info("-----");
            EntityProcessor.Process(typeof(BusinessUnit), xlsXPath,
Constants.SheetNames.BusinessUnits, extraParameters, ignoreDuplicates,
logger);
            logger.Info("Импорт подразделений");
            logger.Info("-----");
            EntityProcessor.Process(typeof(Department), xlsXPath,
Constants.SheetNames.Departments, extraParameters, ignoreDuplicates, logger);
            break;
        case "importcompanies":
            EntityProcessor.Process(typeof(Company), xlsXPath,
Constants.SheetNames.Companies, extraParameters, ignoreDuplicates, logger);
            break;
        case "importpersons":
            EntityProcessor.Process(typeof(Person), xlsXPath,
Constants.SheetNames.Persons, extraParameters, ignoreDuplicates, logger);
            break;
        case "importcontracts":
            EntityProcessor.Process(typeof(Contract), xlsXPath,
Constants.SheetNames.Contracts, extraParameters, ignoreDuplicates, logger);
            break;
        case "importsupagreements":
            EntityProcessor.Process(typeof(SupAgreement), xlsXPath,
Constants.SheetNames.SupAgreements, extraParameters, ignoreDuplicates,
logger);
            break;
        case "importincomingletters":
            EntityProcessor.Process(typeof(IncomingLetter), xlsXPath,
Constants.SheetNames.IncomingLetters, extraParameters, ignoreDuplicates,
logger);
            break;
        case "importoutgoingletters":
            EntityProcessor.Process(typeof(OutgoingLetter), xlsXPath,
Constants.SheetNames.OutgoingLetters, extraParameters, ignoreDuplicates,
logger);
            break;
        case "importorders":
            EntityProcessor.Process(typeof(Order), xlsXPath,
Constants.SheetNames.Orders, extraParameters, ignoreDuplicates, logger);
            break;
        case "addendums":
            EntityProcessor.Process(typeof(Addendum), xlsXPath,
Constants.SheetNames.Addendums, extraParameters, ignoreDuplicates, logger);
            break;
        case "importincominginvoice":
            EntityProcessor.Process(typeof(IncomingInvoice), xlsXPath,
Constants.SheetNames.IncomingInvoice, extraParameters, ignoreDuplicates,
logger);
    }
}

```

**Ошибка! Используйте вкладку "Главная" для применения Заголовков 1;Глава к тексту, который должен здесь отображаться.**

```
        break;  
    default:  
        break;  
}  
}
```

## Создание XLSX-шаблона

При создании шаблона рекомендуется ориентироваться на существующие примеры шаблонов в папке с исходным кодом утилиты.

Чтобы создать шаблон:

1. Создайте XLSX-шаблон – лист Microsoft Excel, в ячейках которого указаны параметры импортируемых сущностей. Подробнее см. инструкцию по загрузке данных в Directum RX, входит в комплект поставки решения.
2. В первой строке укажите наименования параметров. Формат ячеек должен быть «Текстовый».  
**ПРИМЕЧАНИЕ.** Ячейки, в которых хранится дата, приведите к формату «Дата» или «Общий».
3. Зеленым цветом выделите наименование обязательных столбцов, серым – необязательных, как в стандартных XLSX-шаблонах.
4. Переименуйте имя листа на «Вх.Счет». Это наименование было указано в коде и задано в константах утилиты при создании правила импорта. По этому наименованию выполняется поиск необходимого листа в XLSX-шаблоне.

## Реализация импорта табличной части

Реализацию импорта табличной части рассмотрим на примере импорта коллекции «Список рассылки письма» на вкладке «Адресаты» в документе с типом «Исходящее письмо». Чтобы реализовать импорт:

1. Создайте правило импорта табличной части.
2. Создайте XLSX-шаблон.

## Создание правила импорта табличной части

1. В Microsoft Visual Studio откройте проект решения **ImportData**.
2. В узле **EDocs** создайте новый класс **OutgoingLetterAddressees**.
3. В узле **Reference** подключите библиотеки:

```
using System;  
using System.Collections.Generic;  
using NLog;  
using ImportData.IntegrationServicesClient.Models;
```

4. Измените наименование пространства имен.

Было:

```
namespace ImportData.Entities.EDocs
```

Стало:

```
namespace ImportData
```

- Укажите, что класс **OutgoingLetterAddressees** является наследником класса **Entity**:
- В параметре **PropertiesCount** укажите количество колонок используемого XLSX-шаблона:

```
class OutgoingLetterAddressees : Entity
{
    public int PropertiesCount = 4;
    /// <summary>
    /// Получить наименование число запрашиваемых параметров.
    /// </summary>
    /// <returns>Число запрашиваемых параметров.</returns>
    public override int GetPropertiesCount()
    {
        return PropertiesCount;
    }
}
```

- В папке проекта ImportData\IntegrationServicesClient\Models создайте файл IOutgoingLetterAddresseess.cs с описанием модели (класса) для коллекции «Список рассылки письма». Состав нового класса должен соответствовать структуре полей коллекции в Directum RX.

Пример:

```
using System;
namespace ImportData.IntegrationServicesClient.Models
{
    [EntityName("Список рассылки письма")]
    class IOutgoingLetterAddresseess
    {
        public ICounterparties Correspondent { get; set; }
        public IContacts Addressee { get; set; }
        public IMailDeliveryMethods DeliveryMethod { get; set; }
        public IOutgoingLetters OutgoingDocumentBase { get; set; }
    }
}
```

- Перекройте метод **SaveToRX()** и добавьте логику обработки для коллекции документа типа документа «Исходящее письмо». В метод передается одна строка из XLSX-шаблона. Из каждой колонки строки получается значение свойства.

Пример:

```
var documentId = int.TryParse(this.Parameters[shift + 0].Trim());
var outgoingLetter = BusinessLogic.GetEntityWithFilter<IOutgoingLetters>(d => d.Id == documentId, exceptionList, logger);
var counterparty = BusinessLogic.GetEntityWithFilter<ICounterparties>(c => c.Name == this.Parameters[shift + 1].Trim(), exceptionList, logger);
var contact = BusinessLogic.GetEntityWithFilter<IContacts>(d => d.Name == this.Parameters[shift + 2].Trim(), exceptionList, logger);
var deliveryMethod = BusinessLogic.GetEntityWithFilter<IMailDeliveryMethods>(m => m.Name == variableForParameters, exceptionList, logger);
var addressee = new IOutgoingLetterAddresseess
{
    Addressee = contact,
    OutgoingDocumentBase = outgoingLetter,
    DeliveryMethod = deliveryMethod,
    Correspondent = counterparty,
};
```

- Если в табличной части документа уже есть запись с указанными реквизитами, то повторно импортировать его не нужно. Чтобы импортировались только новые сущности, добавьте проверку на наличие дублей в документе.

**Ошибка! Используйте вкладку "Главная" для применения Заголовков 1;Глава к тексту, который должен здесь отображаться.**

Для поиска сущностей используется метод **BusinessLogic.GetEntityWithFilter()**. Пример работы с ним описан в разделе «Доработка существующего правила импорта». Реализовывать собственные методы для поиска не нужно.

10. Реализуйте функционал для добавления новой строки в коллекцию «Список рассылки письма».

```
var result = Client.Instance().For<IOutgoingLetters>()
    .Key(outgoingLetter.Id)
    .NavigateTo(nameof(outgoingLetter.Addressees))
    .Set(new IOutgoingLetterAddresseess()
    {
        Addressee = addressee.Addressee,
        DeliveryMethod = addressee.DeliveryMethod,
        Correspondent = addressee.Correspondent,
        OutgoingDocumentBase = outgoingLetter,
    })
    .InsertEntryAsync().Result;
```

11. В файле Constants.cs в классе **SheetName** добавьте имя листа, которое используется в XLSX-шаблоне, например, **ИсходящиеПисьмаТабличнаяЧасть**. В классе **Actions** добавьте наименование действия **importoutgoinglettersaddressees**. В справочник **dictActions** добавьте действие.

```
public class SheetNames
{
    ...
    public const string OutgoingLettersAddressees =
        "ИсходящиеПисьмаТабличнаяЧасть";
    ...
}

public class Actions
{
    ...
    public const string ImportOutgoingLettersAddressees =
        "importoutgoinglettersaddressees";
    ...

    public static Dictionary<string, string> dictActions = new
        Dictionary<string, string>
    {
        ...
        {ImportOutgoingLettersAddressees, ImportOutgoingLettersAddressees},
        ...
    };
}
```

12. В файле Program.cs в методе **ProcessByAction()** добавьте вызов обработчика для импорта списка сущностей типа «Входящий счет».

```
static void ProcessByAction(string action, string xlsXPath,
    Dictionary<string, string> extraParameters, string ignoreDuplicates,
    NLog.Logger logger)
{
    switch (action)
    {
        ...
        case "importoutgoinglettersaddressees":
```

```

        EntityProcessor.Process(typeof(OutgoingLetterAddressees), xlsXPath,
        Constants.SheetNames.OutgoingLettersAddressees, extraParameters,
        ignoreDuplicates, logger);
        break;
    }
    ...
}

```

## Создание XLSX-шаблона для табличной части

При создании шаблона рекомендуется ориентироваться на существующие примеры шаблонов в папке с исходным кодом утилиты.

Чтобы создать шаблон:

1. Создайте XLSX-шаблон – лист Microsoft Excel, в ячейках которого указаны параметры импортируемых сущностей. Подробнее см. инструкцию по загрузке данных в Directum RX, входит в комплект поставки решения.
2. В первой строке укажите наименования параметров. Формат ячеек должен быть «Текстовый».

**ПРИМЕЧАНИЕ.** Ячейки, в которых хранится дата, приведите к формату «Дата» или «Общий».

3. Зеленым цветом выделите наименование обязательных столбцов, серым – необязательных, как в стандартных XLSX-шаблонах.
4. Переименуйте имя листа на «ИсходящиеПисьмаСписокПолучателей». Это наименование было указано в коде и задано в константах утилиты при создании правила импорта. По этому наименованию выполняется поиск необходимого листа в XLSX-шаблоне.

## Доработка существующего правила импорта

В реализованных классах типов справочников и типов документов можно изменить состав полей импортируемых сущностей.

Доработку существующего правила импорта рассмотрим на примере. Пусть в правиле импорта справочника **Организации** нужно добавить обработку полей **Номер ответственного** и **Ответственный**. Для этого:

1. В классе **Company** в методе **SaveToRX()** добавьте обработку полей **Номер ответственного** и **Ответственный**. Индексы столбцов передаются параметром в виде массива. В примере используется 19 колонок, поэтому у двух добавленных элементов будут индексы 20 и 21.

```
// Стандартный код
```

```
// Номер ответственного.
```

```
var numberResponsible = this.Parameters[shift + 20].Trim();
```

```
// Ответственный.
```

```
var responsible = BusinessLogic.GetEntityWithFilter<IEmployees>(e => e.Id ==
numberResponsible, exceptionList, logger);
```

```
if (!string.IsNullOrEmpty(this.Parameters[shift + 21]) && responsible ==
null)
{
```

**Ошибка! Используйте вкладку "Главная" для применения Заголовков 1;Глава к тексту, который должен здесь отображаться.**

```
var message = string.Format("Не найден Ответственный \"{1}\".  
Наименование организации: \"{0}\". ", name, this.Parameters[shift +  
21].Trim());  
exceptionList.Add(new Structures.ExceptionsStruct { ErrorType =  
Constants.ErrorTypes.Warn, Message = message });  
logger.Warn(message);  
}  
// Стандартный код
```

- В классе **Company** в свойстве **PropertiesCount** увеличьте количество полей. В примере используется 22 поля, т.к. индекс в массиве начинается с 0 и заканчивается на 21. В коде укажите значение переменной:  
`PropertiesCount = 22`
- Доработайте существующий XLSX-шаблон. Для этого в шаблон добавьте новые колонки, соответствующие новым полям **Номер ответственного** и **Ответственный**, и комментарий к ним.

Доработка импортируемых типов документов выполняется аналогично.

## Сборка программы установки

В дистрибутиве решения в папке Install находятся файлы для сборки программы установки:

- DirRxInstaller.nsi – скрипт для сборки;
- \*.ini – конфигурационные файлы.

Чтобы собрать программу установки:

- Откройте скрипт DirRxInstaller.nsi в любом текстовом редакторе, например в Notepad++, и в параметрах **VIProductVersion**, **VIAddVersionKey** укажите актуальную версию решения.  
Пример:  
`VIProductVersion 4.0.4036.0  
VIAddVersionKey FileVersion 4.0.4036.0  
VIAddVersionKey ProductVersion 4.0.4036.0`
- В NSIS выполните действие **Compile NSI scripts** и в открывшемся окне выберите скрипт DirRxInstaller.nsi.
- Дождитесь завершения сборки. В результате в папке Install создается программа установки Setup.exe.