Directum RX

Описание шаблона решения «Утилита импорта данных 4.0»

Быстрый старт для разработчика

Содержание

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.

Решение поддерживает пакетный импорт документов, позволяющий создавать сразу несколько документов в рамках одного запроса к сервису интеграции.

В стандартной поставке реализован импорт документов и справочников следующих типов:

- документы:
    - Договоры;
    - Дополнительные соглашения;
    - Приложений к документу;
    - Входящие письма;
    - Исходящие письма;
    - Приказы;
- справочники:
    - Организации;
    - Наши организации;
    - Подразделения;
    - Должности;
    - Сотрудники;
    - Персоны;
    - Страны;
    - Валюты;
    - Приложений-обработчиков;
    - Категорий;
    - Журналов регистрации;
    - Видов документов;
    - Ролей;

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"/>
     <var name="INTEGRATION_SERVICE_BATCH_REQUESTS_COUNT" value="100"/>
   </settings>
   ```

   ПРИМЕЧАНИЕ. Параметр **INTEGRATION_SERVICE_BATCH_REQUESTS_COUNT** отвечает за количество запросов, выполняющихся в рамках одного пакетного запроса. Максимальное значение – 200, по умолчанию – 100.

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;
- **importaddendums** – импорт приложений;
- **importassociatedapplications** – импорт расширений файлов;
- **importcontractcategories** – импорт категории;
- **importdocumentregisters** – импорт журналов регистрации;
- **importdocumentkinds** – импорт видов документов;
- **importroles** – импорт ролей;

-f, **--file** – путь к заполненному шаблону *.xlsx;

-dr, **--doc_register_id** – журнал регистрации;

-d, **--search_doubles** – признак того, что при создании сущности нужно искать дубли;

-ub, **--update_body** – признак обновления последней версии документа;

-if, **--input_format** – формат загружаемого документа, по умолчанию XLSX;

-csvd, **--csv_delimiter** – разделитель для файлов формата CSV;

-b, **--batch** – использовать пакетные запросы для импорта;

-h, **--help** – справка по работе с утилитой.

Формат строки запуска:

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

Пример:

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

ПРИМЕЧАНИЕ. Пакетный импорт работает только на документах. Не используйте пакетный импорт, если тела импортируемых документов превышают 5 МБ, т.к. это может привести к низкой скорости импорта или к ошибке выполнения запроса к сервису интеграции.

СОВЕТ. Чтобы во время модификации решения быстро запускать утилиту, в Microsoft Visual Studio воспользуйтесь одним из способов:

- откройте свойства проекта и в разделе «Отладка» укажите параметры командной строки;
- в файле program.cs пропишите параметры **args**, например:

```
args = new[] { "-n", "Administrator", "-p", "11111", "-if", "csv", "-a", "importcontacts", "-f", $@"csv/Контактные лица.csv" }.
```

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:

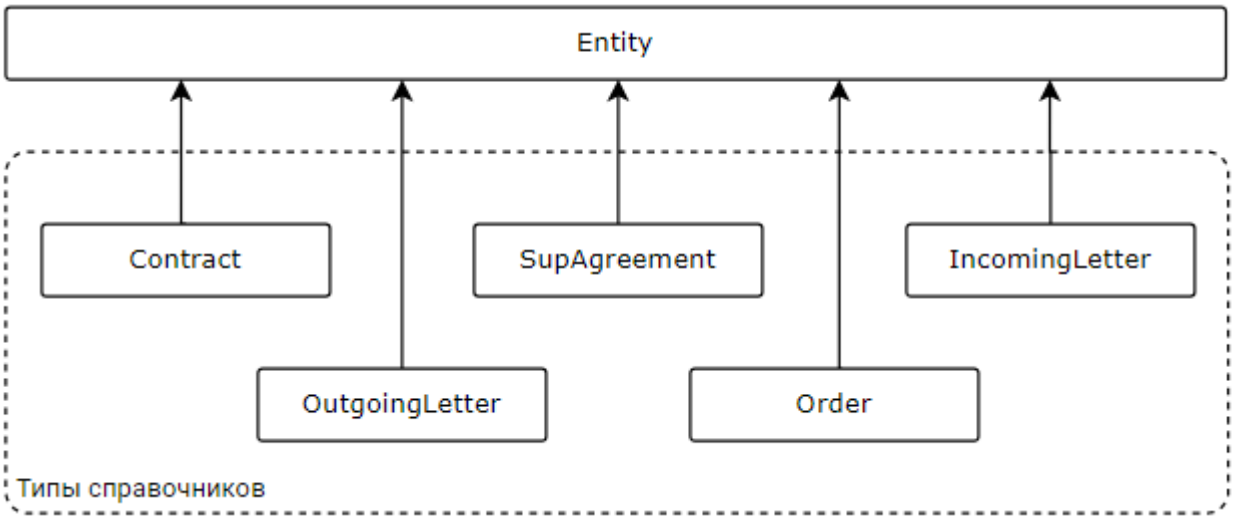
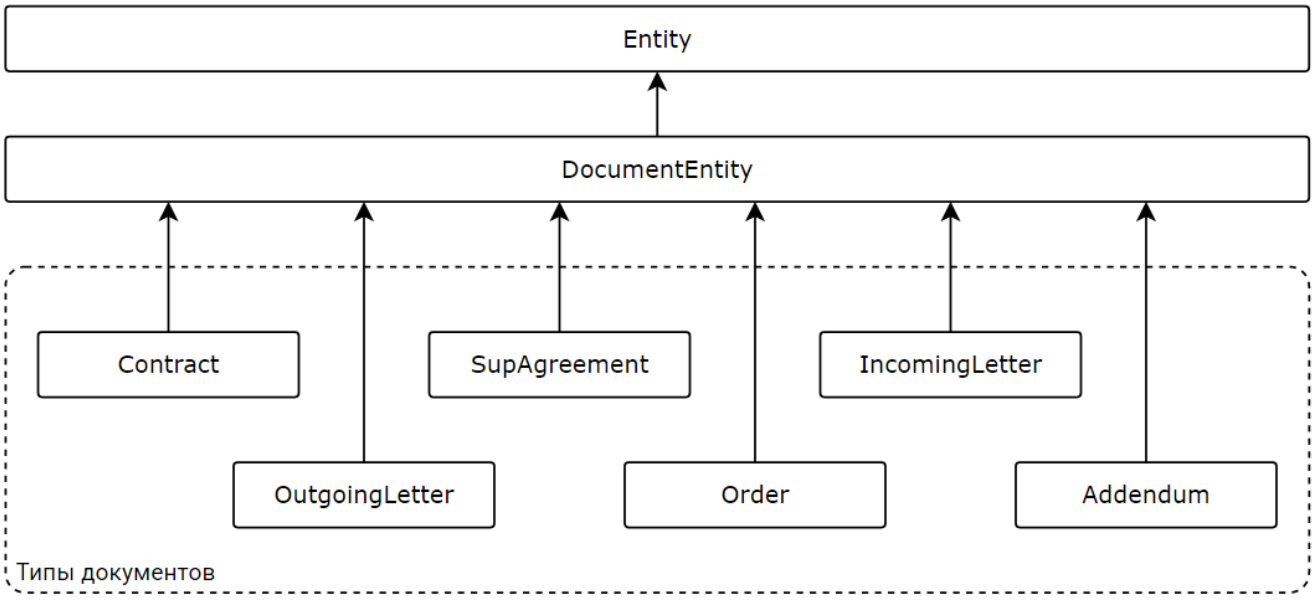


Схема наследования классов для импорта документов:



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 виртуальные свойства класса **Entity**:

- **PropertiesCount** – число запрашиваемых параметров. Используется для сравнения числа параметров, указанных в классе, и полученных при чтении XLSX-шаблона;
- **EntityType** – тип сущности;
- **RequestsPerBatch** – количество HTTP запросов в случае пакетного импорта.

AssociatedApplication, BusinessUnit, **CaseFile**, **Company**, **Contact**, **ContractCategory, Country, Currency, DocumentKind, DocumentRegister, Department**, **Employee**, **Login**, **Person, Role, Substitution** – классы справочников (Databooks). Реализуют процесс импорта исторических данных в справочники системы Directum RX.

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

DocumentEntity – класс наследник класса **Entity**. От него наследуются классы документов (EDocs).

OutgoingLetterAddressees – табличная часть **Список рассылки письма** сущности **Исходящее письмо**.

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.

Класс PropertyOptions

Класс атрибутов свойств, наследник класса **Attribute**. Содержит конструктор:

```
public PropertyOptions(string excelName, RequiredType required, PropertyType type, AdditionalCharacters characters = AdditionalCharacters.Default), где
```

- **excelName** – имя свойства. Совпадает с соответствующим названием столбца в шаблоне;
- **required** – признак обязательности свойства. Перечисление RequiredType:
  - NotRequired (необязательное);
  - Required (обязательное);
- **type** – тип свойства. Перечисление PropertyType:
  - Simple (число, дата, строка и т.п.);
  - Entity (сущность);
  - EntityWithCreate (сущность с последующим созданием);
- **characters** – характеристика свойства. Перечисление AdditionalCharacters:
  - Default (выставляется по умолчанию);
  - ForSearch (указывает о том, что по этому свойству будет происходить поиск уже существующей сущности в системе RX);

- CreateFromOtherProperties (указывает, что свойство будет создано на основании других свойств из шаблона);
- Collection (свойство-коллекция).

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.
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.
3. Создайте CSV-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 **DocumentEntity**:
```
class IncomingInvoice : DocumentEntity
```
6. Укажите количество колонок используемого XLSX-шаблона:
```
public override int PropertiesCount { get { return 16; } }
```
7. Укажите нужно ли тело документа:
```
protected override bool RequiredDocumentBody { get { return true; } }
```
   Сущность не будет импортирована, если значение равно true и указанный в шаблоне документ не найден.
8. Если необходимо реализовать пакетный импорт сущности, то в параметре RequestsPerBatch укажите количество HTTP запросов, необходимых на создание сущности:
```
public override int RequestsPerBatch => 3;
```

9. В папке проекта ImportData\IntegrationServicesClient\Models создайте файл IIncomingInvoices.cs с описанием модели (класса) для сущности «Входящий счет». Состав нового класса должен соответствовать структуре полей сущности в Directum RX. В новом классе реализуйте логику поиска и создания/обновления сущности.

Пример:

```
using System;
namespace ImportData.IntegrationServicesClient.Models
{
  [EntityName("Входящий счет")]
  class IIncomingInvoices : IOfficialDocuments
  {
    // Добавляем необходимые свойства с  атрибутами.
    // "Дата счета" – имя столбца из шаблона,
    // RequiredType.Required – свойство обязательное,
    // PropertyType.Simple – простой тип свойства,
    // AdditionalCharacters.ForSearch – по этому свойству буде происходить поиск уже
    //существующей сущности в системе RX.
    [PropertyOptions("Дата счета", RequiredType.Required, PropertyType.Simple,
      AdditionalCharacters.ForSearch)]
    public DateTimeOffset? Date
    {
     get { return incominInvoiceDate; }
     set { incominInvoiceDate = value.HasValue? new DateTimeOffset(value.Value.Date,
           TimeSpan.Zero) : new DateTimeOffset? (); }
    }

    [PropertyOptions("Номер счета", RequiredType.Required, PropertyType.Simple,
     AdditionalCharacters.ForSearch)]
    public string Number { get; set; }

    // PropertyType.Entity – свойство является сущностью.
    [PropertyOptions("Наша организация", RequiredType.Required, PropertyType.Entity,
     AdditionalCharacters.ForSearch)]
    public IBusinessUnits BusinessUnit { get; set; }

    // PropertyType.EntityWithCreate – будет создана сущность "Контрагент", если в
    // системе RX такой еще нет.
    [PropertyOptions("Контрагент", RequiredType.NotRequired,
     PropertyType.EntityWithCreate, AdditionalCharacters.ForSearch)]
    public ICounterparties Counterparty { get; set; }

    [PropertyOptions("Способ доставки", RequiredType.NotRequired, PropertyType.Entity,
     AdditionalCharacters.ForSearch)]
    public IMailDeliveryMethods DeliveryMethod { get; set; }

    // Метод для поиска сущности в системе RX.
    new public static IEntity FindEntity(Dictionary<string, string> propertiesForSearch,
     Entity entity, bool isEntityForUpdate, List<Structures.ExceptionsStruct>
     exceptionList, NLog.Logger logger)
    {
     IIncomingInvoices doc = null;

     // Для простоты примера не учитываем свойства с атрибутами
     //AdditionalCharacters.ForSearch, а возьмем только свойство Number и DocumentDate,
     // т.е. поиск будет происходить только по Номеру счета и Дате документа (свойство
     // предка класса IOfficialDocuments).
     var number = propertiesForSearch[Constants.KeyAttributes.Number];

      if(propertiesForSearch.ContainsKey(Constants.KeyAttributes.DocumentDate)
         && GetDate(propertiesForSearch[Constants.KeyAttributes.DocumentDate],
        out var documentDate))
```

```
      {
        doc = BusinessLogic.GetEntityWithFilter<IIncomingInvoices>(x => x.Number != null
        && x.Number == number && x.DocumentDate == documentDate, exceptionList, logger);
      }
      else
      {
        doc = BusinessLogic.GetEntityWithFilter<IIncomingInvoices>(x => x.Number != null
        && x.Number == number, exceptionList, logger);
      }

      return doc;
    }

    // Метод для создания и обновления сущности.
    new public static IEntityBase CreateOrUpdate(IEntityBase entity, bool isNewEntity,
      bool isBatch, List<Structures.ExceptionsStruct> exceptionList, NLog.Logger logger)
    {
      if (isNewEntity)
        return BusinessLogic.CreateEntity((IIncomingInvoices)entity, exceptionList,
            logger);
      else
        return BusinessLogic.UpdateEntity((IIncomingInvoices)entity, exceptionList,
            logger);
    }
  }
}
```

10. В созданном ранее классе **IncomingInvoice** укажите тип сущности **IIncomingInvoices**:

    ```
    protected override Type EntityType { get { return typeof(IIncomingInvoices); } }
    ```

11. 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
    {
      ...
      public const string IncomingInvoice = "Вх.Счет";
    }

    public class Actions
    {
      ...
      public const string ImportIncomingInvoice = "importincominginvoice";

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

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 "importincominginvoice":
         EntityProcessor.Process(typeof(IncomingInvoice), xlsxPath,
          Constants.SheetNames.IncomingInvoice, extraParameters, ignoreDuplicates, logger,
          isBatch);
          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.

Создание CSV-шаблона

При создании шаблона ориентируйтесь на примеры в папке Template из репозитория утилиты.

1. Создайте CSV-шаблон одним из способов:
   - вручную по аналогии с примерами из комплекта поставки решения;
   - выгрузкой из системы.
2. Откройте CSV-шаблон на редактирование. В первой строке укажите наименования параметров, в последующих строках – данные.

   При заполнении шаблона учитывайте рекомендации:
   - дробные числа разделяйте через точку;
   - используйте кодировку UTF-8.
4. Сохраните изменения.

Реализация импорта табличной части сущности

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 сущности

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.

   Прежнее имя:

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

   Новое имя:

```
namespace ImportData
```

5. Укажите, что класс **OutgoingLetterAddressees** является наследником класса **Entity**:

```
class OutgoingLetterAddressees : Entity
```

6. Укажите количество колонок используемого XLSX-шаблона:

```
public override int PropertiesCount { get { return 4; } }
```

7. Если необходимо реализовать пакетный импорт сущности, то в параметре RequestsPerBatch укажите количество HTTP запросов, необходимых на создание сущности:

```
public override int RequestsPerBatch => 3;
```

8. 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
 {
  [PropertyOptions("Id исходящего письма в DirectumRX", RequiredType.Required,
   PropertyType.Entity, AdditionalCharacters.ForSearch)]
  public IOutgoingLetters OutgoingDocumentBase { get; set; }

  [PropertyOptions("Корреспондент", RequiredType.Required, PropertyType.Entity,
   AdditionalCharacters.ForSearch)]
  public ICounterparties Correspondent { get; set; }

  [PropertyOptions("Адресат", RequiredType.NotRequired, PropertyType.Entity)]
  public IContacts Addressee { get; set; }

  [PropertyOptions("Способ доставки", RequiredType.NotRequired, PropertyType.Entity)]
  public IMailDeliveryMethods DeliveryMethod { get; set; }

  new public static IEntityBase CreateOrUpdate(IEntityBase entity, bool isNewEntity,
  bool isBatch, List<Structures.ExceptionsStruct> exceptionList, NLog.Logger logger)
  {
    var outgoingLetterAddresseess = (IOutgoingLetterAddresseess)entity;
    var outgoingLetter = outgoingLetterAddresseess.OutgoingDocumentBase;
    var addressee = new IOutgoingLetterAddresseess
    {
      Addressee = outgoingLetterAddresseess.Addressee,
      OutgoingDocumentBase = outgoingLetter,
      DeliveryMethod = outgoingLetterAddresseess.DeliveryMethod,
```

```
        Correspondent = outgoingLetterAddresseess.Correspondent,
      };
      outgoingLetter.CreateAddressee(addressee, logger, isBatch);
      return addressee;
    }
  }
}
```

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

```
public void CreateAddressee(IOutgoingLetterAddresseess addressee)
{
  Client.Instance().For<IOutgoingLetters>()
    .Key(this)
    .NavigateTo(nameof(Addressees))
    .Set(new IOutgoingLetterAddresseess()
     {
       Addressee = addressee.Addressee,
       DeliveryMethod = addressee.DeliveryMethod,
       Correspondent = addressee.Correspondent,
       OutgoingDocumentBase = this
     })
    .InsertEntryAsync();

}


// Пакетная обработка.
private void CreateAddresseeBatch(IOutgoingLetterAddresseess addressee)
{
  BatchClient.AddRequest(odata => odata.For<IOutgoingLetters>()
   .Key(this)
   .NavigateTo(nameof(Addressees))
   .Set(new IOutgoingLetterAddresseess()
    {
      Addressee = addressee.Addressee,
      DeliveryMethod = addressee.DeliveryMethod,
      Correspondent = addressee.Correspondent,
      OutgoingDocumentBase = this
    })
   .InsertEntryAsync());

}
```

10. 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}
  };
```

```
}
```

11. 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, isBatch);
      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.

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:

1. 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
   ```

2. В NSIS выполните действие **Compile NSI scripts** и в открывшемся окне выберите скрипт DirRxInstaller.nsi.
3. Дождитесь завершения сборки. В результате в папке Install создастся программа установки Setup.exe.

Порядок модификации unit-тестов

При проведении unit-теста:

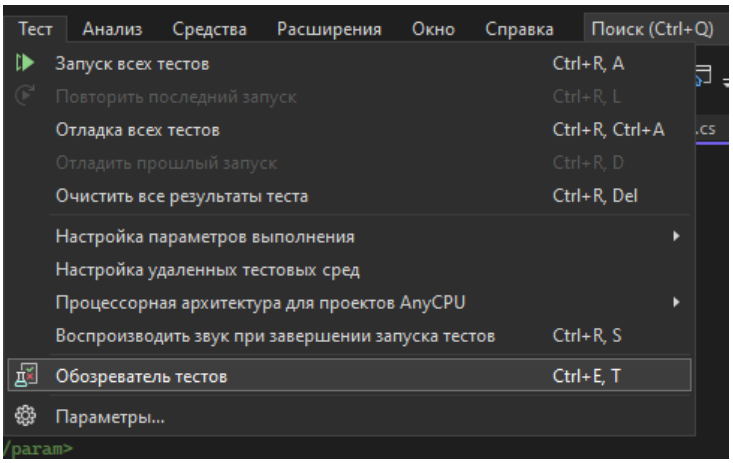
1. Запускается импорт данных из заранее подготовленных файлов в систему Directum RX.
2. Загруженные данные сравниваются с теми, что были отправлены в систему.

В результате при сравнении исходных и загруженных данных учитывается прикладная логика системы.

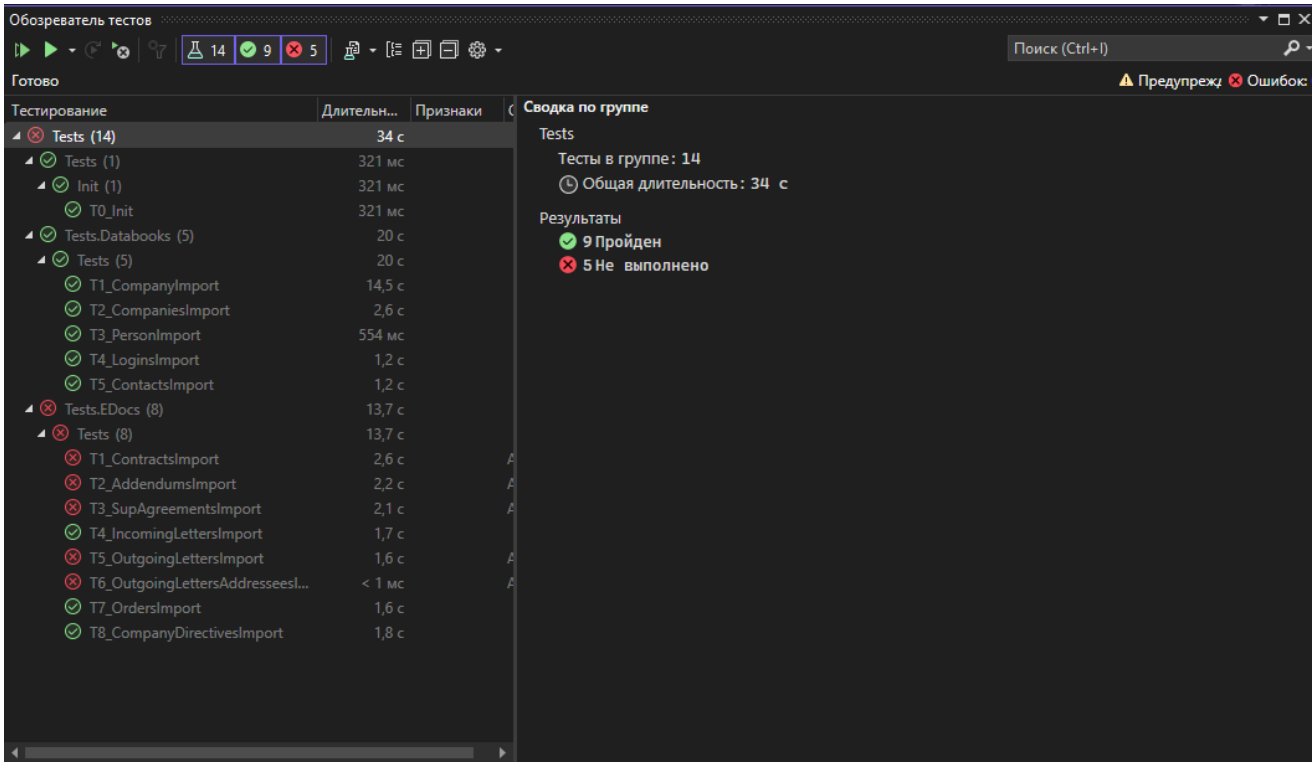
ВАЖНО. Рекомендуется выполнять тесты с чистой базой данных. Если данные были загружены в систему до проведения тестов, то результаты могут быть недостоверными.

Запуск тестов

1. Запустите проект.
2. В конфигурационном файле ConfigSettings.xml задайте URL для подключения к сервису интеграции Directum RX тестового стенда.
3. В классе **TestSettings** укажите актуальный логин и пароль, под которым будет выполняться подключение к системе Directum RX.
4. На вкладке «Тест» выберите пункт **Обозреватель тестов**:



5. В обозревателе тестов запустите все тесты или выберите нужный из узла:



6. Дождитесь завершения тестов и проанализируйте их результаты.

Анализ результатов тестирования

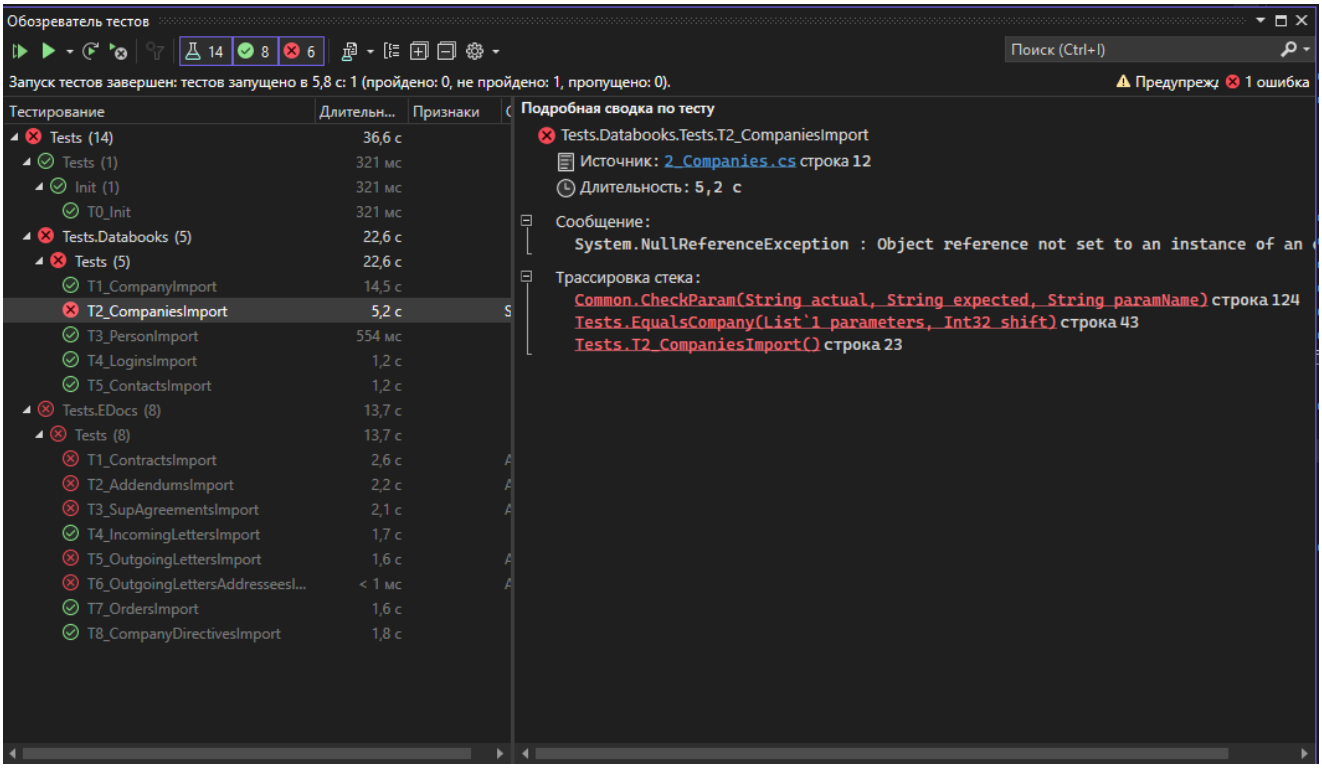
При тестировании возникают ситуации:

- тест пройден успешно;
- ошибка в коде;
- сущность не найдена;
- ошибка в сущности.

Ошибка в коде

В этом случае по стеку можно увидеть, где произошла ошибка, и посмотреть детали.

Чтобы посмотреть входные данные и понять, где произошла ошибка, можно использовать отладчик:

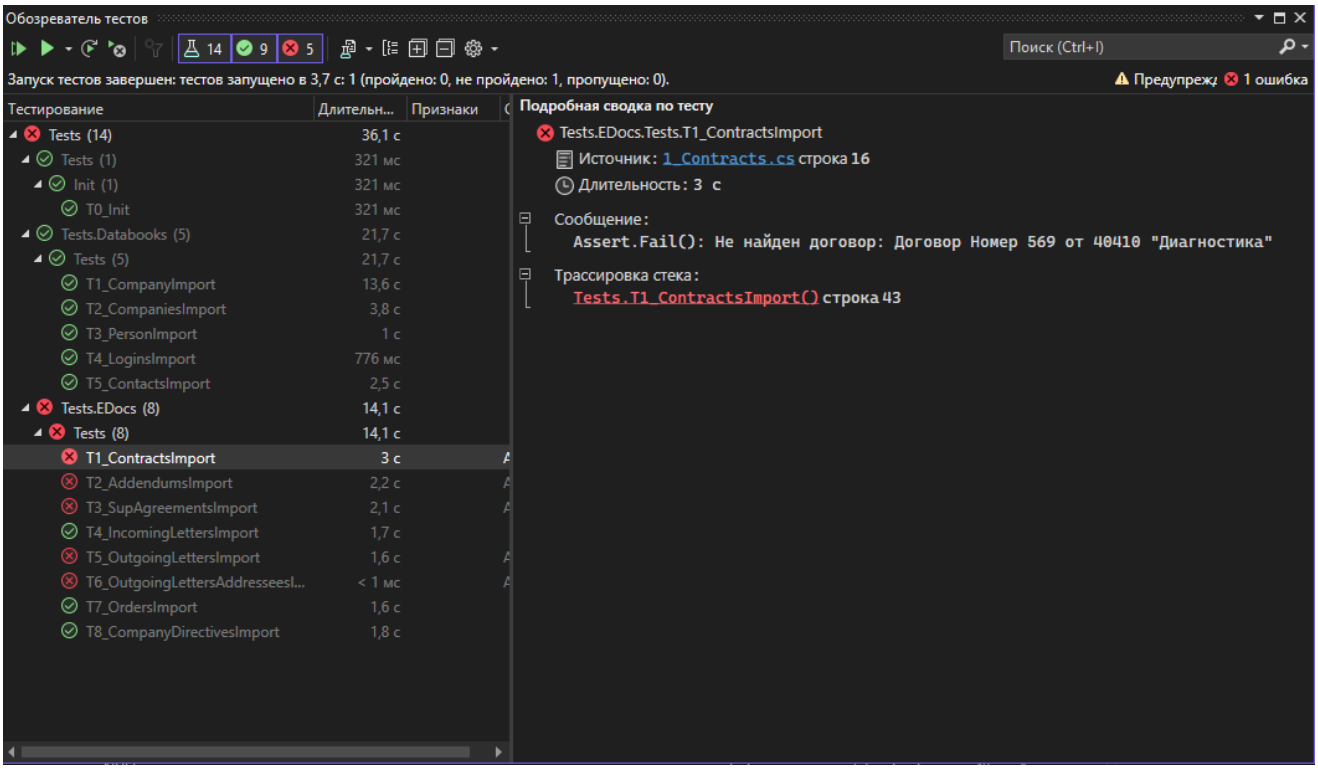


Сущность не найдена

Результат означает, что в Directum RX не создана сущность. Это могло произойти по причинам:

1. Во время импорта возникла ошибка, которая не отобразилась в результатах теста, так как была перехвачена и обработана. В этом случае запись об ошибке можно найти в лог-файле утилиты или с помощью отладчика.
2. В тесте неверно написан поиск сущности. Нужно проверить, по каким параметрам производится поиск сущности.
3. Произошла ошибка, связанная с прикладной разработкой, например, не заполнен обязательный параметр. Такие ошибки фиксируются в лог-файле сервиса интеграции.

4. Произошла ошибка в тестовых данных. Чаще всего ее можно найти в результатах импорта в файле Excel.



Ошибка при создании или изменении сущности

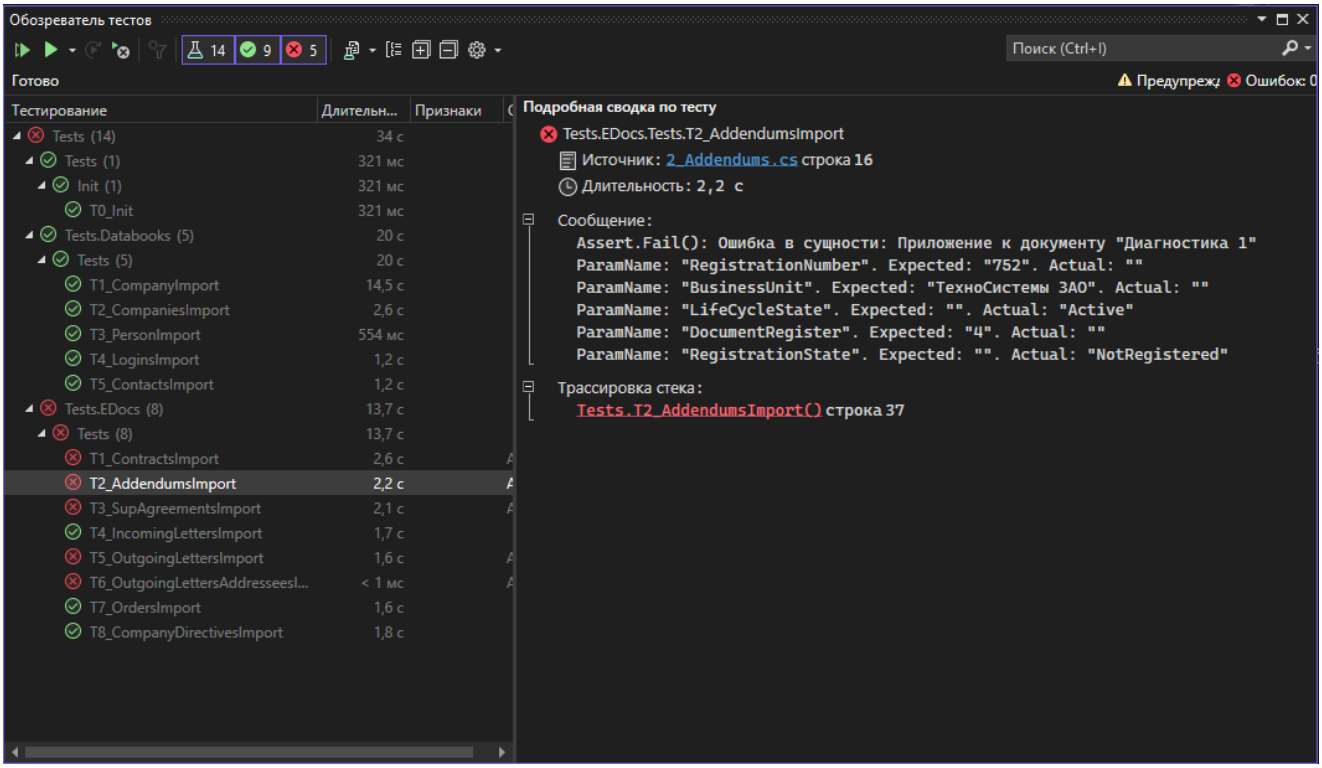
Ошибка означает, что при создании или изменении сущности:

- в Directum RX данные занесены некорректно или их недостаточно;
- данные из тестового шаблона обрабатываются некорректно.

В ошибке используются обозначения:

- Expected – ожидаемый результат из Excel;
- Actual – актуальные данные из Directum RX.

Чтобы выяснить причину ошибки, необходимо проанализировать разработку и тестовые данные. Также ошибка может возникать на стороне сервиса интеграции Directum RX в прикладном коде. Например, некоторые свойства, какие как регион или пол, могут подставляться автоматически.



Рекомендации по доработке утилиты

Базовые сущности

Большая часть классов сравнивается по имени. Для этого создан базовый класс **IEntity**, и большая часть классов наследуется от него. При создании новых сущностей для импорта и добавлении к ним тестов используйте наследование от этого класса, если он удовлетворяет потребностям, в противном случае необходимо наследоваться от класса **IEntityBase** – родителя **IEntity**, как в случае с классом **ILogins**, у сущности которого нет поля **Name**.

```
81 references | Administrator, 93 days ago | 1 author, 2 changes
public class IEntity : IEntityBase
{
    [PropertyOptions("Наименование", RequiredType.Required, PropertyType.Simple, AdditionalCharacters.ForSearch)]
    98 references | Administrator, 93 days ago | 1 author, 2 changes
    public string Name { get; set; }

    0 references | Administrator, 93 days ago | 1 author, 2 changes
    public override string ToString()
    {
        return Name;
    }
}
```

Модификация существующих тестов

Если в сущность добавилось новое поле, добавьте его в область метода Equals() в тесте сущности.

Метод **CheckParam()** принимает в себя:

- сравниваемое свойство;
- содержимое из Excel;

- имя параметра, которое отображается в ошибке;

```
var errorList = new List<string>
{
  Common.CheckParam(actualSupAgreement.RegistrationNumber, parameters[shift + 0], "RegistrationNumber"),
  Common.CheckParam(actualSupAgreement.RegistrationDate, parameters[shift + 1], "RegistrationDate"),
  Common.CheckParam(leadingDocument.RegistrationNumber, parameters[shift + 2], "LeadingDocumentRegNumber"),
  Common.CheckParam(leadingDocument.RegistrationDate, parameters[shift + 3], "LeadingDocumentRegNumberRegDate"),
  Common.CheckParam(actualSupAgreement.Counterparty, parameters[shift + 4], "Counterparty"),
  Common.CheckParam(actualSupAgreement.DocumentKind , parameters[shift + 5], "DocumentKind"),
  Common.CheckParam(actualSupAgreement.Subject, parameters[shift + 6], "Subject"),
  Common.CheckParam(actualSupAgreement.BusinessUnit, parameters[shift + 7], "BusinessUnit"),
  Common.CheckParam(actualSupAgreement.Department, parameters[shift + 8], "Department"),
  Common.CheckParam(actualSupAgreement.LastVersion(), parameters[shift + 9], "LastVersion"),
  Common.CheckParam(actualSupAgreement.ValidFrom, parameters[shift + 10], "ValidFrom"),
  Common.CheckParam(actualSupAgreement.ValidTill, parameters[shift + 11], "ValidTill"),
  Common.CheckParam(actualSupAgreement.TotalAmount, Convert.ToDouble(parameters[shift + 12], CultureInfo.InvariantCulture), "TotalAmount"),
  Common.CheckParam(actualSupAgreement.Currency, parameters[shift + 13], "Currency"),
  Common.CheckParam(actualSupAgreement.LifeCycleState, BusinessLogic.GetPropertyLifeCycleState(parameters[shift + 14]), "LifeCycleState"),
  Common.CheckParam(actualSupAgreement.ResponsibleEmployee , parameters[shift + 15], "ResponsibleEmployee"),
  Common.CheckParam(actualSupAgreement.OurSignatory, parameters[shift + 16], "OurSignatory"),
  Common.CheckParam(actualSupAgreement.Note, parameters[shift + 17], "Note"),
  Common.CheckParam(actualSupAgreement.DocumentRegister?.Id, parameters[shift + 18], "DocumentRegister"),
  Common.CheckParam(actualSupAgreement.RegistrationState, BusinessLogic.GetRegistrationsState(parameters[shift + 19]), "RegistrationState"),
  Common.CheckParam(actualSupAgreement.CaseFile?.Name, parameters[shift + 20], "CaseFile"),
  Common.CheckParam(actualSupAgreement.PlacedToCaseFileDate, parameters[shift + 21], "PlacedToCaseFileDate")
};
```

Если нет подходящей перегрузки метода **CheckParam()**, то можно добавить свою в классе **Common**.

```
/// <summary>
/// Сравнить параметры и получить строку с ошибкой.
/// </summary>
/// <param name="actual">Актуальный (из системы).</param>
/// <param name="expected">Ожидаемый (из xlsx).</param>
/// <param name="paramName">Имя параметра.</param>
/// <returns>Строку с ошибкой, если параметры не равны.</returns>
Ссылок: 65
public static string CheckParam(IEntity? actual, string expected, string paramName) => CheckParam(actual == null || string.IsNullOrEmpty(actual.Name) ? string.Empty : actual.Name, expected.Trim(), paramName);

/// <summary>
/// Сравнить параметры и получить строку с ошибкой.
/// </summary>
/// <param name="actual">Актуальный (из системы).</param>
/// <param name="expected">Ожидаемый (из xlsx).</param>
/// <param name="paramName">Имя параметра.</param>
/// <returns>Строку с ошибкой, если параметры не равны.</returns>
Ссылок: 1
public static string CheckParam(ILogins? actual, string expected, string paramName) => CheckParam(actual == null ? string.Empty : actual.LoginName, expected.Trim(), paramName);
```

Создание новых тестов

Собственные тесты можно создавать на основе стандартных тестов из комплекта поставки решения.

1. Создайте свой XLSX-файл с тестовыми данными.
2. Укажите путь к созданному на предыдущем шаге файлу в разработке класса **TestSettings**.
3. Напишите собственный тест по аналогии с существующим.

Атрибуты

Fact – указывает, что это тест. Методы с этим атрибутом попадают в обозреватель тестов.

Order – приоритет теста. Указывает, в каком порядке выполняются тесты.

Тест T0_Init

Нулевой тест – инициализация. Выполняется первым.

Используется для смены регистрационных номеров в документах. Тест добавлен, так как система не позволяет загружать документы с одним и тем же регистрационным номером. Это позволяет не менять регистрационные номера вручную.

Класс TestSettings

Класс служит для хранения путей к файлам, логина и пароля для подключения к Directum RX.

Класс Common

Хранит общие методы для тестов.

Метод **InitODataClient()** используется для инициализации Simple OData Client, если нужно выполнить действия с сущностями до начала теста. Например, для договоров перед импортом создаются их категории.

Пример создания теста

1. Создать класс, например, 4_IncomingLetters.cs.
2. Присвоить ему пространство имен **namespace Tests.EDocs**.
3. Переименовать в коде имя класса 4_IncomingLetters на разделяемый класс **public partial class Tests**.
4. В классе **TestSettings** добавить путь к файлу:

```
internal static class TestSettings
 {
   ...
   public const string IncomingLettersPathXlsx = $@"{XlsxFolderPath}\IncomingLetters.xlsx";
   ...
 }
```

5. Написать метод для сравнения результата:

```
public static string EqualsIncomingLetter(List<string> parameters, int shift = 0)
{
  // Получаем сущность из системы RX.
  var actualIncomingLetter =
 Common.GetOfficialDocument<IIncomingLetters>(parameters[shift + 0], parameters[shift +
 1], parameters[shift + 12]);

  // Получаем имя сущности.
 var name = Common.GetDocumentName(parameters[shift + 3], parameters[shift + 0],
   parameters[shift + 1], parameters[shift + 4]);

 if (actualIncomingLetter == null)
  return $"Не найдено входящее письмо: {name}";

  // Сравниваем свойства сущности из RX со значениями из файла.
 var errorList = new List<string>
 {
  Common.CheckParam(actualIncomingLetter.RegistrationNumber, parameters[shift + 0],
  "RegistrationNumber"),
  Common.CheckParam(actualIncomingLetter.RegistrationDate, parameters[shift + 1],
  "RegistrationDate"),
  Common.CheckParam(actualIncomingLetter.Correspondent, parameters[shift + 2],
  "Correspondent"),
  Common.CheckParam(actualIncomingLetter.DocumentKind, parameters[shift + 3],
  "DocumentKind"),
  Common.CheckParam(actualIncomingLetter.Subject, parameters[shift + 4], "Subject"),
  Common.CheckParam(actualIncomingLetter.Department, parameters[shift + 5],
  "Department"),
```

```
    Common.CheckParam(actualIncomingLetter.LastVersion(), parameters[shift + 6],
    "LastVersion"),
    Common.CheckParam(actualIncomingLetter.Dated, parameters[shift + 7], "Dated"),
    Common.CheckParam(actualIncomingLetter.InNumber, parameters[shift + 8], "InNumber"),
    Common.CheckParam(actualIncomingLetter.Addressee, parameters[shift + 9],
    "Addressee"),
    Common.CheckParam(actualIncomingLetter.Note, parameters[shift + 10], "Note"),
    Common.CheckParam(actualIncomingLetter.DeliveryMethod, parameters[shift + 11],
    "DeliveryMethod"),
    Common.CheckParam(actualIncomingLetter.DocumentRegister, parameters[shift + 12],
    "DocumentRegister"),
    Common.CheckParam(actualIncomingLetter.RegistrationState,
    BusinessLogic.GetRegistrationsState(parameters[shift + 13].Trim()),
    "RegistrationState")
              };

    errorList = errorList.Where(x => !string.IsNullOrEmpty(x)).ToList();
    if (errorList.Any())
     errorList.Insert(0, $"Ошибка в сущности: {name}");

    return string.Join(Environment.NewLine, errorList);
}
```