

# Пакеты

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

# Содержание

<b>Общие принципы работы с пакетами</b>	<b>5</b>
Классификация пакетов	5
Структура пакета	8
Зависимости и иерархия пакетов	9
Привязка данных к пакету	15
<b>Создать пользовательский пакет</b>	<b>15</b>
1. Создать пакет	15
2. Заполнить свойства пакета	16
3. Определить зависимости пакета	17
4. Проверить зависимости пакета Custom	18
<b>Привязать данные к пакету</b>	<b>18</b>
1. Создать раздел	18
2. Добавить в раздел демонстрационные записи	19
3. Привязать к пакету данные	20
4. Проверить привязки данных	23
<b>Пакет-проект</b>	<b>24</b>
Особенности пакета-проекта	24
Структура пакета-проекта	25
Инструменты для разработки пакета-проекта	25
Импортировать пакет-проект	25
<b>Пакет-сборка</b>	<b>26</b>
Структура и зависимости пакета-сборки	27
Операции с пакетом-сборкой	28
Конвертация пакетов	31
Импорт пакета-сборки	36
<b>Файловый контент пакетов</b>	<b>36</b>
Структура хранения файлового контента пакета	37
Bootstrap-файлы пакета	38
Версионность файлового контента	39
Генерация вспомогательных файлов	40
Предварительная генерация статического файлового контента	40
Генерация файлового контента	41
Совместимость с режимом разработки в файловой системе	43
Перенос изменений между рабочими средами	44
<b>Локализовать файловый контент с помощью конфигурационных ресурсов</b>	<b>44</b>
1. Создать модуль с локализуемыми ресурсами	44

2. Импортировать модуль локализуемых ресурсов	45
<b>Локализовать файловый контент с помощью плагина i18n</b>	<b>45</b>
1. Добавить плагин	45
2. Создать папку с локализуемыми ресурсами	45
3. Создать папки культур	46
4. Добавить файлы с локализуемыми ресурсами	46
5. Отредактировать файл bootstrap.js	47
6. Использовать ресурсы в клиентском модуле	48
<b>Использовать TypeScript при разработке клиентской функциональности</b>	<b>48</b>
1. Установить TypeScript	49
2. Перейти в режим разработки в файловой системе	49
3. Создать структуру хранения файлового контента	50
4. Реализовать валидацию на языке TypeScript	51
5. Выполнить компиляцию исходных кодов TypeScript в исходные коды JavaScript	52
6. Выполнить генерацию вспомогательных файлов	54
7. Проверить результат выполнения примера	54
<b>Создать Angular-компонент для использования в Creatio</b>	<b>57</b>
Создание пользовательского Angular-компонента	57
Подключение Custom Element в Creatio	61
Работа с данными	62
Использование Shadow DOM	64

# Общие принципы работы с пакетами


## Основы

Любой продукт Creatio представляет собой определенный набор пакетов.




**Пакет Creatio** — это совокупность [конфигурационных элементов](#), которые реализуют блок функциональности. Физически пакет представляет собой папку, содержащую определенный набор вложенных папок и файлов.

## Классификация пакетов

### • Простые пакеты:

-  — предустановленные пакеты. Являются частью приложения и по умолчанию устанавливаются в рабочее пространство. Недоступны для изменения.

**Виды** предустановленных пакетов:

- Пакеты с базовой функциональностью (например, `Base`, `NUI` ).
- Пакеты сторонних разработчиков.  
Устанавливаются из \*.zip-архивов с помощью [Creatio IDE](#) или с помощью [утилиты WorkspaceConsole](#).
-  — пользовательские пакеты. Созданы другими пользователями системы и заблокированы для изменения в системе контроля версий. Недоступны для изменения.
-  — пользовательские пакеты. Созданы текущим пользователем либо загружены из системы контроля версий. Доступны для изменения.
- **Пакет-проект** — пакет, который позволяет разрабатывать функциональность как обычный C#-проект. Подробнее читайте в статье [Пакет-проект](#).
- **Пакет-сборка** — пакет, исходный и автогенерируемый код которого компилируется в отдельную сборку. Возможность использования пакета-сборки доступна для приложений Creatio версии 7.18.3 и выше. В Creatio IDE пакет-сборка отображается как пользовательский пакет (). Подробнее читайте в статье [Пакет-сборка](#).

Сравнение пакетов Creatio представлено в таблице ниже.

## Сравнение пакетов Creatio

Ссылки на другие пакеты	Конфигурационные элементы пакета	Файловый контент	Разработка в Creatio IDE	Путь для компиляции кода
Пакет				
+	+	+	+	...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Terrasoft.Configuration.dll
Пакет-проект				
-	-	+	- (используется <a href="#">внешняя IDE</a> )	<p>Для .NET Framework:</p> <pre>...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg\\[Имя пакета]\\Files\\Bin\\[Имя пакета].dll</pre> <p>Для .NET Core:</p> <pre>...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg\\[Имя пакета]\\Files\\Bin\\netstandard\\[Имя пакета].dll</pre>
Пакет-сборка				
+	+(с ограничениями)	+	+	<p>Для .NET Framework:</p> <pre>...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg\\[Имя пакета]\\Files\\Bin\\[Имя пакета].dll</pre> <p>Для .NET Core:</p> <pre>...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg\\[Имя пакета]\\Files\\Bin\\netstandard\\[Имя пакета].dll</pre>

В этой статье рассматриваются простые пакеты.

Для расширения или изменения функциональности необходимо установить пакет с требуемой функциональностью. Разработка дополнительной функциональности и модификация существующей выполняется исключительно в пользовательских пакетах.

## Основные пакеты приложения

К основным пакетам приложения можно отнести пакеты, которые обязательно присутствуют во всех продуктах.

Основные пакеты приложения

Название пакета	Описание
Base	Базовые схемы основных объектов, разделов системы и связанных с ними схем объектов, страниц, процессов и т. д.
Platform	Модули и страницы мастера разделов, дизайнеров реестра и итогов и т. п.
Managers	Клиентские модули менеджеров схем.
NUI	Функциональность, связанная с пользовательским интерфейсом системы.
UIv2	
DesignerTools	Схемы дизайнеров и их элементов.
ProcessDesigner	Схемы дизайнера процессов.

## Пакет Custom

В процессе работы мастер разделов или мастер деталей создает схемы, которые необходимо сохранить в пользовательский пакет. В только что установленном приложении нет пакетов, доступных для изменения, а в предустановленные пакеты невозможно внести изменения. Для этого предназначен специальный предустановленный пакет `Custom`. Он позволяет добавлять схемы как вручную, так и с помощью мастеров.

**Особенности** пакета `Custom`:

- Пакет `Custom` невозможно добавить в систему контроля версий. Поэтому его схемы можно перенести на другую рабочую среду только при помощи функциональности [экспорта и импорта](#) пакетов.
- В отличие от других предустановленных пакетов, пакет `Custom` невозможно выгрузить в файловую систему при помощи [утилиты WorkspaceConsole](#).
- В пакете `Custom` установлены зависимости от всех предустановленных пакетов приложения. При создании или установке пользовательского пакета в пакет `Custom` автоматически добавляется зависимость от пользовательского пакета. Таким образом пакет `Custom` всегда должен быть последним в иерархии пакетов.
- В зависимости пользовательских пакетов невозможно добавить пакет `Custom`.

Рекомендуемые **варианты использования** пакета `Custom`:

- Не предполагается перенос изменений в другую рабочую среду.  
В процессе работы мастер разделов или мастер деталей не только создает различные схемы, но и

привязывает данные к текущему пакету. Для пакета `Custom` не предусмотрено использование стандартного механизма импорта пакетов. Поэтому если текущим пакетом является пакет `Custom`, то перенести привязанные данные в другой пользовательский пакет можно только с помощью запросов к базе данных. Мы настоятельно не рекомендуем использовать этот способ, поскольку изменения могут повлиять на структуру базы данных, что приведет к неработоспособности приложения.

При значительной доработке пользовательской функциональности необходимо [создать пользовательский пакет](#) с использованием системы контроля версий.

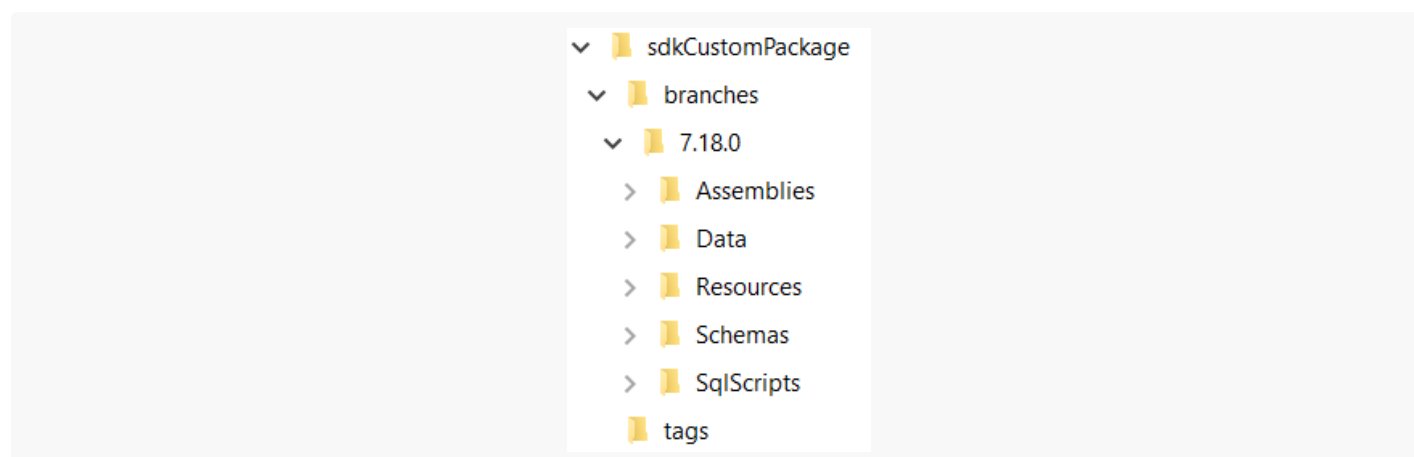
- Изменения выполняются при помощи мастеров или вручную, при этом объем изменений небольшой.
- Нет необходимости использовать систему контроля версий.

## Пользовательский пакет

Чтобы выполнять разработку в пользовательском пакете, необходимо в системной настройке [ *Текущий пакет* ] (код `CurrentPackageId`) указать имя пользовательского пакета.

## Структура пакета

При фиксации пакета в [системе контроля версий](#) в хранилище пакета создается папка с именем пакета.



**Структура** папки с именем пакета:

- Папка `branches`.  
**Назначение** — хранение версий текущего пакета. Версия пакета — отдельная вложенная папка, имя которой совпадает с номером версии пакета в системе (например, 7.18.0).
- Папка `tags`.  
**Назначение** — хранение меток. **Метки** в системе контроля версий — это "снимок" проекта в определенный момент времени, статическая копия файлов, необходимая для фиксации этапа разработки.

В Creatio по умолчанию включен режим работы с SVN. Для настройки работы с SVN необходимо изменить значение атрибута `connectionString` настройки `defPackagesWorkingCopyPath` конфигурационного файла `ConnectionStrings.config`. Эта настройка содержит путь к каталогу на диске, в котором размещаются рабочие копии.



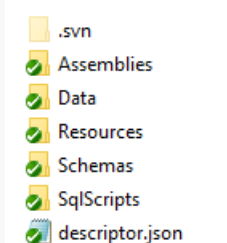
**ConnectionStrings.config**

```
<add name="defPackagesWorkingCopyPath" connectionString="TEMP\APPLICATION\WORKSPACE\TerrasoftPac
```

В режиме работе с SVN приложение Creatio использует собственную рабочую копию каждого пользовательского пакета, для которого подключена версияльность. Содержимое рабочей копии представляет собой пользовательские пакеты в виде набора папок и файлов. SVN-клиент, встроенный в Creatio, синхронизирует это содержимое с SVN-репозиторием. В качестве значения настройки `defPackagesWorkingCopyPath` рекомендуется установить путь на фиксированный каталог, поскольку временный каталог, установленный по умолчанию, может быть очищен операционной системой. Для приложения Creatio на платформе .NET Framework в качестве каталога для хранения рабочих копий запрещено указывать `...\Terrasoft.WebApp\Terrasoft.Configuration\Pkg`.

**Структура папки пакета** в файловой системе:

- Папка `Schemas` — содержит схемы пакета.
- Папка `Assemblies` — содержит внешние сборки, привязанные к пакету.
- Папка `Data` — содержит данные, привязанные к пакету.
- Папка `SqlScripts` — содержит SQL-сценарии, привязанные к пакету.
- Папка `Resources` — содержит локализованные ресурсы пакета.
- Папка `Files` — содержит [файловый контент](#) пакета.
- Файл `descriptor.json` — хранит метаданные пакета в формате JSON. К метаданным пакета относятся идентификатор, наименование, версия, зависимости и т. д.



## Зависимости и иерархия пакетов

Разработка приложения Creatio базируется на основных принципах проектирования программного обеспечения, в частности, **принципа отсутствия повторений (DRY)**.

В архитектуре Creatio этот принцип реализован с помощью **зависимостей пакетов**. Каждый пакет содержит определенную функциональность приложения, которая не должна повторяться в других пакетах. Чтобы такую функциональность можно было использовать в другом пакете, необходимо пакет, содержащий эту функциональность, добавить в зависимости пакета, в котором она будет использоваться.

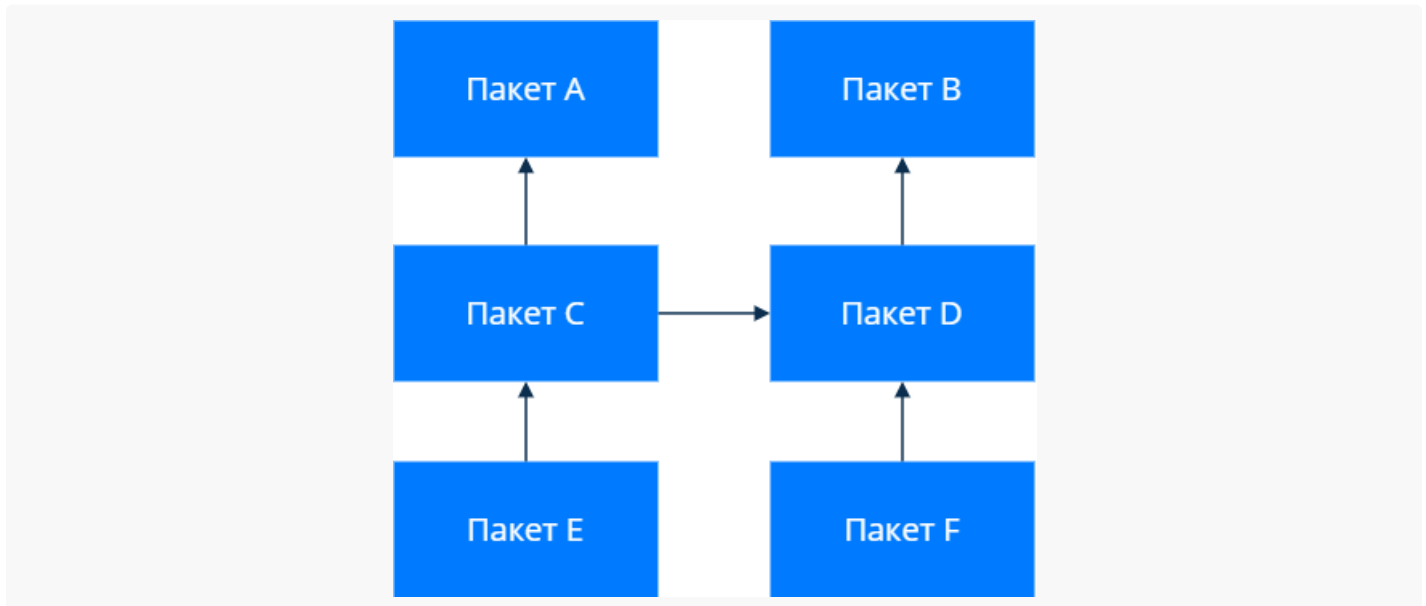
**Виды зависимостей:**

- Чтобы текущий пакет наследовал всю **функциональность приложения**, в качестве родительского

пакета необходимо выбрать пакет, который в иерархии находится следующим после пакета `Custom`.

- Чтобы текущий пакет наследовал **функциональность пакета**, в качестве родительского пакета необходимо выбрать пакет, функциональность которого необходимо наследовать.

Пакет может иметь несколько зависимостей. Например, в пакете С установлены зависимости от пакетов А и D. Таким образом, вся функциональность пакетов А и D доступна в пакете С.



Зависимости пакетов формируют **иерархические цепочки**. Это означает, что в пакете доступна не только функциональность дочернего пакета, но и функциональность всех пакетов, для которых дочерний пакет является родительским. Ближайшей аналогией иерархии пакетов является иерархия наследования классов в объектно-ориентированном программировании. Так, например, в пакете E доступна функциональность не только пакета C, от которого он зависит, но и функциональность пакетов А, В и D. А в пакете F доступна функциональность пакетов В и D.

## Иерархия пакетов приложения

Иерархия и зависимости пакетов отображены на **диаграмме зависимостей пакетов**. Чтобы открыть диаграмму:

1. Перейдите в раздел [ Конфигурация ] ([ Configuration ]).
2. В выпадающем списке [ Действия ] ([ Actions ]) панели инструментов в группе [ Пакеты ] ([ Packages ]) выберите [ Диаграмма зависимостей пакетов ] ([ Package dependencies diagram ]).

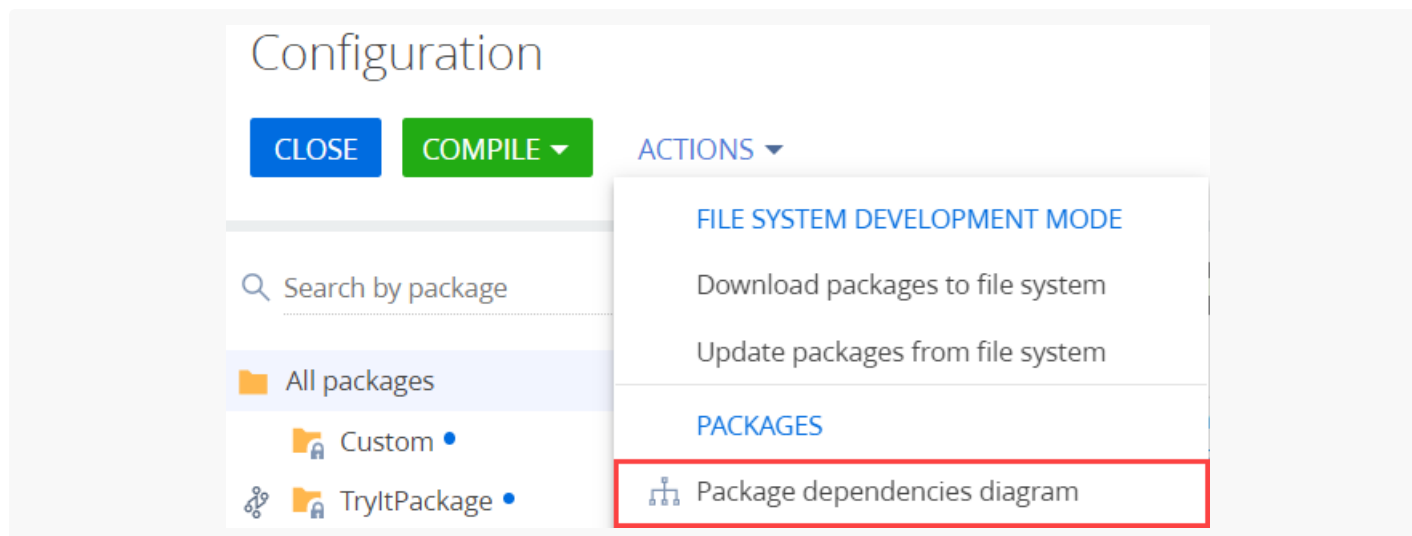
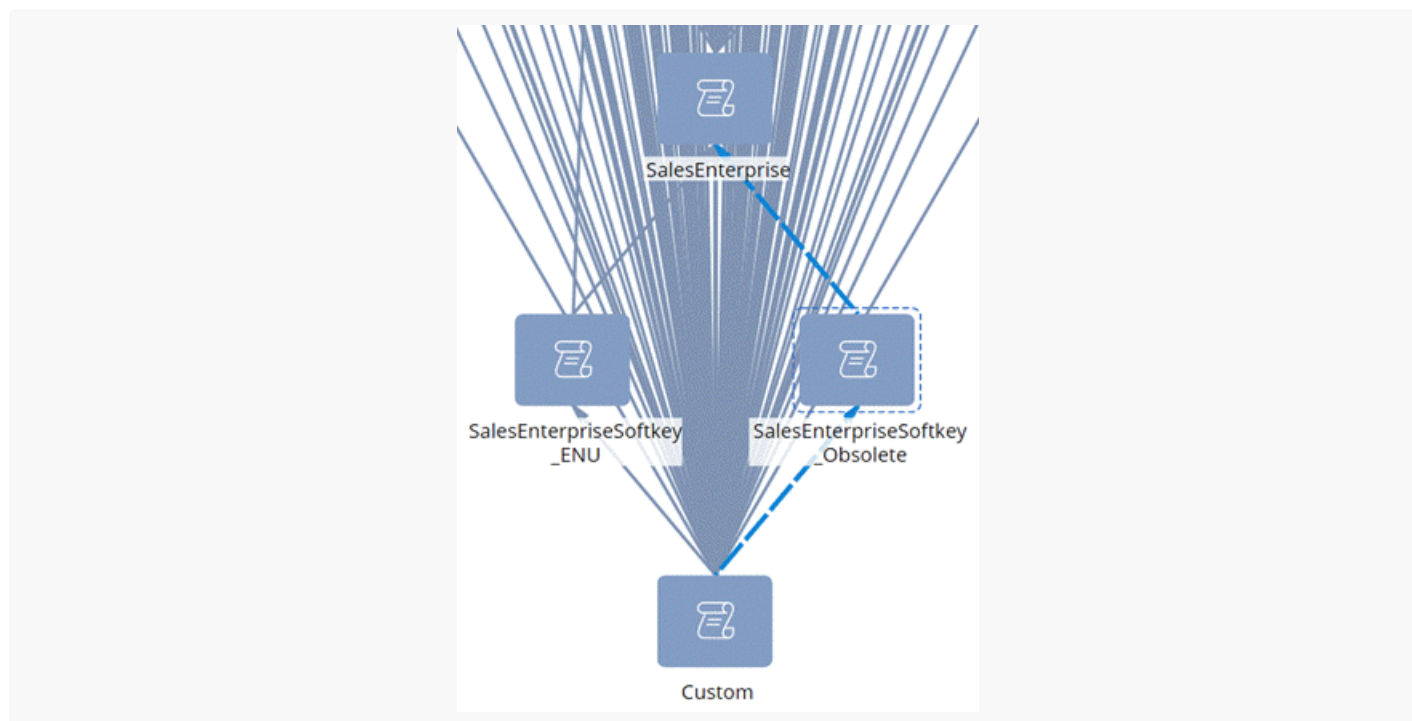


Диаграмма зависимостей будет открыта в новой вкладке.

Если кликнуть по узловому элементу диаграммы с именем пакета, то в виде анимированных стрелок отобразятся связи с другими пакетами. Например, в продукте SalesEnterprise пакет `SalesEnterpriseSoftkey_Obsolete` зависит только от пакета `SalesEnterprise` и всех его родительских пакетов. Также пакет `SalesEnterpriseSoftkey_Obsolete` является родительским для пакета `Custom`.



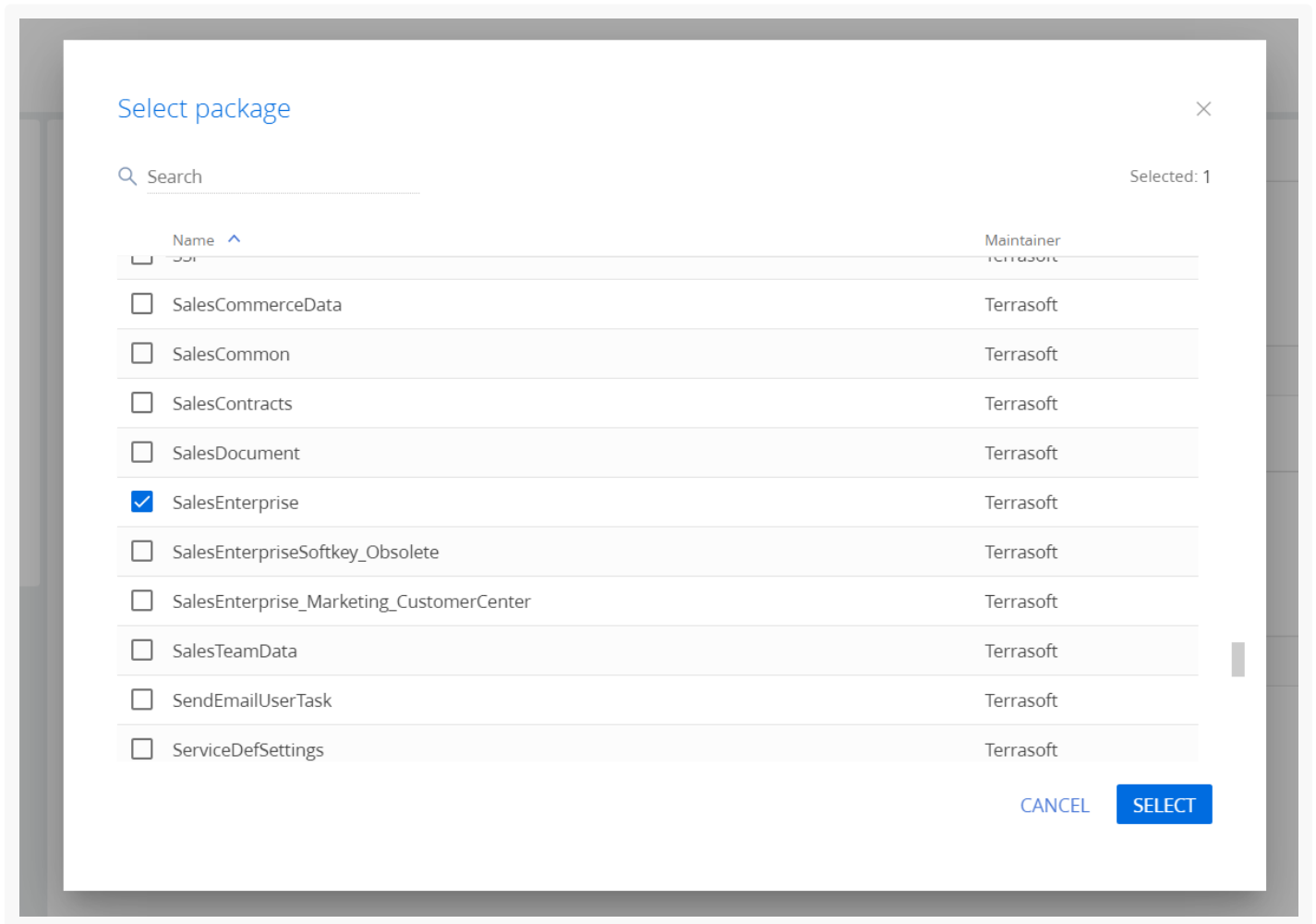
## Добавление зависимостей пакета

Зависимости можно добавить в пользовательский пакет при создании пакета или уже после него.

Чтобы **добавить зависимости**:

1. Перейдите на страницу пакета.

- На вкладке [ *Зависимости* ] ([ *Dependencies* ]) на детали [ *Зависит от пакетов* ] ([ *Depends on packages* ]) нажмите кнопку [ *Добавить* ] ([ *Add* ]).
- В появившемся окне справочника пакетов выберите необходимый пакет и нажмите кнопку [ *Выбрать* ] ([ *Select* ]).



После этого выбранный пакет будет отображен в списке зависимостей текущего пакета, а при добавлении новой зависимости он будет скрыт из справочника пакетов.

## Package properties

[CLOSE](#) [ACTIONS ▾](#)

Name  
TestPackage1

Repository address  
http://tscore-svn:8050/svn/tsfmdoc/SDKP...

Repository version  
SDKPackages

Package version  
1.0.0

Maintainer  
Customer i

Description

DEPENDENCIES

SYSTEM INFORMATION

Depends on Packages i

Search by package

Name ▴

SalesEnterpriseSoftkey\_ENU

+ Add

Dependent Packages i

Search by package

Name ▴

Custom

После создания пакет автоматически добавляется в зависимости предустановленного пакета `Custom`.

© 2022 Terrasoft. Все права защищены.

Package properties

CLOSE
ACTIONS ▼

Name  
TestPackage1

Repository address  
http://tscore-svn:8050/svn/tsfmdoc/SDKP...

Repository version  
SDKPackages

Package version  
1.0.0

Maintainer  
Customer ⓘ

Description

DEPENDENCIES

SYSTEM INFORMATION

Depends on Packages ⓘ

Search by package

Name ^

SalesEnterpriseSoftkey\_ENU

+ Add

Dependent Packages ⓘ

Search by package

Name ^

Custom

## Список зависимостей в метаданных пакета

Список зависимостей хранится в **метаданных пакета**, которые можно посмотреть в свойстве `DependsOn` объекта, определенного в файле `descriptor.json`.

Свойство `DependsOn` — массив объектов, в которых указывается имя пакета, его версия и уникальный идентификатор, по которому можно определить пакет в базе данных приложения. Файл `descriptor.json` создается приложением для каждой версии пакета.

### Пример файла `descriptor.json`

```
{
  "Descriptor": {
    "Uid": "51b3ed42-678c-4da3-bd16-8596b95c0546",
    "PackageVersion": "7.18.0",
    "Name": "UsrDependentPackage",
    "ModifiedOnUtc": "\\Date(1522653150000)\\",
  }
}
```

```

    "Maintainer": "Customer",
    "DependsOn": [
      {
        "Uid": "e14dcfb1-e53c-4439-a876-af7f97083ed9",
        "PackageVersion": "7.18.0",
        "Name": "SalesEnterprise"
      }
    ]
  }
}

```

## Привязка данных к пакету

При переносе изменений между [рабочими средами](#) часто возникает необходимость вместе с разработанной функциональностью предоставлять некоторые данные. Это может быть, например, наполнение справочников, новые системные настройки, демонстрационные записи раздела и т. д.

При создании раздела с помощью мастера к пакету автоматически привязываются данные, необходимые для регистрации и корректной работы раздела.


<div> <div>+ Add ▾</div> <div>Data ▾</div> <div>Filters ▾</div> <div>Search</div> <div>⚙</div> </div>						
Name ▾	Title	Status	Type	Object	Modified on	Package
SysModule_SectionManager_8a3879e5f91c49cf81a9d7dd5b3a47c4			Data	SysModule	4/23/2018, 1:12:06 PM	sdkBookExample
SysModuleInWorkplace_SectionInWorkplaceManager_a69d5bf b41024326adeca22d02f6dde6			Data	SysModuleInWorkplace	4/23/2018, 1:12:07 PM	sdkBookExample
SysModuleEntity_SysModuleEntityManager_8654e87e4fd34e0a91d903ad427373d9			Data	SysModuleEntity	4/23/2018, 1:12:04 PM	sdkBookExample
SysModuleEdit_SysModuleEditManager_e3a3124a5cd346919574d2d5f5f750c4			Data	SysModuleEdit	4/23/2018, 1:12:05 PM	sdkBookExample
SysImage_f8d3f0121ed2433a996610d4b00cf09a			Data	SysImage	4/23/2018, 1:12:07 PM	sdkBookExample
SysDetail_DetailManager_b78e48663abc45cb916779059502af6b			Data	SysDetail	4/23/2018, 1:35:29 PM	sdkBookExample

Привязать необходимые данные к пакету, содержащему разработанную функциональность, можно в разделе [ *Конфигурация* ] ([ *Configuration* ]).

## Создать пользовательский пакет

 Легкий

### 1. Создать пакет

1. Перейдите в дизайнер системы по кнопке .
2. В блоке [ *Конфигурирование разработчиком* ] ([ *Admin area* ]) перейдите по ссылке [ *Управление конфигурацией* ] ([ *Advanced settings* ]).

3. В области работы с пакетами нажмите кнопку .

## 2. Заполнить свойства пакета

При нажатии на кнопку будет отображена карточка пакета, в которой необходимо заполнить свойства пакета.

### Свойства пакета:

- [ *Название* ] ([ *Name* ]) — название пакета (обязательное свойство). Не может совпадать с названием существующих пакетов.
- [ *Описание* ] ([ *Description* ]) — описание пакета, например, расширенная информация о функциональности, которая будет реализована в пакете.
- [ *Хранилище системы контроля версий* ] ([ *Version control system repository* ]) — название хранилища системы контроля версий, в котором будут фиксироваться изменения пакета (обязательное свойство). Хранилища, которые находятся в перечне хранилищ конфигурации, но не помечены как активные, не попадут в выпадающий список доступных хранилищ.

**Важно.** Поле [ *Хранилище системы контроля версий* ] ([ *Version control system repository* ]) заполняется при создании нового пакета и в дальнейшем недоступно для редактирования.

- [ *Версия* ] ([ *Version* ]) — версия пакета (обязательное свойство). Версия пакета может содержать цифры, символы латинского алфавита и знаки "." и "\_". Добавляемое значение должно начинаться с цифры или буквы. Все элементы пакета имеют ту же версию, что и сам пакет. Указываемая версия пакета не обязательно должна совпадать с фактической версией приложения.

### Package

Name\*

sdkTestPackage

Description

Version control system repository

SDKPackages

Version \*

7.18.0

CANCEL

CREATE AND ADD DEPENDENCIES

SAVE



Содержимое свойств пакета будет сохранено в метаданных пакета.

#### Метаданные свойств пакета

```
{
  "Descriptor": {
    "Uid": "1c1443d7-87df-4b48-bfb8-cc647755c4c1",
    "PackageVersion": "7.18.0",
    "Name": "NewPackage",
    "ModifiedOnUtc": "\/Date(1522657977000)\/",
    "Maintainer": "Customer",
    "DependsOn": []
  }
}
```

Кроме указанных выше свойств, метаданные пакета содержат информацию о зависимостях (свойство `DependsOn`) и информацию о разработчике (`Maintainer`). Значение свойства `Maintainer` устанавливается с помощью системной настройки [ *Издатель* ] (код `Maintainer`).

## 3. Определить зависимости пакета

Чтобы текущий пакет наследовал функциональность приложения, необходимо определить **зависимости пакета**.

Чтобы **добавить зависимости** пакета:

1. В карточке пакета нажмите кнопку [ *Создать и добавить зависимости* ] ([ *Create and add dependencies* ]).
2. На вкладке [ *Зависимости* ] ([ *Dependencies* ]) в детали [ *Зависит от пакетов* ] ([ *Depends on packages* ]) установите необходимые зависимости. Чтобы текущий пакет наследовал всю **функциональность приложения**, в качестве родительского пакета необходимо выбрать пакет, который в иерархии находится следующим после пакета [ *Custom* ].

Package properties

CLOSE
ACTIONS

Name  
NewPackage
Repository address
Repository version  
SDKPackages
Package version  
7.17.0
Maintainer  
Customer ⓘ
Description  
new package creation

DEPENDENCIES
SYSTEM INFORMATION

Depends on Packages ⓘ
Search by package
There are no packages
+ Add

Dependent Packages ⓘ
Search by package
Name ^
Custom

## 4. Проверить зависимости пакета [ *Custom* ]

В пакете [ *Custom* ] должны быть установлены зависимости от всех пакетов приложения. Поэтому необходимо удостовериться в том, что в нем установлена зависимость от созданного пакета.

# Привязать данные к пакету

 Средний

**Пример.** Для пользовательского раздела [ *Книги* ] ([ *Books* ]) привязать демонстрационные записи и связанные с ними записи других разделов.

Демонстрационные записи:

- Книга David Flanagan "JavaScript: The Definitive Guide: Activate Your Web Pages", ISBN 978-0596805524, издательство "Apress", стоимость \$33.89.
- Книга Andrew Troelsen "Pro C# 7: With .NET and .NET Core", ISBN 978-1484230176, издательство "Apress", стоимость \$56.99.

## 1. Создать раздел

В нашем примере в [мастере разделов](#) предварительно был создан раздел [ *Книги* ] ([ *Books* ]). Поля раздела представлены в таблице.

Свойства колонок страницы записей раздела

[ <b>Заголовок</b> ] ([ <b>Title</b> ])	[ <b>Код</b> ] ([ <b>Code</b> ])	Тип данных	Обязательность поля
[ <b>Название</b> ] ([ <b>Name</b> ])	UsrName	Строка (String)	Обязательное поле
[ <b>ISBN</b> ]	UsrISBN	Строка (String)	
[ <b>Автор</b> ] ([ <b>Author</b> ])	UsrAuthor	Справочник (Lookup) [ <b>Контакт</b> ] ([ <b>Contact</b> ])	Обязательное поле
[ <b>Издатель</b> ] ([ <b>Publisher</b> ])	UsrPublisher	Справочник (Lookup) [ <b>Контрагент</b> ] ([ <b>Account</b> ])	Обязательное поле
[ <b>Стоимость</b> ] ([ <b>Price</b> ])	UsrPrice	Дробное число (Decimal)	

Создание раздела подробно рассмотрено в статье [Создать новый раздел](#).

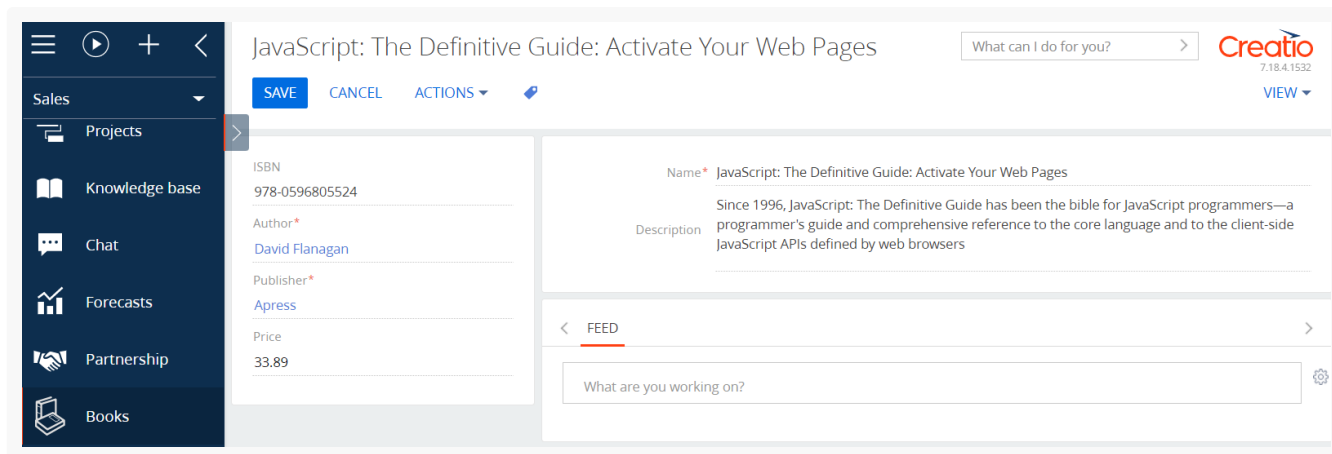
## 2. Добавить в раздел демонстрационные записи

Чтобы **добавить записи** в реестр раздела [ **Книги** ] ([ **Books** ]):

1. В разделе [ **Контакты** ] ([ **Contacts** ]) добавьте запись и заполните поле [ **ФИО** ] ([ **Full name** ]) значением "David Flanagan".
2. В разделе [ **Контакты** ] ([ **Contacts** ]) добавьте запись и заполните поле [ **ФИО** ] ([ **Full name** ]) значением "Andrew Troelsen".
3. В разделе [ **Контрагенты** ] ([ **Accounts** ]) добавьте запись и заполните поле [ **Название** ] ([ **Name** ]) значением "Apress".

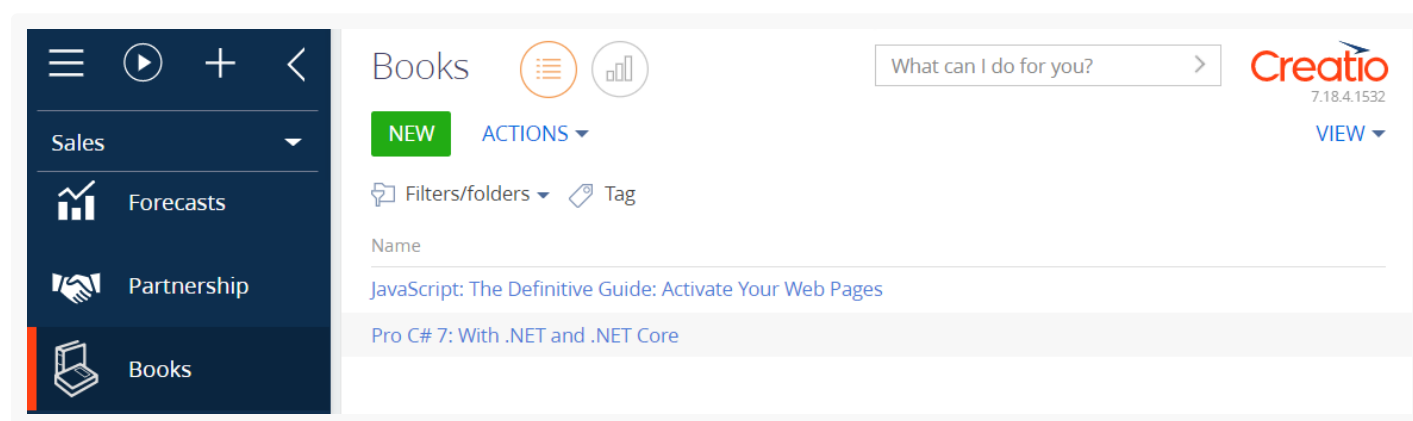
### 4. **Добавьте книгу** JavaScript: The Definitive Guide: Activate Your Web Pages :

- a. Перейдите в раздел [ **Книги** ] ([ **Books** ]).
- b. Нажмите [ **Добавить** ] ([ **New** ]).
- c. Заполните **поля** карточки книги:
  - [ **Название** ] ([ **Name** ]) — "JavaScript: The Definitive Guide: Activate Your Web Pages".
  - [ **ISBN** ] — "978-0596805524".
  - [ **Автор** ] ([ **Author** ]) — выберите "David Flanagan".
  - [ **Издатель** ] ([ **Publisher** ]) — выберите "Apress".
  - [ **Стоимость** ] ([ **Price** ]) — "33.89".



5. Аналогичным образом добавьте книгу `Pro C# 7: With .NET and .NET Core`.

Реестр раздела [ Книги ] ([ Books ]) представлен на рисунке ниже.



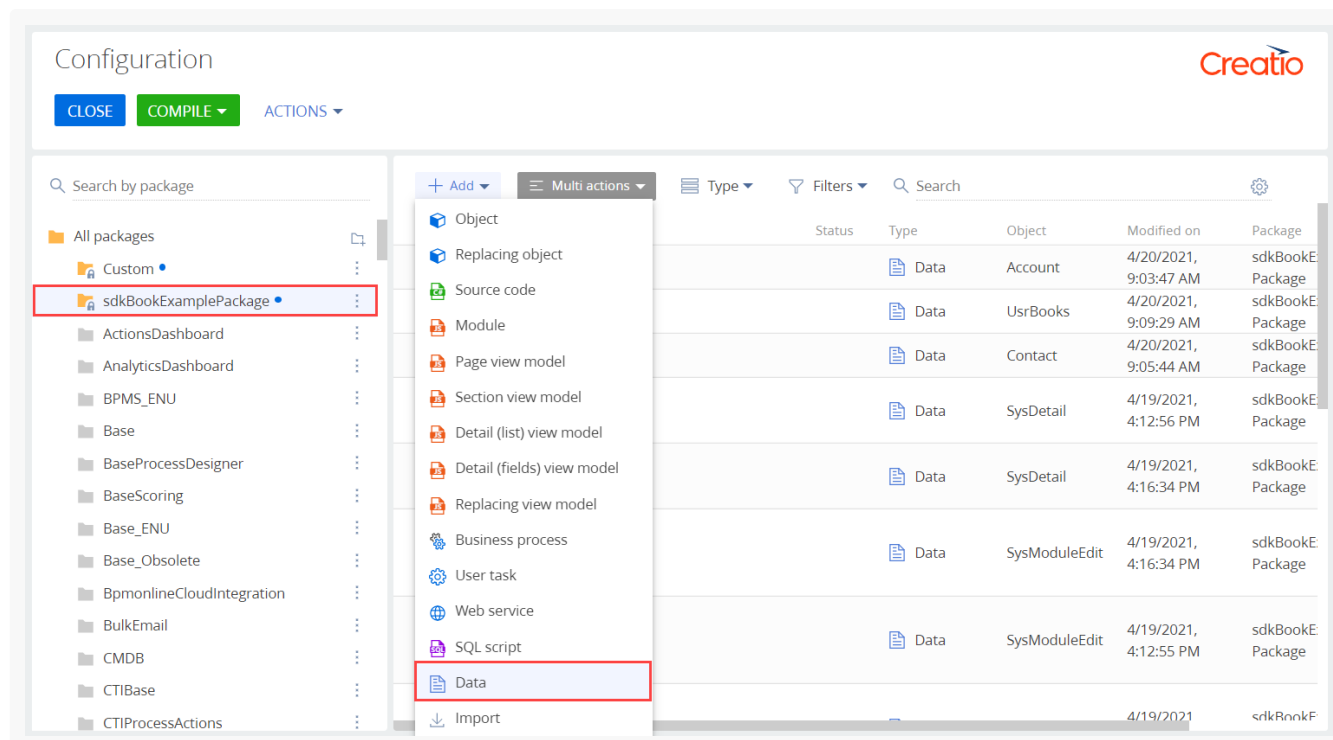
### 3. Привязать к пакету данные

Поскольку записи раздела [ Книги ] ([ Books ]) связаны с записями раздела [ Контакты ] ([ Contacts ]) по колонке [ *UsrAuthor* ], то сначала необходимо привязать к пакету сведения об авторах.

Чтобы **выполнить привязку** данных к пакету:

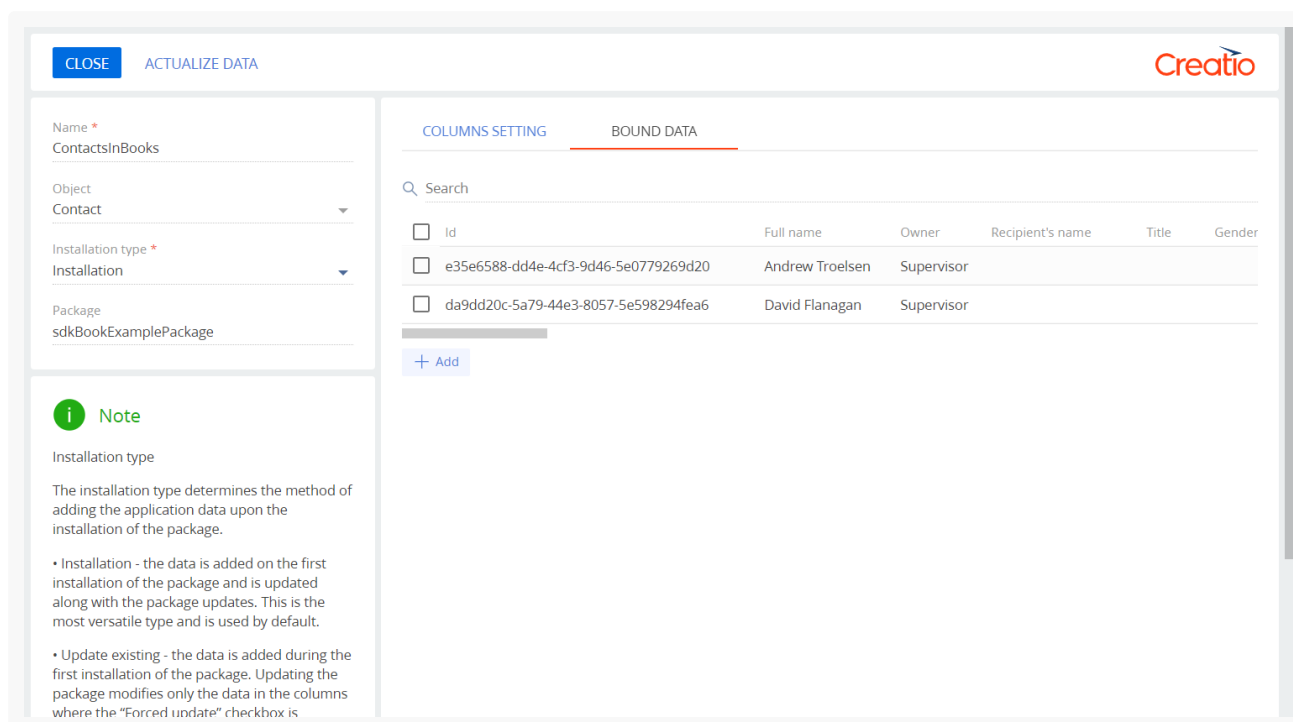
1. Выполните **привязку контактов**:

- Перейдите в раздел [ Конфигурация ] ([ Configuration ]) и выберите пользовательский пакет.
- На панели инструментов рабочей области нажмите кнопку [ Добавить ] ([ Add ]) и выберите в списке вид конфигурационного элемента [ Данные ] ([ Data ]).



с. Заполните **свойства** страницы привязки данных:

- [ *Название* ] ([ *Name* ]) — "ContactsInBooks".
- [ *Объект* ] ([ *Object* ]) — "Контакт" ("Contact").
- [ *Тип установки* ] ([ *Installation type* ]) — "Установка" ("Installation").
- На вкладке [ *Прикрепленные данные* ] ([ *Bound data* ]) выберите записи, которые в колонке [ *ФИО* ] ([ *Full name* ]) содержат значения "David Flanagan" и "Andrew Troelsen".



е. Сохраните данные.

## 2. Выполните **привязку контрагента**:

- Перейдите в раздел [ *Конфигурация* ] ([ *Configuration* ]) и выберите пользовательский пакет.
- На панели инструментов рабочей области нажмите кнопку [ *Добавить* ] ([ *Add* ]) и выберите в списке вид конфигурационного элемента [ *Данные* ] ([ *Data* ]).
- Заполните **свойства** страницы привязки данных:
  - [ *Название* ] ([ *Name* ]) — "AccountsInBooks".
  - [ *Объект* ] ([ *Object* ]) — "Контрагент" ("Account").
  - [ *Тип установки* ] ([ *Installation type* ]) — "Установка" ("Installation").
  - На вкладке [ *Прикрепленные данные* ] ([ *Bound data* ]) выберите запись, которая в колонке [ *Название* ] ([ *Name* ]) содержит значение "Apress".

**BOUND DATA**

Id	Name	Owner	Business entity	Primary contact	Pa
a4707033-30ee-4197-a688-c9bd744638b7	Apress	Supervisor			

[+ Add](#)

**Note**

**Installation type**

The installation type determines the method of adding the application data upon the installation of the package.

- Installation - the data is added on the first installation of the package and is updated along with the package updates. This is the most versatile type and is used by default.
- Update existing - the data is added during the first installation of the package. Updating the package modifies only the data in the columns where the "Forced update" checkbox is

е. Сохраните данные.

## 3. Выполните **привязку книг**:

- Перейдите в раздел [ *Конфигурация* ] ([ *Configuration* ]) и выберите пользовательский пакет.
- На панели инструментов рабочей области нажмите кнопку [ *Добавить* ] ([ *Add* ]) и выберите в списке вид конфигурационного элемента [ *Данные* ] ([ *Data* ]).
- Заполните **свойства** страницы привязки данных:
  - [ *Название* ] ([ *Name* ]) — "Books".
  - [ *Объект* ] ([ *Object* ]) — "UsrBooks".
  - [ *Тип установки* ] ([ *Installation type* ]) — "Установка" ("Installation").

- d. На вкладке [ *Прикрепленные данные* ] ([ *Bound data* ]) выберите записи, которые в колонке [ *Название* ] ([ *Name* ]) содержат значения "JavaScript: The Definitive Guide: Activate Your Web Pages" и "Pro C# 7: With .NET and .NET Core".

**Bound Data**

Search

<input type="checkbox"/> Id	Created on	Created by	Modified on	Modified by	Active processes	Name
<input type="checkbox"/> c44dc8a5-4934-46f4-a3d9-0a0b9ca63409	4/19/2021, 5:05:47 PM	Supervisor	4/19/2021, 5:05:47 PM	Supervisor	0	JavaScript: The Definitive Guide: Activate Your Web Pages
<input type="checkbox"/> ab83e661-cd07-47b8-a646-ef76746a10a4	4/19/2021, 5:11:10 PM	Supervisor	4/19/2021, 5:11:10 PM	Supervisor	0	Pro C# 7: With .NET and .NET Core

[+ Add](#)

**Note**

Installation type

The installation type determines the method of adding the application data upon the installation of the package.

- Installation - the data is added on the first installation of the package and is updated along with the package updates. This is the most versatile type and is used by default.
- Update existing - the data is added during the first installation of the package. Updating the package modifies only the data in the columns where the "Forced update" checkbox is

- e. Сохраните данные.

## 4. Проверить привязки данных

В результате выполнения примера к пользовательскому пакету будут привязаны данные разделов "[ *Книги* ]" ([ *Books* ]), "[ *Контакты* ]" ([ *Contacts* ]), "[ *Контрагенты* ]" ([ *Accounts* ]).

Configuration

CLOSE COMPPILE ACTIONS

Search by package

All packages

- Custom
- sdkBookExamplePackage
- ActionsDashboard
- AnalyticsDashboard
- BPMS\_ENU
- Base
- BaseProcessDesigner
- BaseScoring
- Base\_ENU
- Base\_Obsolete
- BpmonlineCloudIntegration
- BulkEmail
- CMDB
- CTIBase
- CTIProcessActions

+ Add Multi actions Type Filters Search

Name	Title	Status	Type	Object	Modified on	Package
AccountsInBooks *			Data	Account	4/20/2021, 9:03:47 AM	sdkBookExamplePackage
Books *			Data	UsrBooks	4/20/2021, 9:09:29 AM	sdkBookExamplePackage
ContactsInBooks *			Data	Contact	4/20/2021, 9:05:44 AM	sdkBookExamplePackage
SysDetail_DetailManager_987f2dae9a1e4eb79e8861dc9428a1e *			Data	SysDetail	4/19/2021, 4:12:56 PM	sdkBookExamplePackage
SysDetail_DetailManager_a61ff163c0f146f1be7240bb57345693 *			Data	SysDetail	4/19/2021, 4:16:34 PM	sdkBookExamplePackage
SysModuleEdit_SysModuleEditManager_bdf3143752da45e4ab98f2ef7767d57b *			Data	SysModuleEdit	4/19/2021, 4:16:34 PM	sdkBookExamplePackage
SysModuleEdit_SysModuleEditManager_c3f22fb9ee54c148c352bf835ed4e2c *			Data	SysModuleEdit	4/19/2021, 4:12:55 PM	sdkBookExamplePackage
SysModuleEntity_SysModuleEntityManager_62ac...					4/19/2021	sdkBookExamplePackage

Теперь пакет полностью готов для переноса между [рабочими средами](#) с помощью механизма [экспорта и импорта пакетов](#) Creatio IDE. После установки пакета в другую рабочую среду все привязанные записи отобразятся в соответствующих разделах.

## Пакет-проект

Средний

**Пакет-проект** — пакет, который позволяет разрабатывать функциональность приложения в обычном C#-проекте.

## Особенности пакета-проекта

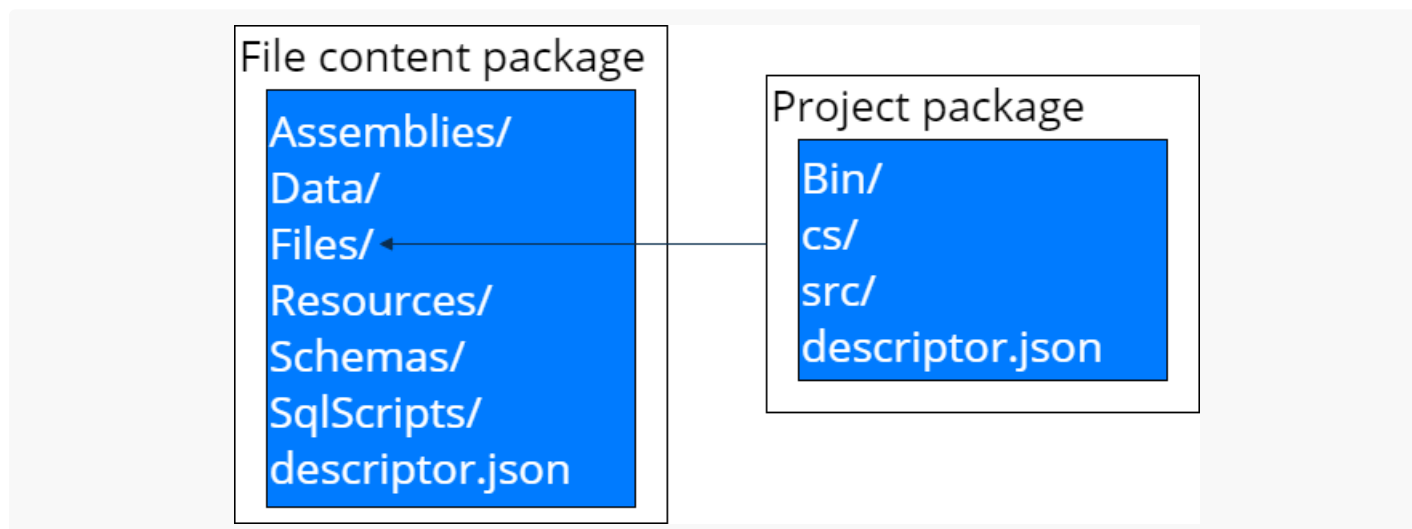
- При наличии в простых пакетах большого количества схем типа [ *Исходный код* ] ([ *Source code* ]) компиляция занимает длительное время. Использование пакетов-проектов позволяет уменьшить скорость компиляции с 30-120 сек до 1-2 сек.
- Пакеты-проекты предоставляют возможность выполнять поставку функциональности на [промышленную среду](#) без прямой поставки.
- Упрощается разработка C#-кода в cloud-приложениях.
- Использование пакетов-проектов предоставляет возможность отслеживать зависимости реализации. Это позволяет составить перечень классов, которые необходимо тестировать при изменениях функциональности.
- Упрощается автоматическое тестирование функциональности.



## Структура пакета-проекта

Структура пакета-проекта в файловой системе не отличается от структуры простого пакета. Основное **отличие** пакета-проекта от простого пакета — наличие файлов `Package.sln` и `Package.csproj`. Структура простого пакета описана в статье [Общие принципы работы с пакетами](#).

Структура папок пакета-проекта представлена на рисунке ниже. Функциональность, разработанная в пакете-проекте, включается в [файловый контент пакета](#) (папка `Files`) в виде скомпилированной библиотеки и \*.cs-файлов.



## Инструменты для разработки пакета-проекта

1. [Creatio command-line interface utility \(clio\)](#) — утилита с открытым исходным кодом для интеграции, разработки и CI/CD.

**Утилита позволяет:**

- Создать пакет-проект.
- Импортировать пакет в on-site или cloud приложение.
- Экспортировать пакет из on-site или cloud приложения.
- Перезапустить приложение.
- Конвертировать существующие пакеты.

2. [CreatioSDK](#) — NuGet-пакет, который предоставляет набор средств разработки. NuGet-пакет позволяет создать приложение на платформе Creatio.

## Импортировать пакет-проект

1. Скомпилируйте пакет-проект.

Пакет-проект, как отдельный C#-проект, компилируется в библиотеку. Имя библиотеки совпадает с именем пакета. Скомпилированные файлы помещаются в папку `../Files/Bin/[PackageName].dll`.

2. Передайте библиотеку.

3. Скопируйте библиотеку в папку.

4. Запустите приложение.

В результате при старте или перезапуске приложения будет выполнен анализ наличия в пакетах подготовленных библиотек. Если такие библиотеки есть, то приложение сразу же подключит их. Для поставки функциональности не требуется компиляция конфигурации.

Импорт пакета-проекта представлен на рисунке ниже.



## Пакет-сборка



Возможность использования пакета-сборки доступна для приложений Creatio версии 7.18.3 и выше.

**Пакет-сборка** — пакет, исходный и автогенерируемый код которого компилируется в отдельную сборку.

**Назначение** пакета-сборки:

- Увеличение скорости разработки за счет снижения времени компиляции, поскольку компилируется только пакет-сборка.
- Увеличение скорости поставки функциональности за счет отсутствия необходимости компиляции, поскольку пакет-сборка содержит в себе предварительно скомпилированную часть.

**Особенности** пакета-сборки:

- Не поддерживается доступ к объектам по протоколу OData версии 3. Для доступа к объектам в пакете-сборке используется протокол OData версии 4. Протокол OData описан в статье [OData](#).
- Не допускается реализация бизнес-логики объектов через событийные подпроцессы дизайнера объекта. Для работы с событийной моделью необходимо использовать `EntityEventListener`. Набор событий `EntityEventListener` не полностью соответствует событийной модели, доступной через Creatio IDE. Событийный слой `Entity` описан в статье [Бизнес-логика объектов](#).
- В C#-коде пакетов не допускается использование типов `Entity` (например, приведение к этим типам), создание экземпляров этих типов через оператор `new` или при помощи рефлексии. Не зависимо от того, где создана эта `Entity`.

- Новые колонки и методы объектов `Entity`, которые размещены в пакете-сборке, не доступны через базовый класс `Entity` (например, `ColumnValues.GetByName(nameValues)`).
- Типы текущего пакета-сборки не доступны в конфигурации и доступны для других пакетов-сборок, которые зависят от текущего.

## Структура и зависимости пакета-сборки

Структура пакета-сборки в файловой системе не отличается от структуры простого пакета. Основное **отличие** пакета-сборки от простого пакета — значение свойств `Type` (тип пакета) и `ProjectPath` (относительный путь от корневого каталога пакета к файлу проекта пакета-сборки) файла `descriptor.json`. Структура простого пакета описана в статье [Общие принципы работы с пакетами](#).

Вспомогательные файлы пакета-сборки создаются или актуализируются:

- При выгрузке пакета-сборки в файловую систему, если режим разработки в файловой системе включен.
- При компиляции пакета-сборки, если режим разработки в файловой системе выключен.
- При фиксации пакета-сборки в системе контроля версий, независимо от статуса режима разработки в файловой системе.

**Вспомогательные файлы** пакета-сборки:

- `[Имя пакета].csproj` — файл проекта, который содержит компилируемое содержимое пакета-сборки.
- `Directory.Build.targets` — файл, который используется для исключения содержимого пакета-сборки из компиляции основной конфигурации.

Начиная с версии 7.18.5, пакет-сборка позволяет ссылаться на внешние сборки.

Чтобы для пакета-сборки **настроить ссылку на внешнюю сборку**:

1. Откройте файл `[Имя пакета].csproj` пакета-сборки.
2. Добавьте необходимую внешнюю сборку в любую секцию файла. Исключением является секция `Label="Package References"`, содержимое которой генерируется автоматически.

Пример настройки ссылок на Quartz и внешнюю библиотеку, которая содержится в каталоге `...\Terrasoft.Configuration\Lib` приведен ниже.

### Пример настройки ссылок на внешние сборки

```
<ItemGroup Label="3rd Party References">
  <Reference Include="Quartz">
    <HintPath>$(CoreLibPath)\Quartz.dll</HintPath>
    <Private>false</Private>
  </Reference>
  <Reference Include="ClassLibrary2">
    <HintPath>$(ConfLibPath)/ClassLibrary2.PackageName.dll</HintPath>
    <SpecificVersion>False</SpecificVersion>
    <Private>False</Private>
  </Reference>
```

```
</ItemGroup>
```

Содержимое файла `[Имя пакета].csproj` пакета-сборки допускается менять вручную. Исключением является секция `Label="Package References"`, содержимое которой генерируется автоматически.

При компиляции через Creatio IDE `TargetFramework` проекта соответствует `TargetFramework` ядра. При компиляции через внешнюю IDE используется значение по умолчанию "net472", которое при необходимости можно изменить.

В отличие от основного проекта конфигурации, в проект пакета-сборки не включены сторонние библиотеки. Для работы со сторонними библиотеками в файл проекта пакета-сборки подключите абстракции ядра или внешние сборки пакета.



**Секции со ссылками**, которые содержатся в пакете-сборке:

- Системные библиотеки.
- Внешние сборки пакета ( `<ItemGroup Label="Package Assembly References"></ItemGroup>` ).
- Библиотеки ядра ( `<ItemGroup Label="Core References"></ItemGroup>` ).
- Библиотеки зависимых пакетов ( `<ItemGroup Label="Package References"></ItemGroup>` ).
- Внешние сборки ( `<ItemGroup Label="3rd Party References"></ItemGroup>` ).

## Операции с пакетом-сборкой

Creatio позволяет создать, компилировать и удалить пакет-сборку.

### Создать пакет-сборку

1. В приложении на платформе .NET Framework перейдите в дизайнер системы по кнопке .
2. В блоке [ *Конфигурирование разработчиком* ] ([ *Admin area* ]) перейдите по ссылке [ *Управление конфигурацией* ] ([ *Advanced settings* ]).
3. В области работы с пакетами нажмите кнопку .
4. Установите признак [ *Компилировать в отдельную сборку* ] ([ *Compile into a separate assembly* ]).

5. Создайте пакет-сборку и установите его зависимости.
6. Разработайте пользовательскую функциональность.
7. В приложении на платформе .NET Framework выполните компиляцию пакета-сборки. Компиляция пакета-сборки описана в пункте [Компилировать пакет-сборку](#).
8. Перенесите пакет-сборку на другую [рабочую среду](#) на платформе .NET Core.
9. Выполните компиляцию перенесенного пакета-сборки. Компиляция пакета-сборки описана в пункте [Компилировать пакет-сборку](#).

Очередность использования платформ (.NET Framework и .NET Core) для разработки пакета-сборки не имеет значения. Важно выполнить компиляцию пакета-сборки в приложениях на платформах .NET Framework и .NET Core. Это позволит пользователям использовать пакет-сборку в приложениях на платформах .NET Framework и .NET Core.

В результате пакет-сборка будет доступен для поставки конечным пользователям.

## Компилировать пакет-сборку

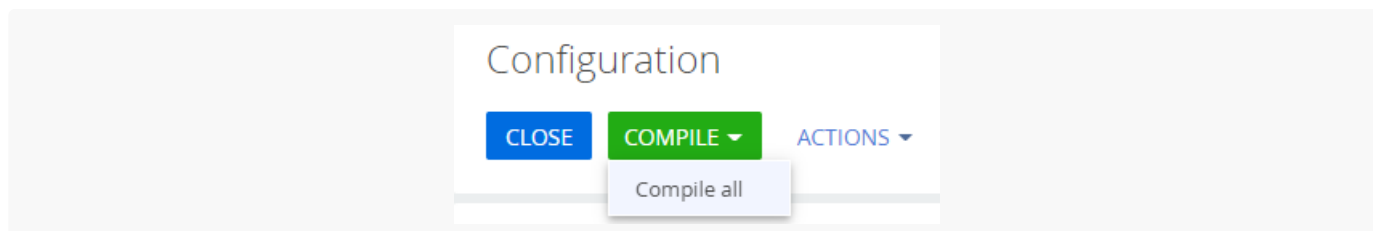
**Назначение** компиляции пакета-сборки — ускорение разработки, поскольку отсутствует необходимость перекомпиляции всей конфигурации.

Начиная с версии 7.18.5, реализована компиляция пакета-сборки, который содержит схемы [клиентских модулей](#) (т. е. схемы с JavaScript-кодом).

**Виды компиляции** пакета-сборки:

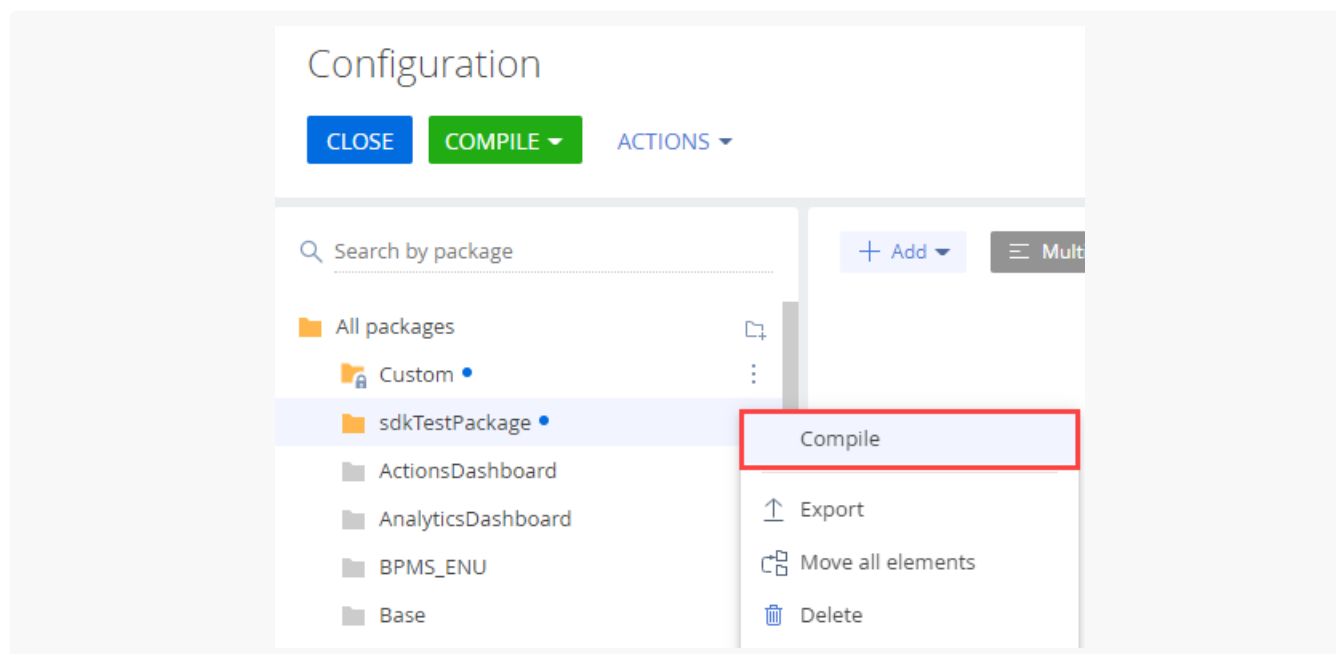
- **Компиляция конфигурации и всех пакетов-сборок.**

Выполняется нажатием кнопки [ *Перекомпилировать все* ] ([ *Compile all* ]) в выпадающем списке кнопки [ *Компилировать* ] ([ *Compile* ]) панели инструментов Creatio IDE.

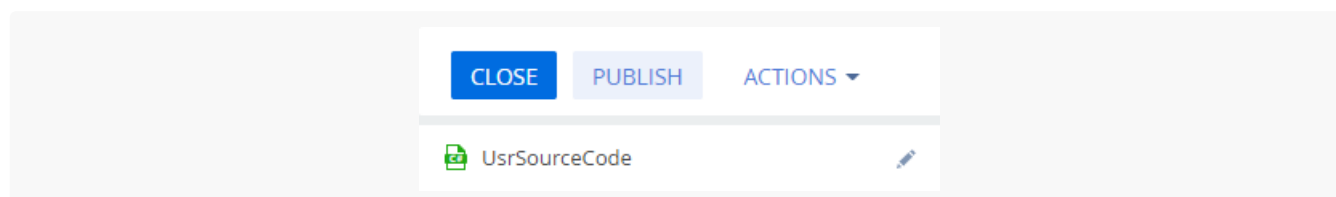


- **Компиляция текущего пакета-сборки.**

- Выполняется выбором пункта [ *Компилировать* ] ([ *Compile* ]) меню пакета Creatio IDE.



- Выполняется нажатием кнопки [ *Опубликовать* ] ([ *Publish* ]) в дизайнера исходного кода. Если редактируемая схема типа [ *Исходный код* ] ([ *Source code* ]) находится в пакете-сборке, то будет выполнена компиляция только пакета-сборки, а не всей конфигурации. В другом случае будет выполнена компиляция всей конфигурации.



В результате пакет-сборка будет скомпилирован. Если при компиляции пакета-сборки будет обнаружено отсутствие пакетов-сборок, от которых зависит текущий пакет-сборка, то приложение также выполнит их компиляцию.

Скомпилированный пакет-сборка будет сохранен в:

- Для приложений на платформе **.NET Framework**

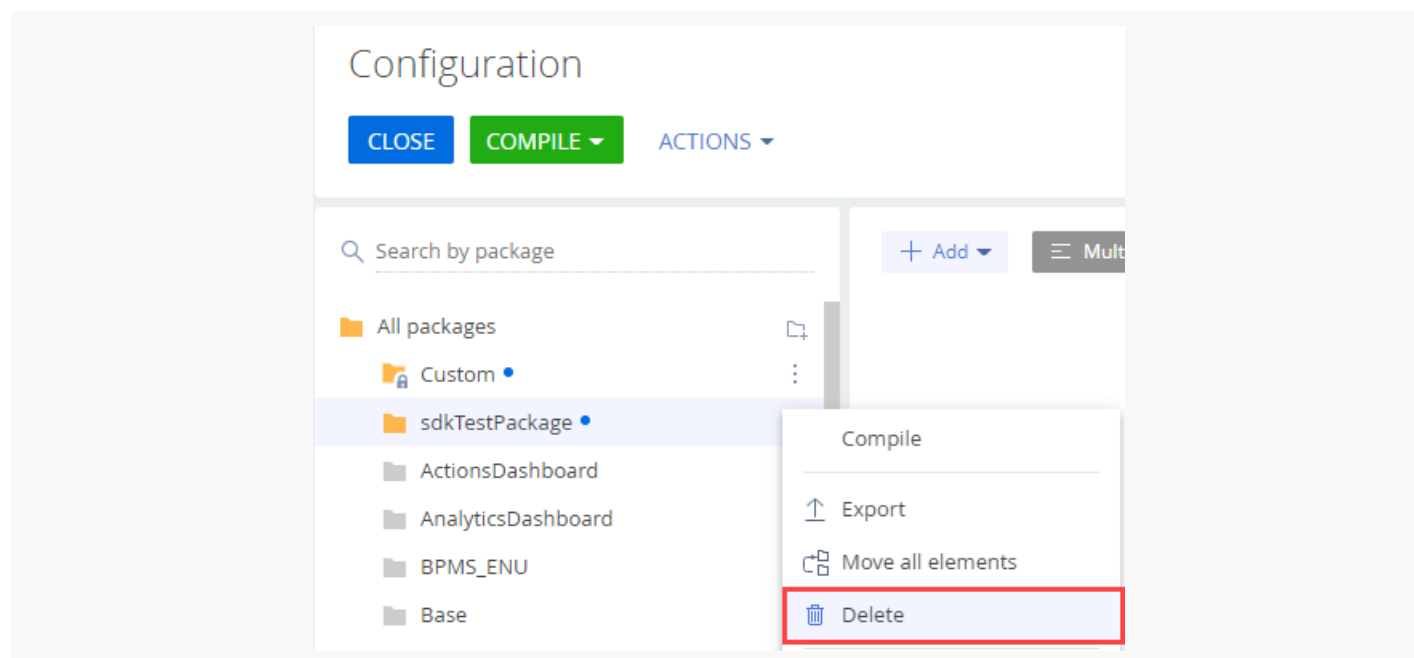
```
...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg\\[Имя пакета]\\Files\\Bin\\[Имя пакета].dll .
```

- Для приложений на платформе **.NET Core**

```
...\\Terrasoft.WebApp\\Terrasoft.Configuration\\Pkg\\[Имя пакета]\\Files\\Bin\\netstandard\\[Имя пакета].dll .
```

## Удалить пакет-сборку

Удаление пакета-сборки не отличается от удаления простого пакета и выполняется в пункте [ *Удалить* ] ([ *Delete* ]) меню пакета Creatio IDE.



Об удалении простого пакета читайте в статье [Принципы разработки в Creatio IDE](#).

В результате пакет-сборка будет удален из конфигурации приложения.

## Конвертация пакетов

**Виды конвертации**, которые позволяет выполнять Creatio:

- Конвертация простого пакета в пакет-сборку.
- Конвертация пакет-сборки в простой пакет.

Конвертация пакетов доступна для **on-site приложений**, поскольку требует доступа к базе данных. Для **приложений cloud** можно выполнить перемещение отдельных конфигурационных элементов в пакет-сборку.

## Конвертировать простой пакет в пакет-сборку

1. Подготовьте конфигурационные элементы простого пакета к конвертации, переписав код в соответствии с особенностями пакета-сборки.
2. С помощью SQL-запроса подготовьте простой пакет к конвертации в пакет-сборку.

MS SQL

```
update [SysPackage]
set [Type] = 1, [ProjectPath] = '', ModifiedOn = GETUTCDATE()
```

```
where [Id] = 'PackageIdGuid'
```

## PostgreSQL

```
update "SysPackage"
set "Type" = 1, "ProjectPath" = '', "ModifiedOn" = current_timestamp
where "Id" = 'PackageIdGuid'
```

Этот запрос выполнит переход в таблицу `[SysPackage]` базы данных. **Действия**, которые выполняются при выполнении запроса:

- Изменение значения колонки `[Type]` с "0" (простой пакет) на "1" (пакет-сборка).
- Заполнение колонки `[ProjectPath]`. Имя проекта должно совпадать с именем пакета.
- Актуализация значения колонки `[ModifiedOn]`.

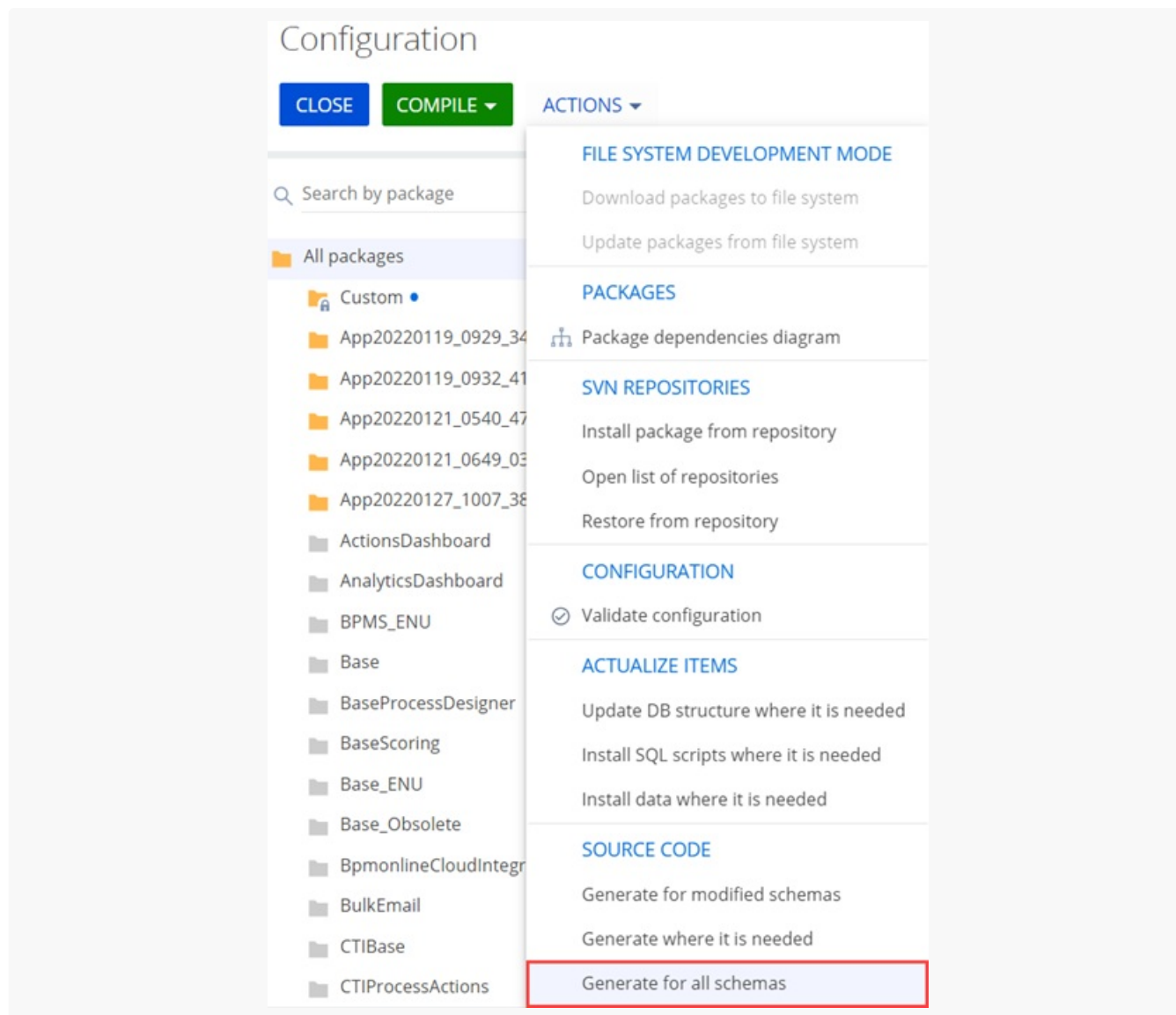
3. При необходимости, [добавьте ссылки на внешние сборки](#).

4. Перезагрузите приложение. Это необходимо, чтобы сбросить кэш.

5. Выполните генерацию всех схем простого пакета и его дочерних схем.

- [Перейдите в раздел \[ Конфигурация \]](#) ([ *Configuration* ]).
- В меню действий в группе [ *Исходный код* ] ([ *Source code* ]) выберите [ *Сгенерировать для всех схем* ] ([ *Generate for all schemas* ]).





6. Выполните компиляцию конфигурации.

В результате простой пакет будет конвертирован в пакет-сборку.

## Конвертировать пакет-сборку в простой пакет

1. Перейдите в базу данных и в таблице [SysPackage] найдите пакет-сборку, который планируется конвертировать в простой пакет.
2. С помощью SQL-запроса актуализируйте значения колонок [Type], [ProjectPath], [ModifiedOn].

MS SQL

```
update [SysPackage]
set [Type] = 0, [ProjectPath] = '', ModifiedOn = GETUTCDATE()
where [Id] = 'PackageIdGuid'
```

## PostgreSQL

```
update "SysPackage"
set "Type" = 0, "ProjectPath" = '', "ModifiedOn" = current_timestamp
where "Id" = 'PackageIdGuid'
```

3. В каталоге `...\Terrasoft.WebApp\Terrasoft.Configuration\Pkg\[Имя пакета]` пакета-сборки удалите каталоги:

- `Autogenerated`
- `Files\Bin`
- `Files\obj`

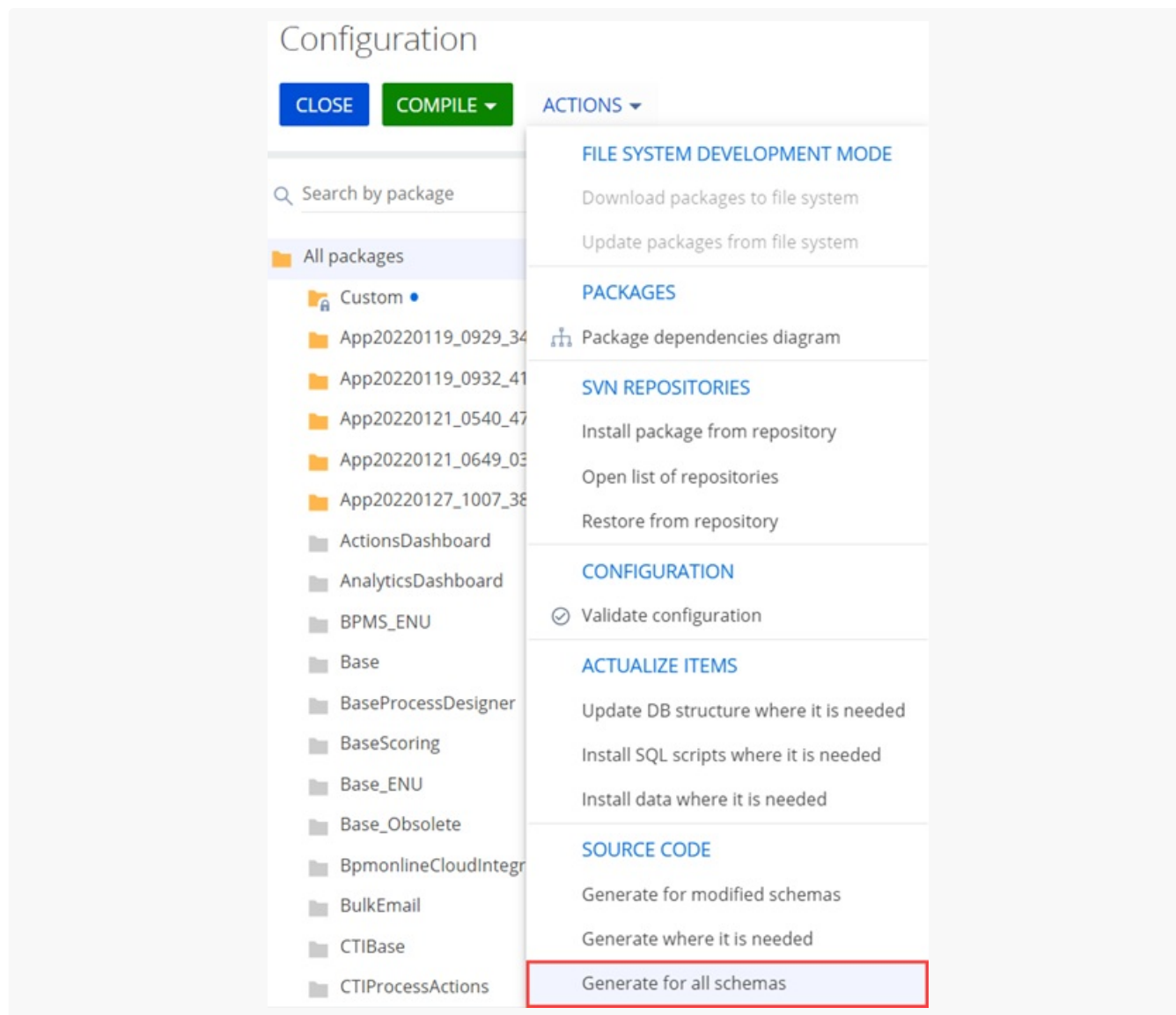
4. В каталоге `...\Terrasoft.WebApp\Terrasoft.Configuration\Pkg\[Имя пакета]` пакета-сборки удалите файлы:

- `Files\[Имя пакета].csproj`
- `Files\Directory.Build.targets`

5. Перезагрузите приложение. Это необходимо, чтобы сбросить кэш.

6. Выполните генерацию всех схем простого пакета и его дочерних схем.

- [Перейдите в раздел \[ Конфигурация \]](#) ([ *Configuration* ]).
- В меню действий в группе [ *Исходный код* ] ([ *Source code* ]) выберите [ *Сгенерировать для всех схем* ] ([ *Generate for all schemas* ]).

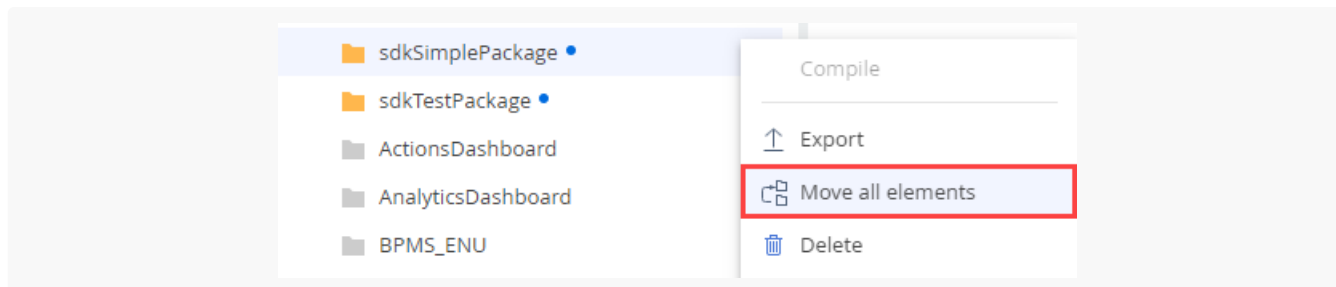


7. Выполните компиляцию конфигурации.

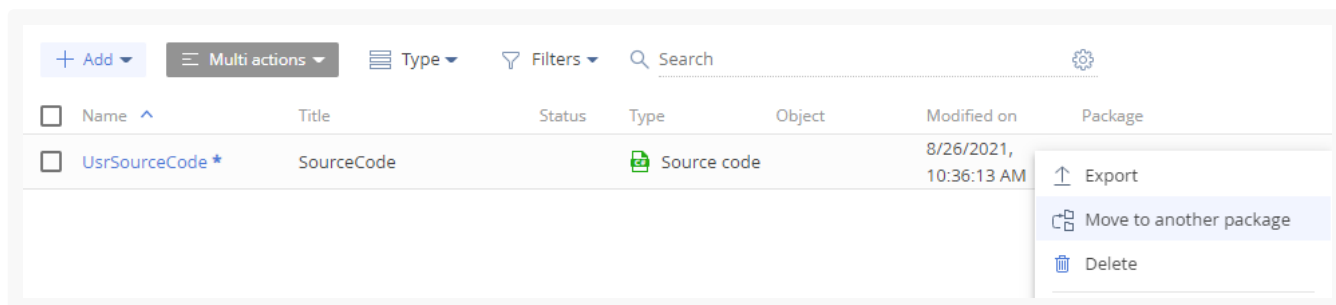
В результате пакет-сборка будет конвертирован в простой пакет.

## Переместить конфигурационные элементы в пакет-сборку

1. Подготовьте конфигурационные элементы к перемещению в пакет-сборку, переписав код в соответствии с особенностями пакета-сборки.
2. Создайте пакет-сборку. Создание пакета-сборки описано в пункте [Создать пакет-сборку](#).
3. Установите [зависимости](#) пакета-сборки. Зависимости пакета-сборки должны совпадать с зависимостями простого пакета, из которого планируется переместить конфигурационные элементы.
4. Выполните перемещение конфигурационных элементов.
  - Чтобы **переместить все конфигурационные элементы**, в меню пакета, из которого планируется переместить конфигурационные элементы, нажмите [ *Переместить все элементы* ] ([ *Move all elements* ]).



- Чтобы **переместить отдельный конфигурационный элемент**, в меню конфигурационного элемента нажмите [ *Переместить в другой пакет* ] ([ *Move to another package* ]).



5. Выполните компиляцию конфигурации.
6. Обновите зависимости дочерних пакетов: замените родительский пакет на пакет-сборку, в который были перенесены конфигурационные элементы.

В результате конфигурационные элементы будут перемещены в пакет-сборку.

## Импорт пакета-сборки

Импорт пакета-сборки не отличается от импорта простого пакета. Об импорте простого пакета читайте в статье [Перенос пакета](#).

**Особенности** импорта пакета-сборки:

- Содержимое пакета-сборки помещается в базу данных.
- Копируется файловый контент пакета-сборки.
- Компиляция конфигурации не запускается, поскольку пакет-сборка был скомпилирован при разработке и содержит в себе runtime-сборку.
- Выполняется перезапуск сайта, инициализация веб-сервисов и фабрики классов, которые содержатся в пакете-сборке.

## Файловый контент пакетов



**Файловый контент пакетов** — файлы (\*.js, \*.css, изображения и др.), добавленные в пользовательские пакеты приложения. Файловый контент является статическим и не обрабатывается веб-сервером, что позволяет повысить скорость работы приложения.

**Виды файлового контента:**

- Клиентский контент, генерируемый в режиме реального времени.
- Предварительно сгенерированный клиентский контент.

**Особенности использования** клиентского контента, генерируемого в режиме реального времени:

- Нет необходимости предварительно генерировать клиентский контент.
- При вычислении иерархии пакетов, схем и формировании контента присутствует нагрузка на процессор (CPU).
- При получении иерархии пакетов, схем и формировании контента присутствует нагрузка на базу данных.
- Потребление памяти для кэширования клиентского контента.

**Особенности использования** предварительно сгенерированного клиентского контента:

- Присутствует минимальная нагрузка на процессор.
- Необходимо предварительно генерировать клиентский контент.
- Отсутствуют запросы в базу данных.
- Клиентский контент кэшируется средствами IIS.

## Структура хранения файлового контента пакета

Файловый контент является частью пакета. Для повышения производительности приложения и снижения нагрузки на базу данных весь файловый контент можно предварительно сгенерировать в специальной папке приложения `...\Terrasoft.WebApp\Terrasoft.Configuration\Pkg\[Имя пакета]\Files`. При запросе сервер IIS ищет запрашиваемый контент в этой папке и сразу же отправляет его приложению. В пакет могут быть добавлены любые файлы, однако использоваться будут только файлы, необходимые для клиентской части Creatio.

Рекомендуется использовать **структуру** папки `Files`, приведенную ниже.

**Рекомендуемая структура папки Files**

```
-PackageName
  ...
  -Files
    -src
      -js
        bootstrap.js
        [другие *.js-файлы]
      -css
        [*.css-файлы]
      -less
        [*.less-файлы]
      -img
        [файлы изображений]
```

```

    -res
      [файлы ресурсов]
    descriptor.json
    ...
  descriptor.json

```

`js` — папка с \*.js-файлами исходных кодов на языке JavaScript.

`css` — папка с \*.css-файлами стилей.

`less` — папка с \*.less-файлами стилей.

`img` — папка с изображениями.

`res` — папка с файлами ресурсов.

`descriptor.json` — дескриптор файлового контента, который хранит информацию о bootstrap-файлах пакета.

Структура файла `descriptor.json` представлена ниже.

Структура файла `descriptor.json`

```

{
  "bootstraps": [
    ... // Массив строк, содержащих относительные пути к bootstrap-файлам.
  ]
}

```

Пример файла `descriptor.json`

```

{
  "bootstraps": [
    "src/js/bootstrap.js",
    "src/js/anotherBootstrap.js"
  ]
}

```

Чтобы добавить файловый контент в пакет, необходимо поместить файл в соответствующую вложенную папку папки `Files` необходимого пакета.

## Bootstrap-файлы пакета

**Bootstrap-файлы** пакета — это \*.js-файлы, которые позволяют управлять загрузкой клиентской конфигурационной логики. Структура файла может варьироваться.

### Структура файла bootstrap.js

```
(function() {
  require.config({
    paths: {
      "Название модуля": "Ссылка на файловый контент",
      ...
    }
  });
})();
```

### Пример файла bootstrap.js

```
(function() {
  require.config({
    paths: {
      "MyPackage1-ContactSectionV2": Terrasoft.getFileContentUrl("MyPackage1", "src/js/Cor
      "MyPackage1-Utilities": Terrasoft.getFileContentUrl("MyPackage1", "src/js/Utilities.
    }
  });
})();
```

Bootstrap-файлы загружаются асинхронно после загрузки ядра, но до загрузки конфигурации. Для корректной загрузки bootstrap-файлов в папке статического контента генерируется вспомогательный файл `_FileContentBootstraps.js`, который содержит информацию о bootstrap-файлах всех пакетов.

### Пример содержимого файла `_FileContentBootstraps.js`

```
var Terrasoft = Terrasoft || {};
Terrasoft.configuration = Terrasoft.configuration || {};
Terrasoft.configuration.FileContentBootstraps = {
  "MyPackage1": [
    "src/js/bootstrap.js"
  ]
};
```

## Версионность файлового контента

Для корректной работы версионности файлового контента в папке статического контента генерируется вспомогательный файл `_FileContentDescriptors.js`. Это файл, в котором в виде коллекции "ключ-значение" содержится информация о файлах в файловом контенте всех пакетов. Каждому ключу (названию файла) соответствует значение — уникальный хэш-код. Таким образом обеспечивается

гарантированная загрузка в браузер актуальной версии файла.

#### Пример содержимого файла `_FileContentDescriptors.js`

```
var Terrasoft = Terrasoft || {};
Terrasoft.configuration = Terrasoft.configuration || {};
Terrasoft.configuration.FileContentDescriptors = {
  "MyPackage1/descriptor.json": {
    "Hash": "5d4e779e7ff24396a132a0e39cca25cc"
  },
  "MyPackage1/Files/src/js/Utilities.js": {
    "Hash": "6d5e776e7ff24596a135a0e39cc525gc"
  }
};
```

## Генерация вспомогательных файлов

Для **генерации вспомогательных файлов** (`_FileContentBootstraps.js` и `FileContentDescriptors.js`) необходимо с помощью [утилиты WorkspaceConsole](#) выполнить операцию `BuildConfiguration`.

Параметры операции `BuildConfiguration`

Параметр	Описание
<code>operation</code>	Название операции. Необходимо установить значение <code>BuildConfiguration</code> — операция компиляции конфигурации.
<code>useStaticFileContent</code>	Признак использования статического контента. Необходимо установить значение <code>false</code> .
<code>usePackageFileContent</code>	Признак использования файлового контента пакетов. Необходимо установить значение <code>true</code> .

#### Генерация вспомогательных файлов

```
Terrasoft.Tools.WorkspaceConsole.exe -operation=BuildConfiguration -workspaceName=Default -desti
```

В результате выполнения операции в папке со статическим контентом `...\Terrasoft.WebApp\conf\content` будут сгенерированы вспомогательные файлы `_FileContentBootstraps.js` и `_FileContentDescriptors.js`.

Описание параметров утилиты `WorkspaceConsole` содержится в статье [Параметры утилиты WorkspaceConsole](#).

## Предварительная генерация статического файлового



## КОНТЕНТА

Файловый контент генерируется в специальную папку `.\Terrasoft.WebApp\conf`, которая содержит \*.js-файлы с исходным кодом схем, \*.css-файлы стилей и \*.js-файлы ресурсов для всех культур приложения, а также изображения.

**Важно.** Для генерации статического контента папки `.\Terrasoft.WebApp\conf` пользователю пула IIS, в котором запущено приложение, необходимо иметь права на модификацию. Права настраиваются на уровне сервера в секции Handler Mappings. Подробнее описано в статье [Настроить сервер приложения на IIS](#).

Имя пользователя пула IIS устанавливается в свойстве `[ Identity ]`. Доступ к этому свойству можно получить в менеджере IIS на вкладке `[ Application Pools ]` через команду `[ Advanced Settings ]`.

**Условия** для выполнения первичной или повторной генерации статического файлового контента:

- Сохранение схемы через дизайнеры клиентских схем и объектов.
- Сохранение через мастера разделов и деталей.
- Установка и удаление приложений из Marketplace и \*.zip-архива.
- Применение переводов.
- Действия `[ Компилировать ]` (`[ Compile ]`) и `[ Перекомпилировать все ]` (`[ Compile all ]`) раздела `[ Конфигурация ]` (`[ Configuration ]`).  
Эти действия необходимо выполнять при **удалении схем или пакетов** из раздела `[ Конфигурация ]` (`[ Configuration ]`).  
Действие `[ Перекомпилировать все ]` (`[ Compile all ]`) необходимо выполнять при **установке или обновлении пакета** из системы контроля версий [SVN](#).

**На заметку.** Действие `[ Перекомпилировать все ]` (`[ Compile all ]`) выполняет полную регенерацию файлового статического контента. Остальные действия в системе выполняют регенерацию только измененных схем.

## Генерация файлового контента

Для **генерации файлового контента** необходимо с помощью [утилиты WorkspaceConsole](#) выполнить операцию `BuildConfiguration`.

Параметры операции `BuildConfiguration`

Параметр	Описание
<code>workspaceName</code>	Название рабочего пространства. По умолчанию — <code>Default</code> .
<code>destinationPath</code>	Папка, в которую будет сгенерирован статический контент.
<code>webApplicationPath</code>	<p>Путь к веб-приложению, из которого будет вычитана информация по соединению с базой данных.</p> <p>Необязательный параметр. Если значение не указано, то соединение будет установлено с базой данных, указанной в строке соединения в файле <code>Terrasoft.Tools.WorkspaceConsole.config</code>. Если значение указано, то соединение будет установлено с базой данных из файла <code>ConnectionStrings.config</code> веб-приложения.</p>
<code>force</code>	<p>Необязательный параметр. По умолчанию — <code>false</code> (выполняется генерация файлового контента для измененных схем).</p> <p>Если установлено значение <code>true</code>, то выполняется генерация файлового контента по всем схемам.</p>

Генерация файлового контента (вариант 1)

```
Terrasoft.Tools.WorkspaceConsole.exe -operation=BuildConfiguration -workspaceName=Default -desti
```

Генерация файлового контента (вариант 2)

```
Terrasoft.Tools.WorkspaceConsole.exe -operation=BuildConfiguration -workspaceName=Default -webAp
```

## Генерация клиентского контента при добавлении новой культуры

После **добавления новых культур** из интерфейса приложения необходимо в разделе [ *Конфигурация* ] ([ *Configuration* ]) выполнить действие [ *Перекомпилировать все* ] ([ *Compile all* ]).

**Важно.** Если пользователь не может войти в систему после добавления новой культуры, то необходимо перейти в раздел [ *Конфигурация* ] ([ *Configuration* ]) по ссылке `http://[Путь к приложению]/0/dev`.

## Получение URL изображения

Изображения в клиентской части Creatio запрашиваются браузером по URL, который устанавливается в атрибуте `src` html-элемента `img`. Для формирования этого URL в Creatio используется модуль `Terrasoft.ImageUrlBuilder (imageurlbuilder.js)`, в котором реализован `getUrl(config)` — публичный метод для получения URL изображения. Этот метод принимает конфигурационный JavaScript-объект `config`, в свойстве `params` которого содержится объект параметров. На основе этого свойства формируется URL изображения для вставки на страницу.

#### Структура объекта `params`

```
config: {
  params: {
    schemaName: "",
    resourceItemName: "",
    hash: "",
    resourceItemExtension: ""
  }
}
```

`schemaName` — название схемы (строка).

`resourceItemName` — название изображения в Creatio (строка).

`hash` — хэш изображения (строка).

`resourceItemExtension` — расширение файла изображения (например, ".png").

Пример формирования конфигурационного объекта параметров для получения URL статического изображения представлен ниже.

#### Пример формирования конфигурационного объекта параметров

```
var localizableImages = {
  AddButtonImage: {
    source: 3,
    params: {
      schemaName: "ActivityMiniPage",
      resourceItemName: "AddButtonImage",
      hash: "c15d635407f524f3bbe4f1810b82d315",
      resourceItemExtension: ".png"
    }
  }
}
```

## Совместимость с режимом разработки в файловой системе

Режим разработки в файловой системе несовместим с получением клиентского контента из

предварительно сгенерированных файлов. Для корректной работы с режимом разработки в файловой системе необходимо **отключить получение статического клиентского контента** из файловой системы. Для отключения данной функциональности необходимо в файле `web.config` для флага `UseStaticFileContent` установить значение `false`.

#### Отключить получение статического клиентского контента из файловой системы

```
<fileDesignMode enabled="true" />
...
<add key="UseStaticFileContent" value="false" />
```

## Перенос изменений между рабочими средами

Файловый контент является неотъемлемой частью пакета. Он фиксируется в хранилище системы контроля версий наравне с остальным содержимым пакета. В дальнейшем файловый контент может быть **перенесен на другую рабочую среду**:

- Для переноса изменений на [среду разработки](#) рекомендуется использовать систему контроля версий SVN.
- Для переноса изменений на [предпромышленную](#) и [промышленную](#) среды рекомендуется использовать механизм [экспорта и импорта](#) Creatio IDE.

**Важно.** При установке пакетов папка `Files` будет создана только в том случае, если она не пустая. Если эта папка не была создана, то для начала разработки ее необходимо создать вручную.

# Локализовать файловый контент с помощью конфигурационных ресурсов

 Сложный

## 1. Создать модуль с локализуемыми ресурсами

Для **перевода ресурсов** на разные языки рекомендуется использовать отдельный модуль с локализуемыми ресурсами, созданный встроенными инструментами Creatio в разделе [ *Конфигурация* ] ([ *Configuration* ]).

#### Пример модуля с локализуемыми ресурсами

```
define("Module1", ["Module1Resources"], function(res) {
    return res;
});
```

## 2. Импортировать модуль локализуемых ресурсов

Чтобы из клиентского модуля **получить доступ к модулю локализуемых ресурсов**, необходимо в качестве зависимости импортировать модуль локализуемых ресурсов в клиентский модуль.

### Пример подключения локализуемых ресурсов в модуль

```
define("MyPackage-MyModule", ["Module1"], function(module1) {
  console.log(module1.localizableStrings.MyString);
});
```

# Локализовать файловый контент с помощью плагина i18n



**i18n** — это плагин для AMD-загрузчика (например, RequireJS), предназначенный для загрузки локализуемых строковых ресурсов. Исходный код плагина можно найти в [GitHub-репозитории](#).

## 1. Добавить плагин

Добавьте плагин в папку с \*.js-файлами исходных кодов

```
..\Terrasoft.WebApp\Terrasoft.Configuration\Pkg\MyPackage1\content\js\i18n.js .
```

Здесь `MyPackage1` — рабочая папка пакета `MyPackage1`.

## 2. Создать папку с локализуемыми ресурсами

Создайте папку `..\MyPackage1\content\nls` и добавьте в нее \*.js-файл с локализуемыми ресурсами.

Можно добавлять несколько \*.js-файлов с локализуемыми ресурсами. Имена файлов могут быть произвольными. Содержимое файлов — AMD модули, которые содержат объекты.

**Структура объектов** AMD модулей:

- **Поле** `root`.

Поле содержит коллекцию "ключ-значение", где ключ — это название локализуемой строки, а значение — локализуемая строка на языке по умолчанию. Значение будет использоваться, если запрашиваемый язык не поддерживается.

- **Поля культур.**

В качестве имен полей установите стандартные коды поддерживаемых культур (например, `en-US`, `ru-RU`), а значение имеет логический тип (`true` — поддерживаемая культура включена, `false` — поддерживаемая культура отключена).

Ниже представлен пример файла `..\MyPackage1\content\js\nls\ContactSectionV2Resources.js`.

**Пример файла** `ContactSectionV2Resources.js`

```
define({
  "root": {
    "FileContentActionDescr": "File content first action (Default)",
    "FileContentActionDescr2": "File content second action (Default)"
  },
  "en-US": true,
  "ru-RU": true
});;
```

### 3. Создать папки культур

В папке `..\MyPackage1\content\nls` создайте папки культур. В качестве имен папок установите код той культуры, локализация которой будет в них размещена (например, `en-US`, `ru-RU`).

Структура папки `MyPackage1` с русской и английской культурами представлена ниже.

**Структура папки** `MyPackage1`

```
content
  nls
    en-US
    ru-RU
```

### 4. Добавить файлы с локализуемыми ресурсами

В каждую созданную папку локализации поместите такой же набор \*.js-файлов с локализуемыми ресурсами, как и в корневой папке `..\MyPackage1\content\nls`. Содержимое файлов — AMD модули, объекты которых являются коллекциями "ключ-значение", где ключ — это наименование локализуемой строки, а значение — строка на языке, соответствующем названию папки (коду культуры).

Например, если поддерживаются только русская и английская культуры, то создайте два файла

`ContactSectionV2Resources.js`.

Файл `ContactSectionV2Resources.js`, соответствующий английской культуре

```
define({
  "FileContentActionDescr": "File content first action",
  "FileContentActionDescr2": "File content second action"
});;
```

Файл ContactSectionV2Resources.js, соответствующий русской культуре

```
define({
  "FileContentActionDescr": "Первое действие файлового контента"
});
```

Поскольку для русской культуры перевод строки "FileContentActionDescr2" не указан, то будет использовано значение по умолчанию — "File content second action (Default)".

## 5. Отредактировать файл bootstrap.js

Чтобы отредактировать файл `bootstrap.js`:

1. Подключите плагин `i18n`, указав его название в виде псевдонима `i18n` в конфигурации путей `RequireJS` и прописав соответствующий путь к нему в свойстве `paths`.
2. Укажите плагину культуру, которая является текущей для пользователя. Для этого свойству `config` объекта конфигурации библиотеки `RequireJS` присвойте объект со свойством `i18n`, которому, в свою очередь, присвойте объект со свойством `locale` и значением, полученным из глобальной переменной `Terrasoft.currentUserCultureName` (код текущей культуры).
3. Для каждого файла с ресурсами локализации укажите соответствующие псевдонимы и пути в конфигурации путей `RequireJS`. При этом псевдоним должен являться URL-путем относительно директории `nls`.

**Пример файла** `..\MyPackage1\content\js\bootstrap.js`

```
(function() {
  require.config({
    paths: {
      "MyPackage1-Utilities": Terrasoft.getFileContentUrl("MyPackage1", "content/js/Utilit
      "MyPackage1-ContactSectionV2": Terrasoft.getFileContentUrl("MyPackage1", "content/js
      "MyPackage1-CSS": Terrasoft.getFileContentUrl("MyPackage1", "content/css/MyPackage.c
      "MyPackage1-LESS": Terrasoft.getFileContentUrl("MyPackage1", "content/less/MyPackage
      "i18n": Terrasoft.getFileContentUrl("MyPackage1", "content/js/i18n.js"),
      "nls/ContactSectionV2Resources": Terrasoft.getFileContentUrl("MyPackage1", "content/
      "nls/ru-RU/ContactSectionV2Resources": Terrasoft.getFileContentUrl("MyPackage1", "cc
      "nls/en-US/ContactSectionV2Resources": Terrasoft.getFileContentUrl("MyPackage1", "c
    },
    config: {
      i18n: {
        locale: Terrasoft.currentUserCultureName
      }
    }
  });
})();
```

## 6. Использовать ресурсы в клиентском модуле

Чтобы использовать ресурсы в клиентском модуле, укажите в массиве зависимостей модуль с ресурсами с префиксом "i18n!".

Ниже представлен пример использования локализуемой строки `FileContentActionDescr` в качестве заголовка для нового действия раздела [ *Контакты* ] ([ *Contacts* ]).

### Пример файла `..\MyPackage1\content\js\ContactSectionV2.js`

```
define("MyPackage1-ContactSectionV2", ["i18n!nls/ContactSectionV2Resources",
  "css!MyPackage1-CSS", "less!MyPackage1-LESS"], function(resources) {
  return {
    methods: {
      getSectionActions: function() {
        var actionMenuItems = this.callParent(arguments);
        actionMenuItems.addItem(this.getButtonMenuItem({"Type": "Terrasoft.MenuSeparator"}));
        actionMenuItems.addItem(this.getButtonMenuItem({
          "Click": {"bindTo": "onFileContentActionClick"},
          "Caption": resources.FileContentActionDescr
        })));
        return actionMenuItems;
      },
      onFileContentActionClick: function() {
        console.log("File content clicked!")
      }
    },
    diff: /**SCHEMA_DIFF*/[/**SCHEMA_DIFF*/
  ]
});
```

# Использовать TypeScript при разработке клиентской функциональности



При разработке клиентской функциональности файловый контент позволяет использовать компилируемые в JavaScript языки, например, **TypeScript**. Подробнее о TypeScript можно узнать на официальном [сайте TypeScript](#).

**Пример.** При сохранении записи контрагента выводить для пользователя сообщение о правильности заполнения поля [ *Альтернативные названия* ] ([ *Also known as* ]). Поле должно содержать только буквенные символы. Логику валидации поля реализовать на языке TypeScript.



# 1. Установить TypeScript

Одним из способов установки TypeScript является использование **менеджера пакетов NPM** для `Node.js`.

Чтобы **установить TypeScript**:

1. Проверьте наличие среды выполнения `Node.js` в вашей операционной системе.  
Скачать инсталлятор можно на сайте <https://nodejs.org>.
2. В консоли Windows выполните команду:

## Команда для установки TypeScript

```
npm install -g typescript
```

# 2. Перейти в режим разработки в файловой системе

Чтобы **настроить Creatio для работы в файловой системе**:

1. **Включите режим разработки в файловой системе.**

В файле `web.config`, который находится в корневом каталоге приложения, установите значение `true` для атрибута `enabled` элемента `fileDesignMode`.

2. **Отключите получение статического клиентского контента из файловой системы.**

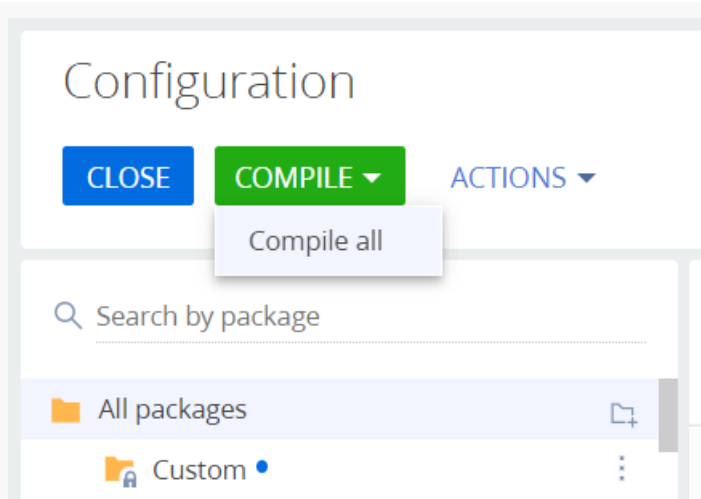
В файле `web.config`, который находится в корневом каталоге приложения, установите значение `false` для флага `UseStaticFileContent`.

## Web.config

```
<filedesignmode enabled="true"/>
...
<add key="UseStaticFileContent" value="false"/>
```

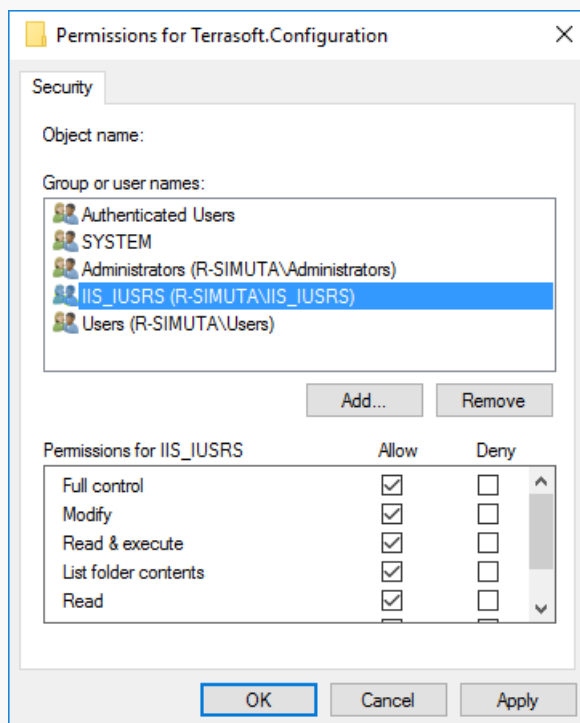
3. **Скомпилируйте приложение.**

В разделе [ *Конфигурация* ] ([ *Configuration* ]) выполните действие [ *Компилировать все* ] ([ *Compile all items* ]).



#### 4. Предоставьте доступ IIS к каталогу конфигурации.

Чтобы приложение могло корректно работать с конфигурационным проектом, необходимо предоставить полный доступ пользователю операционной системы, от имени которого запущен пул приложений IIS, к каталогу `[Путь к приложению]\Terrasoft.WebApp\Terrasoft.Configuration`. Как правило, это встроенный пользователь `IIS_IUSRS`.



Режим разработки в файловой системе описан в статье [Внешние IDE](#).

## 3. Создать структуру хранения файлового контента

Чтобы **создать структуру хранения файлового контента**:

1. В пользовательском пакете, выгруженном в файловую систему, создайте каталог `Files`.

- В каталоге `Files` создайте вложенный каталог `src`.
- В каталоге `src` создайте вложенный каталог `js`.
- В каталоге `Files` создайте файл `descriptor.json`.

`descriptor.json`

```
{
  "bootstraps": [
    "src/js/bootstrap.js"
  ]
}
```

- В каталоге `Files\src\js` создайте файл `bootstrap.js`.

`bootstrap.js`

```
(function() {
  require.config({
    paths: {
      "LettersOnlyValidator": Terrasoft.getFileContentUrl("sdkTypeScript", "src/js/Lett
    }
  });
})();
```

## 4. Реализовать валидацию на языке TypeScript

Чтобы **реализовать валидацию на языке TypeScript**:

- В каталоге `Files\src\js` создайте файл `Validation.ts`, в котором объявите интерфейс `StringValidator`.

`Validation.ts`

```
interface StringValidator {
  isAcceptable(s: string): boolean;
}
export = StringValidator;
```

- В каталоге `Files\src\js` создайте файл `LettersOnlyValidator.ts`. Объявите в нем класс `LettersOnlyValidator`, реализующий интерфейс `StringValidator`.

`LettersOnlyValidator.ts`

```
// Импорт модуля, в котором реализован интерфейс StringValidator.
import StringValidator = require("Validation");

// Создаваемый класс должен принадлежать пространству имен (модулю) Terrasoft.
module Terrasoft {
    // Объявление класса валидации значений.
    export class LettersOnlyValidator implements StringValidator {
        // Регулярное выражение, допускающее использование только буквенных символов.
        lettersRegexp: any = /^[A-Za-z]+$/;
        // Валидирующий метод.
        isAcceptable(s: string) {
            return !Ext.isEmpty(s) && this.lettersRegexp.test(s);
        }
    }
}

// Создание и экспорт экземпляра класса для require.
export = new Terrasoft.LettersOnlyValidator();
```

## 5. Выполнить компиляцию исходных кодов TypeScript в исходные коды JavaScript

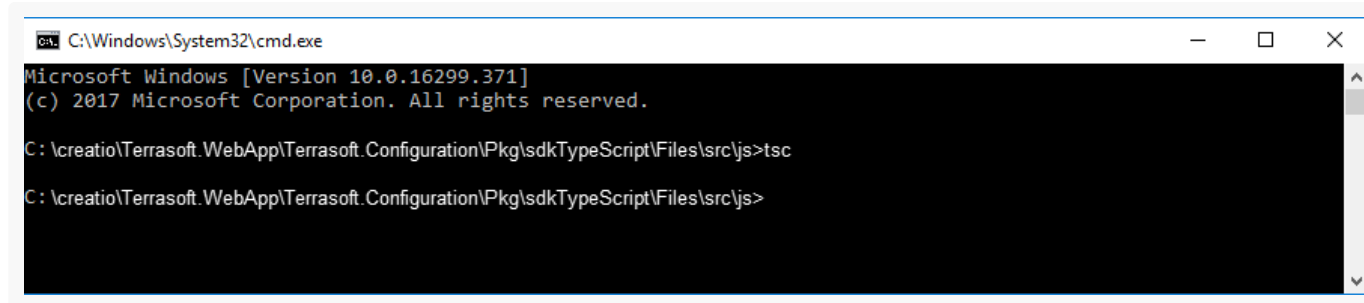
Чтобы **выполнить компиляцию исходных кодов TypeScript в исходные коды JavaScript**:

1. Для настройки компиляции добавьте в каталог `Files\src\js` конфигурационный файл `tsconfig.json`.

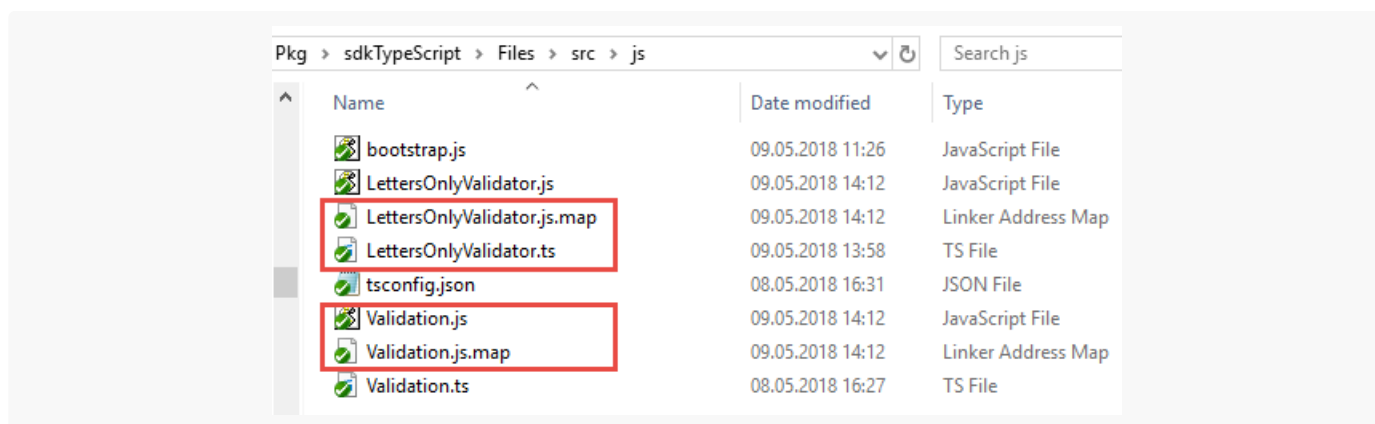
`tsconfig.json`

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "amd",
    "sourceMap": true
  }
}
```

2. В консоли Windows перейдите в каталог `Files\src\js` и выполните команду `tsc`.



В результате выполнения компиляции в каталоге `Files\src\js` будут созданы JavaScript-версии файлов `Validation.ts` и `LettersOnlyValidator.ts`, а также \*.map-файлы, облегчающие отладку в браузере.



Содержимое файла `LettersOnlyValidator.js`, который будет использоваться в Creatio, получено автоматически.

#### LettersOnlyValidator.js

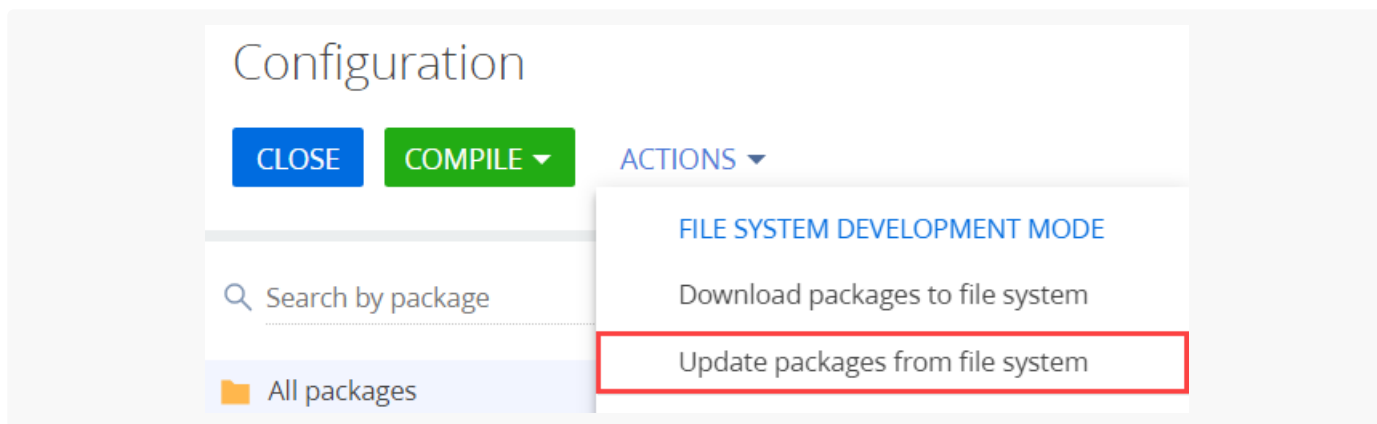
```
define(["require", "exports"], function (require, exports) {
    "use strict";
    var Terrasoft;
    (function (Terrasoft) {
        var LettersOnlyValidator = /** @class */ (function () {
            function LettersOnlyValidator() {
                this.lettersRegexp = /^[A-Za-z]+$/;
            }
            LettersOnlyValidator.prototype.isAcceptable = function (s) {
                return !Ext.isEmpty(s) && this.lettersRegexp.test(s);
            };
            return LettersOnlyValidator;
        })();
        Terrasoft.LettersOnlyValidator = LettersOnlyValidator;
    })(Terrasoft || (Terrasoft = {}));
    return new Terrasoft.LettersOnlyValidator();
});
```

```
//# sourceMappingURL=LettersOnlyValidator.js.map
```

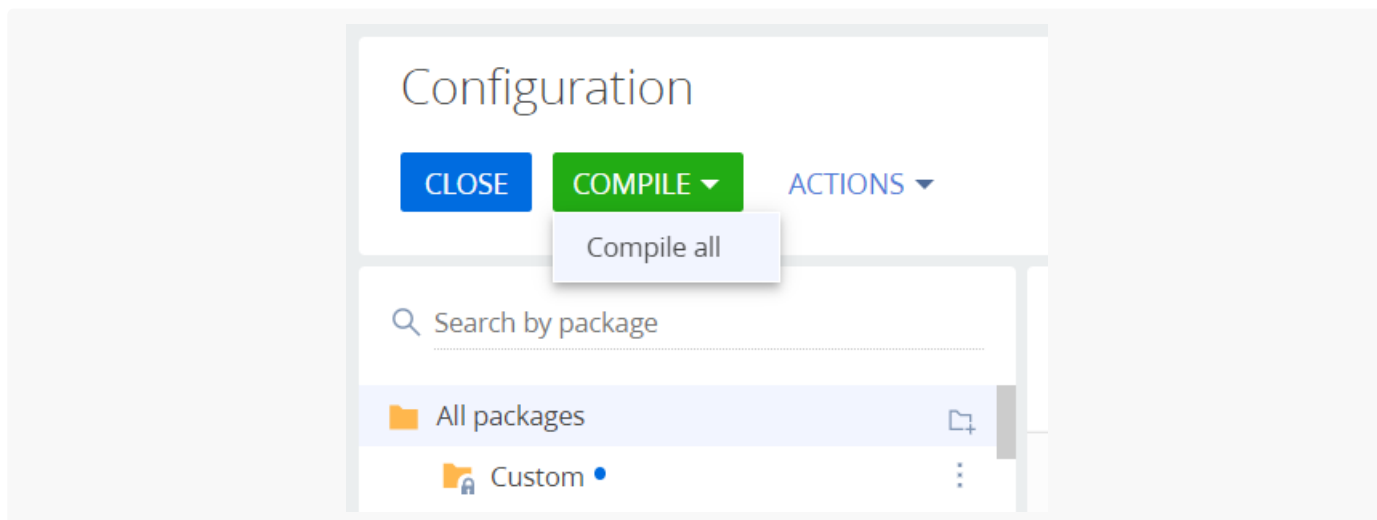
## 6. Выполнить генерацию вспомогательных файлов

Чтобы **выполнить генерацию вспомогательных файлов** `_FileContentBootstraps.js` и `FileContentDescriptors.js`:

1. Перейдите в раздел [ Конфигурация ] ([ *Configuration* ]).
2. Выполните загрузку пакетов из файловой системы (действие [ Обновить пакеты из файловой системы ] ([ *Update packages from file system* ])).



3. Для применения изменений в файле `bootstrap.js` выполните компиляцию приложения (действие [ Компилировать все ] ([ *Compile all items* ])).

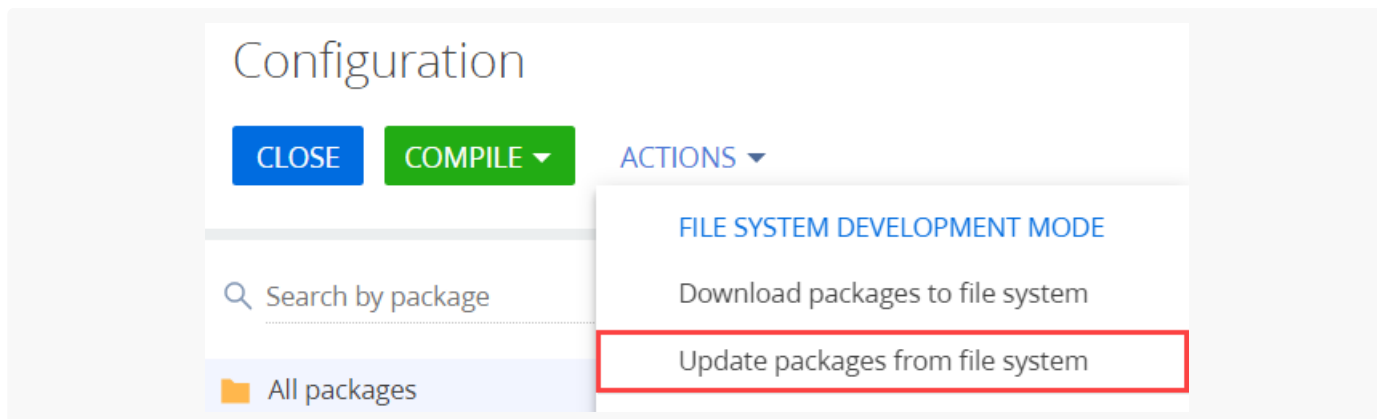


## 7. Проверить результат выполнения примера

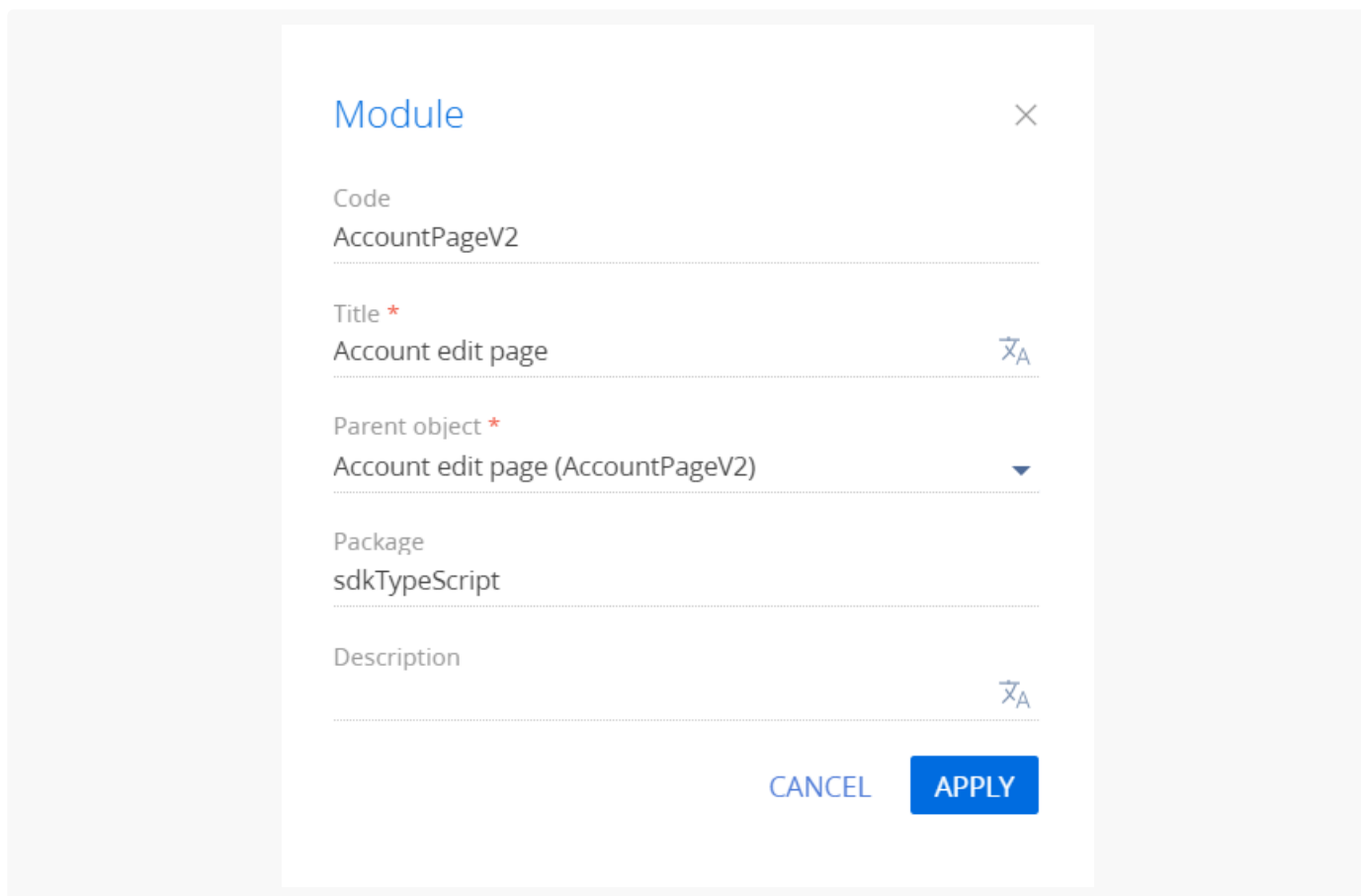
Чтобы **использовать валидацию**:

1. Перейдите в раздел [ Конфигурация ] ([ *Configuration* ]).

2. Выполните загрузку пакетов из файловой системы (действие [ Обновить пакеты из файловой системы ] ([ Update packages from file system ])).



3. Создайте [схему замещающей модели представления](#) страницы контрагента.



4. Выполните выгрузку пакетов в файловую систему (действие [ Выгрузить пакеты в файловую систему ] ([ Download packages to file system ])).
5. В файловой системе измените файл `..\sdkTypeScript\Schemas\AccountPageV2\AccountPageV2.js`.

```
..\sdkTypeScript\Schemas\AccountPageV2\AccountPageV2.js
```

```
// Объявление модуля и его зависимостей.
```

```

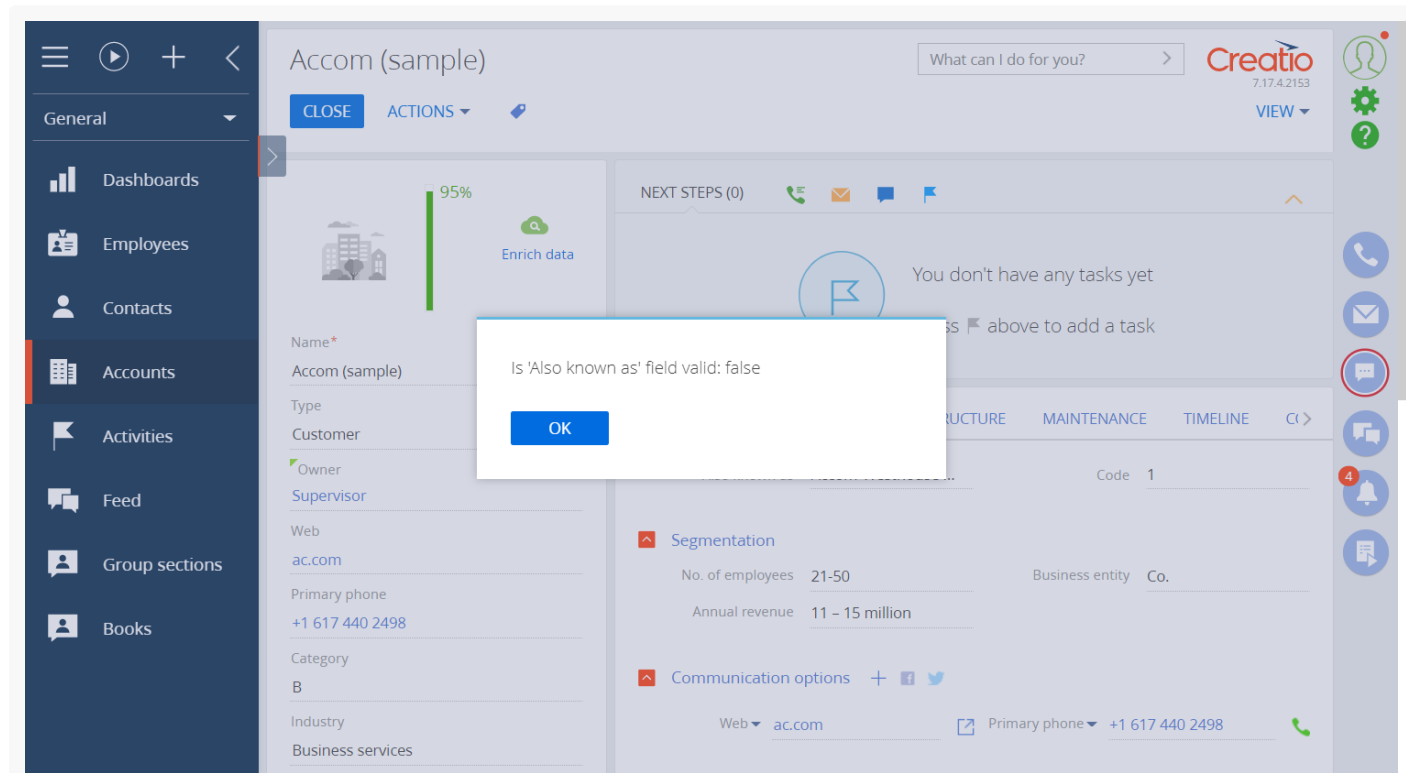
define("AccountPageV2", ["LettersOnlyValidator"], function(LettersOnlyValidator) {
    return {
        entitySchemaName: "Account",
        methods: {
            // Метод валидации.
            validateMethod: function() {
                // Определение правильности заполнения колонки AlternativeName.
                var res = LettersOnlyValidator.isAcceptable(this.get("AlternativeName"));
                // Вывод результата пользователю.
                Terrasoft.showInformation("Is 'Also known as' field valid: " + res);
            },
            // Переопределение метода родительской схемы, вызываемого при сохранении записи.
            save: function() {
                // Вызов метода валидации.
                this.validateMethod();
                // Вызов базовой функциональности.
                this.callParent(arguments);
            }
        },
        diff: /**SCHEMA_DIFF*/ [] /**SCHEMA_DIFF*/
    };
});

```

6. Сохраните файл с исходным кодом схемы и обновите страницу контрагента.

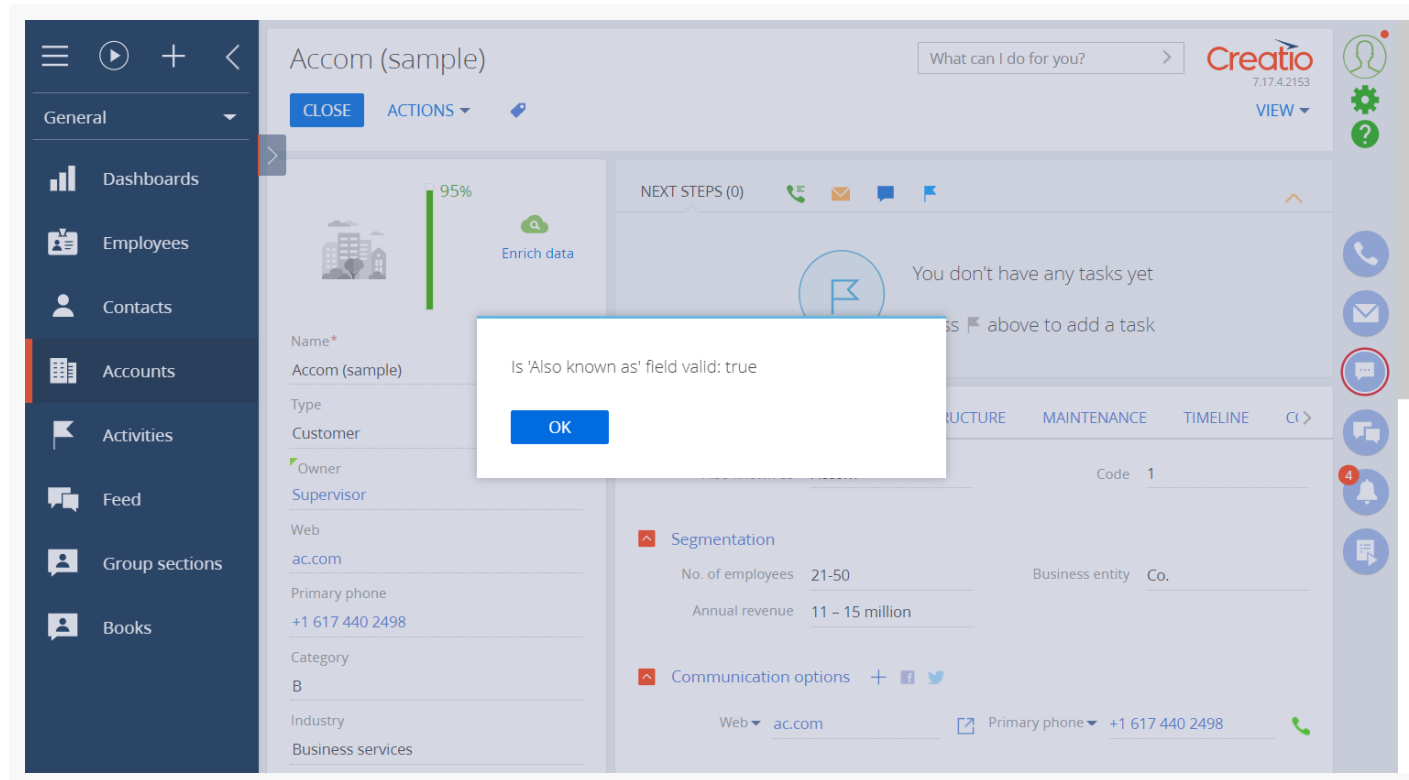
При сохранении записи будет выполняться [валидация](#) и отображаться соответствующее сообщение.

Сообщение о некорректно заполненном поле





Сообщение о корректно заполненном поле



# Создать Angular-компонент для использования в Creatio

 Сложный

Для встраивания Angular-компонентов в приложение Creatio используется функциональность Angular Elements. **Angular Elements** — это `npm`-пакет, который позволяет упаковывать Angular-компоненты в Custom Elements и определять новые HTML-элементы со стандартным поведением (Custom Elements является частью стандарта Web-Components).

## Создание пользовательского Angular-компонента

### 1. Настроить окружение для разработки компонентов средствами Angular CLI

Для этого установите:

1. [Node.js® и npm package manager](#).

2. Angular CLI.

Чтобы установить Angular CLI выполните в системной консоли команду:

#### Установка Angular CLI

```
npm install -g @angular/cli
```

#### Пример установки Angular CLI версии 8

```
npm install -g @angular/cli@8
```

## 2. Создать Angular приложение

Выполните в консоли команду `ng new` и укажите имя приложения, например `angular-element-test`.

#### Создание Angular приложения

```
ng new angular-element-test --style=scss
```

## 3. Установить пакет Angular Elements

Из папки приложения, созданного на предыдущем шаге, выполните в консоли команду.

#### Установка пакета Angular Elements

```
ng add @angular/elements
```

## 4. Создать компонент Angular

Чтобы создать компонент выполните в консоли команду.

#### Создание компонента Angular

```
ng g c angular-element
```

## 5. Зарегистрировать компонент как Custom Element

Чтобы настроить трансформацию компонента в пользовательский HTML-элемент, необходимо внести изменения в файл `app.module.ts`:

1. Добавьте импорт модуля `createCustomElement`.
2. В модуле в секции `entryComponents` укажите имя компонента.

3. В методе `ngDoBootstrap` зарегистрируйте компонент под HTML-тегом.

`app.module.ts`

```
import { BrowserModule } from "@angular/platform-browser";
import { NgModule, DoBootstrap, Injector, ApplicationRef } from "@angular/core";
import { createCustomElement } from "@angular/elements";
import { AppComponent } from "../app.component";
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  entryComponents: [AngularElementComponent]
})
export class AppModule implements DoBootstrap {
  constructor(private injector: Injector) {
  }
  ngDoBootstrap(appRef: ApplicationRef): void {
    const el = createCustomElement(AngularElementComponent, { injector: this._injector });
    customElements.define('angular-element-component', el);
  }
}
```

## 6. Выполнить сборку приложения

1. При сборке проекта сгенерируются несколько \*.js-файлов. Для простоты дальнейшего использования веб-компонента в Creatio, созданные после сборки файлы рекомендуется поставлять в одном файле. Для этого необходимо в корне приложения создать скрипт `build.js`.

**Пример** `build.js`

```
const fs = require('fs-extra');
const concat = require('concat');
const componentPath = './dist/angular-element-test/angular-element-component.js';

(async function build() {
  const files = [
    './dist/angular-element-test/runtime.js',
    './dist/angular-element-test/polyfills.js',
    './dist/angular-element-test/main.js',
    './tools/lodash-fix.js',
  ].filter((x) => fs.pathExistsSync(x));
  await fs.ensureFile(componentPath);
  await concat(files, componentPath);
})();
```

Если в веб-компоненте используется библиотека `lodash`, то для ее работы в Creatio необходимо `main.js` (и при необходимости `styles.js`) объединять со скриптом, устраняющим конфликты по `lodash`. Для этого в корне Angular-проекта создаем папку `tools` и файл `lodash-fix.js`.

```
lodash-fix.js
```

```
window._.noConflict();
```

**Важно.** Если Вы не используете библиотеку `lodash`, то файл `lodash-fix.js` создавать не нужно и строку  `'./tools/lodash-fix.js'`  из массива `files` необходимо убрать.

Дополнительно для выполнения скрипта в `build.js` необходимо установить в проекте пакеты `concat` и `fs-extra` как dev-dependency. Для этого выполните в командной строке команды:

#### Установка дополнительных пакетов

```
npm i concat -D
npm i fs-extra -D
```

По умолчанию для созданного приложения могут быть установлены настройки файла `browserslist`, которые создают сразу несколько сборок для браузеров, которые поддерживают ES2015, и для тех, которым нужен ES5. Для данного примера мы собираем Angular элемент для современных браузеров.

#### Пример `browserslist`

```
# This file is used by the build system to adjust CSS and JS output to support the specified
# For additional information regarding the format and rule options, please see:
# https://github.com/browserslist/browserslist#queries
```

```
# You can see what browsers were selected by your queries by running:
#   npx browserslist
```

```
last 1 Chrome version
last 1 Firefox version
last 2 Edge major versions
last 2 Safari major versions
last 2 iOS major versions
Firefox ESR
not IE 11
```

**Важно.** Если Вам необходимо поставлять веб-компонент в браузеры, которые не поддерживают ES2015, нужно либо править массив файлов в `build.js`, либо изменить `target` в `tsconfig.json` (`target: "es5"`). Внимательно проверяйте названия файлов после сборки в папке `dist`. Если они не совпадают с названиями в массиве `build.js`, их нужно изменить в файле.

- Добавьте в `package.json` команды, которые отвечают за сборку элемента. В результате их выполнения, вся бизнес логика помещается в один файл `angular-element-component.js`, с которым мы будем работать далее.

`package.json`

```
....
"build-ng-element": "ng build --output-hashing none && node build.js",
"build-ng-element:prod": "ng build --prod --output-hashing none && node build.js",
...
```

**Важно.** Рекомендуем при разработке, выполнять сборку приложения без параметра `--prod`.

## Подключение Custom Element в Creatio

Созданный в результате сборки файл `angular-element-component.js` необходимо встроить в пакет Creatio как [файловый контент](#).

### 1. Разместить файл в статическом контенте пакета

Для этого скопируйте файл в папку `Название пользовательского пакета\Files\src\js`, например, `MyPackage\Files\src\js`.

### 2. Встроить билд в Creatio

Для этого необходимо в файле `bootstrap.js` (пакета Creatio, куда Вы хотите загрузить веб-компонент) настроить конфиг с указанием пути к билду.

#### Настройка конфига

```
(function() {
  require.config({
    paths: {
      "angular-element-component": Terrasoft.getFileContentUrl("MyPackageName", "src/js/ar
    }
  });
})();
```

Для загрузки `bootstrap` укажите путь к данному файлу. Для этого создайте `descriptor.json` в `Название пользовательского пакета\Files`.

**descriptor.json**

```
{
  "bootstraps": [
    "src/js/bootstrap.js"
  ]
}
```

Выполните загрузку из файловой системы и компиляцию.

### 3. Выполнить загрузку компонента в необходимой схеме/модуле

Создайте в пакете схему или модуль, в котором должен быть использован созданный пользовательский элемент, и выполните его загрузку в блоке подключения зависимостей модуля.

#### Выполнение загрузки компонента

```
define("MyModuleName", ["angular-element-component"], function() {
```

### 4. Создать HTML-элемент и добавить его в модель DOM

#### Пример добавления пользовательского элемента `angular-element-component` в модель DOM ...

```
/**
 * @inheritDoc Terrasoft.BaseModule#render
 * @override
 */
render: function(renderTo) {
  this.callParent(arguments);
  const component = document.createElement("angular-element-component");
  component.setAttribute("id", this.id);
  renderTo.appendChild(component);
}
```

## Работа с данными

Передача данных в Angular-компонент выполняется через публичные свойства/поля, помеченные декоратором `@Input`.

**Важно.** Описанные в camelCase свойства без указания в декораторе явного имени будут переведены в HTML-атрибуты в kebab-case.

#### Пример создания свойства компонента ( `app.component.ts` )

```
@Input('value')
public set value(value: string) {
    this._value = value;
}
```

#### Пример передачи данных в компонент ( `CustomModule.js` )

```
/**
 * @inheritDoc Terrasoft.BaseModule#render
 * @override
 */
render: function(renderTo) {
    this.callParent(arguments);
    const component = document.createElement("angular-element-component");
    component.setAttribute("value", 'Hello');
    renderTo.appendChild(component);
}
```

Получение данных от компонента реализовано через механизм событий. Для этого необходимо публичное поле (тип `EventEmitter<T>` ) пометить декоратором `@Output` . Для инициализации события необходимо у поля вызвать метод `emit(T)` и передать необходимые данные.

#### Пример реализации события в компоненте ( `app.component.ts` )

```
/**
 * Emits btn click.
 */
@Output() btnClicked = new EventEmitter<any>();

/**
 * Handles btn click.
 * @param eventData - Event data.
 */
public onBtnClick(eventData: any) {
    this.btnClicked.emit(eventData);
}
```

Добавьте кнопку в `angular-element.component.html`.

#### Пример добавления кнопки в `angular-element.component.html`

```
<button (click)="onBtnClick()">Click me</button>
```

#### Пример обработки события в Creatio ( `CustomModule.js` )

```
/**
 * @inheritDoc Terrasoft.Component#initDomEvents
 * @override
 */
initDomEvents: function() {
  this.callParent(arguments);
  const el = this.component;
  if (el) {
    el.on("itemClick", this.onItemClickHandler, this);
  }
}
```

## Использование Shadow DOM

Некоторые компоненты, созданные с помощью Angular и встроенные в Creatio могут быть сконфигурированы так, чтобы реализация компонента была закрыта от внешнего окружения так называемым Shadow DOM.

Shadow DOM — это механизм инкапсуляции компонентов внутри DOM. Благодаря ему, в компоненте есть собственное «теневое» DOM-дерево, к которому нельзя просто так обратиться из главного документа, у него могут быть изолированные CSS-правила и т. д.

Для использования Shadow DOM необходимо в декоратор компонента добавить свойство

```
encapsulation: ViewEncapsulation.ShadowDom.
```

#### `angular-element.component.ts`

```
import { Component, OnInit, ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'angular-element-component',
  templateUrl: './angular-element-component.html',
  styleUrls: [ './angular-element-component.scss' ],
  encapsulation: ViewEncapsulation.ShadowDom,
})
export class AngularElementComponent implements OnInit {
}
```



## Создание Acceptance Tests для Shadow DOM

Shadow DOM создает проблему для тестирования компонентов в приложении с помощью приемочных cucumber тестов. К компонентам внутри Shadow DOM нельзя обратиться через стандартные селекторы из корневого документа.

Для этого необходимо использовать `shadow root` как корневой документ и через него обращаться к элементам компонента.

Shadow root — корневая нода компонента внутри Shadow DOM.

Shadow host — нода компонента, внутри которой размещается Shadow DOM.

В классе `BPMonline.BaseItem` реализованы базовые методы по работе с Shadow DOM.

**Важно.** В большинстве методов необходимо передавать селектор компонента, в котором находится Shadow DOM — `shadow host`.

Метод	Описание
<code>clickShadowItem</code>	Нажать на элемент внутри Shadow DOM компонента.
<code>getShadowRootElement</code>	По заданному css-селектору Angular компонента возвращает его <code>shadow root</code> , который можно использовать для дальнейших выборов элементов.
<code>getShadowWebElement</code>	Возвращает экземпляр элемента внутри Shadow DOM по заданному css-селектору. В зависимости от параметра <code>waitForVisible</code> ожидает его появления либо нет.
<code>getShadowWebElements</code>	Возвращает экземпляры элементов внутри Shadow DOM по заданному css-селектору.
<code>mouseoverShadowItem</code>	Навести курсор на элемент внутри Shadow DOM.
<code>waitForShadowItem</code>	Ожидает появления элемента внутри Shadow DOM компонента и возвращает его экземпляр.
<code>waitForShadowItemExist</code>	Ожидает появления элемента внутри Shadow DOM компонента.
<code>waitForShadowItemHide</code>	Ожидает скрывания элемента внутри Shadow DOM компонента.

**К сведению.** Примеры использования методов можно найти в классе `BPMonline.pages.ForecastTabUIV2`.

