

# Пакеты

Версия 7.17



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>4</b>
Классификация пакетов	4
Структура пакета	4
Зависимости и иерархия пакетов	5
Основные пакеты приложения	10
Пакет Custom	10
<b>Создать пользовательский пакет</b>	<b>11</b>
1. Создать новый пакет	11
2. Заполнить основные свойства пакета	12
3. Определить зависимости пакета	13
4. Проверить зависимости пакета Custom	14
<b>Привязать данные к пакету</b>	<b>14</b>
Алгоритм реализации примера	15
<b>Файловый контент пакетов</b>	<b>20</b>
Рекомендованная структура хранения файлового контента пакета	21
Версионирование файлового контента	24
Генерация вспомогательных файлов	24
Предварительная генерация статического файлового контента	25
Перенос изменений между средами	26
Совместимость с режимом разработки в файловой системе	27
Генерация клиентского контента при добавлении новой культуры	27
Изменения в объекте параметров, необходимом для формирования URL изображения	27
<b>Локализовать файловый контент</b>	<b>28</b>
Локализация с использованием конфигурационных ресурсов	28
Локализация с использованием плагина i18n	29
<b>Использовать TypeScript при разработке клиентской функциональности</b>	<b>32</b>
Установка TypeScript	32
Алгоритм реализации примера	32

# Общие принципы работы с пакетами

## Основы




Любой продукт Creatio представляет собой определенный набор пакетов. С их помощью выполняются все конфигурационные изменения.

## Классификация пакетов

**Пакет Creatio** — это совокупность конфигурационных элементов (схем, данных, SQL-скриптов, дополнительных библиотек), которые реализуют определенный блок функциональности. Физически пакет представляет собой каталог, содержащий определенный набор подкаталогов и файлов.

Чтобы расширить или изменить функциональность решения Creatio, нужно установить пакет, в котором реализованы все необходимые изменения.

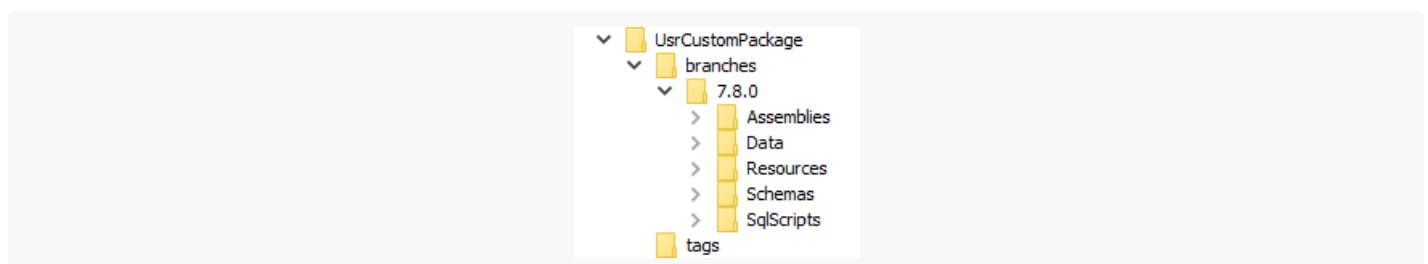
### Типы пакетов:

-  — предустановленные пакеты (недоступны для изменения). Поставляются вместе с системой и устанавливаются в рабочее пространство по умолчанию. К таким относятся пакеты с базовой функциональностью (например, `Base`, `NUI`), а также пакеты, созданные сторонними разработчиками. Такие пакеты устанавливаются из zip-архивов [как приложения marketplace](#) или с помощью [утилиты WorkspaceConsole](#).
-  — пользовательские пакеты, недоступные для изменения (созданы другими пользователями системы и заблокированы для изменения в системе контроля версий).
-  — пользовательские пакеты, доступные для изменения (созданы текущим пользователем либо загружены из системы контроля версий и доступны к изменению).

Конфигурационные элементы из предустановленных пакетов недоступны для изменения. Разработка дополнительной функциональности и модификация существующей выполняется исключительно в пользовательских пакетах.

## Структура пакета

При фиксации пакета в систему контроля версий в хранилище пакета создается папка с именем пакета, а внутри нее — каталоги `branches` и `tags`.



### 1. Папка `branches`

Здесь хранятся все **версии** данного пакета, каждая в отдельной вложенной папке, имя которой совпадает с номером версии пакета в системе, например, 7.8.0.

**Важно.** Структура, учитывающая версии пакета, осталась для совместимости с приложениями версий ниже 7.9.

## 2. Папка tags

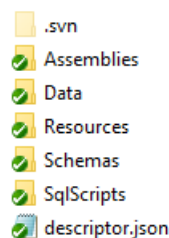
Предназначена для хранения меток. **Метки** в системе контроля версий — это "снимок" проекта в определенный момент времени, статическая копия файлов, необходимая для фиксации определенного важного этапа разработки.

Рабочая копия пакета сохраняется локально в файловой системе. Путь для хранения пакетов задается в конфигурационном файле `ConnectionStrings.config` в атрибуте `connectionString` элемента `defPackagesWorkingCopyPath`.

### ConnectionStrings.config

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

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 все текстовые ресурсы пакета, переведенные на разные языки.
- Папка `Files` — содержит [файловый контент](#) (начиная с версии 7.11.3).
- Файл `descriptor.json` — хранит метаданные пакета в формате JSON — его идентификатор, наименование, версия, зависимости и т. д.

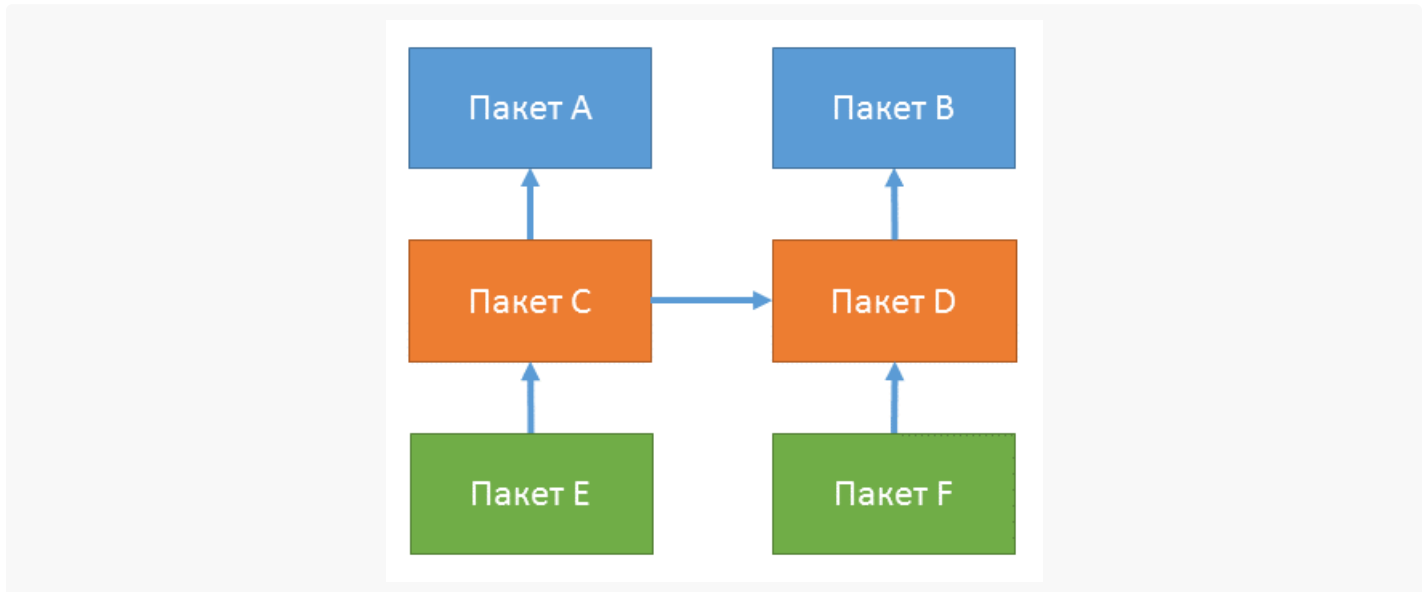
## Зависимости и иерархия пакетов

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

В архитектуре Creatio этот принцип был применен к механизму пакетов и реализован с помощью

**зависимостей пакетов** друг от друга. Каждый пакет содержит определенную функциональность приложения, которая не должна повторяться в других пакетах. Для того чтобы такую функциональность можно было использовать в любом другом пакете, необходимо пакет, содержащий эту функциональность, добавить в зависимости пакета, в котором она будет использоваться.

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.

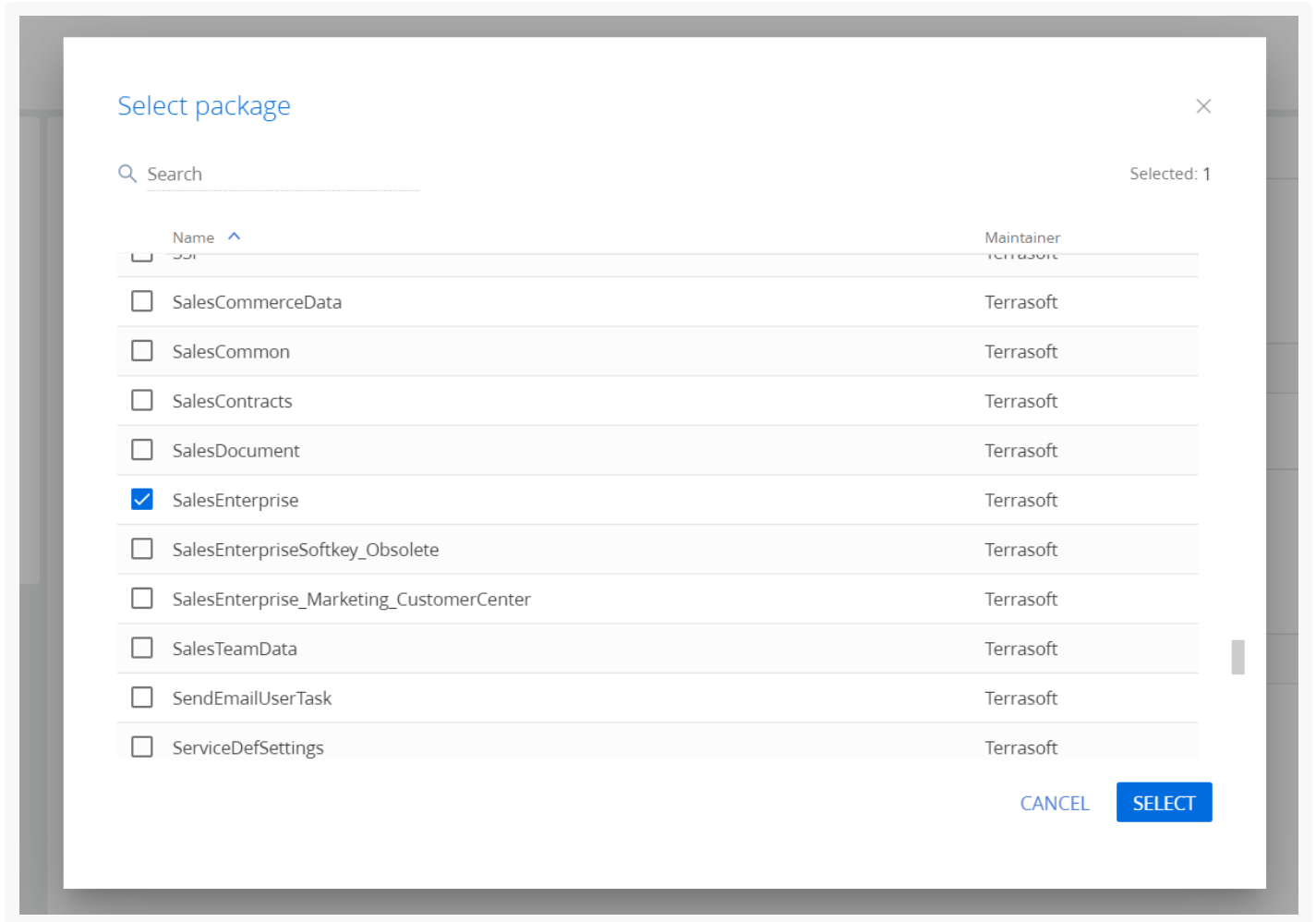


Зависимости пакетов формируют **иерархические цепочки**. Это означает, что в пакете доступна не только функциональность дочернего пакета, но и функциональность всех пакетов для которых дочерний пакет является родительским. Ближайшей аналогией иерархии пакетов является иерархия наследования классов в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

## Добавление зависимостей пакета

Зависимости можно добавить только в пользовательский пакет после его создания.

Для этого необходимо на странице редактирования пакета на вкладке [ *Зависимости* ] ([ *Dependencies* ]) на детали [ *Зависит от пакетов* ] ([ *Depends on packages* ]) нажать кнопку [ *Добавить* ] ([ *Add* ]). В появившемся окне справочника пакетов необходимо выбрать нужный пакет и нажать 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

Description

DEPENDENCIES

SYSTEM INFORMATION

Depends on Packages ⓘ

🔍 Search by package

Name ▴

📁 SalesEnterpriseSoftkey\_ENU

+ Add

Dependent Packages ⓘ

🔍 Search by package

Name ▴

📁 Custom

После создания пакета он автоматически добавляется в зависимости предустановленного пакета [ *Custom* ].

© 2021 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

Список зависимостей пакета хранится в его метаданных в свойстве `DependsOn` объекта, определенного в файле `descriptor.json`.

Свойство `DependsOn` является массивом 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": "7.8.0",
    "Name": "UsrDependentPackage",
    "ModifiedOnUtc": "\\Date(1522653150000)\\",
  }
}
```

```

    "Maintainer": "Customer",
    "DependsOn": [
      {
        "UId": "e14dcfb1-e53c-4439-a876-af7f97083ed9",
        "PackageVersion": "7.8.0",
        "Name": "SalesEnterprise"
      }
    ]
  }
}

```

## Основные пакеты приложения

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

Основные пакеты приложения

Название пакета	Содержимое
Base	Базовые схемы основных объектов, разделов системы и связанных с ними схем объектов, страниц, процессов и др.
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	Функциональность, связанная с пользовательским интерфейсом системы
DesignerTools	Схемы дизайнеров и их элементов
ProcessDesigner	Схемы дизайнера процессов

## Пакет [ Custom ]

В процессе своей работы мастер разделов или мастер деталей создает различные схемы, которые необходимо сохранить в пакет. Однако в только что установленном приложении доступных для изменения пользовательских пакетов нет, а в предустановленные пакеты изменения внести нельзя.

Для разрешения подобных конфликтов предназначен специальный предустановленный пакет [ Custom ]. Он позволяет добавлять схемы как вручную, так и с помощью мастеров.

### Особенности пакета [ Custom ]

1. Как и все предустановленные пакеты, пакет [ Custom ] нельзя добавить в систему контроля версий

(SVN). Поэтому его схемы можно перенести в другое приложение только при помощи [экспорта и импорта](#).

2. В отличие от других предустановленных пакетов, пакет [ *Custom* ] нельзя выгрузить в файловую систему при помощи [утилиты WorkspaceConsole](#).
3. 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 пакет [ *Custom* ] добавить нельзя.

**Важно.** Технически пользовательский пакет можно сделать последним в иерархии при помощи системной настройки Идентификатор пользовательского пакета (CustomPackageUId). Однако добавить в его зависимости предустановленные пакеты (в том числе и пакет Custom) можно только в том случае, если разработка ведется без использования SVN. Устанавливать вместо пакета Custom любой другой пакет в качестве последнего в иерархии крайне не рекомендуется!

В процессе своей работы мастер разделов или мастер деталей не только создает различные схемы, но и привязывает данные к **текущему пакету**. При этом, если текущим пакетом является пакет [ *Custom* ], то перенести привязанные данные в другой пользовательский пакет практически невозможно. Поэтому рекомендуется в качестве текущего пакета использовать любой пользовательский пакет, но не [ *Custom* ].

Для того чтобы поменять **текущий пакет**, необходимо использовать системную настройку [ *Текущий пакет* ] ( `CurrentPackageId` ).

Пакет [ *Custom* ] **рекомендуется использовать** в следующих случаях:


- Когда не предполагается перенос изменений в другое приложение.
- Если изменения выполняются при помощи мастеров или вручную, при этом объем изменений небольшой.
- Если нет необходимости использовать SVN.

При необходимости разработать значительный объем новой функциональности более целесообразным будет [создание нового пользовательского пакета](#) с использованием SVN.

# Создать пользовательский пакет

 Легкий

## 1. Создать новый пакет

Чтобы создать новый пользовательский пакет перейдите в раздел [ *Конфигурация* ] ([ *Configuration* ]) и нажмите кнопку  в области работы с пакетами.

При нажатии на кнопку будет отображено окно для создания нового пакета, в котором можно задать название и описание пакета.

**Package**

Name\*  
NewPackage

Description  
new package creation

Version control system repository  
SDKPackages

Version  
1.0.0

CANCEL CREATE AND ADD DEPENDENCIES SAVE

## 2. Заполнить основные свойства пакета

Поля карточки пакета:

- [ *Название* ] ([ *Name* ]) — название пакета. Обязательное для заполнения поле. Не может совпадать с названием уже существующих пакетов.
- [ *Описание* ] ([ *Description* ]) — описание пакета, например, расширенная информация о функциональности, которая реализуется в пакете. Не обязательное поле.
- [ *Хранилище системы контроля версий* ] ([ *Version Control System Repository* ]) — название хранилища системы контроля версий, в котором будут фиксироваться изменения пакета. Список доступных хранилищ формируется из списка хранилищ системы контроля версий. Хранилища, которые находятся в списке хранилищ конфигурации, но не помечены как активные, не попадут в выпадающий список доступных хранилищ. Поле является обязательным для заполнения.

**Важно.** Поле [ *Хранилище системы контроля версий* ] заполняется при создании нового пакета и в дальнейшем недоступно для редактирования. Если система контроля версий не используется, то это поле не отображается.

- [ *Версия* ] ([ *Version* ]) — версия пакета. Обязательное для заполнения поле. 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. Версия пакета не обязательно должна совпадать с версией приложения.

Содержимое основных полей карточки пакета будет сохранено в его метаданных.

### Метаданные основных полей карточки пакета

```
{
  "Descriptor": {
    "UId": "1c1443d7-87df-4b48-bfb8-cc647755c4c1",
    "PackageVersion": "7.8.0",
    "Name": "NewPackage",
    "ModifiedOnUtc": "\/Date(1522657977000)\/",
    "Maintainer": "Customer",
    "DependsOn": []
  }
}
```

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

## 3. Определить зависимости пакета

Чтобы созданный пакет имел всю функциональность, которая заложена в систему, определите для него зависимости. Нажмите на кнопку [ *Создать и добавить зависимости* ] ([ *Create and add dependencies* ]). Пакет будет создан, а затем откроется страница редактирования пакета.

На вкладке [ *Зависимости* ] ([ *Dependencies* ]) в детали [ *Зависит от пакетов* ] ([ *Depends on packages* ])

добавьте необходимые зависимости.

При этом достаточно указать самый последний пакет в иерархии предустановленных пакетов.

**На заметку.** [Пакет \[ Custom \]](#) добавить в зависимости нового пакета нельзя.

## 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



При поставке пользователям пакетов часто возникает потребность предоставлять вместе с разработанной функциональностью также и некоторые данные. Это может быть, например, наполнение справочников, новые системные настройки, демонстрационные записи раздела и т. п.

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

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

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

+ Add ▾ Data ▾ Filters ▾ Search ⚙						
Name ▾	Title	Status	Type	Object	Modified on	Package
SysModule_SectionManager_8a3879e5f91c49cf81a9d7dd5b3a47c4			Data	SysModule	4/23/2018, 1:12:06 PM	sdkBookExample
SysModuleInWorkplace_SectionInWorkplaceManager_a69d5bf641024326adeca22d02f6dde6			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

# Алгоритм реализации примера

## 1. Создать новый раздел [ Книги ]

**Важно.** Функциональность нового раздела следует создавать в [отдельном пакете разработки](#). Чтобы мастер раздела создавал схемы в пакете разработки, необходимо выбрать этот пакет в колонке [ Значение по умолчанию ] ([ Default value ]) системной настройки [ Текущий пакет ] ([ Current package ]). После завершения работы мастера в качестве текущего можно установить пакет [ Custom ].

Для создания нового раздела [ Книги ] ([ Books ]) воспользуйтесь [мастером разделов](#).

Свойства раздела [ Книги ] ([ Books ])

Books: General section properties

SAVE CANCEL < SECTION PAGE BUSINESS RULES CASES BUSINESS PROCESSES >

Select basic properties for section:

Title\* Books

Code\* UsrBook

Menu icon

Page settings:

☒ One page for all records

☐ Multiple pages

Свойства страницы редактирования записей

Books: Page

SAVE

CANCEL

<

SECTION

PAGE

BUSINESS RULES

CASES

BUSINESS PROCESSES

>

Page elements

Books

New column

Boolean

Date

0.5 Decimal

123 Integer

Lookup

String

ISBN

Author \*

Publisher \*

0.5 Price

Name \*

Description

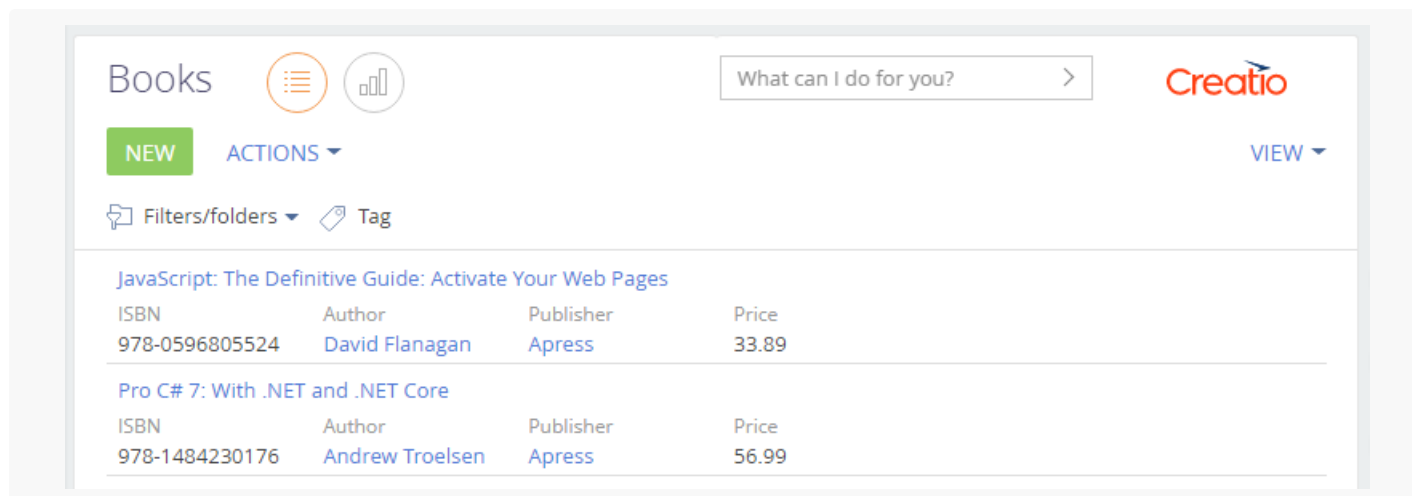
Свойства колонок страницы редактирования записей раздела

Заголовок	Название (Код в БД)	Тип данных
Название (Name)	UsrName	Строка (String).
Описание (Description)	UsrDescription	Строка (String). Многострочный текст (Multiline text).
ISBN	UsrISBN	Строка (String).
Автор (Author)	UsrAuthor	Справочник [ Контакт ] ([ Contact ]). Значение колонки будет привязано к одной из записей раздела [ Контакты ] ([ Contacts ]).
Издатель (Publisher)	UsrPublisher	Справочник [ Контрагент ] ([ Account ]). Значение колонки будет привязано к одной из записей раздела [ Контрагенты ] ([ Accounts ]).
Стоимость (Price)	UsrPrice	Дробное число (Decimal).

2. Добавить в раздел необходимые записи

Добавьте в раздел две демонстрационные записи. При необходимости также создайте записи в связанных разделах [ Контакты ] ([ Contacts ]) и [ Контрагенты ] ([ Accounts ]).

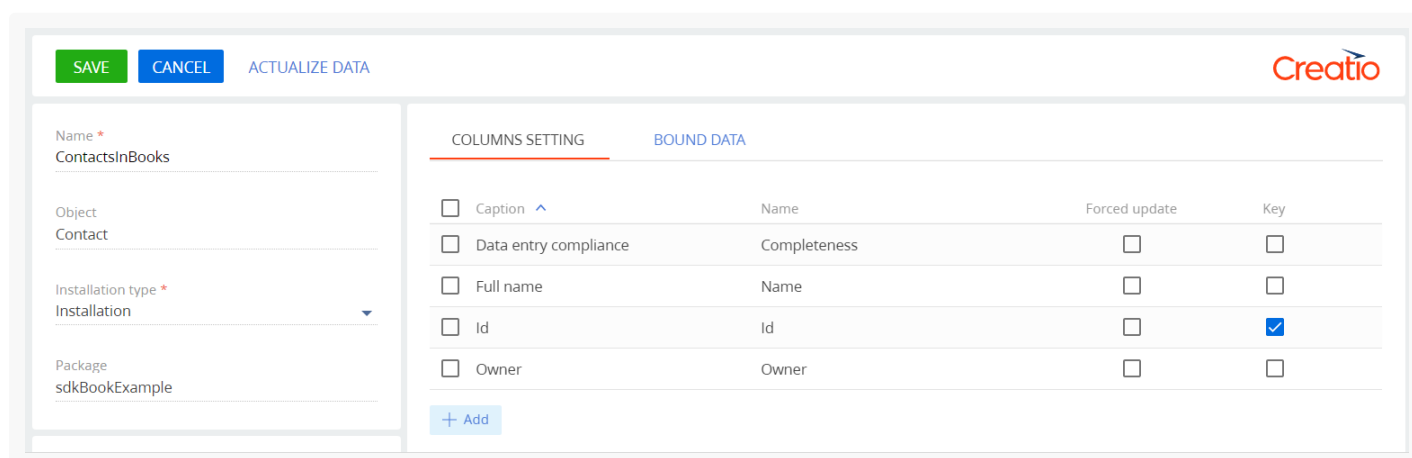




### 3. Привязать к пакету данные о контактах

Поскольку записи раздела [ Книги ] ([ Books ]) связаны с записями раздела [ Контакты ] ([ Contacts ]) по колонке [ UsrAuthor ], то сначала необходимо привязать к пакету сведения об авторах. Для этого выполните действие [ Добавить ] -> [ Данные ] ([ Add ] -> [ Data ]) в рабочей области раздела [ Конфигурация ] ([ Configuration ]) и установите следующие свойства страницы привязки данных:

1. [ Название ] ([ Name ]) — "ContactsInBooks".
2. [ Объект ] ([ Object ]) — "Контакт" ("Contact").
3. [ Тип установки ] ([ Installation type ]) — "Установка" ("Installation").
4. На вкладке [ Настроить колонки ] ([ Columns setting ]), нажав кнопку [ Добавить ] ([ Add ]), добавьте заполненные колонки. Колонка [ Id ] должна быть выбрана обязательно.
5. На вкладке [ Прикрепленные данные ] ([ Bound data ]), нажав кнопку [ Добавить ] ([ Add ]), добавьте все необходимые данные.



SAVE

CANCEL

ACTUALIZE DATA

Name \*

ContactsInBooks

Object

Contact

Installation type \*

Installation

Package

sdkBookExample

COLUMNS SETTING

BOUND DATA

Search

<input type="checkbox"/>	Data entry compliance	Full name	Id	Owner
<input type="checkbox"/>	10	David Flanagan	147b3eb4-fb4d-4c99-ba71-177bd716b63c	Supervisor
<input type="checkbox"/>	10	Andrew Troelsen	aae71a25-26e0-4d7c-9c45-dac444cf5095	Supervisor

+ Add

## 4. Привязать к пакету данные о контрагентах

Выполните действие [ *Добавить* ] -> [ *Данные* ] ([ *Add* ] -> [ *Data* ]) в рабочей области раздела [ *Конфигурация* ] ([ *Configuration* ]) и установите следующие свойства страницы привязки данных:

- [ *Название* ] ([ *Name* ]) — "AccountsInBooks".
- [ *Объект* ] ([ *Object* ]) — "Контрагент" ("Account").
- [ *Тип установки* ] ([ *Installation type* ]) — "Установка" ("Installation").
- На вкладке [ *Настроить колонки* ] ([ *Columns setting* ]), нажав кнопку [ *Добавить* ] ([ *Add* ]), добавьте заполненные колонки. Колонка [ *Id* ] должна быть выбрана обязательно.
- На вкладке [ *Прикрепленные данные* ] ([ *Bound data* ]), нажав кнопку [ *Добавить* ] ([ *Add* ]), добавьте все необходимые данные.

CLOSE

ACTUALIZE DATA

Name \*

AccountsInBooks

Object

Account

Installation type \*

Installation

Package

sdkBookExample

COLUMNS SETTING

BOUND DATA

Caption ^

Name

Forced update

Key

<input type="checkbox"/>	Data entry compliance	Completeness	<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>	Id	Id	<input type="checkbox"/>	<input checked="" type="checkbox"/>	
<input type="checkbox"/>	Name	Name	<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>	Owner	Owner	<input type="checkbox"/>	<input type="checkbox"/>	

+ Add

SAVE

CANCEL

ACTUALIZE DATA

Name \*

AccountsInBooks

Object

Account

Installation type \*

Installation

Package

sdkBookExample

COLUMNS SETTING

BOUND DATA

Search

<input type="checkbox"/>	Id	Owner	Name	Data entry compliance
<input type="checkbox"/>	097ced7b-d9cf-428a-a4b4-654c9adf2e47	Supervisor	Apress	10

+ Add

## 5. Привязать к пакету данные пользовательского раздела

Выполните действие [ *Добавить* ] -> [ *Данные* ] ([ *Add* ] -> [ *Data* ]) в рабочей области раздела [ *Конфигурация* ] ([ *Configuration* ]) и установите следующие свойства страницы привязки данных:

1. [ *Название* ] ([ *Name* ]) — "Books".
2. [ *Объект* ] ([ *Object* ]) — "Книги" ("Books").
3. [ *Тип установки* ] ([ *Installation type* ]) — "Установка" ("Installation").
4. На вкладке [ *Настроить колонки* ] ([ *Columns setting* ]), нажав кнопку [ *Добавить* ] ([ *Add* ]), добавьте заполненные колонки. Колонка [ *Id* ] должна быть выбрана обязательно.
5. На вкладке [ *Прикрепленные данные* ] ([ *Bound data* ]), нажав кнопку [ *Добавить* ] ([ *Add* ]), добавьте все необходимые данные.

Column	Name	Forced update	Key
<input type="checkbox"/> Caption			
<input type="checkbox"/> Author	UsrAuthor	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Description	UsrDescription	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> ISBN	UsrISBN	<input type="checkbox"/>	<input type="checkbox"/>
<input checked="" type="checkbox"/> Id	Id	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/> Name	UsrName	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Price	UsrPrice	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Publisher	UsrPublisher	<input type="checkbox"/>	<input type="checkbox"/>

Id	Name	Description	Publisher	Price	Author	ISBN
<input type="checkbox"/> ea13a9e3-e9b1-46d7-bef2-063532e0916b	JavaScript: The Definitive Guide: Activate Your Web Pages	Since 1996, JavaScript: The Definitive Guide has been the bible for JavaScript programmers —a programmer's guide and comprehensive reference to the core language and to the client-side JavaScript APIs defined by web browsers	Apress		David Flanagan	978-0596805524
<input type="checkbox"/> c2564aa7-5aff-4ace-8d00-1e5ccf5d2b64	Pro C# 7: With .NET and .NET Core	Dive in and discover why Pro C# has been a favorite of C# developers worldwide for over 15 years.	Apress		Andrew Troelsen	978-1484230176

В результате выполнения примера к пакету будут привязаны три дополнительных набора данных для трех разделов.

<div> <div>+ Add</div> <div>Data</div> <div>Filters</div> <div>Search</div> <div></div> </div>							
Name	Title	Status	Type	Object	Modified on	Package	
AccountsInBooks			Data	Account	5/23/2018, 11:26:58 AM	sdkBookExample	
Books			Data	UsrBook	4/23/2018, 4:45:09 PM	sdkBookExample	
ContactsInBooks			Data	Contact	5/23/2018, 11:26:00 AM	sdkBookExample	
SysDetail_DetailManager_b78e48663abc45cb916779059502af6b			Data	SysDetail	4/23/2018, 1:35:29 PM	sdkBookExample	
SysImage_f8d3f0121ed2433a996610d4b00cf09a			Data	SysImage	4/23/2018, 1:12:07 PM	sdkBookExample	
SysModuleEdit_SysModuleEditManager_e3a3124a5cd346919574d2d5f5f750c4			Data	SysModuleEdit	4/23/2018, 1:12:05 PM	sdkBookExample	
SysModuleEntity_SysModuleEntityManager_8654e87e4fd34e0a91d903ad427373d9			Data	SysModuleEntity	4/23/2018, 1:12:04 PM	sdkBookExample	
SysModuleInWorkplace_SectionInWorkplaceManager_a69d5bf b41024326adeca22d02f6dde6			Data	SysModuleInWorkplace	4/23/2018, 1:12:07 PM	sdkBookExample	
SysModule_SectionManager_8a3879e5f91c49cf81a9d7dd5b3a47c4			Data	SysModule	4/23/2018, 1:12:06 PM	sdkBookExample	

Пакет можно выгрузить в архив, используя [функциональность экспорта](#). После установки пакета в другое приложение все привязанные записи отобразятся в соответствующих разделах.

## Файловый контент пакетов

 **Сложный**

**Файловый контент пакетов** — любые файлы (\*.js -файлы, \*.css -файлы, изображения и др.), используемые приложением, добавленные в пользовательские пакеты.

Файловый контент не обрабатывается web-сервером и является статическим, что в свою очередь повышает скорость работы приложения.

Для повышения общей производительности приложения и снижения нагрузки на БД весь файловый контент можно предварительно сгенерировать в специальном каталоге приложения. При запросе файлового контента сервер IIS ищет запрашиваемый контент в этом каталоге и сразу же отправляет его клиентскому приложению.

Преимущества и недостатки разных способов использования файлового контента

Преимущества	Недостатки
Генерация клиентского контента "на лету"	
Не нужно предварительно генерировать клиентский контент	Нагрузка на процессор при вычислении иерархии пакетов, схем и формировании контента
	Нагрузка на базу данных для получения иерархии пакетов, схем и формирования их контента
	Потребление памяти для кеширования клиентского контента
Использование предварительно сгенерированного клиентского контента	
Минимальная нагрузка на процессор (CPU)	Необходимо предварительно генерировать клиентский контент
Отсутствуют запросы в базу данных	
Клиентский контент кешируется средствами IIS	

**Важно.** Файловый контент является неотъемлемой частью приложения Creatio и всегда хранится в каталоге `...\Terrasoft.WebApp\Terrasoft.Configuration\Pkg\<Название пакета>\Files`.

В пакет могут быть добавлены любые файлы, однако использоваться будут только файлы, необходимые для клиентской части Creatio.

## Рекомендованная структура хранения файлового контента пакета

Для использования файлового контента в [структуру пакета](#) добавлен каталог `Files`. Рекомендуется соблюдать структуру каталога `Files`, приведенную ниже.

### Рекомендуемая структура каталога `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` — каталог с `*.js`-файлами исходных кодов на языке JavaScript.
- `css` — каталог с `*.css`-файлами стилей.
- `less` — каталог с `*.less`-файлами стилей.
- `img` — каталог с изображениями.
- `res` — каталог с файлами ресурсов.
- `descriptor.json` — дескриптор файлового контента.

Чтобы добавить файловый контент в пакет достаточно просто поместить файл в соответствующий подкаталог директории `Files` необходимого пакета. Каталог `Files` будет размещен по пути

```
... \Terrasoft.WebApp\Terrasoft.Configuration\Pkg\<Название пакета>\Files .
```

## Дескриптор файлового контента

В файле `descriptor.json`, размещенном в каталоге `Files`, хранится информация о `bootstrap`-файлах пакета.

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"
  ]
}

```

```
]
}
```

## Bootstrap-файлы пакета

Это `*.js`-файлы, которые позволяют управлять загрузкой клиентской конфигурационной логики. Файл не имеет четкой структуры.

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.
    }
  });
})();
```

**Важно.** Все bootstrap-файлы загружаются асинхронно после загрузки ядра, но до загрузки конфигурации.

## Загрузка bootstrap-файлов

Для корректной загрузки bootstrap-файлов в директории статического контента генерируется вспомогательный файл `_FileContentBootstraps.js`. Это файл, в котором содержится информация 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"
  ]
};
```

## Версионирование файлового контента

Для корректной работы версионирования файлов файлового контента в директории 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.

**На заметку.** После установки файлового контента нет необходимости в очистке кэша браузера.

### 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`

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

- `operation` — название операции. `BuildConfiguration` — операция компиляции конфигурации.
- `useStaticFileContent` — признак использования статического контента. Должен иметь значение `false`.



- `usePackageFileContent` — признак использования файлового контента пакетов. Должен иметь значение `true`.

Остальные параметры `WorkspaceConsole` описаны в [статье](#).

В результате выполнения операции в каталоге со статическим контентом

`...\Terrasoft.WebApp\conf\content` будут сгенерированы вспомогательные файлы `_FileContentBootstraps.js` и `_FileContentDescriptors.js`.

## Предварительная генерация статического файлового контента

Файловый контент в данном случае генерируется в специальный каталог `.\Terrasoft.WebApp\conf`. В нем содержатся `*.js`-файлы с исходным кодом схем, `*.css`-файлы стилей и `*.js`-файлы ресурсов для всех культур приложения, а также изображения.

**Важно.** Для каталога `.\Terrasoft.WebApp\conf` должны быть установлены права на модификацию (чтение и запись файлов и вложенных каталогов, а также удаление каталога) для пользователя пула IIS, в котором запущено приложение. В противном случае приложение Creatio не сможет сгенерировать статический контент.

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

## Действия, при которых выполняется генерация файлового контента

Первичная или повторная генерация статического файлового контента выполняется при следующих действиях в системе:

- 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 все ]` и `[ Компилировать измененное ]` в разделе `[ Конфигурация ]`.

**Важно.** При удалении схем или пакетов из раздела `[ Конфигурация ]` необходимо выполнить действие `[ Компилировать измененное ]` или `[ Компилировать все ]`.

При установке или обновлении пакета из SVN также необходимо выполнить действие `[ Компилировать все ]`.

**На заметку.** Только действие `[ Компилировать все ]` выполняет полную регенерацию файлового статического контента. Остальные действия выполняют регенерацию только измененных схем.

## Генерация файлового контента с помощью утилиты WorkspaceConsole

Параметры операции `BuildConfiguration`

Параметр	Описание
<code>workspaceName</code>	Название рабочего пространства. По умолчанию <code>Default</code> .
<code>destinationPath</code>	Каталог, в который 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>Если установлено значение <code>true</code>, то выполняется генерация контента по всем схемам. Если <code>false</code>, то выполняется генерация для измененных схем.</p> <p>Необязательный параметр. По умолчанию установлено значение <code>false</code>.</p>

### Пример использования 1

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

### Пример использования 2

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

## Перенос изменений между средами

Файловый контент является неотъемлемой частью пакета. Он фиксируется в хранилище системы контроля версий наравне с остальным содержимым пакета. В дальнейшем он может быть перенесен на другую среду разработки при помощи [SVN](#).

**Важно.** Для переноса изменений на тестовую и промышленную среды рекомендуется использовать [встроенные средства Creatio](#).

**Важно.** При установке пакетов каталог `files` будет создан только в том случае, если он не пустой. Если этот каталог создан не был, то для начала разработки его нужно создать вручную.

## Совместимость с режимом разработки в файловой системе

На текущий момент режим разработки в файловой системе (РФС) не совместим с получением клиентского контента из предварительно сгенерированных файлов. Для корректной работы с РФС необходимо отключить получение статического клиентского контента из файловой системы. Для отключения данной функциональности нужно установить значение `false` для флага

`UseStaticFileContent` в файле `Web.config`.

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

## Генерация клиентского контента при добавлении новой культуры

После добавления новых культур из интерфейса приложения необходимо выполнить действие [ *Компилировать все* ] в разделе [ *Конфигурация* ].

**Важно.** Если пользователь не может войти в систему после добавления новой культуры, то необходимо зайти в раздел [ *Конфигурация* ] по ссылке `http://[Путь к приложению]/0/dev` и выполнить действие [ *Компилировать все* ].

## Изменения в объекте параметров, необходимом для формирования URL изображения

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

### Структура объекта `params`

```
config: {
  params: {
    schemaName: "",
```

```

    resourceName: "",
    hash: "",
    resourceItemExtension: ""
  }
}

```

- `schemaName` — название схемы (строка);
- `resourceItemName` — название изображения в Creatio (строка);
- `hash` — хэш изображения (строка);
- `resourceItemExtension` — расширение файла изображения (например, ".png").

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

#### Пример корректного формирования конфигурационного объекта параметров

```

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

```

## Локализовать файловый контент



Сложный

### Локализация с использованием конфигурационных ресурсов

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

#### Пример исходного кода модуля

```

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

```

Для подключения локализуемых ресурсов в модуль, который определяется в файловом контенте пакета, достаточно сослаться на модуль с ресурсами.

### Подключение локализуемых ресурсов в модуль

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

## Локализация с использованием плагина i18n

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

Чтобы выполнить локализацию файлового контента с помощью плагина `RequireJS i18n`, выполните следующие действия:

1. Поместите плагин в директорию с `*.js`-файлами исходных кодов

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

Здесь `MyPackage1` — рабочий каталог пакета `MyPackage1`.

2. Создайте каталог `..\MyPackage1\content\nls` и поместите в него один или несколько `*.js`-файлов с локализуемыми ресурсами. Имена файлов могут быть произвольными. Содержимое файлов — AMD модули, объекты которых имеют следующую структуру:

- Поле `"root"`, содержащее 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, имена которых являются стандартными кодами поддерживаемых культур (например `"en-US"`, `"ru-RU"`), а значение имеет логический тип. Может быть `true`, если поддерживаемая культура включена, и `false` — если отключена.

Например, добавлен файл `..\MyPackage1\content\js\nls\ContactSectionV2Resources.js`.

```
..\MyPackage1\content\js\nls\ContactSectionV2Resources.js
```

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

3. В каталоге `..\MyPackage1\content\nls` создайте папки, названия которых соответствуют коду той культуры, локализация которой будет в них размещена (например `"en-US"`, `"ru-RU"`).

#### Структура каталога `MyPackage1` когда поддерживаются русская и английская культуры

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

4. В каждый созданный каталог локализации поместите такой же набор файлов `*.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 необходимо создать два файла `ContactSectionV2Resources.js`.

`ContactSectionV2Resources.js`, соответствующий английской культуре

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

`ContactSectionV2Resources.js`, соответствующий русской культуре

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

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

5. Отредактируйте содержимое файла `bootstrap.js`:

- Подключите плагин `i18n`, указав его название в виде псевдонима `"i18n"` в конфигурации путей `RequireJS` и прописав соответствующий путь к нему в свойстве `paths`.
- 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).

- 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 `..\MyPackage1\content\js\bootstrap.js`

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

- Используйте ресурсы в нужном модуле, указав в массиве зависимостей требуемый модуль с ресурсами с префиксом "i18n!". Например, если необходимо использовать локализуемую строку

`FileContentActionDescr` как заголовок для нового действия в разделе [ *Контакты* ], то в файл `..\MyPackage1\content\js\ContactSectionV2.js` нужно добавить содержимое.

#### `..\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.MenuSepara
                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



Файловый контент позволяет использовать при разработке клиентской функциональности компилируемые в JavaScript языки, например, TypeScript. Подробнее о TypeScript можно узнать на сайте <https://www.typescriptlang.org>.

## Установка TypeScript

Одним из способов установки инструментария TypeScript является использование менеджера пакетов NPM для `Node.js`. Для этого необходимо выполнить в консоли Windows следующую команду:

### Команда для установки инструментария TypeScript

```
npm install -g typescript
```

**Важно.** Прежде чем устанавливать TypeScript с помощью NPM, проверьте наличие среды выполнения `Node.js` в вашей операционной системе. Скачать инсталлятор можно по на сайте <https://nodejs.org>.

**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.

## Алгоритм реализации примера



## 1. [Перейти](#) в режим разработки в файловой системе

## 2. Создать структуру хранения файлового контента

Общий принцип [создания рекомендуемой структуры хранения файлового контента](#):

1. В выгруженном в файловую систему пользовательском пакете создайте каталог `Files`.
2. В каталог `Files` добавьте папку `src`, а внутри нее создайте подкаталог `js`.
3. В каталог `Files` добавьте файл `descriptor.json`.

`descriptor.json`

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

4. В каталог `Files\src\js` добавьте файл `bootstrap.js`.

`bootstrap.js`

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

**На заметку.** Указанный в `bootstrap.js` файл `LettersOnlyValidator.js` будет скомпилирован на шаге 4.

## 3. Реализовать класс валидации значения 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 этом же каталоге 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();
```

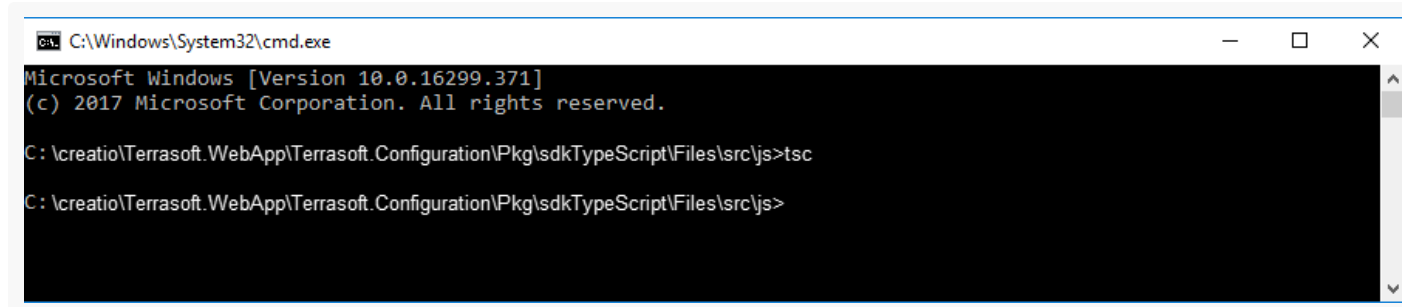
## 4. Выполнить компиляцию исходных кодов TypeScript в исходные коды JavaScript

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

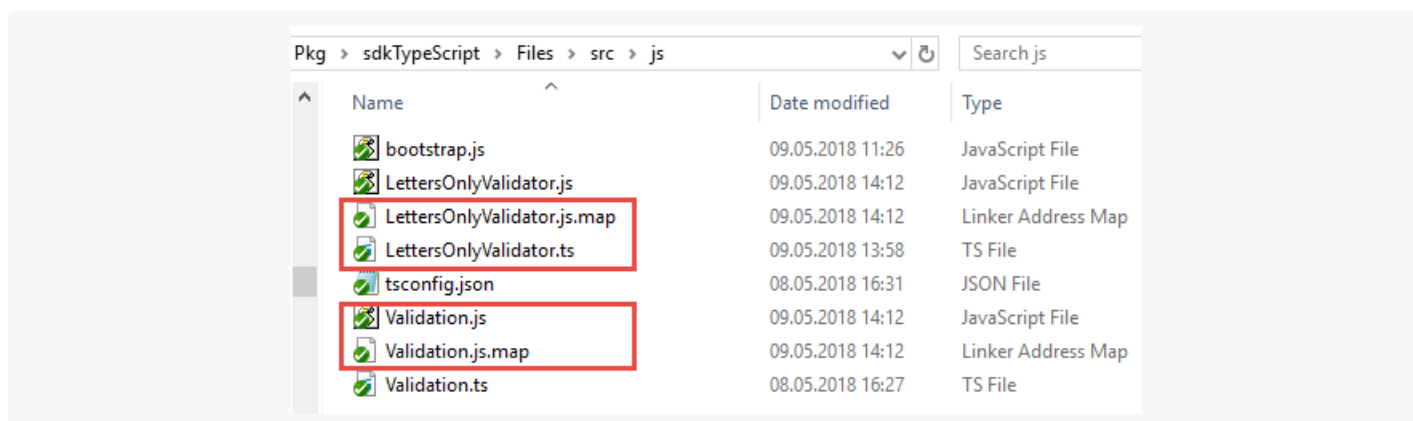
#### `tsconfig.json`

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

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

```

## 5. Выполнить генерацию вспомогательных файлов

Для [генерации](#) вспомогательных файлов `_FileContentBootstraps.js` и `FileContentDescriptors.js` выполните следующие действия:

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. Выполните компиляцию приложения (действие [ *Компилировать все* ] ([ *Compile all items* ])).

**На заметку.** Этот шаг необходимо выполнять для применения изменений в файле `bootsrtap.js`. Для его выполнения также можно использовать утилиту `WorkspaceConsole`.

## 6. Использовать валидатор в схеме Creatio

В разделе [ *Конфигурация* ] ([ *Configuration* ]):

1. Выполните загрузку пакетов из файловой системы (действие [ *Обновить пакеты из файловой системы* ] ([ *Update packages from file system* ])).
2. [Создайте замещающую схему](#) страницы редактирования записи контрагента.

Module

Code  
AccountPageV2

Title \*  
Account edit page

Parent object \*  
Account edit page

Package  
sdkTypeScript

Description

CANCEL APPLY

3. Выполните выгрузку пакетов в файловую систему (действие [ *Выгрузить пакеты в файловую систему* ] ([ *Download packages to file system* ])).
4. 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() {
```

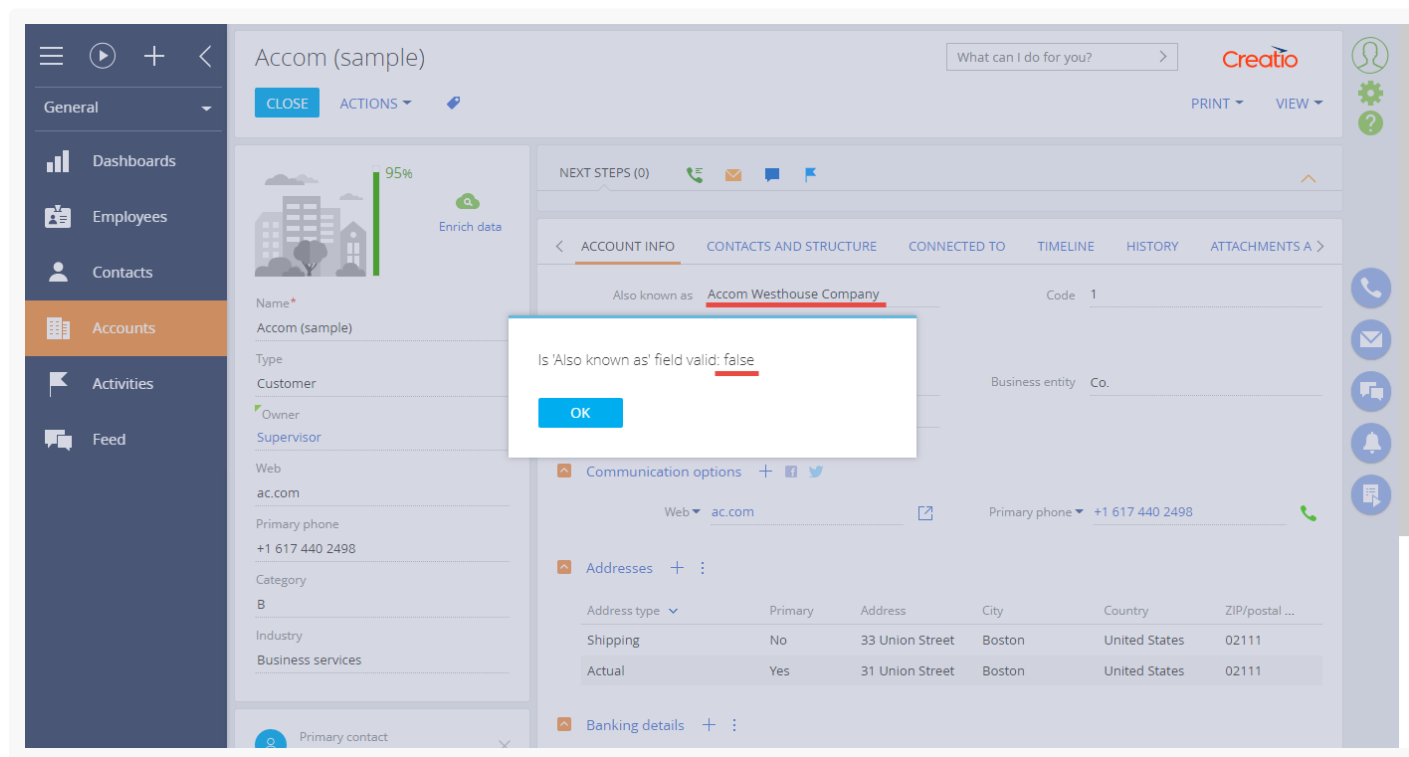
```

        // Вызов метода валидации.
        this.validateMethod();
        // Вызов базовой функциональности.
        this.callParent(arguments);
    }
},
diff: /**SCHEMA_DIFF*/ [] /**SCHEMA_DIFF*/
});
});

```

После сохранения файла с исходным кодом схемы и обновления страницы приложения на странице редактирования контрагента при сохранении записи будет выполняться [валидация](#) и отображаться соответствующее предупреждение.

Неправильно заполненное поле



Правильно заполненное поле

